PROGRAMMING

CT103

Dr. Karl Mason

Karl.Mason@nuigalway.ie

How to contact me

- Dr. Karl Mason
  - karl.mason@nuigalway.ie

  - Room 418 Top Floor IT Building

  - Staff profile: https://www.nuigalway.ie/our-research/people/engineering-and-informatics/karljohnmason/

Course Info

- Lectures – 2 hours per week
  - **Monday 11 am**, Dillon Theatre.
  - **Wednesday 11am**, McMunn Theatre.
- Attendance will be take at each lecture

- Labs – 2 hours per week
  - Tuesdays IT102
  - **Group 1**: 2pm-4pm **Group 2**: 4pm-6pm

- Tutorials
  - **Monday 4pm**, IT202
  - Attendance not mandatory, only if you need extra help.
  - **Starts 4th October**, none in September.

Lab Info

- We will be using Microsoft Visual Studio in the labs to do the assignments
  - You will have access to these tools via the College licence (http://nuigalway-engineering-dreamspark.onthehub.com/)
  - There are many other tools that can also be used (basically called “C Compilers” – more on that later)

Lab Groups

- **Group 1:**
  - 2pm – 4pm surnames **A** to **K**

- **Group 2:**
  - 4pm to 6pm surnames **L** to **Z**

Book

- Course text:
  - Any decent introduction to programming in C will do
    - “Absolute Beginners Guide to C” by Greg Perry, Published by SAMS
    - “C for Dummies” by Dan Gookin, published by Wiley
    - “C How to Program” by Deitel & Deitel, published by Prentice Hall

  - ***<u>DO NOT</u> get a book on C++ or C# by mistake***

  - **Plenty of C books in the library to borrow for free**

Marking

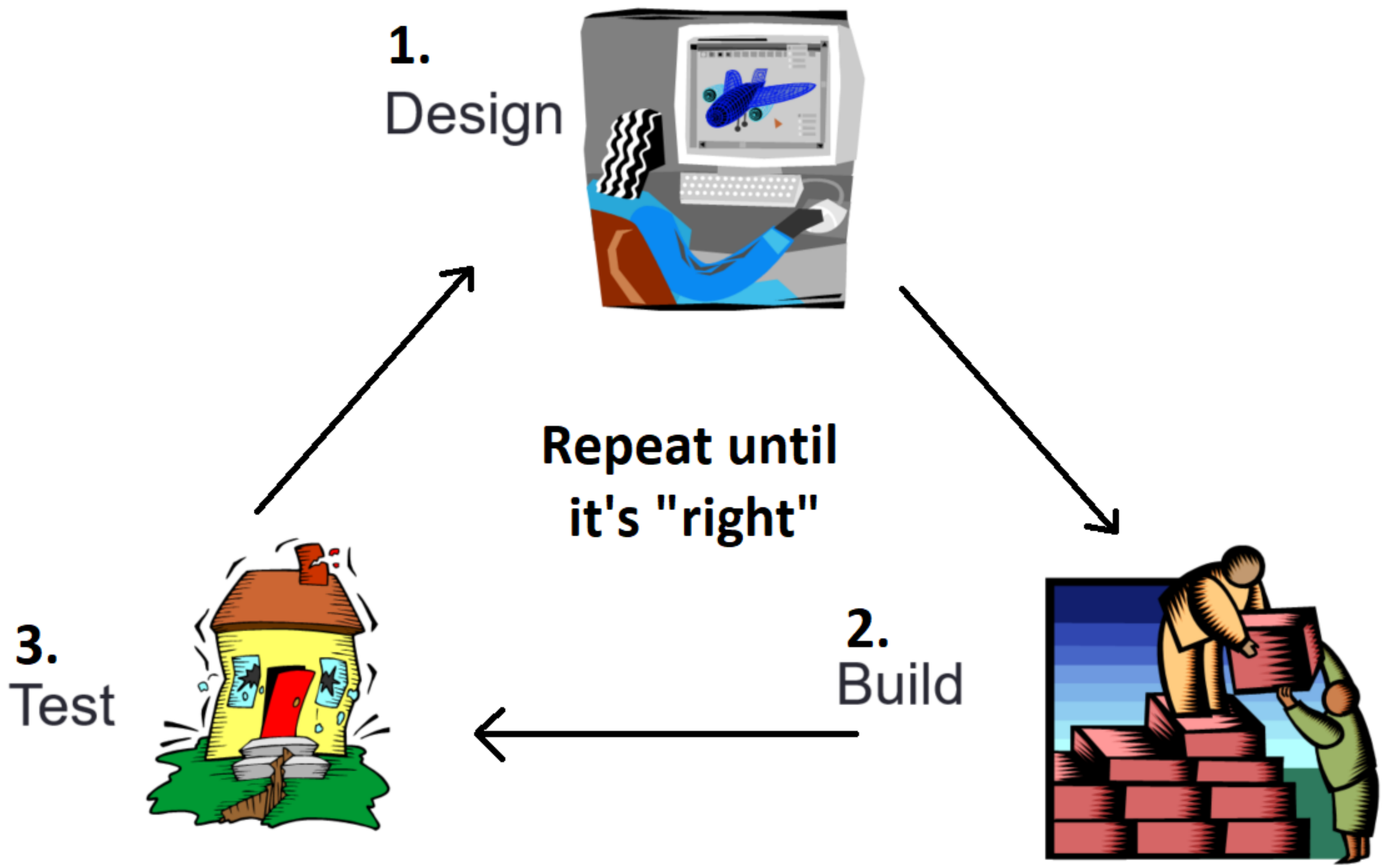
- Lab Assignments – 25%
  - Submitted on Blackboard at the end of each lab.
  - Lab assignments in semester 1 **and** 2.
- Written exam – 75%
  - End of semester **2** only.

Algorithm

- An Algorithm is a sequence of instructions for the computer to follow

- It usually describes:
    - The inputs you need to accomplish the task
    - The formula you need to apply to the inputs or any other manipulation of the inputs required
    - The end result or output

Steps involved in writing software



Programming building blocks

- Lists of instructions
  - Like cooking, e.g. “beat eggs; add flour and sugar; mix; pour into baking tin; bake at 180 for 20 minutes”
- **IF Statements** / Conditions
  - Like “IF it is raining, take an umbrella”
- **Loops** - Repeating behavior
  - 2 loop types: **For** loops and **While** loops.
  - For example “jump up and down 3 times”, or “while there is petrol left, keep driving”
- Computing results
  - Performing a sequence of steps in a particular order to get the result
  - For example to calculate your BMI divide your weight by your height squared

PseudoCode

- Example #1 - Computing Sales Tax
  - Pseudo-code for task of computing the **final price** of an item after calculating **sales tax**. Note the three types of instructions: input (get), process/calculate (=) and output (display)

```
1.  get price of item
2.  get sales tax rate
3.  sales tax = price of time x sales tax rate
4   final price = price of item + sales tax
5.  display final price
6.  halt
```

- Variables: price of item, sales tax rate, sales tax, final price
- Note that the operations are numbered and each operation is unambiguous
- We also extract and list all variables used in our pseudo-code. This will be useful when translating pseudo-code into a programming language

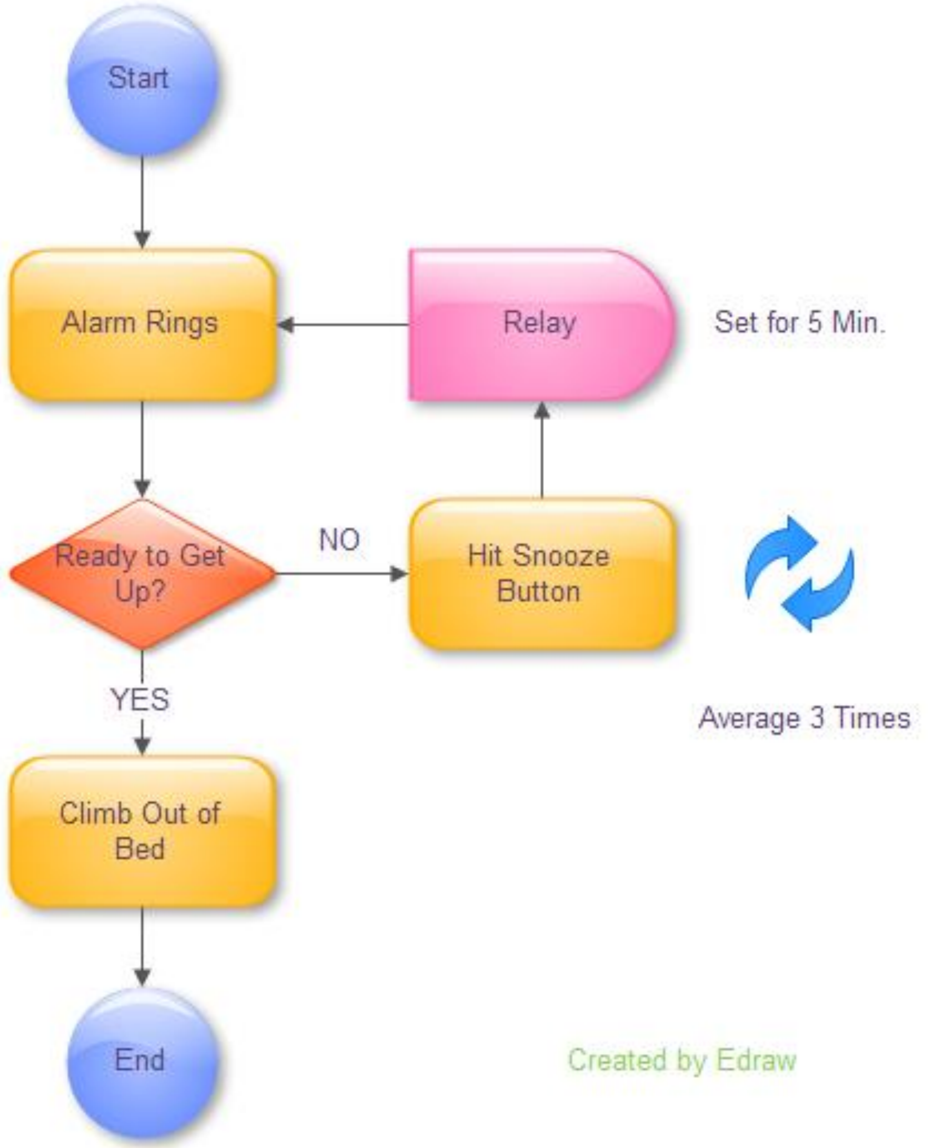
PseudoCode

- Example #2 - Computing Weekly Wages:
- **Gross pay** depends on the pay rate and the number of hours worked per week. However, if you work more than 40 hours, you get paid time-and-a-half for all hours worked over 40. Pseudo-code the task of computing gross pay given pay rate and hours worked.

```
1. get hours worked
2. get pay rate
3 if hours worked ≤ 40 then
   3.1  gross pay = pay rate x hours worked
4.else
   4.1 gross pay = pay rate x 40 + (1.5 x pay rate x (hours worked - 40)
5.display gross pay
6.halt
```

- Variables: hours worked, pay rate, gross pay

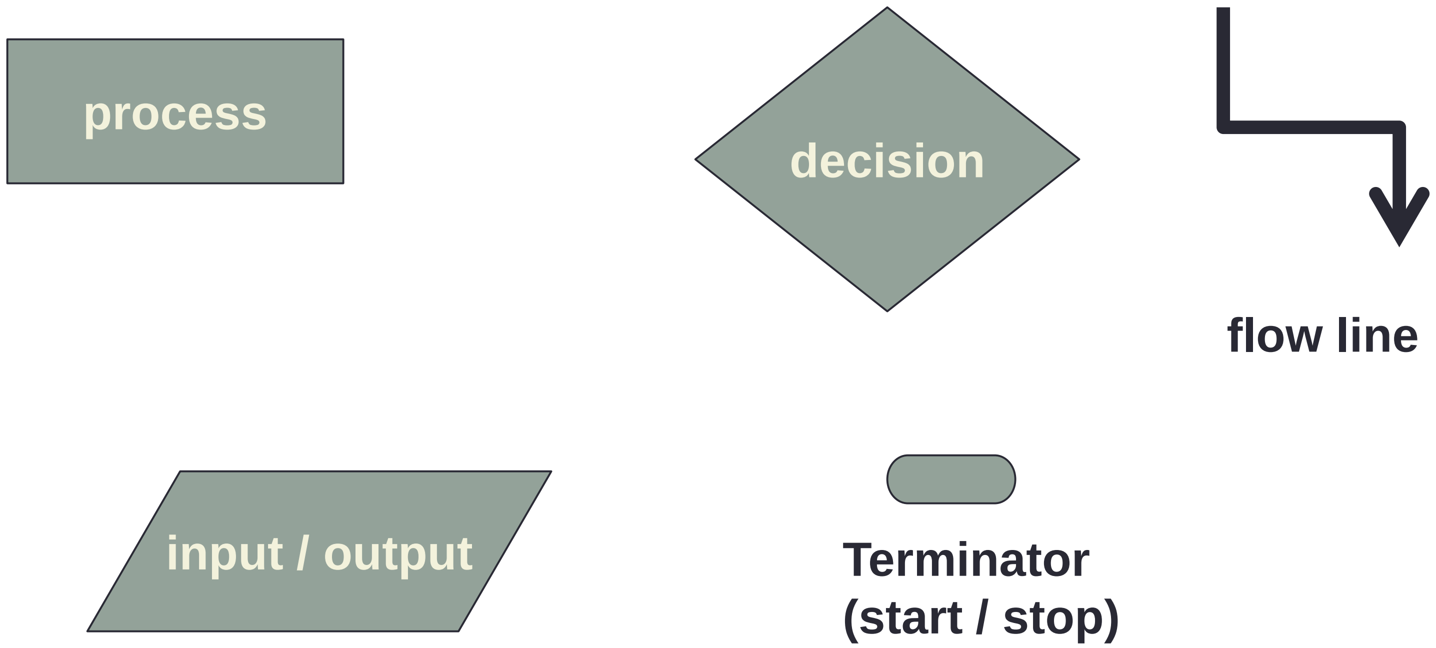
Flowcharts

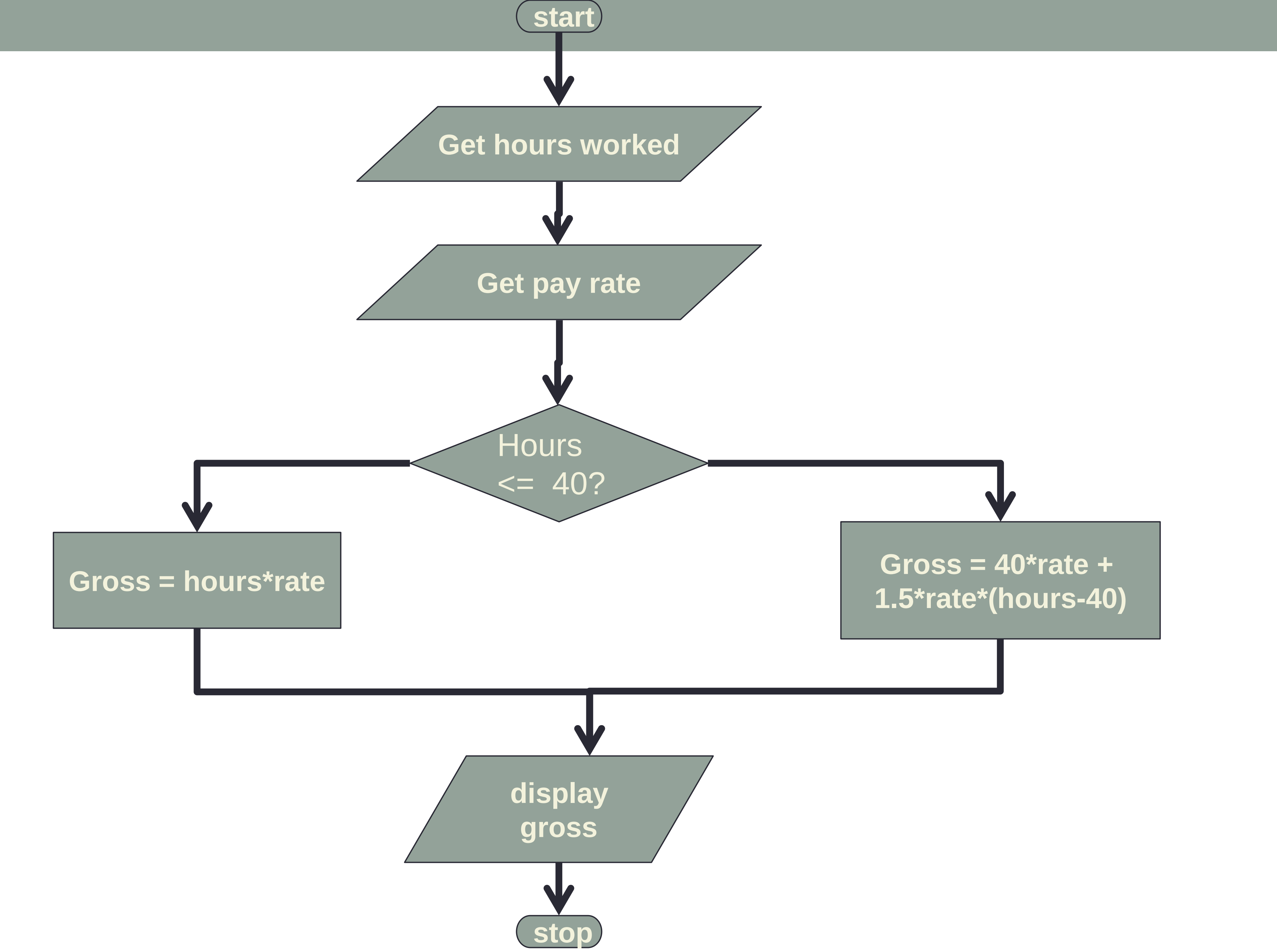


www.edrawsoft.com

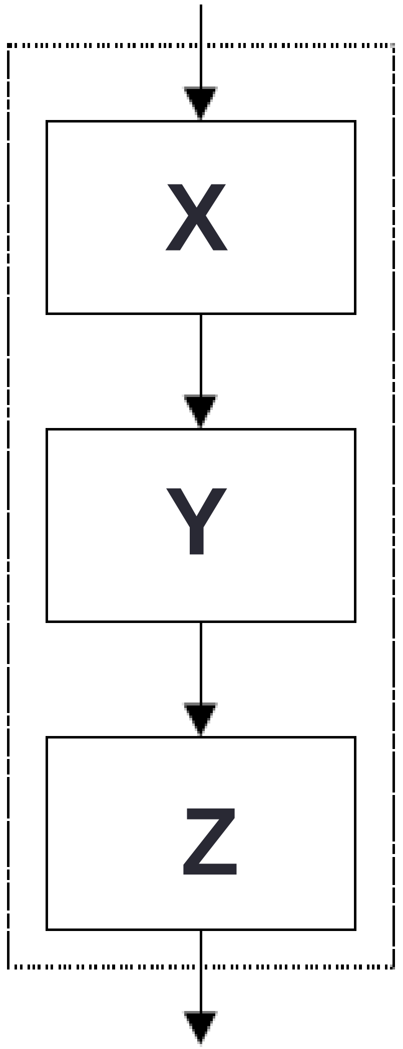
Planning a program - flowcharts

- Popular symbols:




start
Get hours worked
Get pay rate
Hours <= 40?
Gross = hours*rate
Gross = 40*rate + 1.5*rate*(hours-40)
display gross
stop

Sequence



Do X

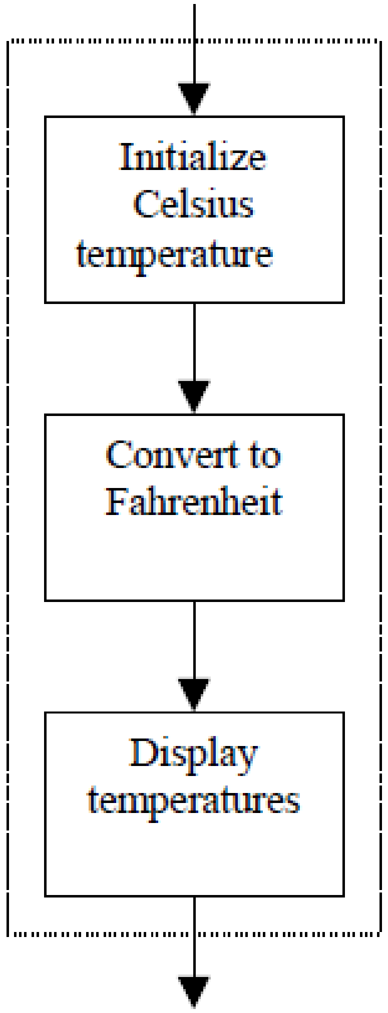
Do Y

Do Z

http://www.eod.gvsu.edu/~blaucha/c2d2/Structured%20Design%20Using%20Flowcharts.pdf

Sequence Flowchart to C Code

Flowchart



Pseudocode

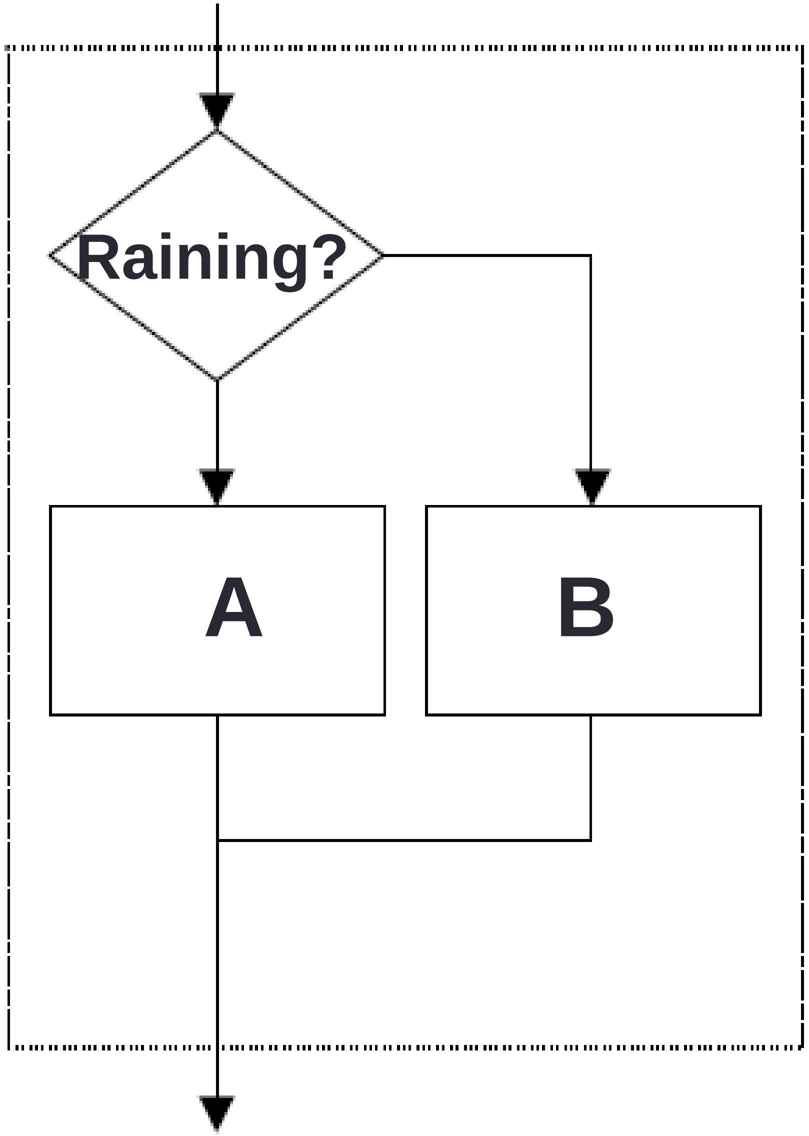
- Set Celsius temperature to 10
- Convert to Fahrenheit temperature
- Display temperatures

C Code

```
x = 10;

y = 9*x/5+32;

DispTemp(x,y);
```

http://www.eod.gvsu.edu/~blaucha/c2d2/Structured%20Design%20Using%20Flowcharts.pdf

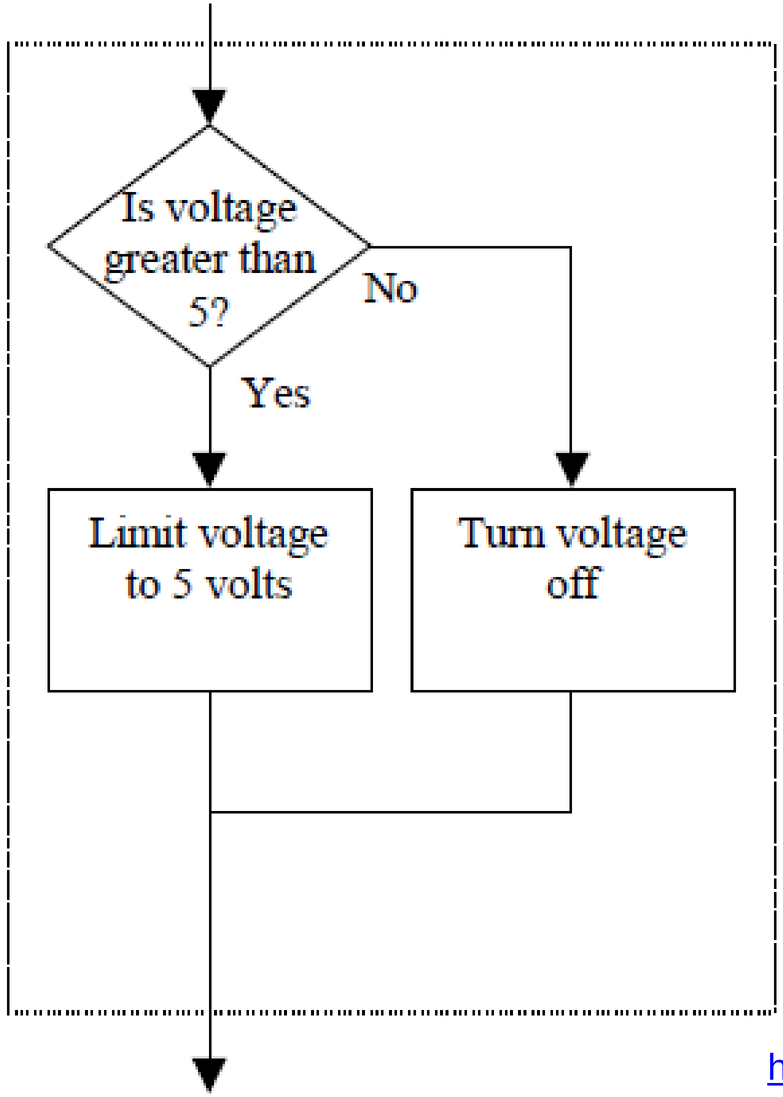
IF-THEN-ELSE



- IF Raining? TRUE THEN
  - Do A
- Otherwise
  - Do B

http://www.eod.gvsu.edu/~blaucha/c2d2/Structured%20Design%20Using%20Flowcharts.pdf

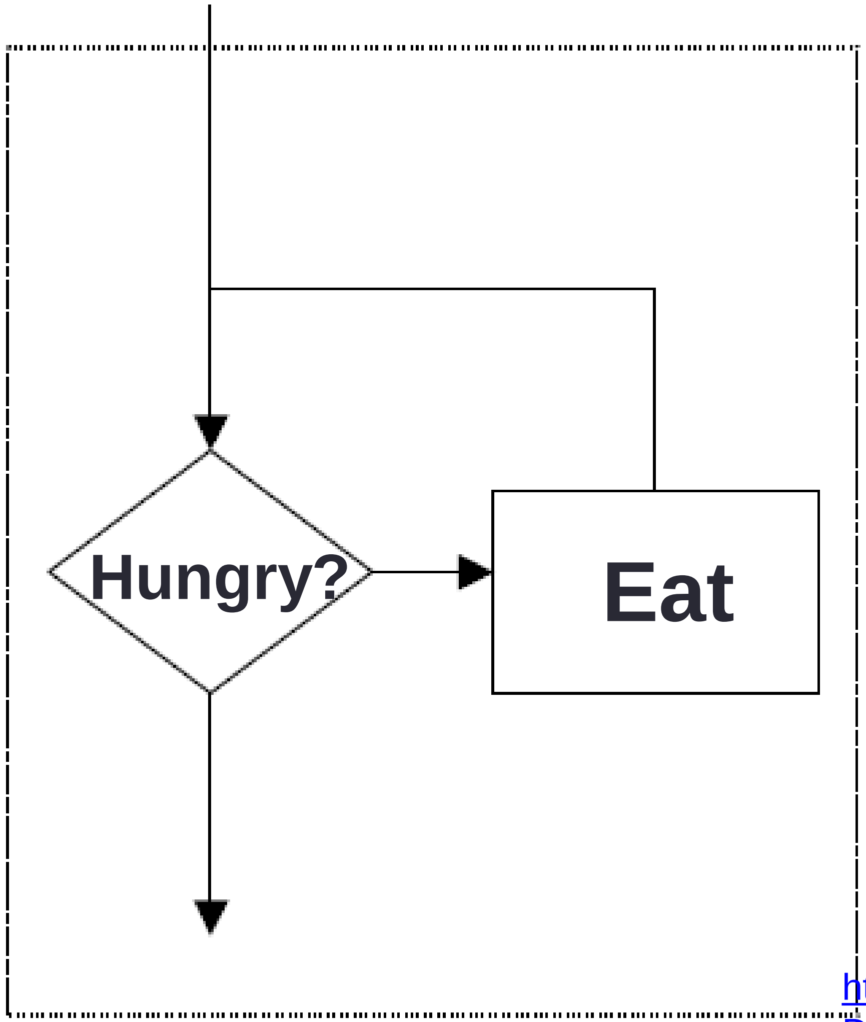
IF Statement Example



```
if ( x > 5 )
{
        x = 5;
}
else
{
        x = 0;
}
```

http://www.eod.gvsu.edu/~blaucha/c2d2/Structured%20Design%20Using%20Flowcharts.pdf

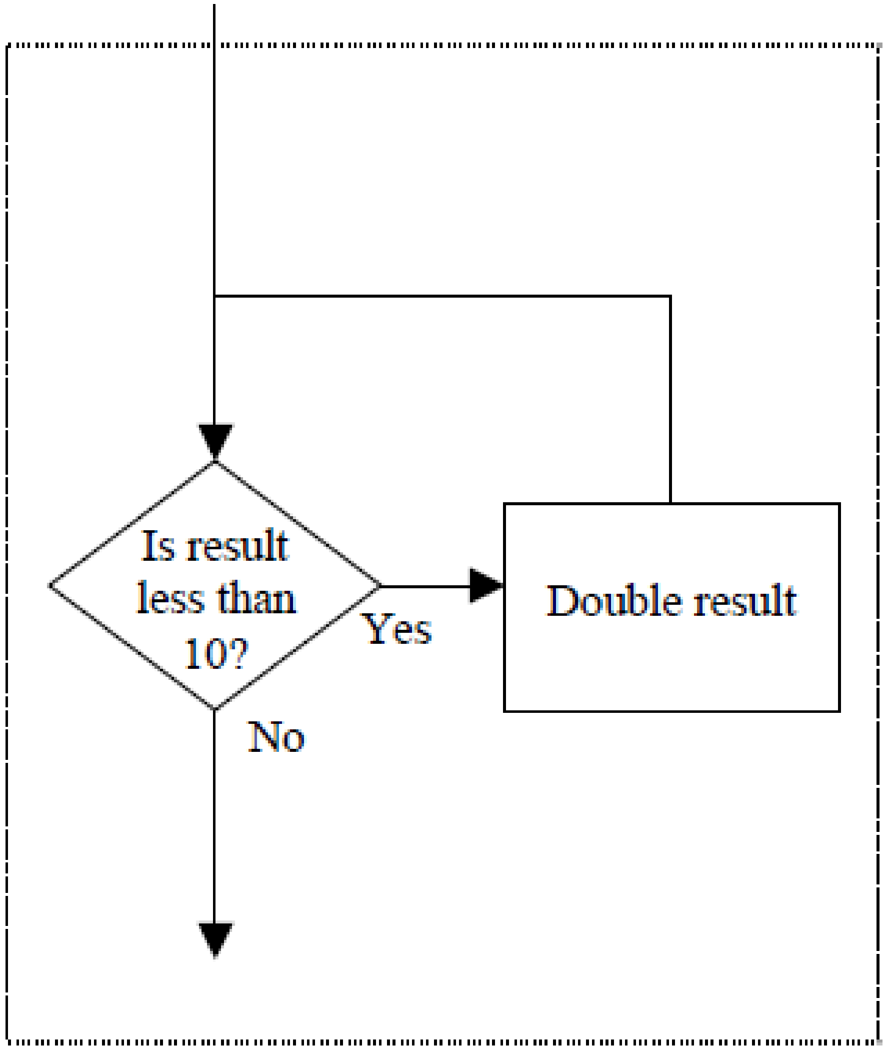
WHILE



- While Hungry
  - Eat

http://www.eod.gvsu.edu/~blaucha/c2d2/Structured%20Design%20Using%20Flowcharts.pdf

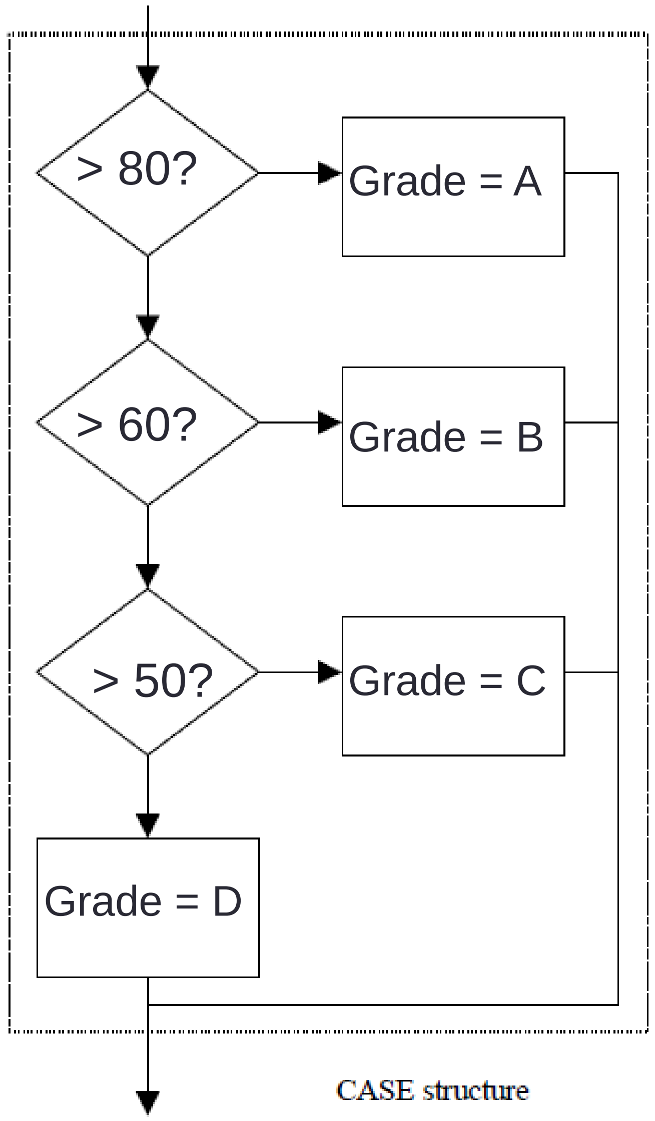
example



```
while ( x < 10 )
{
        x = 2*x;
}
```

http://www.eod.gvsu.edu/~blaucha/c2d2/Structured%20Design%20Using%20Flowcharts.pdf

CASE



CASE structure

IF Mark >= 80? **THEN** Grade = A

ELSE

IF Mark >= 60? **THEN** Grade = B

ELSE

IF Mark >= 50? **THEN** Grade = C

Else

IF Mark >= 40? **THEN** Grade = D

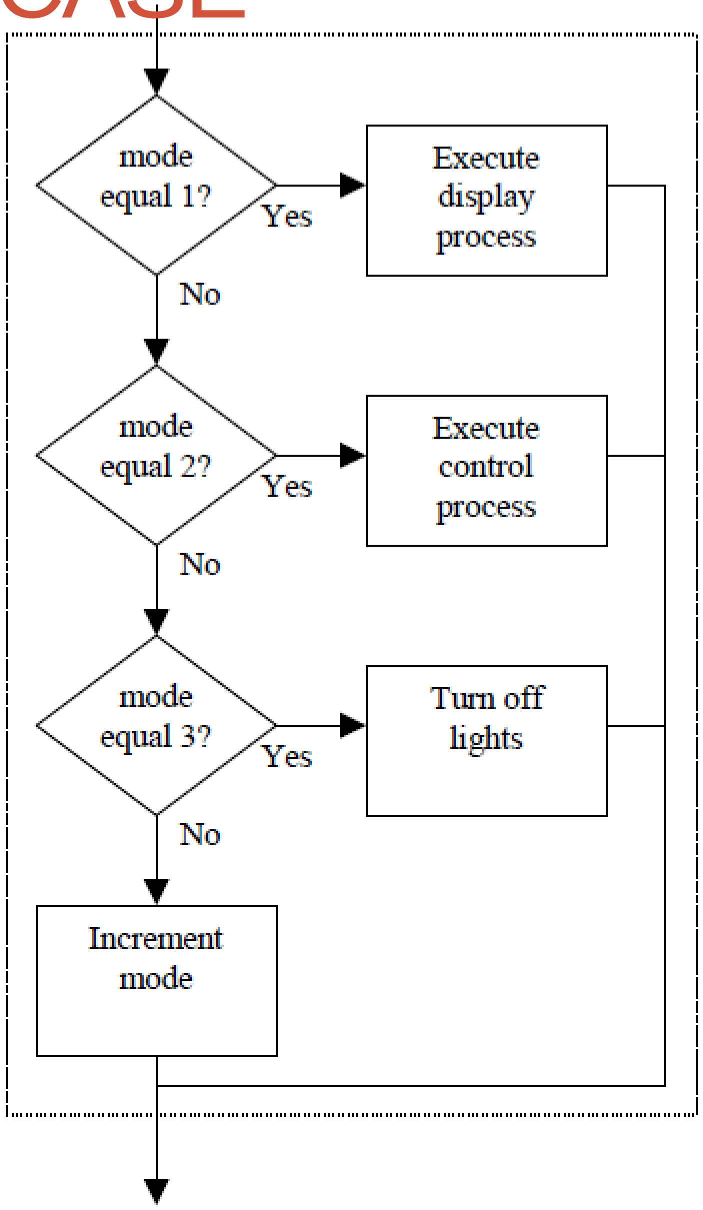
ELSE

Grade = F

END IF

http://www.eod.gvsu.edu/~blaucha/c2d2/Structured%20Design%20Using%20Flowcharts.pdf

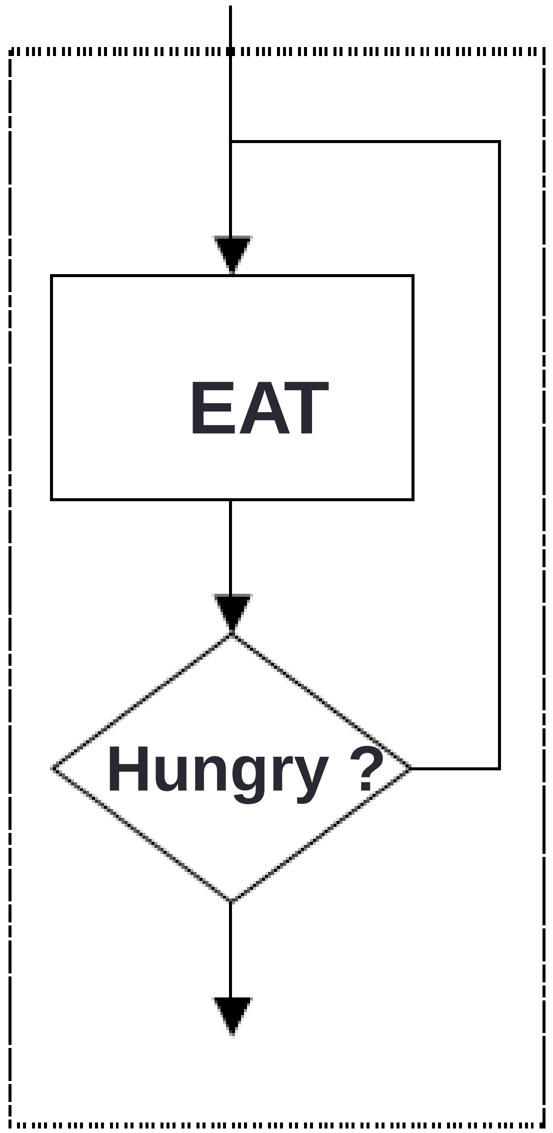
CASE



```
switch (mode) {
case 1:
      /* Display process */
      break;
case 2:
      /* Control process */
      break;
case 3:
      red_light = 0;
      blue_light = 0;
      break;
default:
      mode = mode + 1;
      break;
}
```

http://www.eod.gvsu.edu/~blaucha/c2d2/Structured%20Design%20Using%20Flowcharts.pdf

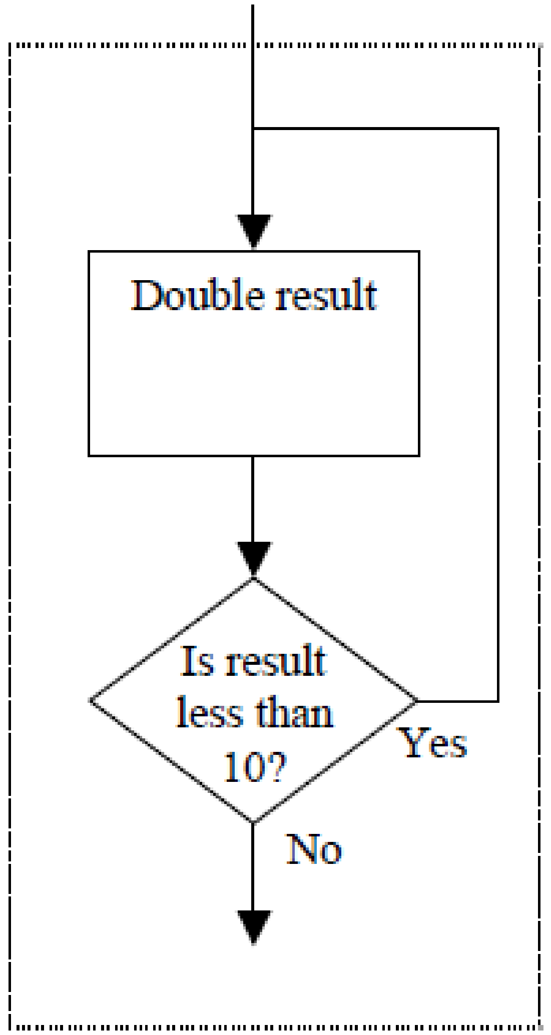
DO-WHILE



- **Do**
  - EAT
- **While** Hungry = TRUE

http://www.eod.gvsu.edu/~blaucha/c2d2/Structured%20Design%20Using%20Flowcharts.pdf

DO-WHILE



```
do
{
        x = 2*x;
}
while ( x < 10 );
```

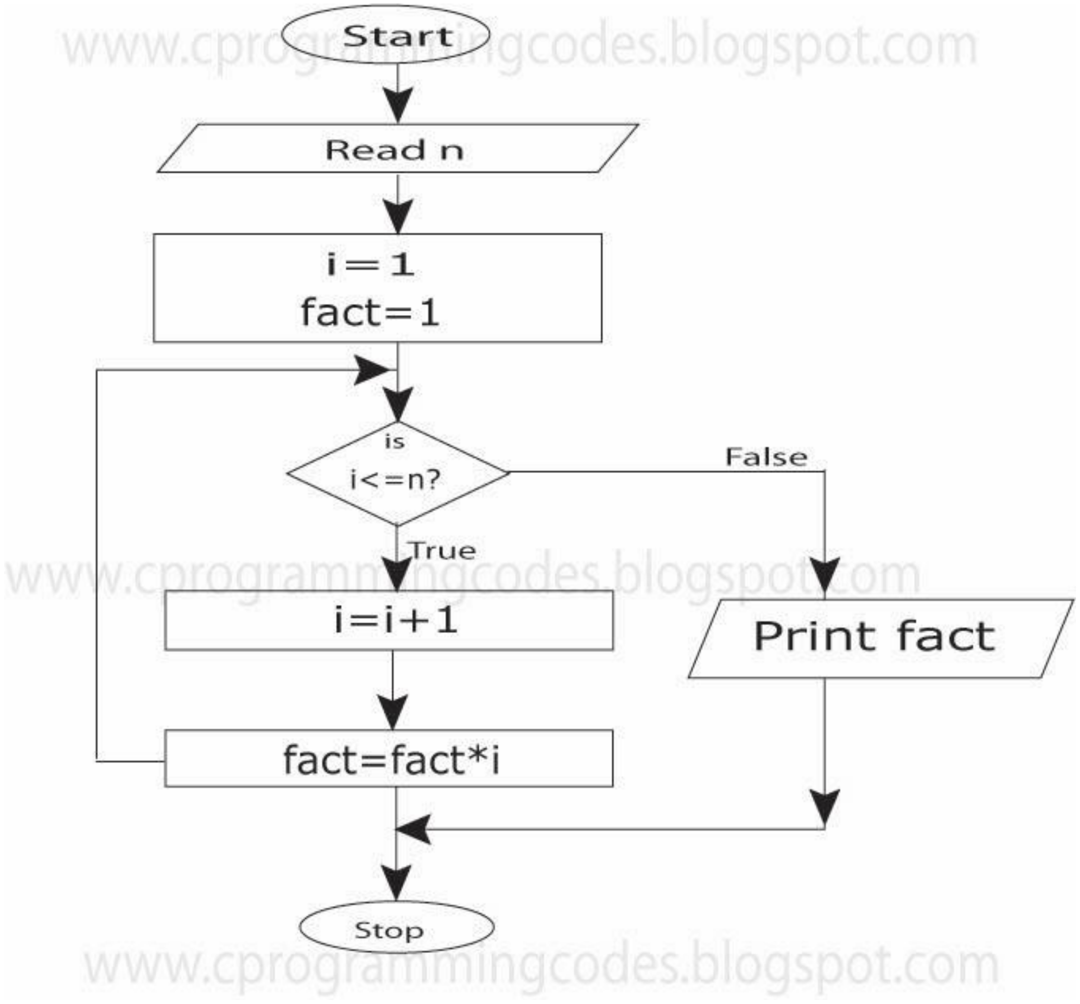
http://www.eod.gvsu.edu/~blaucha/c2d2/Structured%20Design%20Using%20Flowcharts.pdf

Algorithm to calculate factorial of a number

```
1. Start
2. Read the number n
3. [Initialize]
     i=1, fact=1
4. i=i+1
5. fact=fact*i
6. Repeat step 4 through 6 until i=n
7. Print fact
8. Stop
```

http://cprogrammingcodes.blogspot.ie/2012/10/factorial-c-programalgorithmflowchart.html

Flowchart



http://cprogrammingcodes.blogspot.ie/2012/10/factorial-c-programalgorithmflowchart.html

PROGRAMMING

CT103

Week 1b

Lecture Content

- Last lecture (Week 1a):
  - Module overview: Grading, etc.
  - Introduction to algorithms.
  - Pseudocode and flowcharts.

- Today’s lecture (Week 1b):
  - Computer programs.
  - Data types.
  - Example C program.

COMPUTER PROGRAMS

What is a Program?

- **Definition**: A program is a set of *instructions* that are run by the *Central Processing Unit (CPU)* on a computer.

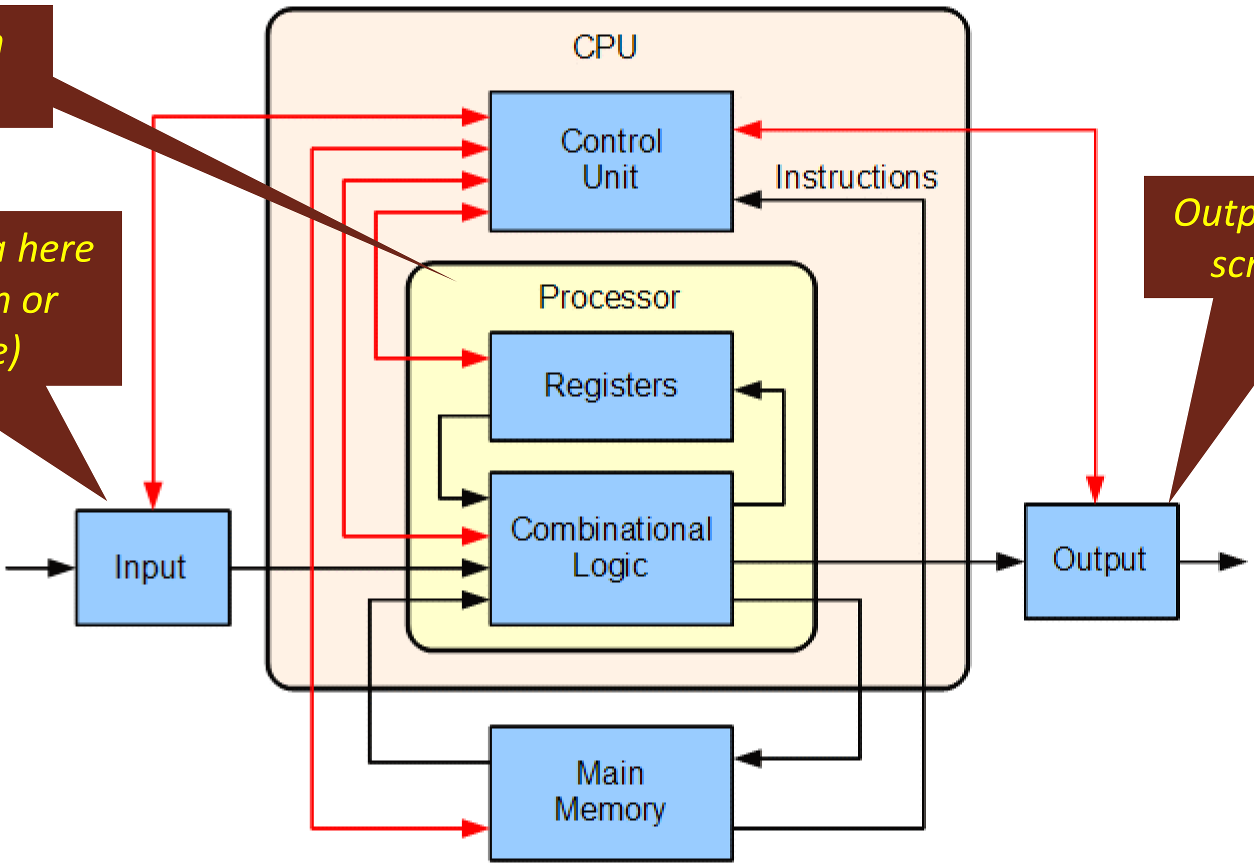
- The instructions are designed to accomplish a specific task and are written in a programming language like C.

What is a Program?

- C is what is called a *high-level language* – this has to be translated into instructions called *machine language* that the CPU can execute.

- Distinction between Program and Algorithm:
  - A program is a set of instructions that the computer executes.
  - An algorithm is a series of steps to complete a task.
  - A program contains the algorithm.
  - Algorithm is the logic, program is the implementation.

Where our Program Runs



The program runs here

We can input data here (e.g. on a screen or command line)

Output can go to the screen, files, etc.

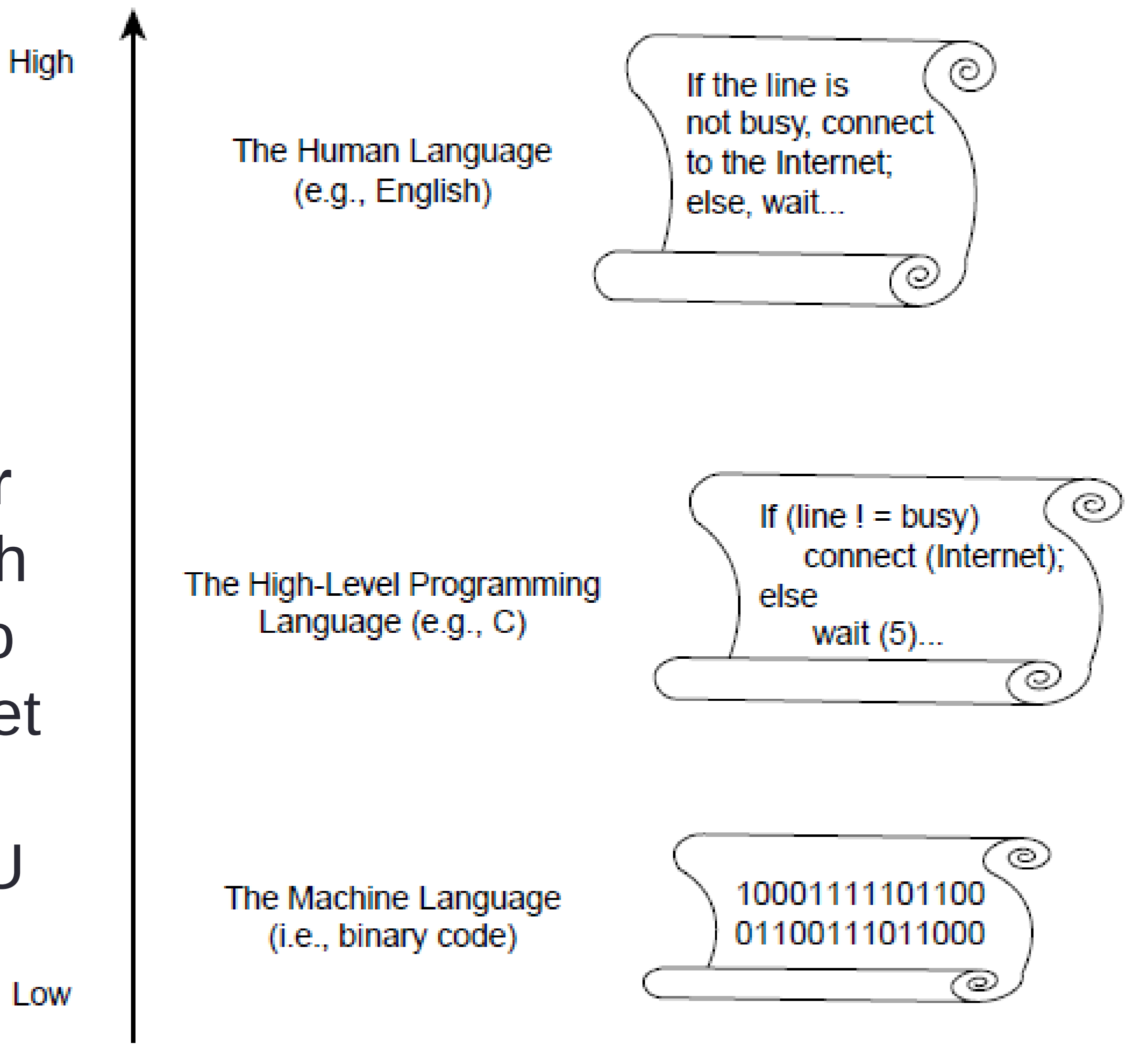
CPU and RAM

Central Processing Unit (CPU)

Random Access Memory (RAM)

CPU Runs What?

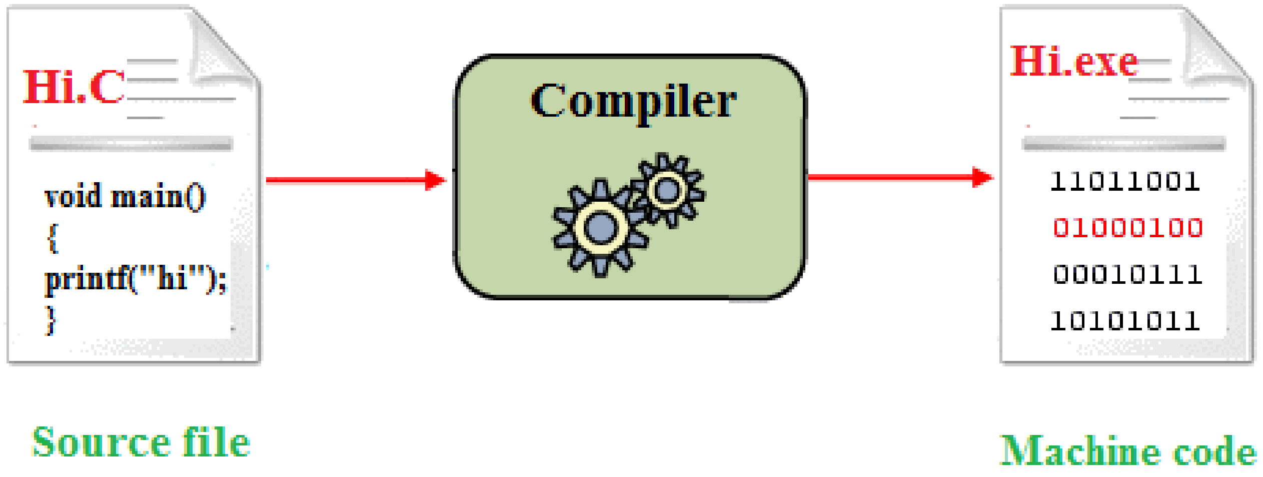
- Each CPU uses a specific set of instructions, called ***machine language***. We write our programs in a **higher level language** which is then translated into a machine-specific set of instructions for execution by the CPU


High
Low
The Human Language (e.g., English)
If the line is not busy, connect to the Internet; else, wait...
The High-Level Programming Language (e.g., C)
If (line ! = busy) connect (Internet); else wait (5)...
The Machine Language (i.e., binary code)
10001111101100 01100111011000

Sams Teach Yourself C in 24 Hours ©2000 by Sams Publishing

Compiler

- If the CPU understands machine language (1s and 0s), how do we convert our C program to machine language?
  - Answer: The compiler will do this for you!
- The compiler will convert your C program source code (.c file) into binary code for the CPU to understand.



Programming Software

- Applications used to write software:
  - Assist in writing program in high level language (e.g. C)
  - Compile it into machine language
  - Link various bits of machine code together to create an application
  - Run, test and debug the application
- Typically also called IDE (Integrated Development Environment), such as **MS Visual Studio** or NetBeans

Writing Programs

- We use an ***editor*** to write the program (the ***source code***), and then a ***compiler*** to compile it.

- A compiler turns the program into ***machine-language*** instructions that the computer can understand.

C Compiler

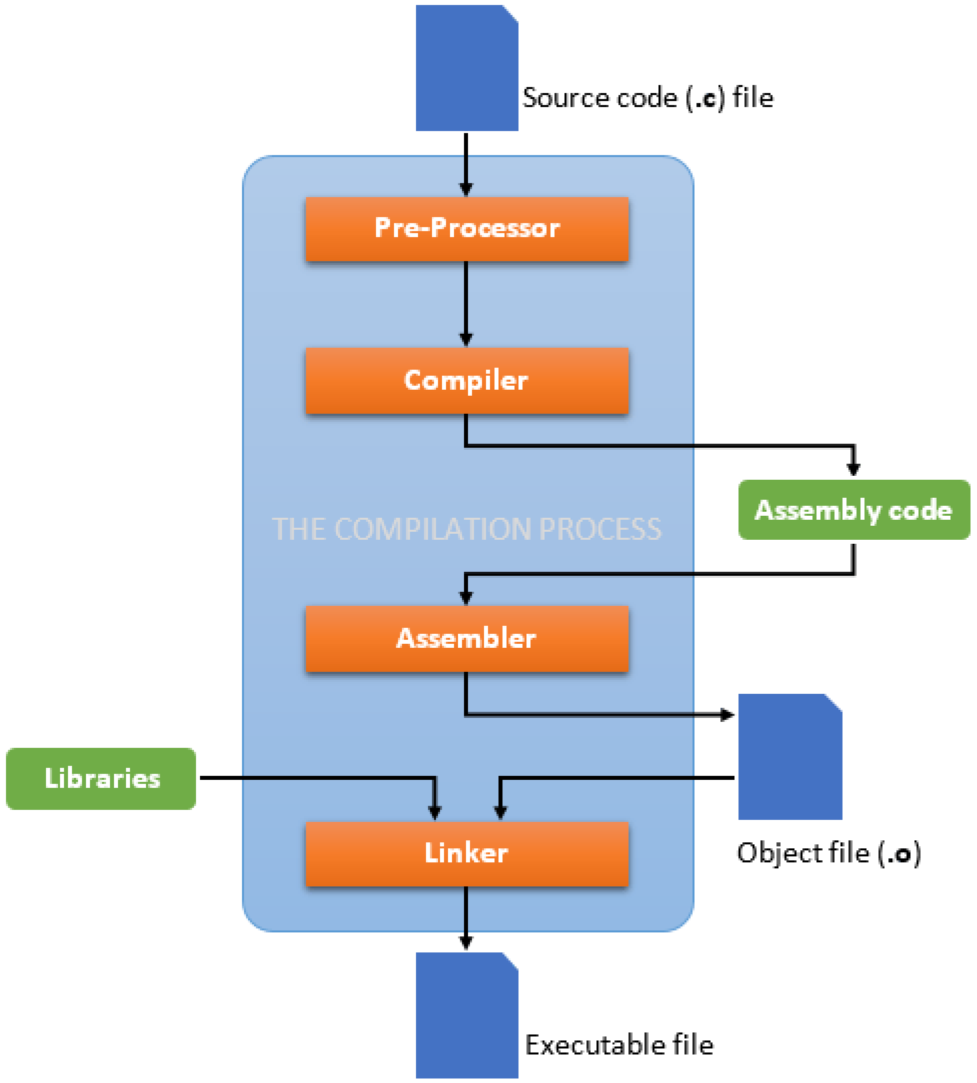


Image source: Medium.com

DATA & VARIABLES

Variables

- How we temporarily store data that we are using in our programs
- They often represent some *real-world* piece of data, e.g.
  - salary, temperature, interest rate
- In most programming languages, including C, we have to decide on the type of variable most suitable for the data we want to store and manipulate
- Variable examples in C:
  - float salary;
  - float temperate;
  - int age;
  - char exam_grade;

C Number types

- There are actually different kinds of numbers:
  - Integers (no decimal point)
    - E.g. 10 54 0 -121
  - Floating-point or real (with decimal point)
    - E.g. 4343.34 0.0 0.123234 -34.223
  - The choice of integer or floating-point depends on what it represents
    - Age (integer), No. of people in family (integer), Interest Rate (floating-point), price of litre of petrol (floating-point)

Kinds of Data

- So we can see that we need different variable types, or data types, to hold information
- The basic set of C data types is:
  - **int** - this holds an integer

    e.g. 10   21   456   -6899
  - **float** – holds a floating point number

    e.g. 125.467
  - **double** – holds a very big floating point number

    e.g. up to 1.797e+308
  - **char** – holds a character

    e.g. ‘A’   ‘c’   ‘%’
  - Also **strings** – holds multiple characters

    e.g. ‘hello’

Modifiers

- **Short,** i.e. smaller (less memory)
- **Long**, i.e. larger (more memory)
- **Signed**, i.e. positive or negative
- **Unsigned**, i.e. non negative
- The amount of storage used for each data type (+ modifier) is not set in stone
- ANSI has the following rules:

  short int <= int <= long int

  float <= double <= long double

Size (bytes)

- Actual space used to store numbers can vary between machines and operating systems, but in general:

Data Type	Memory (Bytes)	Min Value	Max Value
short int	2	-32768	32768
unsigned short int	2	0	65,535
unsigned int	4	0	4294967295
int	4	-2147483648	2147483648
unsigned char	1	0	255
float	4		
double	8		
long double	12		

Characters

- A *character* is any single character that your computer can represent – usually there are 256 of them
- We usually use the 128 most common (called the *Standard* ASCII character set)

Back to Characters

- The following are all characters:

  A a 4 % ^ . Q + =] #

- A group of multiple characters is called a *string* e.g.

  “I love Programming!”

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1 (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2 (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Functions

- A function is a piece of self-contained code that performs a task
- For example, to print out the text “Hello”, we can use the standard C function *printf()*
  - printf (“Hello”);
- To read an integer input from the keyboard, we could use:
  - scanf (“%d”, &age);

- We will learn more about functions later in the course!

PROGRAM RECAP

Designing your Program

- The most basic way of describing what should happen is to just write it down
- The easiest way of doing this is to use *Structured English*
- This means using keywords like IF, THEN, ELSE, DO, to express what should happen
- Another common way is to use a Flowchart

Sequence

- Actions which take place one after the other

Find a teapot

Put in the tea

Pour in boiling water

IF-THEN-ELSE

- Used where you need to decide on what action to take

```
IF condition A
  THEN action B
ELSE action C
ENDIF
```

“IF” Example

```
IF you like tea
  THEN drink tea
  ELSE drink coffee
ENDIF
```

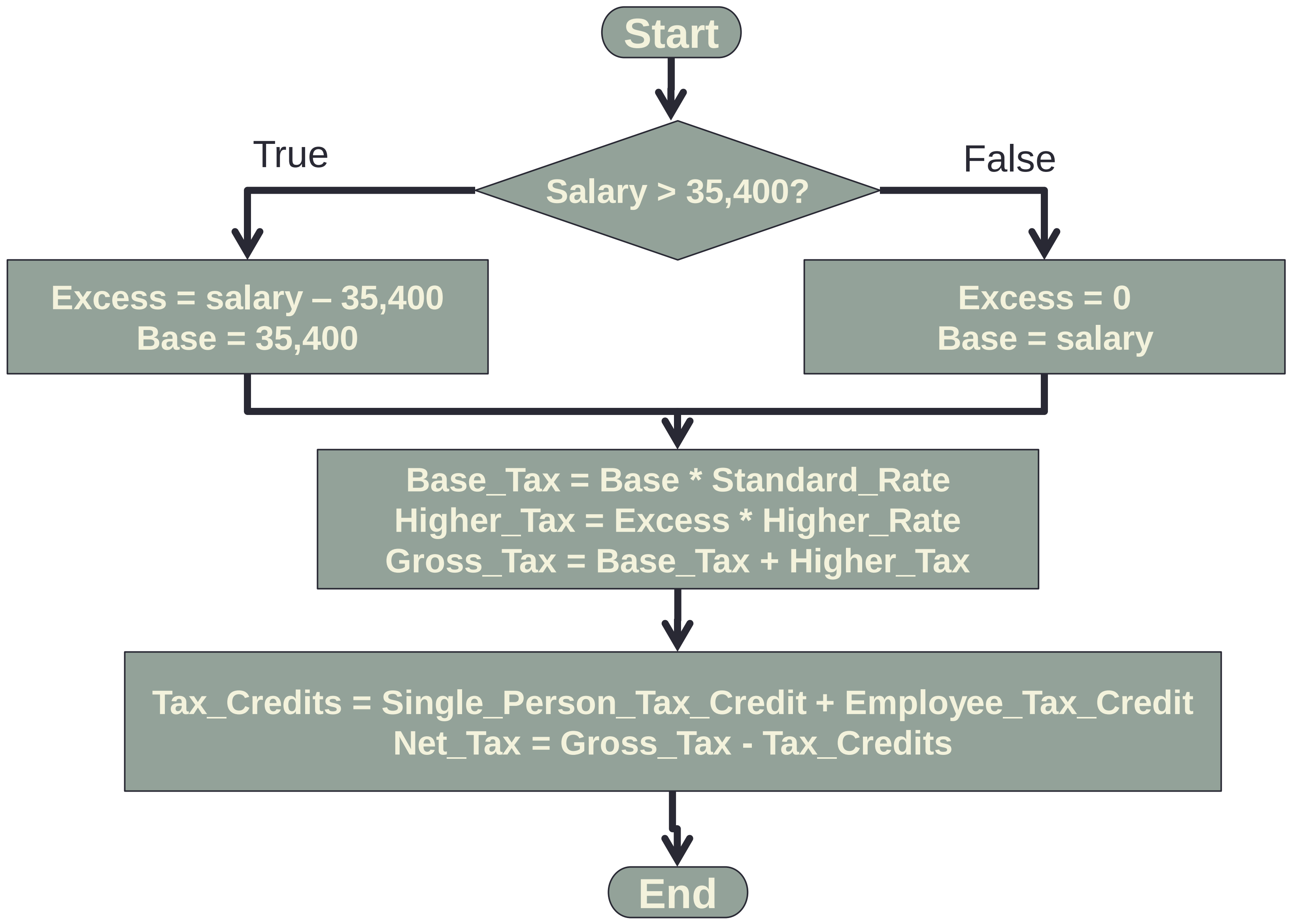
More Realistic IF Example

```
IF customer_order_total  > €400
  THEN
  IF days customer_balance is due > 60 days
          THEN
                    hold the customer_order
                    send reminder letter
          ELSE
                    process the customer order
  END-IF
ELSE
  process the customer order
END-IF
```

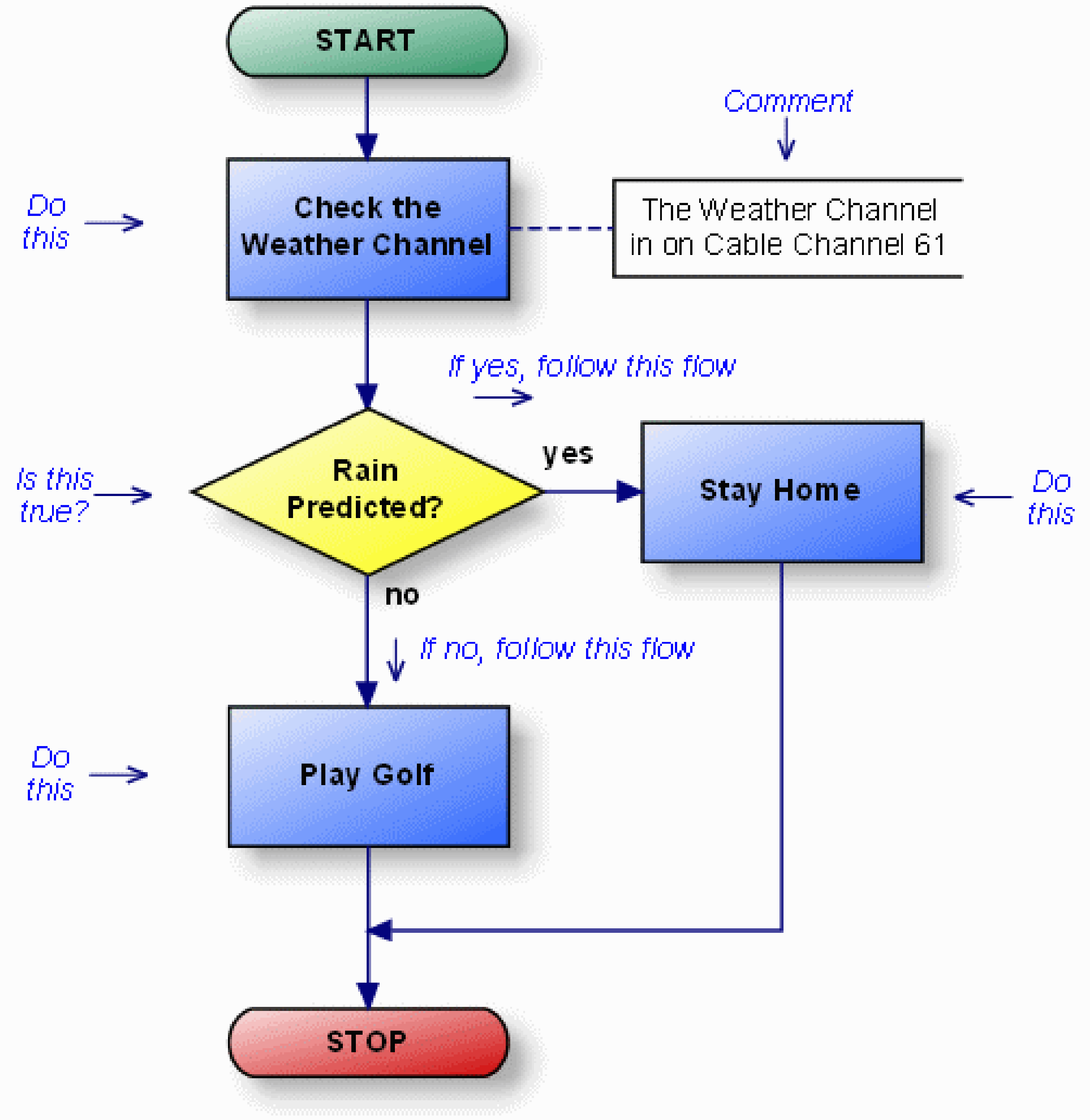
Structured English

```
Read in salary
IF salary > 35,400 THEN
   Excess = salary - 35,400
   BASE = 35,400
ELSE
   Excess = 0
   BASE = salary
ENDIF
Base_Tax = Base * Standard_Rate
Higher_Tax = Excess * Higher_Rate
Gross_Tax = Base_Tax + Higher_Tax
Tax_Credits = Single_Person_Tax_Credit + Employee_Tax_Credit
Net_Tax = Gross_Tax - Tax_Credits
```

Flowchart


Start
Salary > 35,400?
True
False
Excess = salary – 35,400
Base = 35,400
Excess = 0
Base = salary
Base_Tax = Base * Standard_Rate
Higher_Tax = Excess * Higher_Rate
Gross_Tax = Base_Tax + Higher_Tax
Tax_Credits = Single_Person_Tax_Credit + Employee_Tax_Credit
Net_Tax = Gross_Tax - Tax_Credits
End

Workflow



http://www.pacestar.com

PROGRAM EXAMPLES

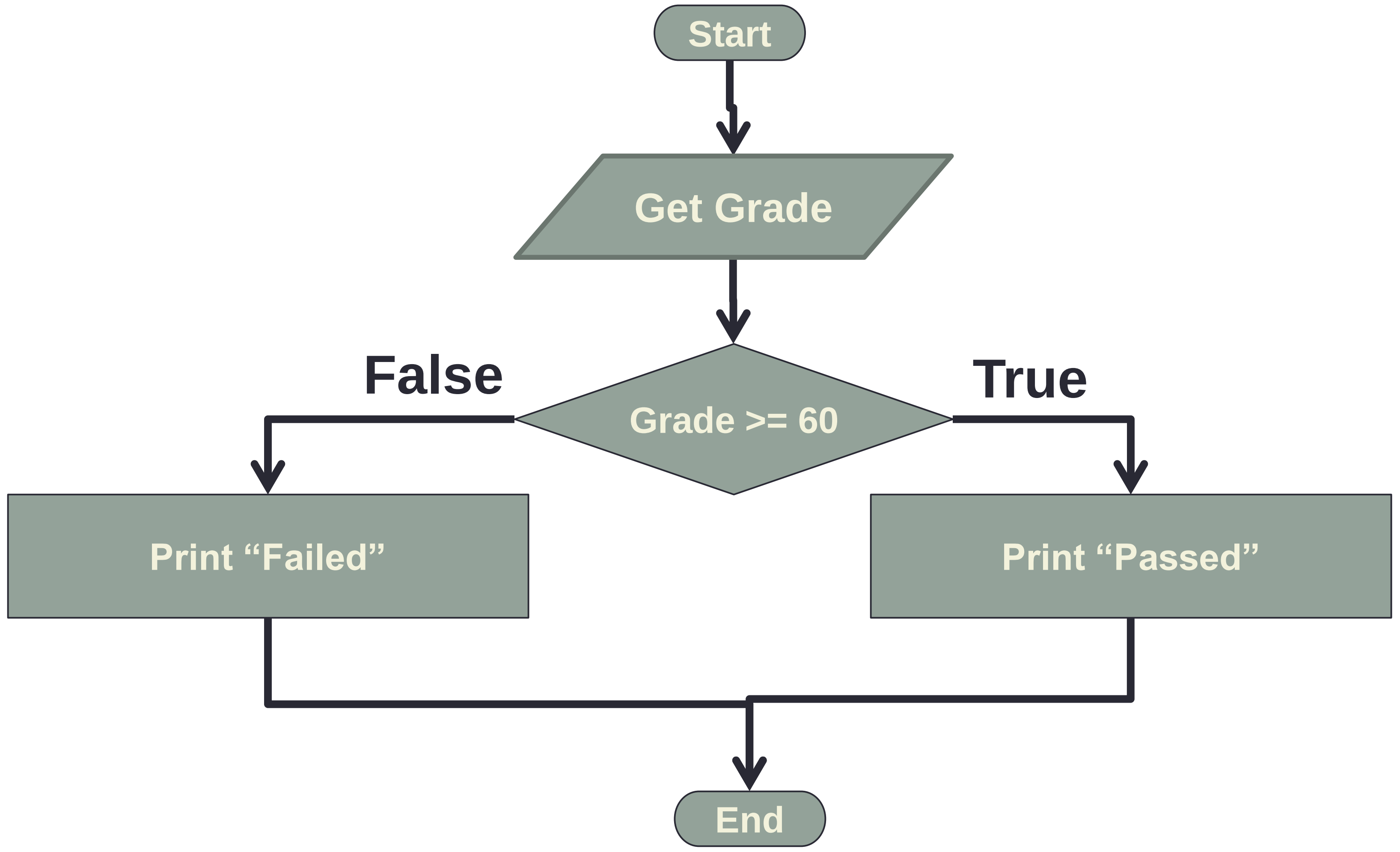
Worked Through Example

- Problem Description:
  - Write a program that reads in an exam mark and outputs “Passed” if the mark is 60 or more. Otherwise print out “Failed”.

Pseudocode

```
Get exam grade
If grade is greater than or equal to 60
  Print “Passed”
else
  Print “Failed”
```

Flowchart


Start
Get Grade
Grade >= 60
False
True
Print “Failed”
Print “Passed”
End

```
#include <stdio.h>
void main()
{
        int grade = 0;

        printf ("Enter grade: ");
        scanf_s("%d", &grade);

        if (grade >= 60)
        {
                printf ("Passed \n");
        }
        else
        {
                printf ("Failed \n");
        }
}
```

Code in Visual Studio

- Here I will go through some C code

Programming

CT103

Week 2a

Lecture Content

- Last lecture (Week 1b):
  - Computer programs.
  - Data types.
  - Example C program.

- Today’s lecture (Week 2a):
  - C basic variable types and their size in bytes.
  - Naming variables.
  - Declaring and initialising variables.
  - Comments.
  - C program.

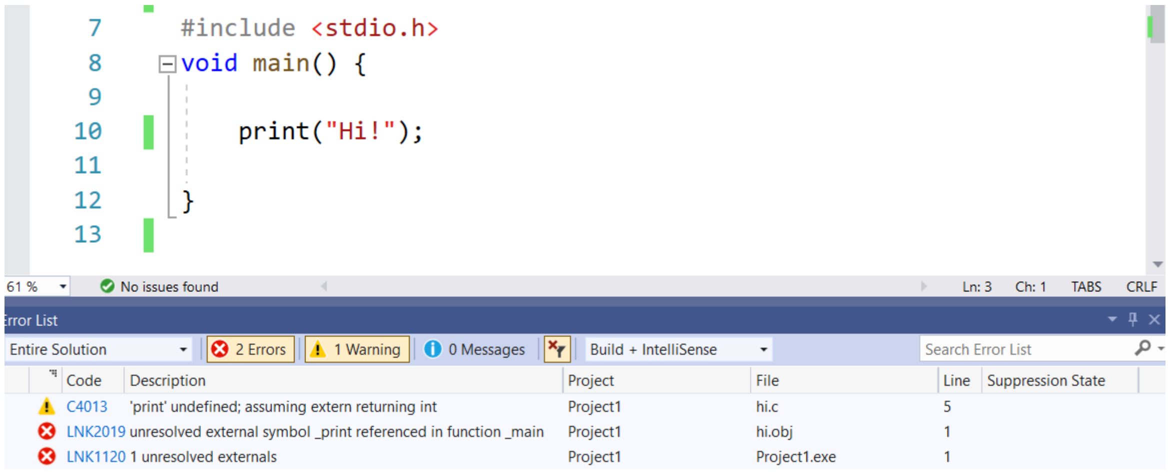
Before We Start

- What did ‘\n’ do in the C code example from week 1b?
- Why won’t my code run?
- Xcode for Mac.
- Access to PCs in the IT Building.

Before We Start

- What did ‘\n’ do in the C code example from week 1b?
  - Answer: ‘\n’ starts a new line when printing text to the screen.
  - E.g., **printf (“1 \n 2 \n 3");** will print “1”, “2” and “3” on new lines.

- Why won’t my code run?



```
#include <stdio.h>
void main() {

    print("Hi!");

}

```

Code	Description	Project	File	Line	Suppression State
C4013	'print' undefined; assuming extern returning int	Project1	hi.c	5	
LNK2019	unresolved external symbol _print referenced in function _main	Project1	hi.obj	1	
LNK1120	1 unresolved externals	Project1	Project1.exe	1	

Before We Start

- Xcode for Mac.
  - If anyone is still having difficulty getting Xcode set up for Mac, there are plenty of online tutorials that can help you:
    - https://www.youtube.com/watch?v=_gwPhmyiuVo
    - https://www.youtube.com/watch?v=_cDXKReugEU
    - Search ‘Mac Xcode C’ on YouTube and you will find many results.

- Access to PCs in the IT Building.
  - You should have received an email instructing you to go to http://www.it.nuigalway.ie/accounts to get an initial password.
  - If you haven’t yet, you should go to this link and follow the instructions.

VARIABLES

Variables

- We need to be able hold data in our programs and change it as we do calculations
- Variables are pieces of memory that C reserves to hold our data
- Data is stored in binary form
- The more memory a variable uses, then the more data (1's and 0's) it can hold – hence the bigger numbers need more memory, e.g. (on a 64-bit windows machine):

char	1 byte
short int	2 bytes
int	4 bytes
float	4 bytes
double	8 bytes

Bits and Bytes

- What is a bit?
  - A bit is the most basic unit of information, i.e. 1 or a 0.
- What is byte?
  - A byte is 8 bits, e.g. 10101100.
- A kilobyte is 1024 bytes, i.e. 1 KB = 1024 B (\$2^{10} = 1024\$)
- A megabyte is 1024 kilobytes…
- Etc.
- However…

Bits and Bytes

- However…
- In International System (SI) Units, kilo means 1000.
- A kilobyte is 1000 bytes, i.e. 1 KB = 1000 B
- A megabyte is 1000 kilobytes… etc.
- These SI units are used for:
  - Data transfer rates
  - Hard drive capacities
- Other definition (1KB = 1024B) used for operating systems.

Types of C variables

Name	Description
char	Holds character data such as ‘x’ and ‘*’
short int	Holds integer data such as 1, 32, -456 Stores data between -32768 and 32767 Or 0 to 65535 if unsigned
int	Holds integer between -2,147,483,648 and 2,147,483,647 (double this if unsigned)
long int	Same as for int on a 32-bit compiler, but on 64 bit compiler: −9,223,372,036,854,775,808 to +9,223,372,036,854,775,807
float	Holds floating point data such as 0.003, -12.4
double	Holds extremely large and small floating point data (bigger/smaller than $\pm 3.4\text{x}10^{38}$!!)

Floating point

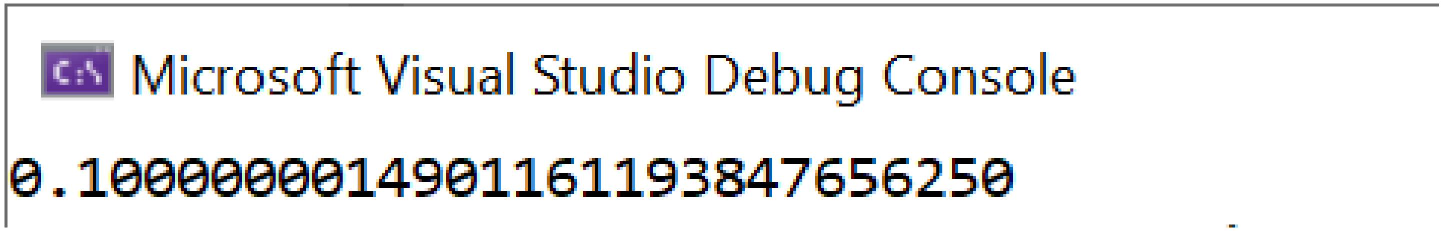
- A floating point number is one with a decimal number after the point.
- Decimal fractions difficult to represent exactly as binary fractions, so the binary is as close as possible, but there will be an approximation, but as we usually only display to a certain number of decimal places, we don’t usually notice.
- So we are usually seeing the rounded display of the actual machine value.

Rounding

- So for example if I execute this:

```
float f = 0.1;

printf("%30.28f", f);
```

- I see this output (printf allows me to specify the number of decimal places I see)!
  - In this case I am telling printf to print the variable ***f*** 30 characters wide, of which 28 are after the decimal point



```
0.1000000014901161193847656250
```

Float vs double

- The double (64 bits) occupies twice the memory of a float (32 bits), therefore can store bigger, and more precise, numbers.

- So you can convert from a float to a double, but not necessarily the other way, without loss of precision.

- Depending on the platform you might use float if you don’t need doubles, to save on memory, performance and bandwidth.

NAMING VARIABLES

Naming variables

- Every variable you want to use needs a different name.
- A variable name can be from 1 to 32 characters long.
- The name must begin with a letter followed by any letter, number, underscore combination.
- The following are valid examples of variable names:

  myData   pay94   age_limit   amount
- The following are invalid examples of variable names :

  95Pay   my age   rate*pay   printf

Variable naming conventions

- These are just some of the variable naming conventions (also called ‘cases’ or identifier formats).

- Many companies have their own conventions.

```
double annualsalary; // flat case
double annualSalary; // camel case
double annual_salary; // snake case
double Annual_Salary; // camel snake case
```

Declaring and Initialising variables

- We usually **declare** variables at the start of the program and we can optionally **initialise** them at the same time

```
float salary, pension; // variables declared but not initialised
char initial = 'c'; // declared and initalised
int departmentNumber; // not initialised
int age = 0; // declared and initalised


// now assign a value to the variable 'salary'
salary = 35000.00;
```

- Note how we can put comments at the end of a line.
- We will talk more about these later today!

Storing data in variables

- We use the *assignment operator* (=) to put data in variables

```
age = 34;
salary = 50000;
pension = salary + age*1000;
```

In general, we take what is on the right hand side (or what it evaluates to if it is an equation or a function call), and put it into the left hand side (usually a variable)

Printing out values of variables

- We can use printf() to do the work for us here
- For example:

```
int age = 25;
float salary = 34000.00;
char initial = 'D';

printf("age = %d \n", age);
printf("salary = %.2f \n", salary);
printf("initial = %c \n", initial);

printf("you are %d years old, you earn %.2f and your middle
initial is %c \n", age, salary, initial);
```

Using printf

- The printf function takes in a number of inputs
- The first input is always the text you want to print out, which may include placeholders (actually called *conversion characters*) for 1 or more pieces of data
- The data is supplied in the inputs following the formatted text input, with inputs separated by commas, for example:

```
int age = 35;
float salary = 35000.00;

printf("you are %d years old and earn %.2f per year", age, salary);
```

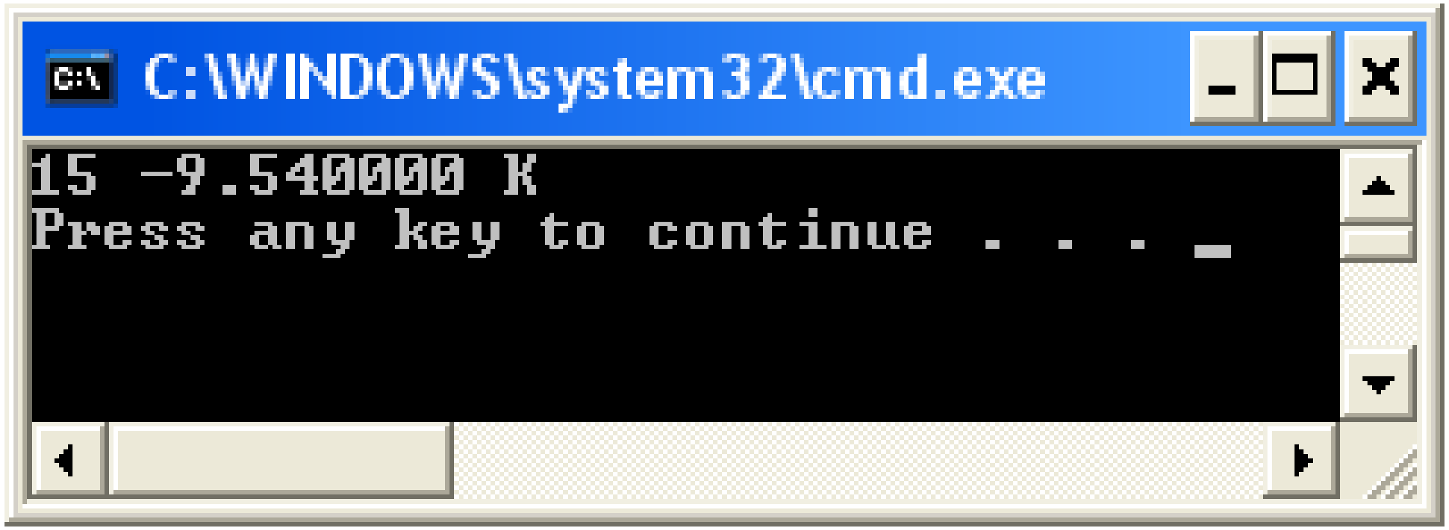
Conversion Characters

- Remember that we have to tell C exactly how to print numbers and characters
  - We have to use *conversion characters* (also called *format specifiers*)

Conversion Character	Description
%d	Integer
%f	Floating point
%c	Character
%s	String
%lf	Double
%X	Hexadecimal

Example

printf ("%d %f %c\n", 15, -9.54, 'K');

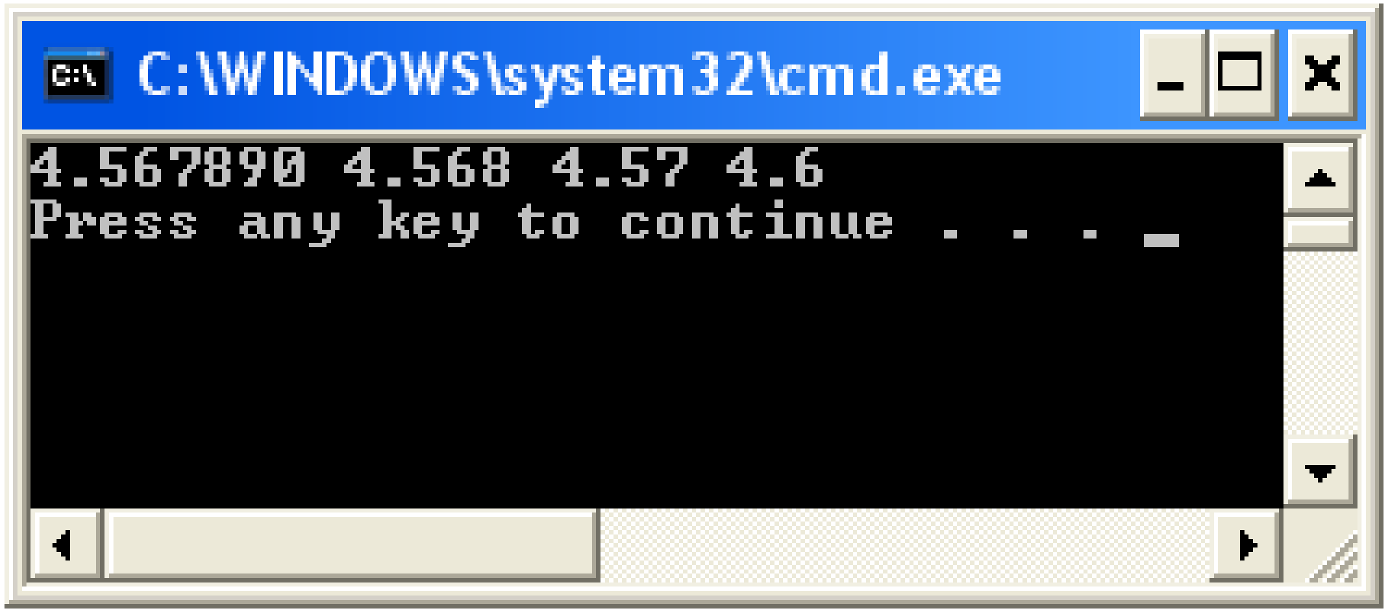


```
15 -9.540000 K
Press any key to continue . . . _
```

Note: if we don't specify the number of decimal places, C automatically puts in 6!

Example

printf ("%f %.3f %.2f %.1f\n", 4.56789, 4.56789, 4.56789, 4.56789);



```
4.567890 4.568 4.57 4.6
Press any key to continue . . . _
```

Note: C rounds to the number of decimal places specified

Escape Sequences

- C uses Escape Sequences a lot to represent characters that can’t easily be represented in text. They are converted into the correct character for example when output to screen.
- They are just special characters – we already used \n which gives us a new line.
- Some other ones are:

\t	tab
\\	just a backslash
\”	double quote
\’	single quote
\a	beep or alarm

Sample Program

- Try out this program yourself

```
#include <stdio.h>
void main()
{
        float grade1, grade2, grade3;
        float average = 0.0;

        printf("Enter 3 grades separated by spaces: ");
        scanf("%f %f %f", &grade1, &grade2, &grade3);

        average = (grade1 + grade2 + grade3) / 3.0;

        printf("average grade = %.2f", average);

}
```

How *scanf* is used

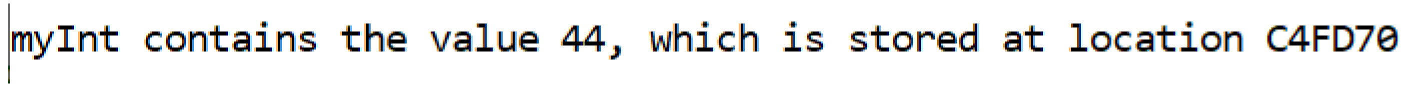
- The first input in scanf is the format text which tells scanf what the text the user inputs will contain, and how to parse it.
- In this example we are telling scanf that the input text will contain 3 floating point numbers, separated by spaces.
- After you enter the text via the keyboard and press enter, scanf *parses* the input to find the 3 floating point numbers.
- It then stores them in the variables which you provide to it.
- Putting the & in front of the variable name gives scanf access to the address of the variable, so it knows where to put the value it parses from the input text.

```
scanf("%f %f %f", &grade1, &grade2, &grade3);
```

&var gives you the address of var !

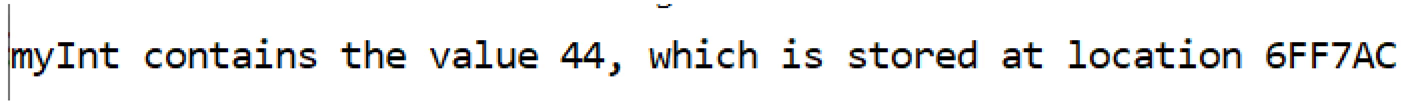
- So this code prints out the value of myInt and also the memory address where it is stored

```
int myInt = 44;

printf("myInt contains the value %d, which is stored at location %X \n", myInt, &myInt);
```



```
myInt contains the value 44, which is stored at location C4FD70
```

- See what happens when I run it again – different memory location used when program is ‘reloaded’



```
myInt contains the value 44, which is stored at location 6FF7AC
```

So….

- To repeat…when I run this command, I am giving scanf the addresses of the three variables (grade1, grade2, grade3) so that it can store values there when it reads from the input (keyboard)

```
scanf("%f %f %f", &grade1, &grade2, &grade3);
```

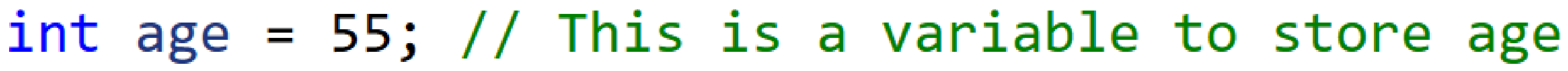
COMMENTS

What are Comments?

- Comments are non-code text that you can add into your program.
- They are generally used to make the code more readable.
- You should use these to explain what your code is doing.

Using Comments

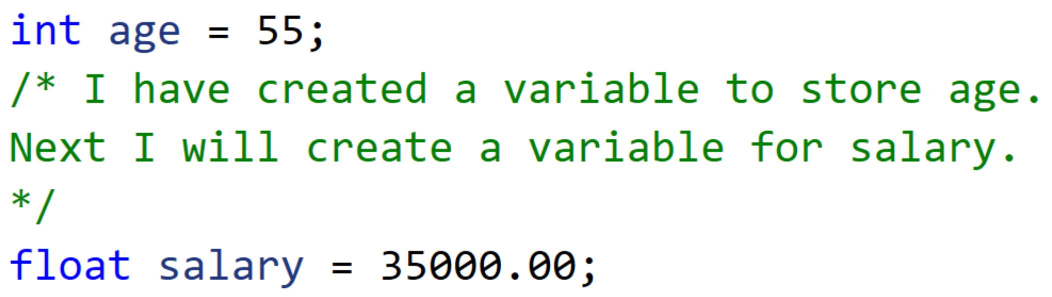
- In C, you can write a comment using //
- Anything that comes after // will be ignored by the compiler.
- E.g.



```c
int age = 55; // This is a variable to store age
```

Comment Blocks

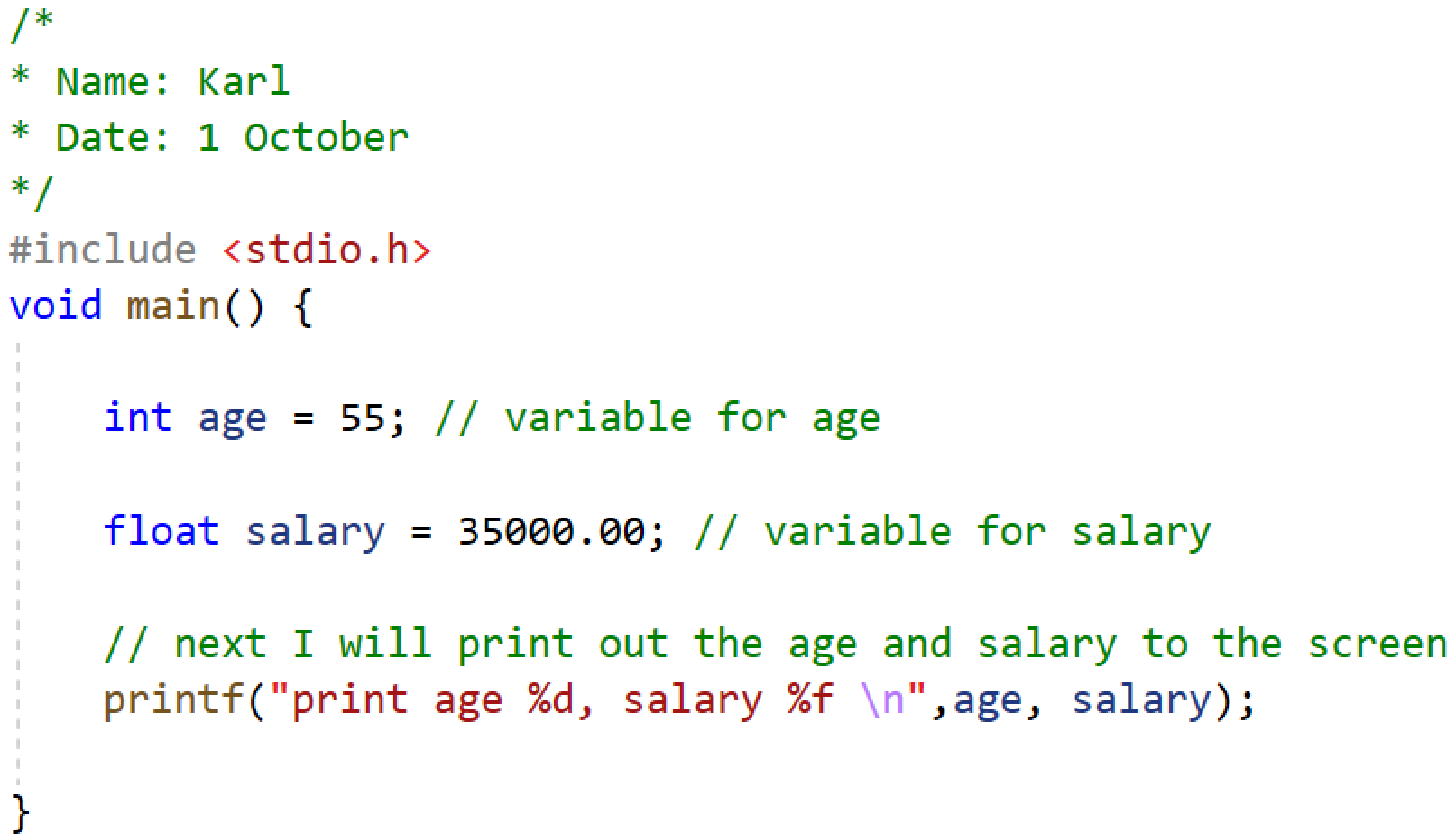
- In C, you can comment multiple lines of code using /**/
- Anything that comes in between /* and */ will be ignored by the compiler.
- E.g.



```c
int age = 55;
/* I have created a variable to store age.
Next I will create a variable for salary.
*/
float salary = 35000.00;
```

Comments Example

- You can see how comments make the following code easier to understand:



```
/*
* Name: Karl
* Date: 1 October
*/
#include <stdio.h>
void main() {

    int age = 55; // variable for age

    float salary = 35000.00; // variable for salary

    // next I will print out the age and salary to the screen
    printf("print age %d, salary %f \n",age, salary);

}
```

CODE EXAMPLES

C Program Example

- Lets now look at a C program.

Programming

CT103

Week 2b

Lecture Content

- Last lecture (Week 2a):
  - C basic variable types and their size in bytes.
  - Naming variables.
  - Declaring and initialising variables.
  - Comments.

- Today’s lecture (Week 2b):
  - Basic maths operators.
  - Modulus.
  - Else if statements.
  - Nested if statements.

MATH OPERATORS

Math Operators

- Addition: **+**
- Subtraction: **-**
- Multiplication: *****
- Division: **/**

- Modulus (same as ‘remainder’ in maths): **%**
  - This one is very useful!

Modulus

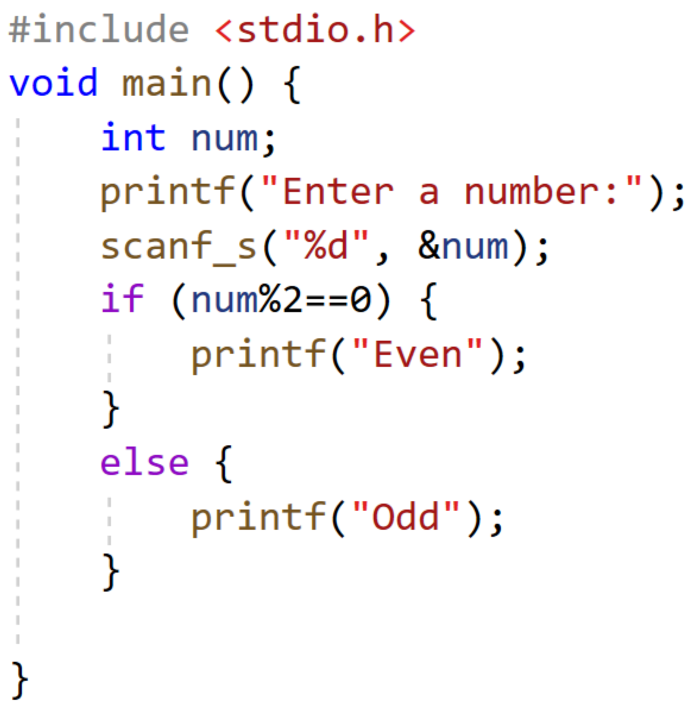
- Why is modulus % useful?
- The modulus operator allows us to get the remainder when doing integer division.
- What is the remainder?
  - When dividing two numbers that don't divide evenly, the remainder is what is left over.
  - E.g. 9/4 = 2 with a remainder of 1.
  - In C: 9%4 = 1.

Modulus

- I still don’t understand why is modulus % useful?
- Lets say I want you to write a program that tells me if a number is even or odd.
- How would you do it?

Odd or Even?

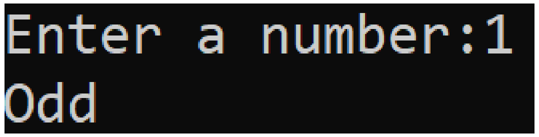
- Lets say I want you to write a program that tells me if a number is even or odd. How would you do it?
  - Answer: Use Modulus!



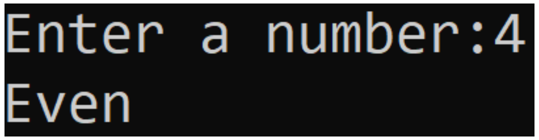
```c
#include <stdio.h>
void main() {
    int num;
    printf("Enter a number:");
    scanf_s("%d", &num);
    if (num%2==0) {
        printf("Even");
    }
    else {
        printf("Odd");
    }

}
```

Odd or Even C Program

```c
#include <stdio.h>
void main() {
    int num;
    printf("Enter a number:");
    scanf_s("%d", &num);
    if (num%2==0) {
        printf("Even");
    }
    else {
        printf("Odd");
    }

}
```



```
Enter a number:1
Odd
```



```
Enter a number:4
Even
```

ORDER OF OPERATORS

Order of operators

- C doesn't always compute maths operations in the order you might expect
- For example, is ans = 21 or 11?

```
ans = 5 + 2 * 3;
```

- C always does the multiplication before the addition

To be sure to be sure ….

```
avg = i + j + k + l / 4;
```

- C computes the division first, which means that

```
avg = 10 + 2 + 4 + 8/4;
```

  Would be equal to 18…. Not what we want

- Always use parentheses, like:

```
avg = (10 + 2 + 4 + 8)/4;
```

  Equal to 6

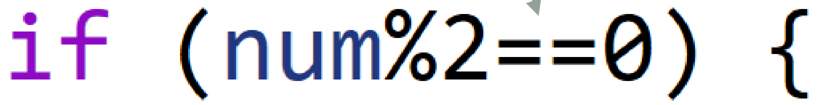
- In effect you are dictating explicitly the order you want operations evaluated in – much safer!

CHECKING IF NUMBERS ARE EQUAL

= or ==

- You might have noticed in the previous example of modulus that we used == in our IF statement.

- Why did you do this? Was this a typo?



```
if (num%2==0) {
```

= or ==

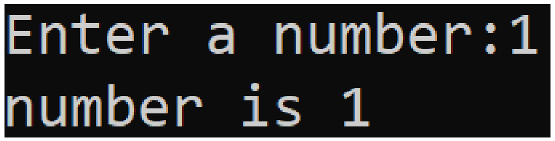
- **Assigning value:** A single equals sign (=) assigns a value to something (e.g. int i = 5;)
  - Used when initializing or setting variables.

- **Checking equality:** Two equals signs together (==) is a relational operator to check what is on either side of the operator is the same
  - Often used in IF statements and While loops.
  - The result is either true or false
  - In C, true is 1, and false is 0

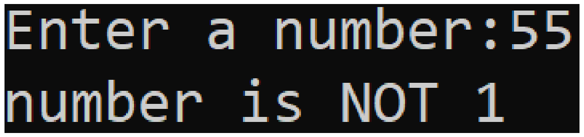
Equality Example

- **Checking equality:**

```
if (num==1) {
    printf("number is 1 \n");
}
else {
    printf("number is NOT 1 \n");
}
```



```
Enter a number:1
number is 1
```



```
Enter a number:55
number is NOT 1
```

Testing Data

- The *if* statement works like:
  - If something is true then do A, otherwise do B

- C’s Relational Operators:

Relational Operator	Description
==	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
!=	Not equal to

Examples

```
int i= 5;
int j = 10;
int k = 15;
int l = 5;
```

- The following are **true**

```
i == l          j < k          k > l          j != k
```

- The following are **false**

```
i > j           k< j           k == l
```

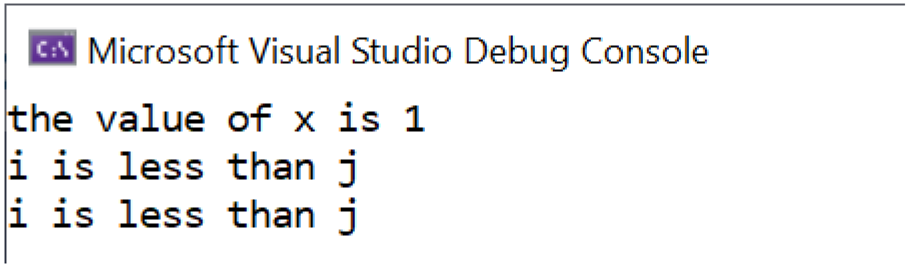
True vs False

- In standard C there is no Boolean data type, so we usually use an integer to store a true / false value

- It is very easy, because in C
  - True is represented by 1
  - False is represented by 0

Example – try it out yourself

```
int i = 4, j = 7;

int x = (i < j);

printf("the value of x is %d \n", x);

if (x)
{
        printf("i is less than j \n");
}

if (i < j)
{
        printf("i is less than j \n");
}
```



```
the value of x is 1
i is less than j
i is less than j
```

So…

- What is in a and b after these lines are executed?:

```
int a,b;
a = (4 < 10);
b = (8 == 9);
```

- Put them in a program and see for yourself if you are not sure

IF, ELSE IF, ELSE

More than one Decision

- Up until now, we have only considered a simple if – else statement,
  - i.e. if (True){// do something} else{//do something different}
- What do we do if we have multiple conditions?
  - If the grade > 85, A. If grade > 70, B. If grade > 55, C… etc.
- Will simply using multiple if statements work?

Grade Example

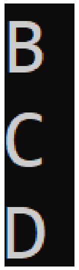
- Will the following code work as we want?

```
int grade = 81;
if (grade>85) {
    printf("A \n");
}
if (grade > 70){
    printf("B \n");
}
if (grade > 55) {
    printf("C \n");
}
if (grade > 40) {
    printf("D \n");
}
else {
    printf("F \n");
}
```

Grade Example

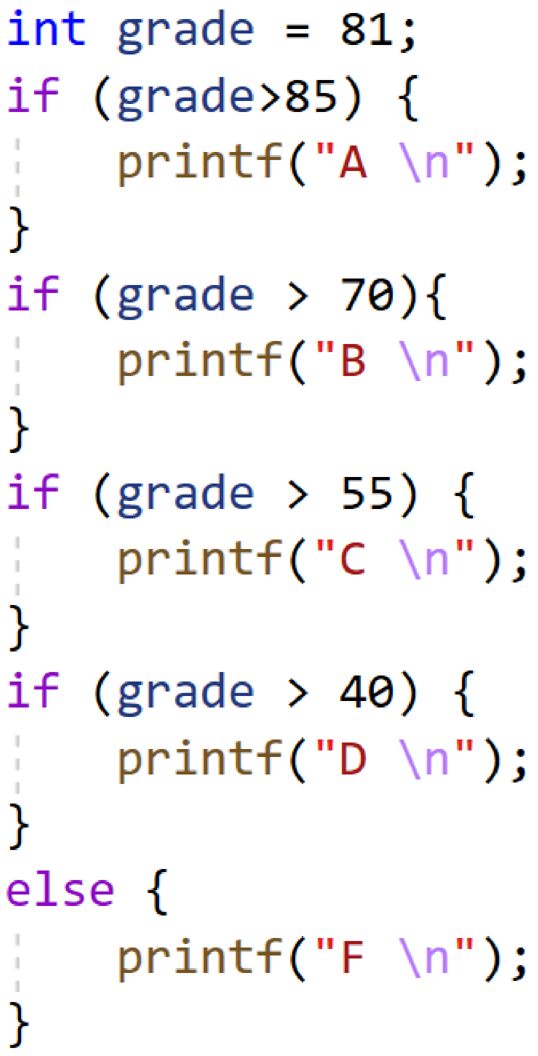
- Will the following code work as we want?

- Output:



```
B
C
D
```

- The student got a B. Why is the program also printing C and D?



```
int grade = 81;
if (grade>85) {
    printf("A \n");
}
if (grade > 70){
    printf("B \n");
}
if (grade > 55) {
    printf("C \n");
}
if (grade > 40) {
    printf("D \n");
}
else {
    printf("F \n");
}
```

Else if

- We need to use ‘Else if’ statements!
- Will the new code now work as we want?

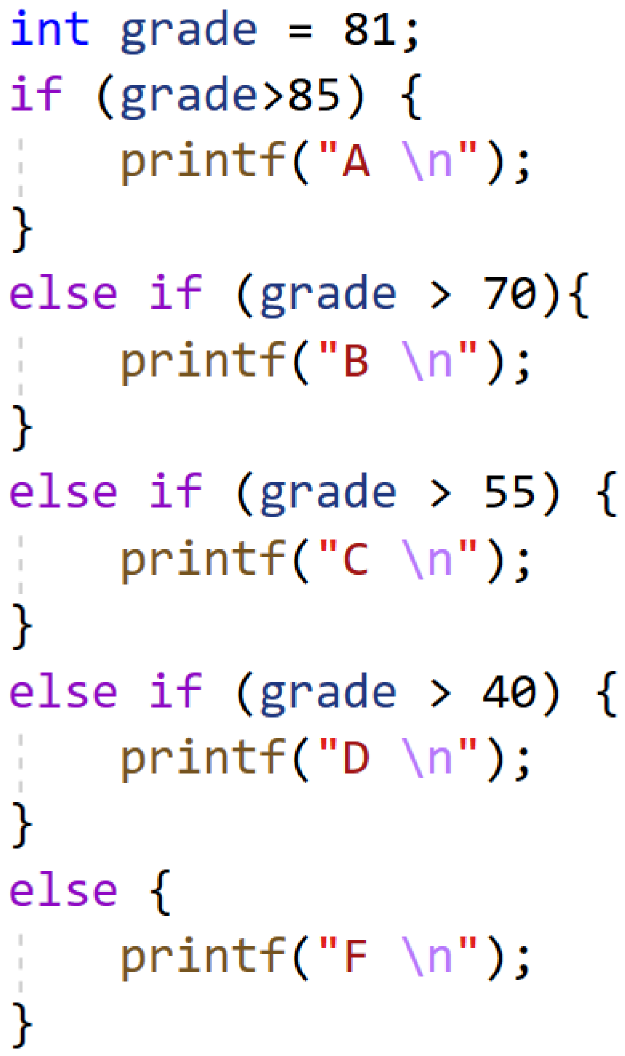
```c
int grade = 81;
if (grade>85) {
    printf("A \n");
}
else if (grade > 70){
    printf("B \n");
}
else if (grade > 55) {
    printf("C \n");
}
else if (grade > 40) {
    printf("D \n");
}
else {
    printf("F \n");
}
```

Else if

- Will the new code now work as we want?
- Output:

```
B
```

- Success!
- ‘Else if’ will not check subsequent ‘If’ statements after a condition is **True**.



```
int grade = 81;
if (grade>85) {
    printf("A \n");
}
else if (grade > 70){
    printf("B \n");
}
else if (grade > 55) {
    printf("C \n");
}
else if (grade > 40) {
    printf("D \n");
}
else {
    printf("F \n");
}
```

CHECKING TWO CONDITIONS

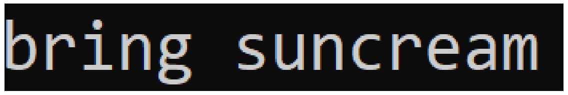
Two Conditions in an If Statement

- Up until now, we have only considered a single condition in our if statement.
- What if we want to check if two conditions are true?
- For example:
  - If there is no rain and it is warm, bring suncream.
- How would we write a program to do this?

Two Conditions in an If Statement

- There are two ways of doing this.
- The first method is to use one if statement within another if statement. These are called ‘**nested**’ if statements.

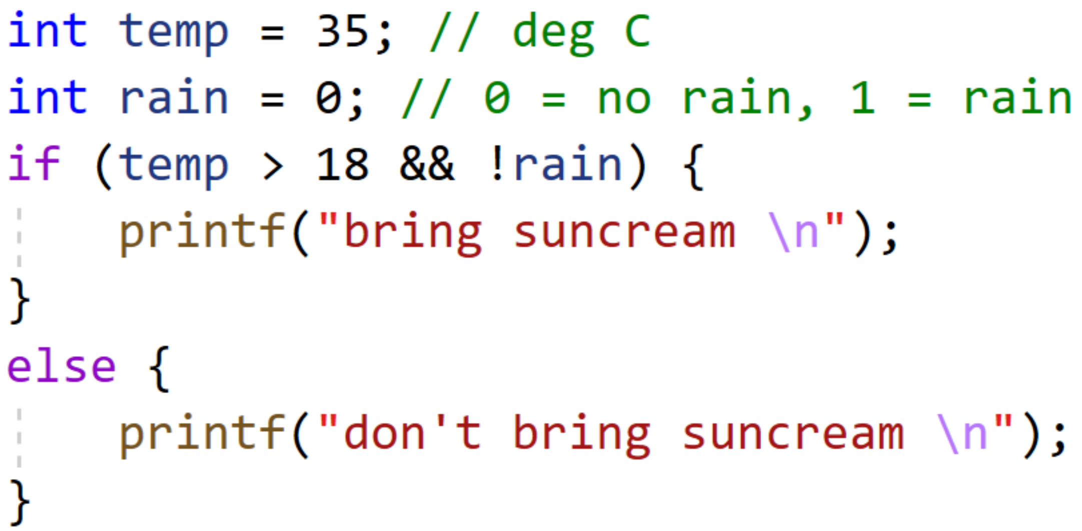
```
int temp = 35; // deg C
int rain = 0; // 0 = no rain, 1 = rain
if (temp > 18) {
    if (!rain) {
        printf("bring suncream \n");
    }
}
else {
    printf("don't bring suncream \n");
}
```



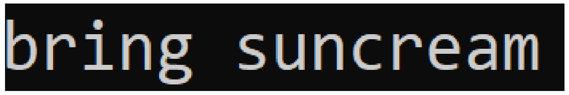
```
bring suncream
```

Two Conditions in an If Statement

- The second way to do this is to use **Boolean logic**.
- This involves using **AND**, represented by **&&** in C.
- This makes our code shorter.
- We will <u>discuss Boolean logic next Monday </u>in more detail.



```c
int temp = 35; // deg C
int rain = 0; // 0 = no rain, 1 = rain
if (temp > 18 && !rain) {
    printf("bring suncream \n");
}
else {
    printf("don't bring suncream \n");
}
```



```
bring suncream
```

PROGRAMMING

CT103

Week 3a

Sign in on Blackboard

- Please sign in on blackboard for CT103.
- I will leave the sign in option open for an hour after the lecture.

Lecture Content

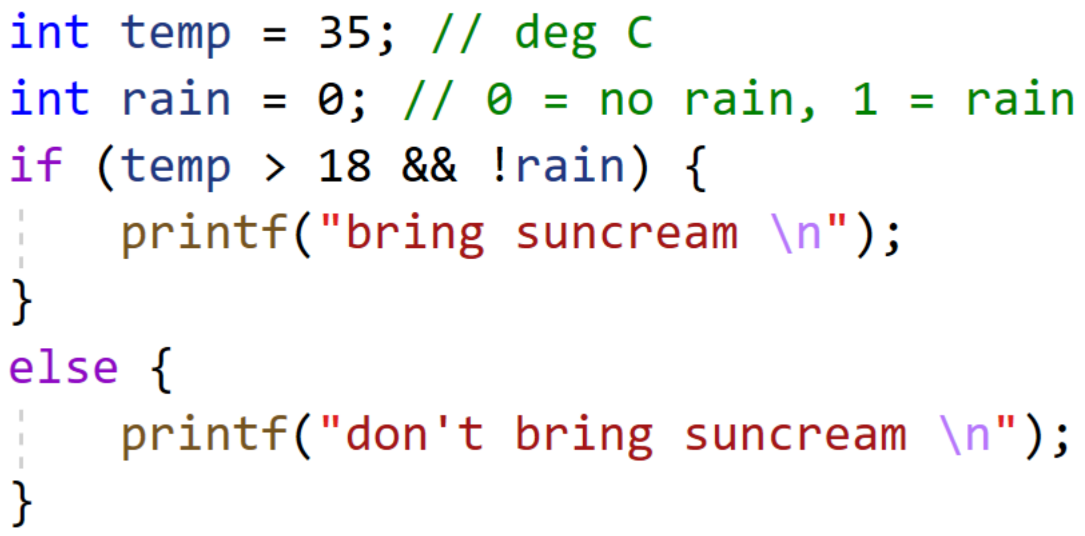
- Last lecture (Week 2b):
  - Basic maths operators.
  - Modulus.
  - Else if statements.
  - Nested if statements.

- Today's lecture (Week 3a):
  - Boolean Logic.
  - Switch Statements.
  - Characters.

BOOLEAN ALGEBRA

Boolean Algebra

- We introduced Boolean logic last week.
- We saw AND which is && in C.
- We also saw NOT which is ! in C.



```c
int temp = 35; // deg C
int rain = 0; // 0 = no rain, 1 = rain
if (temp > 18 && !rain) {
    printf("bring suncream \n");
}
else {
    printf("don't bring suncream \n");
}
```

Boolean Algebra

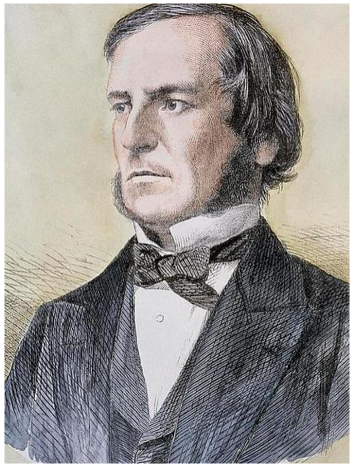
- What is Boolean Algebra?

- Definition: Boolean Algebra is a form of algebra in which all variables are either True or False.

- Boolean operators can then applied to these variables.

George Boole

- Boolean algebra is named after George Boole who first introduced it.
- George Boole was a Professor in UCC, Cork Ireland.



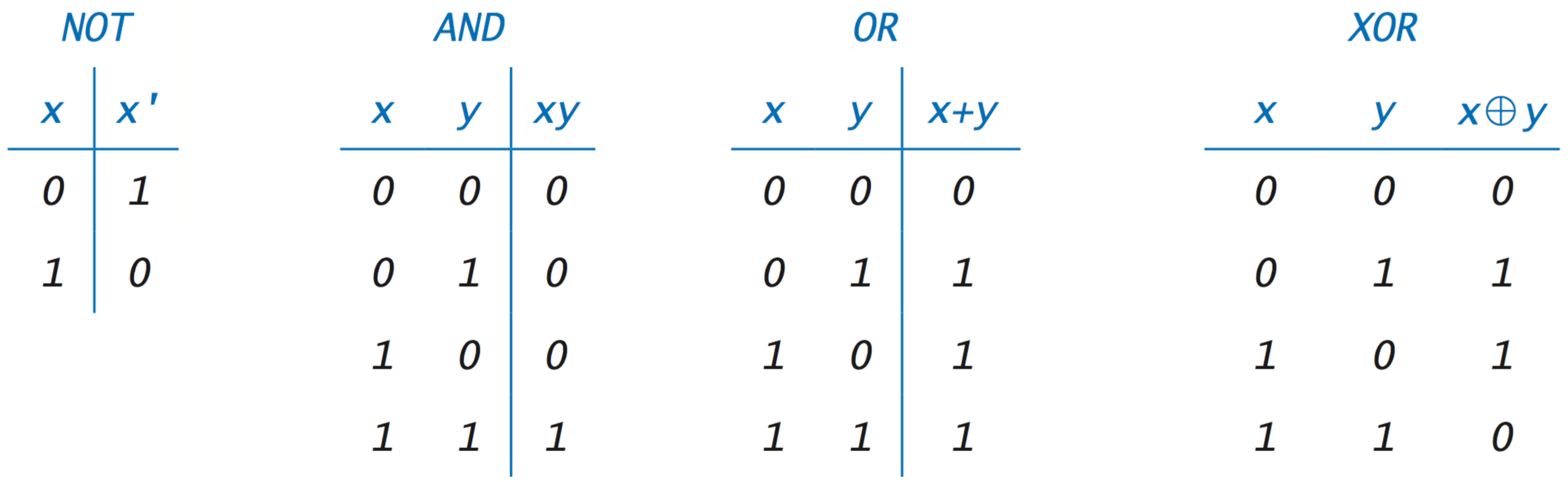
George Boole
Image from: Wikipedia

Boolean Operators

- The primary Boolean operators are:
  - AND (In C: **&&**)
  - OR (In C: **||**)
  - NOT (In C: **!**)
  - XOR (In C: **!=**)

Truth Tables

- The following truth table shows how each of these operators work.
- In C: 1 = True, 0 = False



NOT

x	x'
0	1
1	0

AND

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

OR

x	y	$x+y$
0	0	0
0	1	1
1	0	1
1	1	1

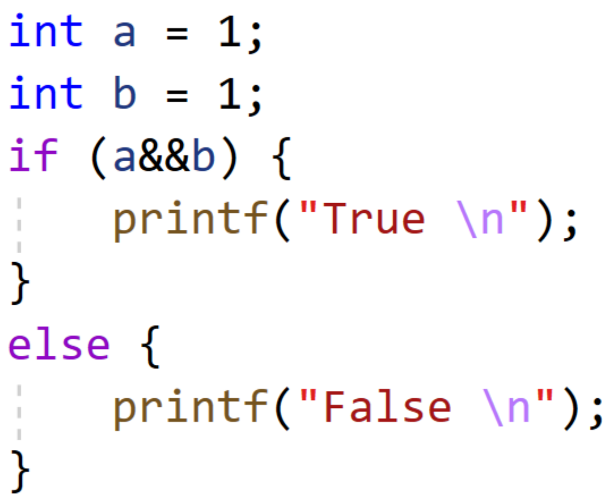
XOR

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Source: https://introcs.cs.princeton.edu/java/71boolean/

Boolean Operators in C

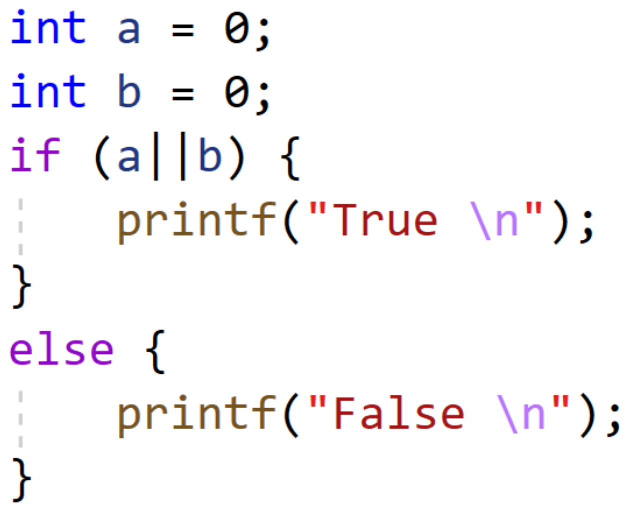
- AND
- What will the following code output?



```
int a = 1;
int b = 1;
if (a&&b) {
    printf("True \n");
}
else {
    printf("False \n");
}
```

Boolean Operators in C

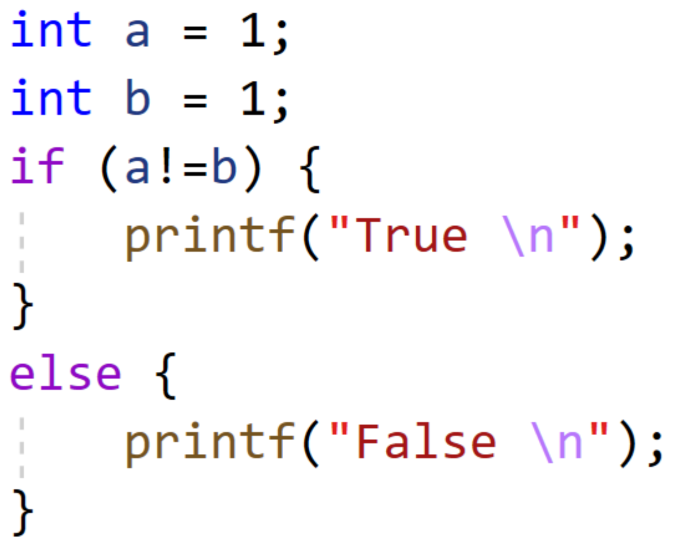
- OR
- What will the following code output?



```
int a = 0;
int b = 0;
if (a||b) {
    printf("True \n");
}
else {
    printf("False \n");
}
```

Boolean Operators in C

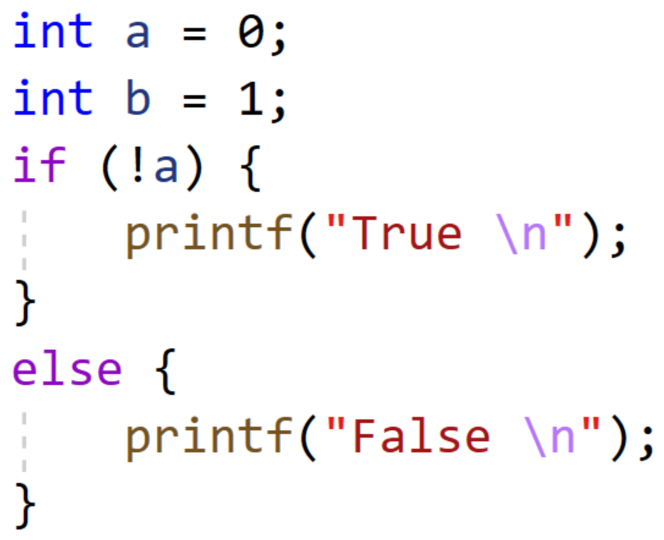
- XOR
- What will the following code output?



```c
int a = 1;
int b = 1;
if (a!=b) {
    printf("True \n");
}
else {
    printf("False \n");
}
```

Boolean Operators in C

- NOT
- What will the following code output?



```c
int a = 0;
int b = 1;
if (!a) {
    printf("True \n");
}
else {
    printf("False \n");
}
```

SWITCH STATEMENTS

Switch statement

- Switch statements test the value of a variable and compares it with multiple cases.

- If case match is not found, default statement is executed.

- Benefits of switch statements:
  - Switch can be tidier.
  - Can be executed faster.

Switch Template

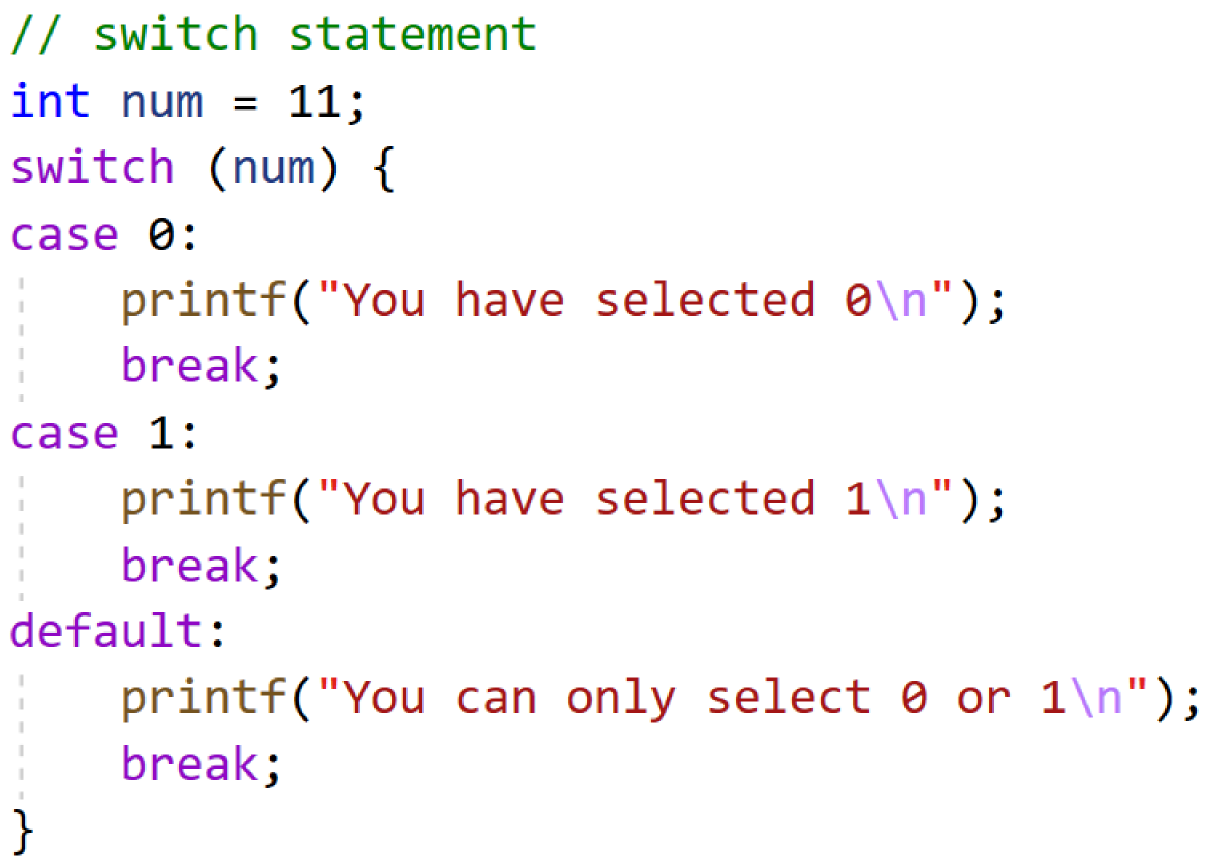
Note : not ;

```
switch (expression)
{
    case value1:
     // do something
       break;
    case value2:
       // do something else
       break;
       ...
    default:
       break;
 }
```

- Expression is evaluated.
    - **Expression** must return an <u>int</u>.
    - Expression can be an int.
- Value of expression compared to each case.
- *Break* important to avoid running on and executing the next case (if you leave it out, it will!)

Switch Example in C

- Switch statement that checks if a number is 0 or 1.

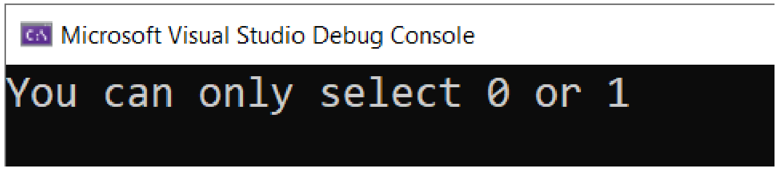


```c
// switch statement
int num = 11;
switch (num) {
case 0:
    printf("You have selected 0\n");
    break;
case 1:
    printf("You have selected 1\n");
    break;
default:
    printf("You can only select 0 or 1\n");
    break;
}
```

Sample Output

- If we run the following statement with num =11, we get the default response.

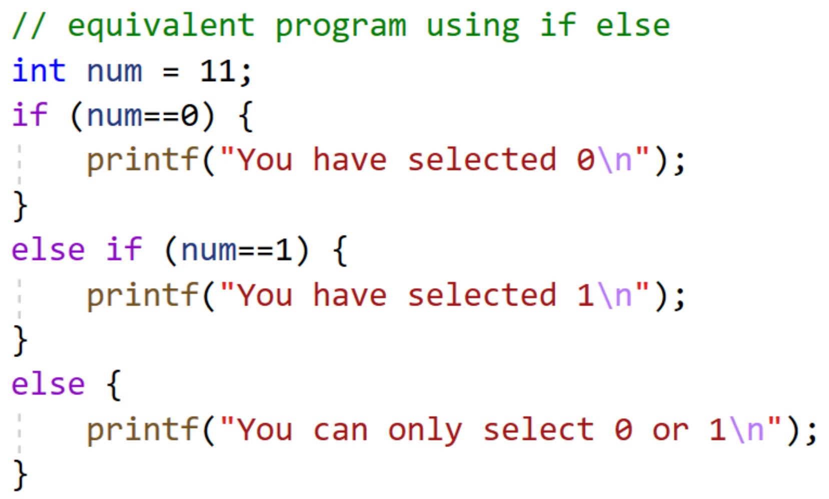
```c
// switch statement
int num = 11;
switch (num) {
case 0:
    printf("You have selected 0\n");
    break;
case 1:
    printf("You have selected 1\n");
    break;
default:
    printf("You can only select 0 or 1\n");
    break;
}
```



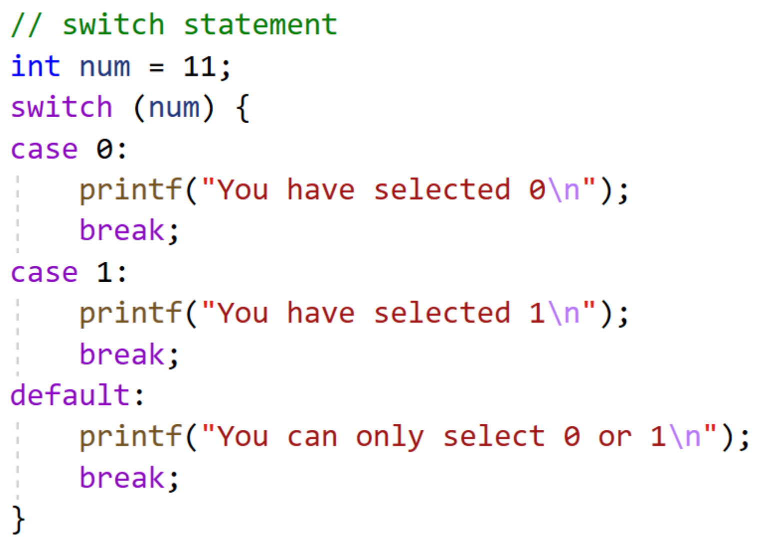
```
You can only select 0 or 1
```

Equivalent Program using IF Else

- Below we can compare both programs side by side using If Else and using Switch.



```
// equivalent program using if else
int num = 11;
if (num==0) {
    printf("You have selected 0\n");
}
else if (num==1) {
    printf("You have selected 1\n");
}
else {
    printf("You can only select 0 or 1\n");
}
```



```
// switch statement
int num = 11;
switch (num) {
case 0:
    printf("You have selected 0\n");
    break;
case 1:
    printf("You have selected 1\n");
    break;
default:
    printf("You can only select 0 or 1\n");
    break;
}
```

CHARACTERS

Characters in C

- What are they really?
- How are they stored?
- How do read them in.
- Hanging newline characters in the input
  - And how to get rid of them

How are variable values stored

- 1’s and 0’s – everything is stored in binary format.
- That includes characters also. Each character has a different binary value.
- char c = ’a’;
- Note: Singe quotations for characters. We learnt this last week…
- Other languages, e.g. python, are less strict with quotations.

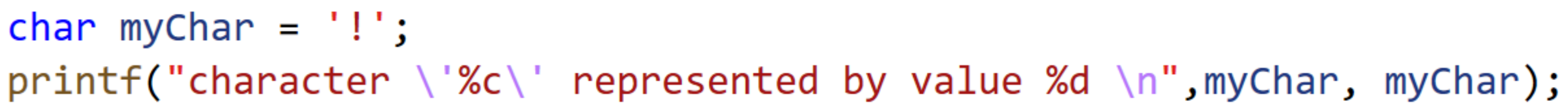
What are the values behind the characters?

- This is where having a standard character table comes in.
- Enough people in industry got together and decided what the value of each character should be.
- So for example:
  - ‘a’ is stored as the number 97 (binary 1100001)
  - ‘A’ is stored as the number 65 (binary 1000001)
  - ‘?’ is stored as the number 63 (binary 111111)
  - ‘#’ is stored as the number 35 (binary 100011)
  - … and so on

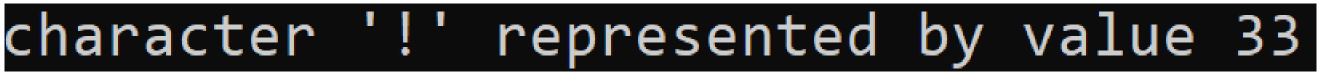
  - The full set is called a character set, such as the original ASCII (American Standard Code for Information Interchange) table
  - Since superseded by UTF, but UTF includes the basic ASCII English character set

How to see the value of a character

- The following will show you the value of a character:



```c
char myChar = '!';
printf("character \'%c\' represented by value %d \n",myChar, myChar);
```



```
character '!' represented by value 33
```

EXAMPLE C PROBLEM

Quality Control Program

- You are writing a computer program for a manufacturer to check if the quality of a product. Write a C program:
  1. Begin with the width and height of the product in meters.
  2. Convert the width and height to millimetres.
  3. Check if the product width is outside of the acceptable region. Min width = 200mm. Max width = 230mm.
  4. Do step 3. twice, first using AND, then using OR.
  5. Categorize the height as short, medium, or tall based on the table below:

Category	Min Height (mm)	Max Height (mm)
Short	-	100
Medium	100	120
Tall	120	-

PROGRAMMING

CT103

Week 3b

Sign in on Blackboard

- Please sign in on blackboard for CT103.
- I will leave the sign in option open for an hour after the lecture.

Note on Lab Assignments

- Please make sure to bring your laptop if you can.

- Submit your **.c file** and **.doc file** in a **zipped** up **folder**.

- Make sure your code in your .c file matches your code in your .doc file.

- Make sure you do the assignments yourself. It is fine to ask each other questions or use C code I provide in lectures/tutorials.
- Do not copy another students assignment. Plagiarism is taken seriously by the university.
- If instances of plagiarism are detected, all students involved will receive a grade of zero for the assignment and may be subject to further disciplinary proceedings.

Lecture Content

- Last lecture (Week 3a):
  - Boolean Logic.
  - Switch Statements.
  - Characters.

- Today’s lecture (Week 3b):
  - Loops.
  - While loops.
  - Do while loops.
  - Example C program.

WHY DO WE NEED LOOPS?

Loops

- Up until now we have not actually looked at any C programs that use loops.

- Loops are useful if we want to do the same task more than once.

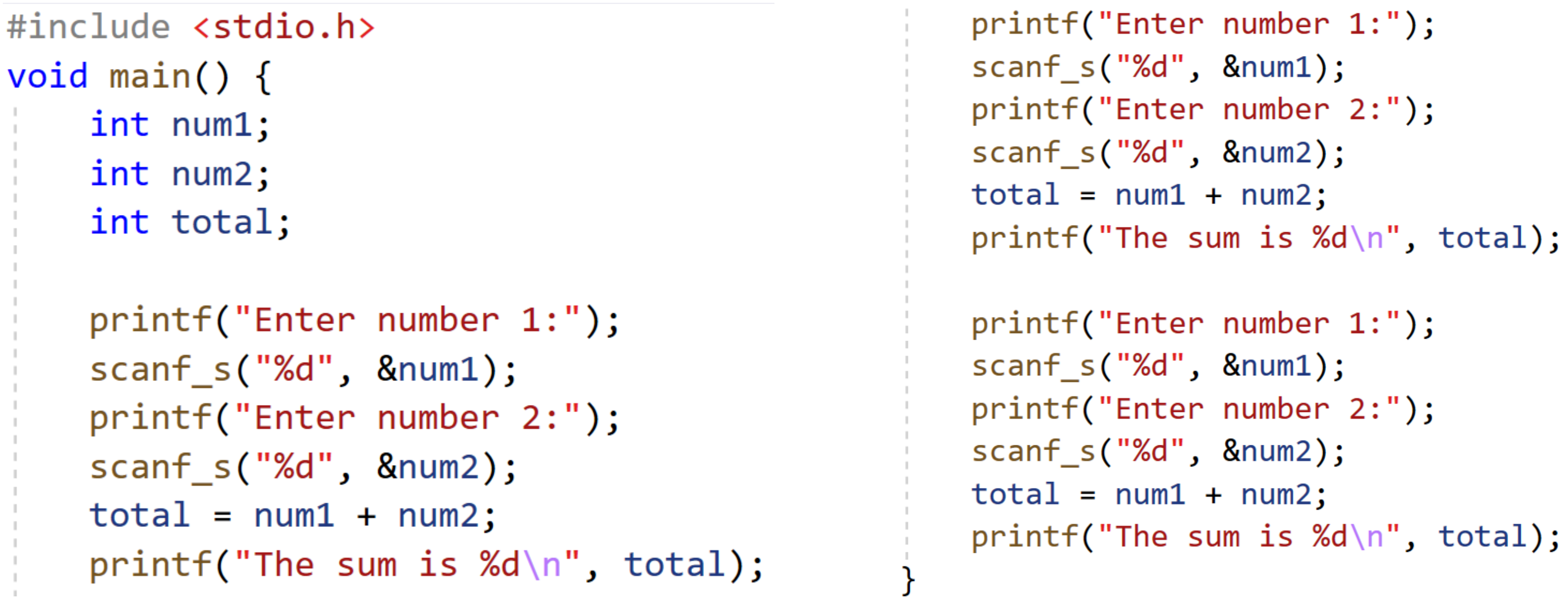
- If we did not have loops, we would have to rewrite the same code over and over.
  - This would be time consuming, unreadable and difficult to change.

Motivating Loops Example

- How would I write a program that would do the following 3 times **without using loops**.
  - Read in two numbers.
  - Add them together.
  - Print the result.

Motivating Loops Example

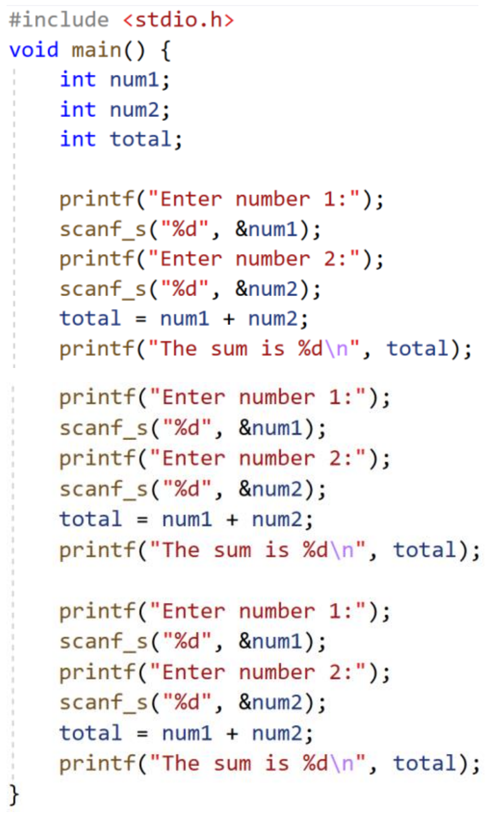
- You could do the following:



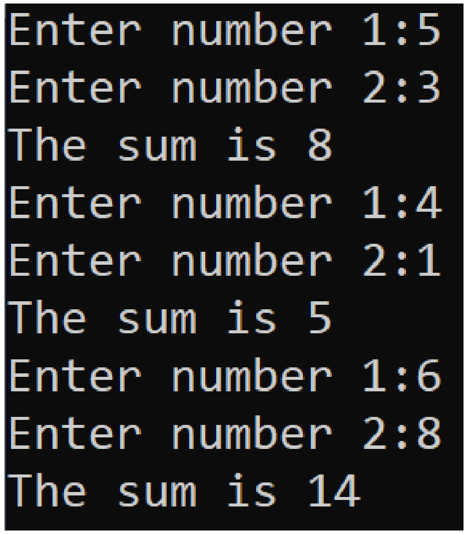
```c
#include <stdio.h>
void main() {
    int num1;
    int num2;
    int total;

    printf("Enter number 1:");
    scanf_s("%d", &num1);
    printf("Enter number 2:");
    scanf_s("%d", &num2);
    total = num1 + num2;
    printf("The sum is %d\n", total);
    printf("Enter number 1:");
    scanf_s("%d", &num1);
    printf("Enter number 2:");
    scanf_s("%d", &num2);
    total = num1 + num2;
    printf("The sum is %d\n", total);

    printf("Enter number 1:");
    scanf_s("%d", &num1);
    printf("Enter number 2:");
    scanf_s("%d", &num2);
    total = num1 + num2;
    printf("The sum is %d\n", total);
}
```

Motivating Loops Example

- When you run the code, it works and gives the following output:



```
#include <stdio.h>
void main() {
    int num1;
    int num2;
    int total;

    printf("Enter number 1:");
    scanf_s("%d", &num1);
    printf("Enter number 2:");
    scanf_s("%d", &num2);
    total = num1 + num2;
    printf("The sum is %d\n", total);

    printf("Enter number 1:");
    scanf_s("%d", &num1);
    printf("Enter number 2:");
    scanf_s("%d", &num2);
    total = num1 + num2;
    printf("The sum is %d\n", total);

    printf("Enter number 1:");
    scanf_s("%d", &num1);
    printf("Enter number 2:");
    scanf_s("%d", &num2);
    total = num1 + num2;
    printf("The sum is %d\n", total);
}
```

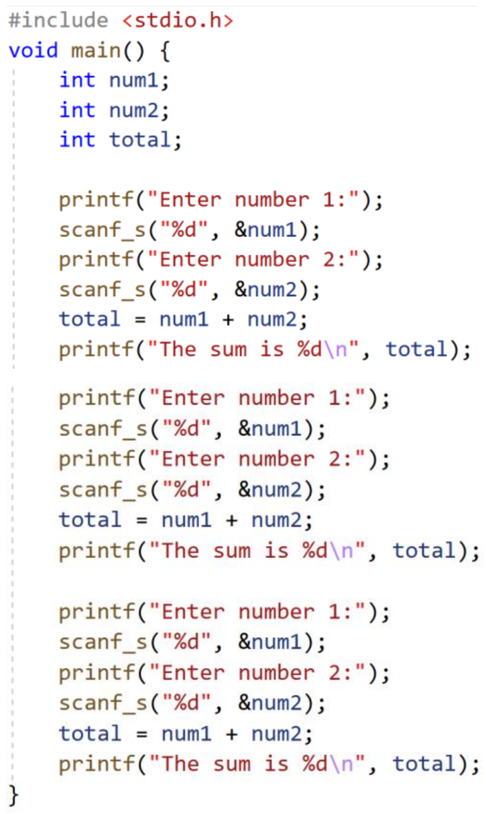


```
Enter number 1:5
Enter number 2:3
The sum is 8
Enter number 1:4
Enter number 2:1
The sum is 5
Enter number 1:6
Enter number 2:8
The sum is 14
```

Motivating Loops Example

- There are plenty of problems with this:
- The code is longer than it needs to be.
- What if we want to change it so that we are subtracting numbers instead of adding numbers?
- This is doable for repeating this task 3 times, what if we want to do it 100 times? Or 10000 times?

- Loops are a way of solving these issues!



```
#include <stdio.h>
void main() {
    int num1;
    int num2;
    int total;

    printf("Enter number 1:");
    scanf_s("%d", &num1);
    printf("Enter number 2:");
    scanf_s("%d", &num2);
    total = num1 + num2;
    printf("The sum is %d\n", total);

    printf("Enter number 1:");
    scanf_s("%d", &num1);
    printf("Enter number 2:");
    scanf_s("%d", &num2);
    total = num1 + num2;
    printf("The sum is %d\n", total);

    printf("Enter number 1:");
    scanf_s("%d", &num1);
    printf("Enter number 2:");
    scanf_s("%d", &num2);
    total = num1 + num2;
    printf("The sum is %d\n", total);
}
```

WHILE LOOPS

While Loops

- The first type of loop we will cover is called the **while loop**.

- The while loop will repeat a block of code over and over while some condition is true.

While Loops Template

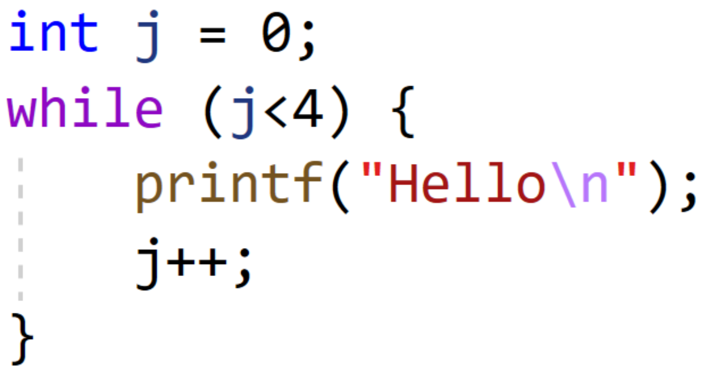
- While loops have the following structure:

```
while (condition) {
    // Do something
}
```

- We have some **condition**, e.g. number<10.
- While this condition is **True**, whatever is inside the curly brackets {} gets executed.
- This is useful for doing something more than once!

While Loops Example

- Lets look at the following simple while loop.



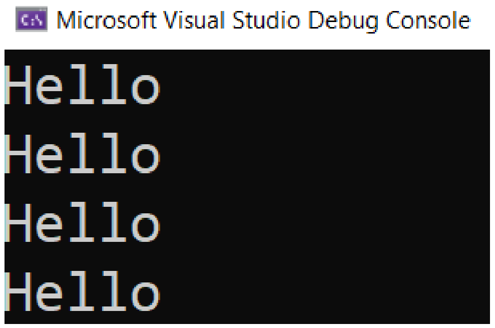
```c
int j = 0;
while (j<4) {
    printf("Hello\n");
    j++;
}
```

- This code will print “Hello” to the screen 4 times.

While Loops Example

- See the output of this code:

```c
int j = 0;
while (j<4) {
    printf("Hello\n");
    j++;
}
```


Microsoft Visual Studio Debug Console

```
Hello
Hello
Hello
Hello
```

- Of course don’t forget # include and void main!

Loops Example

- Remember the problem from earlier:
- How would I write a program that would do the following 3 times **without using loops**.
  - Read in two numbers.
  - Add them together.
  - Print the result.

- How would I now do this **using a while loop**?

While Loops Example

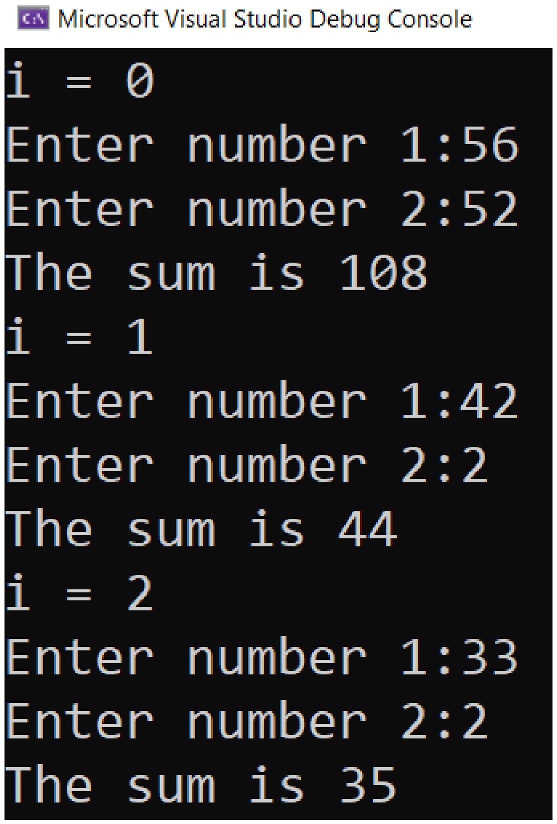
- You would do the following:

```
#include <stdio.h>
void main() {
    int num1;
    int num2;
    int total;
    int i = 0;

    while (i<3){
        printf("i = %d\n",i);
        printf("Enter number 1:");
        scanf_s("%d", &num1);
        printf("Enter number 2:");
        scanf_s("%d", &num2);
        total = num1 + num2;
        printf("The sum is %d\n", total);
        i++;
    }
}
```

While Loops Example

- Running the program will give you the following output:

```c
#include <stdio.h>
void main() {
    int num1;
    int num2;
    int total;
    int i = 0;

    while (i<3){
        printf("i = %d\n",i);
        printf("Enter number 1:");
        scanf_s("%d", &num1);
        printf("Enter number 2:");
        scanf_s("%d", &num2);
        total = num1 + num2;
        printf("The sum is %d\n", total);
        i++;
    }
}
```

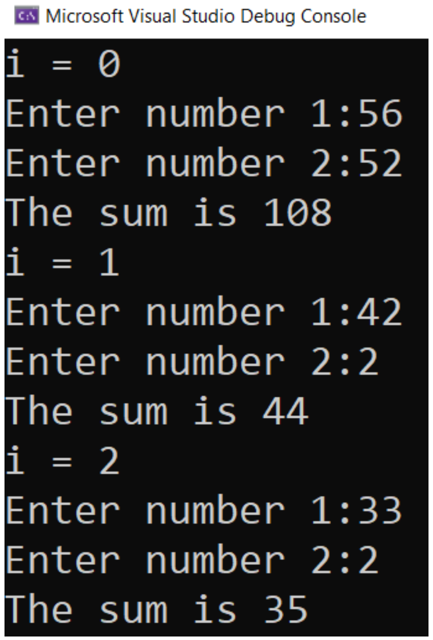

Microsoft Visual Studio Debug Console

```
i = 0
Enter number 1:56
Enter number 2:52
The sum is 108
i = 1
Enter number 1:42
Enter number 2:2
The sum is 44
i = 2
Enter number 1:33
Enter number 2:2
The sum is 35
```

While Loops Example

- This program gives the same output as the previous example (except I am printing an extra line that shows i).

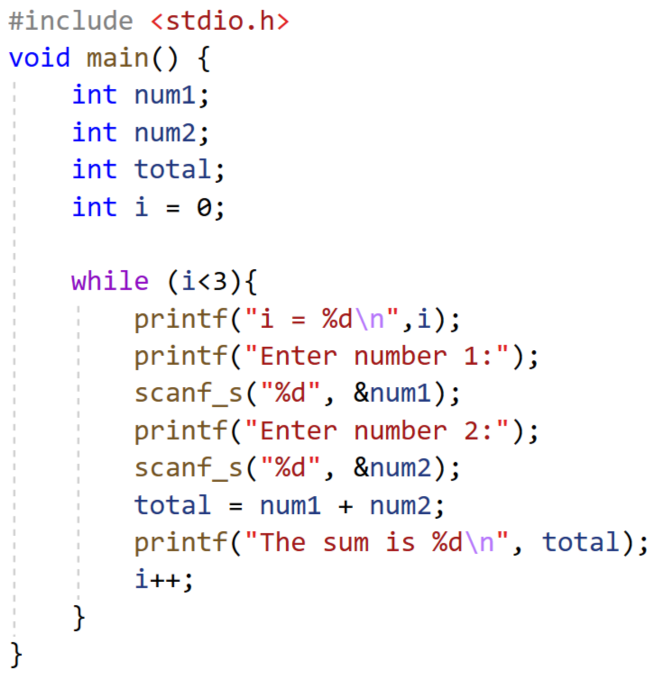
```c
#include <stdio.h>
void main() {
    int num1;
    int num2;
    int total;
    int i = 0;

    while (i<3){
        printf("i = %d\n",i);
        printf("Enter number 1:");
        scanf_s("%d", &num1);
        printf("Enter number 2:");
        scanf_s("%d", &num2);
        total = num1 + num2;
        printf("The sum is %d\n", total);
        i++;
    }
}
```


Microsoft Visual Studio Debug Console

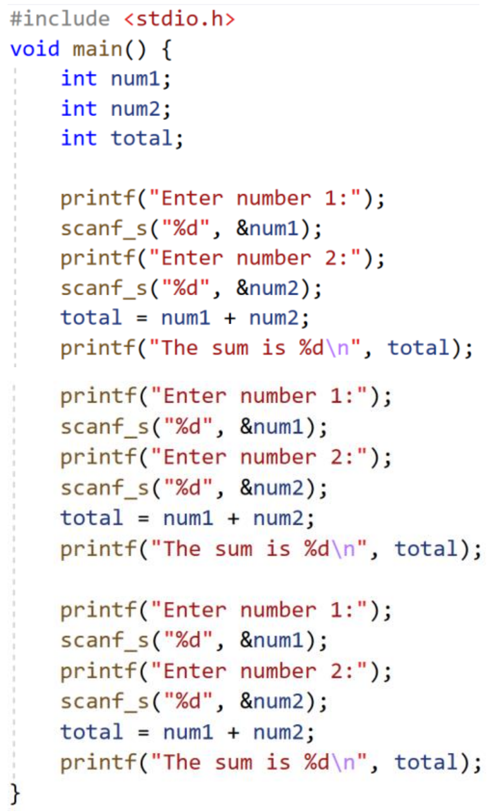
```
i = 0
Enter number 1:56
Enter number 2:52
The sum is 108
i = 1
Enter number 1:42
Enter number 2:2
The sum is 44
i = 2
Enter number 1:33
Enter number 2:2
The sum is 35
```

Solution Comparison

- The solution on the left uses while loops. The solution on the right does not.



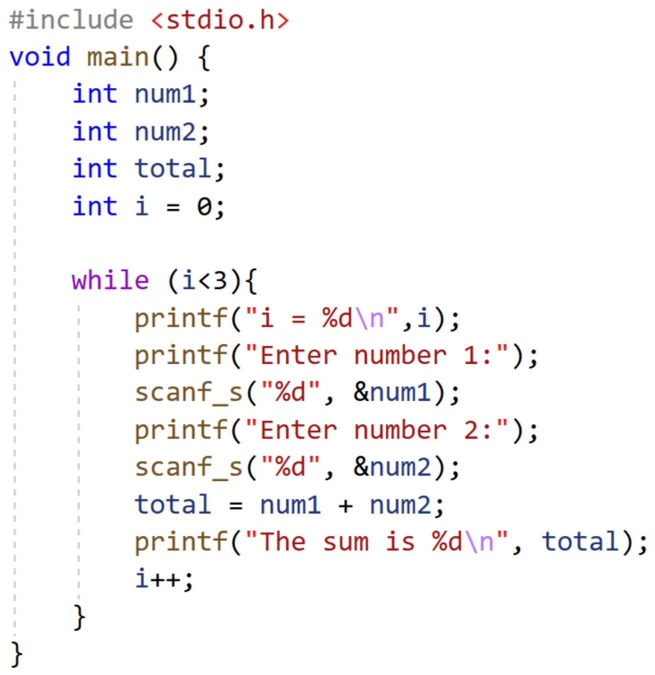
```c
#include <stdio.h>
void main() {
    int num1;
    int num2;
    int total;
    int i = 0;

    while (i<3){
        printf("i = %d\n",i);
        printf("Enter number 1:");
        scanf_s("%d", &num1);
        printf("Enter number 2:");
        scanf_s("%d", &num2);
        total = num1 + num2;
        printf("The sum is %d\n", total);
        i++;
    }
}
```



```c
#include <stdio.h>
void main() {
    int num1;
    int num2;
    int total;

    printf("Enter number 1:");
    scanf_s("%d", &num1);
    printf("Enter number 2:");
    scanf_s("%d", &num2);
    total = num1 + num2;
    printf("The sum is %d\n", total);

    printf("Enter number 1:");
    scanf_s("%d", &num1);
    printf("Enter number 2:");
    scanf_s("%d", &num2);
    total = num1 + num2;
    printf("The sum is %d\n", total);

    printf("Enter number 1:");
    scanf_s("%d", &num1);
    printf("Enter number 2:");
    scanf_s("%d", &num2);
    total = num1 + num2;
    printf("The sum is %d\n", total);
}
```

Advantages of While Loops

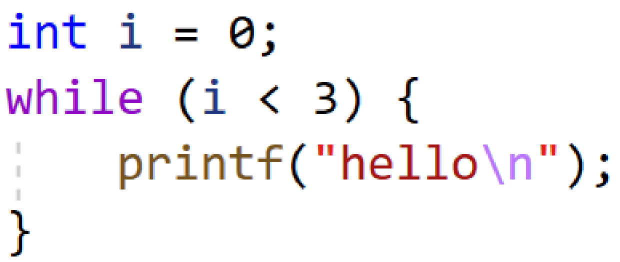
- Using while loops gives us the following advantages:
  - The code is easier to read.
  - If I want to run the same block of code 10000 times, all I need to do is change the condition to (i<10000).
  - If I want to make a change to multiply numbers together instead of adding them, I only need to do it once!



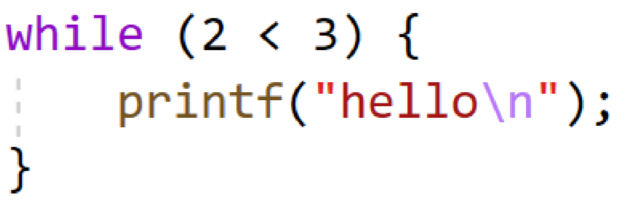
```c
#include <stdio.h>
void main() {
    int num1;
    int num2;
    int total;
    int i = 0;

    while (i<3){
        printf("i = %d\n",i);
        printf("Enter number 1:");
        scanf_s("%d", &num1);
        printf("Enter number 2:");
        scanf_s("%d", &num2);
        total = num1 + num2;
        printf("The sum is %d\n", total);
        i++;
    }
}
```

Avoid infinite loops!

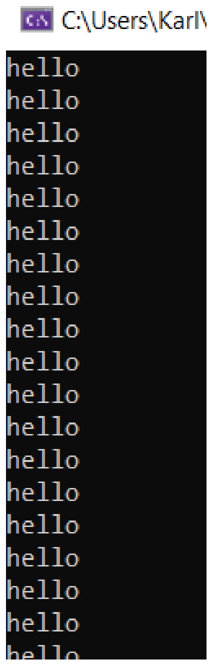
- An infinite loop is a loop that can never end.
- Therefore be careful with your while **condition**.
- You must change a variable inside the while loops body that is used in the condition – otherwise you could end up in an infinite loop.
- Below are both examples of infinite loops:



```c
int i = 0;
while (i < 3) {
    printf("hello\n");
}
```



```c
while (2 < 3) {
    printf("hello\n");
}
```

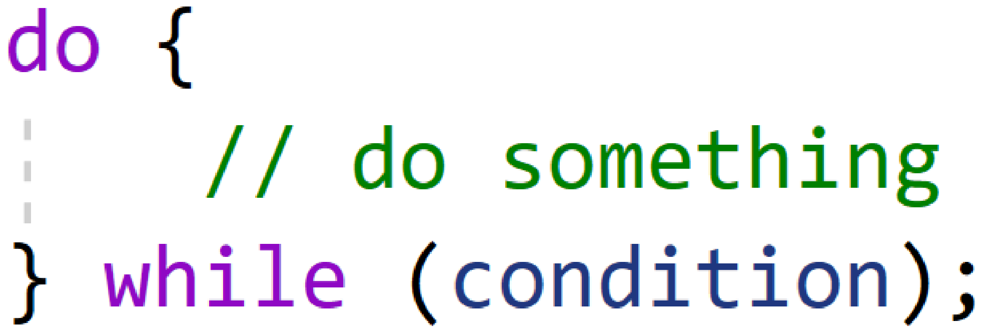


```
C:\Users\Karl\
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
hello
```

DO WHILE LOOPS

Do While Loops

- Do While Loops are a variant of while loops.
- They work in much the same way as while loops.
- Do While Loops have the following structure.

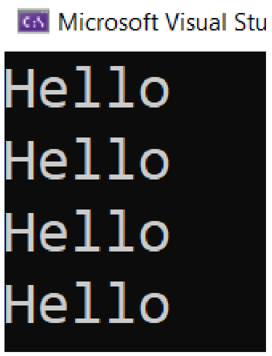


```
do {
    // do something
} while (condition);
```

Do While Loops Example

- Print “Hello” 4 times using a do while loop.

```c
int j = 0;
do {
    printf("Hello\n");
    j++;
} while (j<4);
```

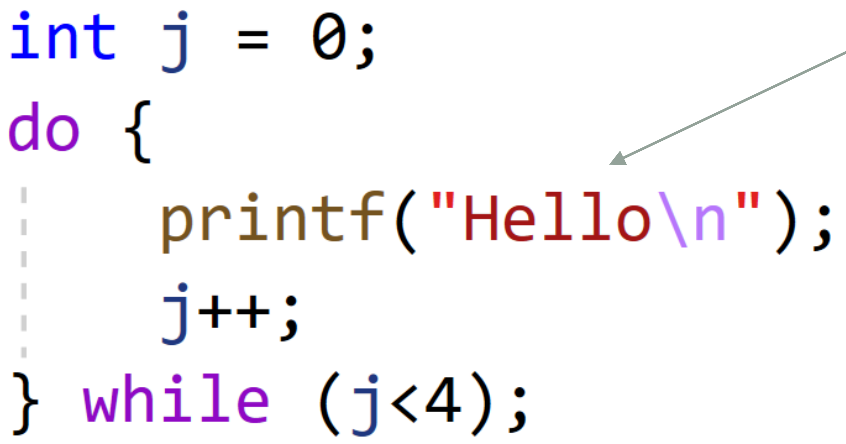


Microsoft Visual Stu

```
Hello
Hello
Hello
Hello
```

Why Use Do While Loops?

- You can use a do while loop if you want to ensure that you execute a block of code **at least once**.

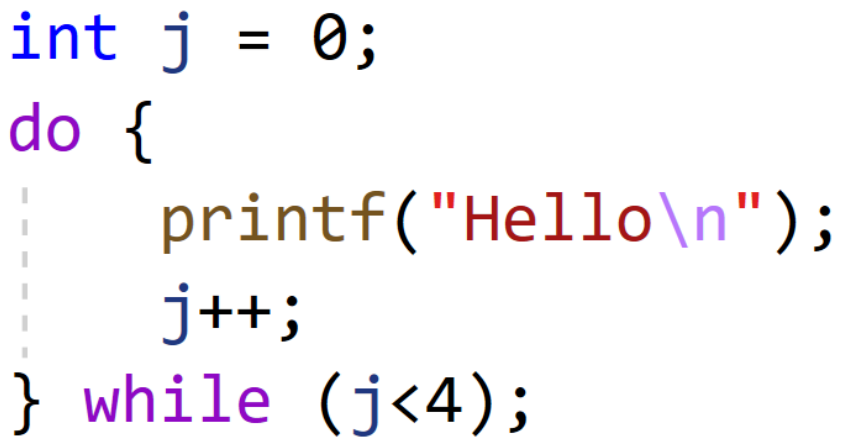


```
int j = 0;
do {
    printf("Hello\n");
    j++;
} while (j<4);
```

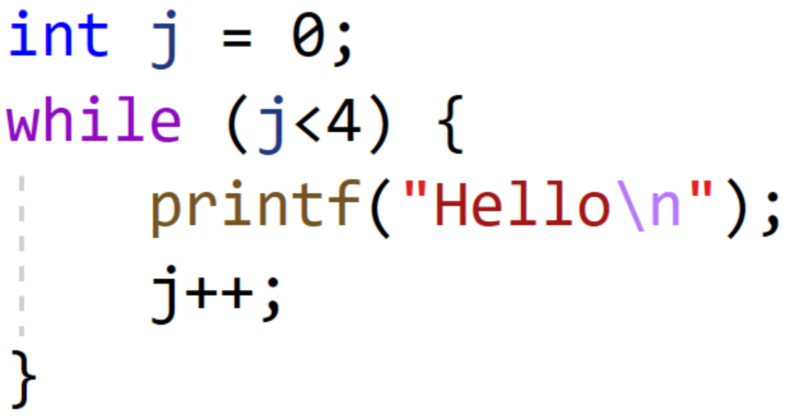
“Hello” will be printed at a minimum of once, irrespective of what value j has.

Do While Comparison with While

- Both of the following programs with print “Hello” 4 times.
- The program on the **left** uses a **do while** loop.
- The program on the **right** uses a **while** loop.



```
int j = 0;
do {
    printf("Hello\n");
    j++;
} while (j<4);
```



```
int j = 0;
while (j<4) {
    printf("Hello\n");
    j++;
}
```

EXAMPLE PROBLEM

Example Problem

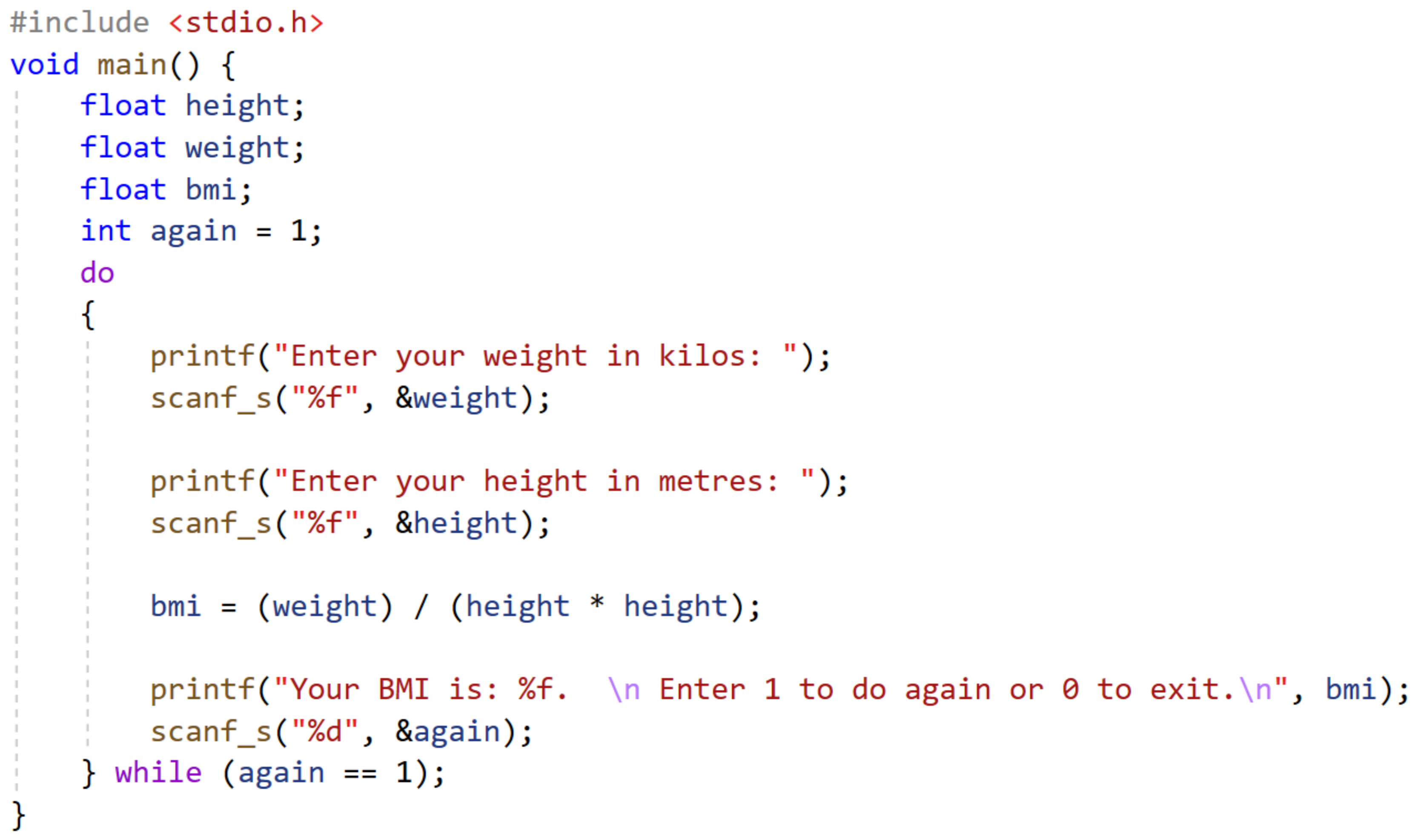
- Write a program that reads in the users weight in kg and height in meters.
- Calculate their BMI as: BMI = weight / height^2.
- Your program should then give the user an option to go again or to end the program.
- The user should be able to do as many BMI calculations as possible.

Example Problem

- Go to C program solution.

BMI C Program

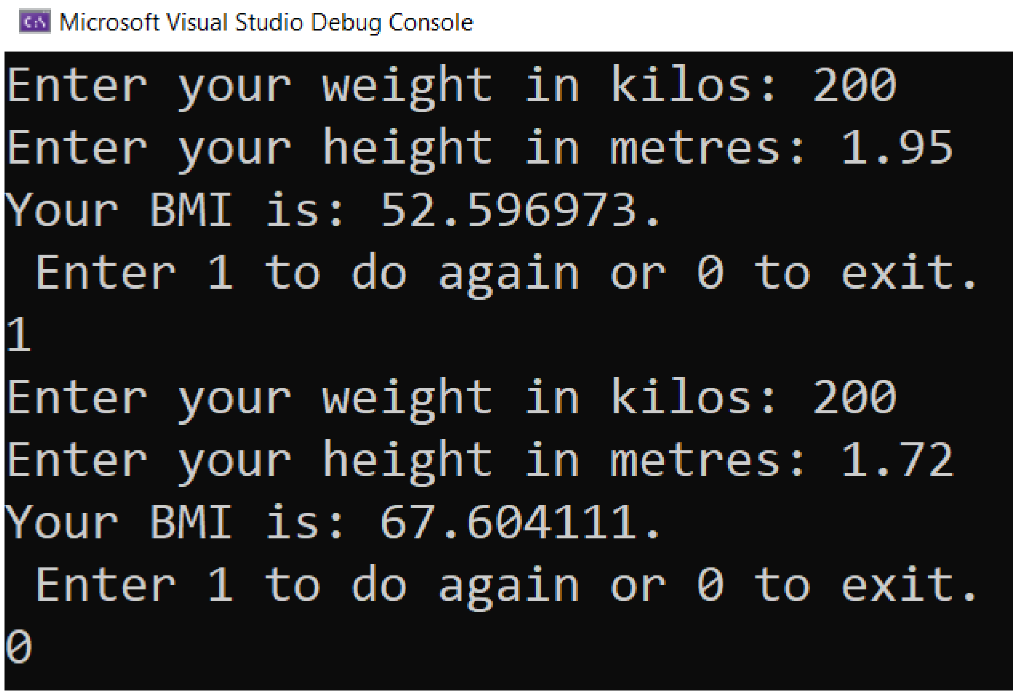
- The following will code will work:



```
#include <stdio.h>
void main() {
    float height;
    float weight;
    float bmi;
    int again = 1;
    do
    {
        printf("Enter your weight in kilos: ");
        scanf_s("%f", &weight);

        printf("Enter your height in metres: ");
        scanf_s("%f", &height);

        bmi = (weight) / (height * height);

        printf("Your BMI is: %f.  \n Enter 1 to do again or 0 to exit.\n", bmi);
        scanf_s("%d", &again);
    } while (again == 1);
}
```

BMI C Program Output

- The code will produce the following output:



```
Enter your weight in kilos: 200
Enter your height in metres: 1.95
Your BMI is: 52.596973.
 Enter 1 to do again or 0 to exit.
1
Enter your weight in kilos: 200
Enter your height in metres: 1.72
Your BMI is: 67.604111.
 Enter 1 to do again or 0 to exit.
0
```

PROGRAMMING

CT103

Week 4a

Sign in on Blackboard

- Please sign in on blackboard for CT103.
- I will leave the sign in option open for an hour after the lecture.

Lecture Content

- Last lecture (Week 3b):
  - Loops.
  - While loops.
  - Do while loops.

- Today’s lecture (Week 4a):
  - Recap on while loops.
  - For loops.
  - Example C program.

WHILE LOOPS RECAP

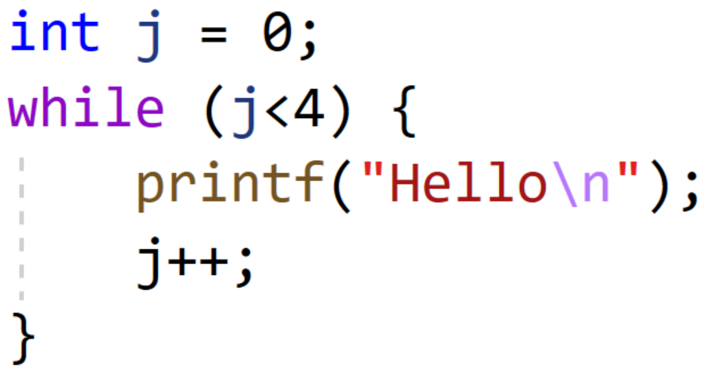
While Loops

- Last week we learned about **while** and **do while** loops.

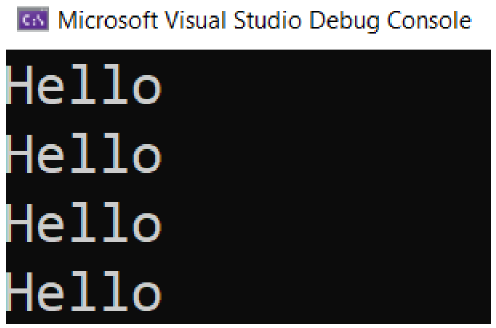
- The while loop will repeat a block of code over and over while some condition is true.

While Loops

- See the output of this code:



```c
int j = 0;
while (j<4) {
    printf("Hello\n");
    j++;
}
```

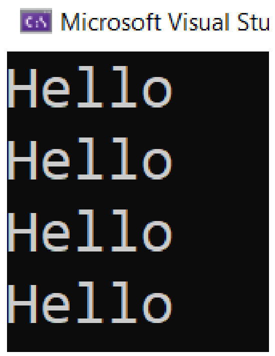


```
Hello
Hello
Hello
Hello
```

Do While Loops

- Print “Hello” 4 times using a do while loop.

```
int j = 0;
do {
    printf("Hello\n");
    j++;
} while (j<4);
```



Microsoft Visual Stu

```
Hello
Hello
Hello
Hello
```

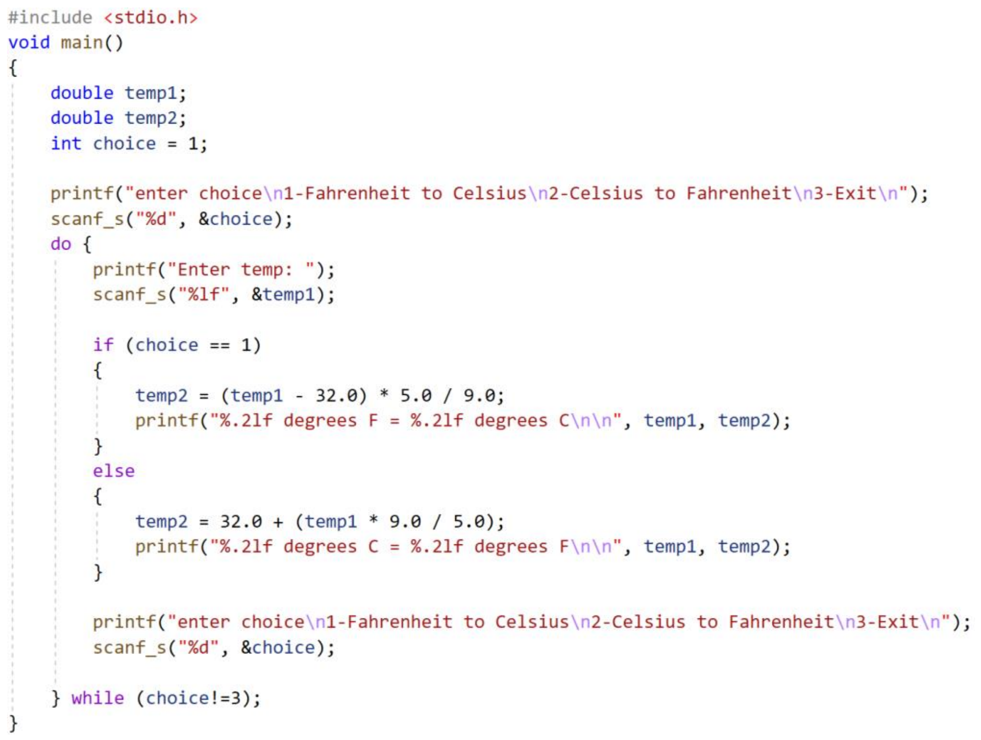
Another Do While Example

- Write a program that:
  - Asks the user if they want to convert a temperature from Celsius to Fahrenheit or the other way around.
  - The program should convert temperature from one unit to the other, e.g. Fahrenheit to Celsius.
  - The user should be able to do as many temperature conversions as they like.

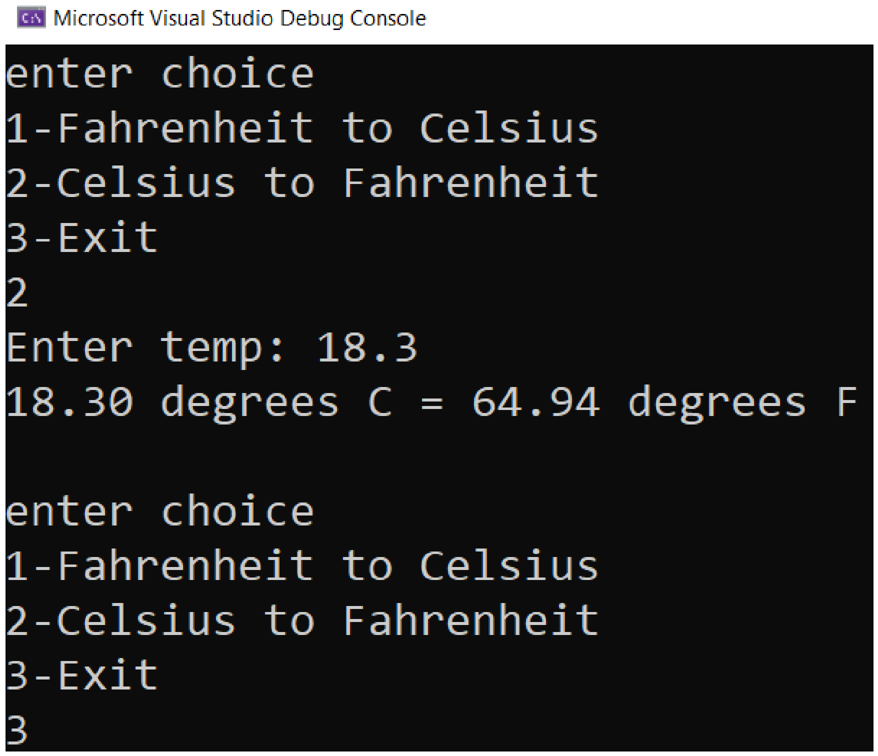
Another Do While Example

```
#include <stdio.h>
void main()
{
    double temp1;
    double temp2;
    int choice = 1;

    printf("enter choice\n1-Fahrenheit to Celsius\n2-Celsius to Fahrenheit\n3-Exit\n");
    scanf_s("%d", &choice);
    do {
        printf("Enter temp: ");
        scanf_s("%lf", &temp1);

        if (choice == 1)
        {
            temp2 = (temp1 - 32.0) * 5.0 / 9.0;
            printf("%.2lf degrees F = %.2lf degrees C\n\n", temp1, temp2);
        }
        else
        {
            temp2 = 32.0 + (temp1 * 9.0 / 5.0);
            printf("%.2lf degrees C = %.2lf degrees F\n\n", temp1, temp2);
        }

        printf("enter choice\n1-Fahrenheit to Celsius\n2-Celsius to Fahrenheit\n3-Exit\n");
        scanf_s("%d", &choice);

    } while (choice!=3);
}
```

Another Do While Example



```c
#include <stdio.h>
void main()
{
    double temp1;
    double temp2;
    int choice = 1;

    printf("enter choice\n1-Fahrenheit to Celsius\n2-Celsius to Fahrenheit\n3-Exit\n");
    scanf_s("%d", &choice);
    do {
        printf("Enter temp: ");
        scanf_s("%lf", &temp1);

        if (choice == 1)
        {
            temp2 = (temp1 - 32.0) * 5.0 / 9.0;
            printf("%.2lf degrees F = %.2lf degrees C\n\n", temp1, temp2);
        }
        else
        {
            temp2 = 32.0 + (temp1 * 9.0 / 5.0);
            printf("%.2lf degrees C = %.2lf degrees F\n\n", temp1, temp2);
        }

        printf("enter choice\n1-Fahrenheit to Celsius\n2-Celsius to Fahrenheit\n3-Exit\n");
        scanf_s("%d", &choice);

    } while (choice!=3);
}
```



Microsoft Visual Studio Debug Console

```
enter choice
1-Fahrenheit to Celsius
2-Celsius to Fahrenheit
3-Exit
2
Enter temp: 18.3
18.30 degrees C = 64.94 degrees F

enter choice
1-Fahrenheit to Celsius
2-Celsius to Fahrenheit
3-Exit
3
```

Points to Remember

- Loops allow us to repeat a piece of code.
- While loops allow us to keep repeating as long as the condition is true.
- Avoid infinite loops. Do this by changing “*something*” in the body of the loop.

Points to Remember

- Do while loops are similar to while loops except that they ensure what is in body of loop is executed at least once.

- Loops allow us to have shorter and more readable code.

FOR LOOPS

For Loops

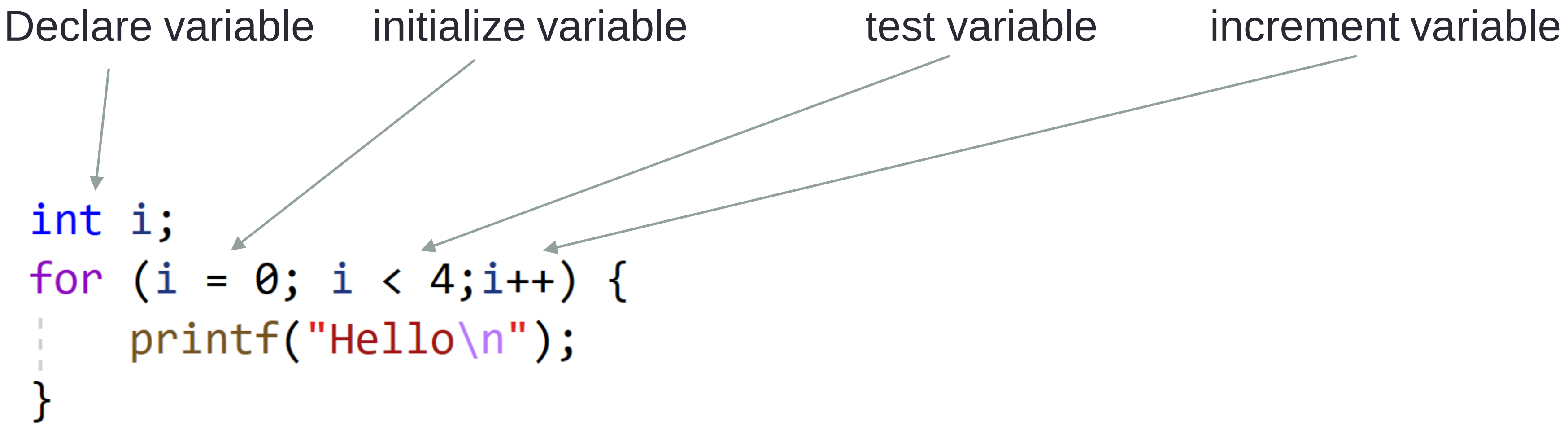
- **For loops** are useful if we want to repeat some code a predetermined number of times.

- We saw how we do this with **while loops**. We use:
  - A variable.
  - A condition.
  - An increment of the variable.

- **For loops** are a shorter way of doing this!

For Loop Template

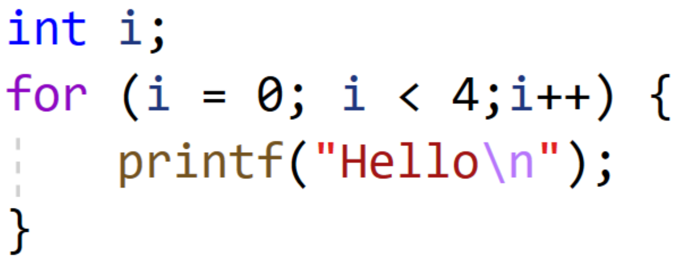
- So what does a **for loop** look like?
- A **for loop** will look something like the following:



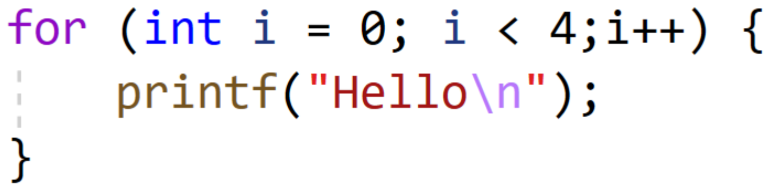
```c
int i;
for (i = 0; i < 4;i++) {
    printf("Hello\n");
}
```

For Loop Template

- A **for loop** can also look as follows:



```c
int i;
for (i = 0; i < 4;i++) {
    printf("Hello\n");
}
```



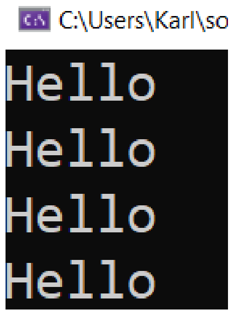
```c
for (int i = 0; i < 4;i++) {
    printf("Hello\n");
}
```

- We can declare and initialize the variable in the for loop.

For Loop Example

- When we run the program, it outputs “Hello” 4 times:

```c
int i;
for (i = 0; i < 4;i++) {
    printf("Hello\n");
}
```

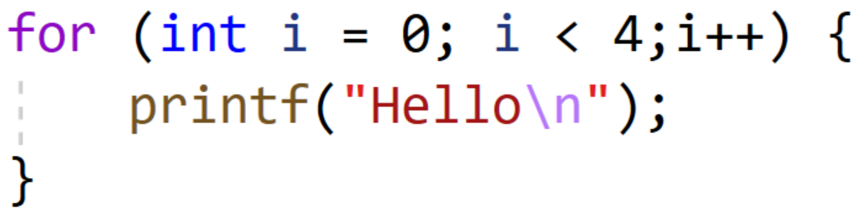


C:\Users\Karl\so

```
Hello
Hello
Hello
Hello
```

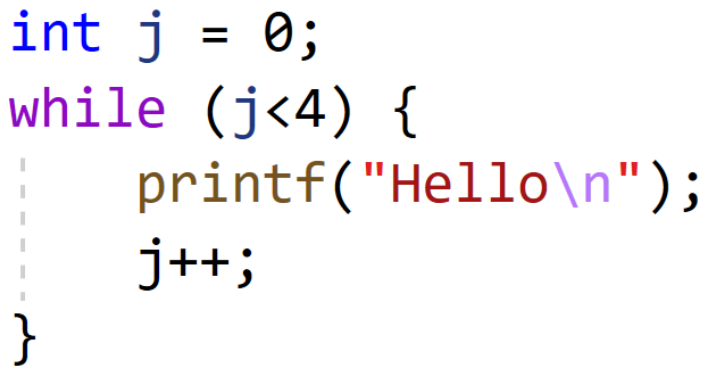
For Loop vs While Loop

- Lets compare the structure of for loops and while loops.



```
for (int i = 0; i < 4;i++) {
    printf("Hello\n");
}
```

For Loop

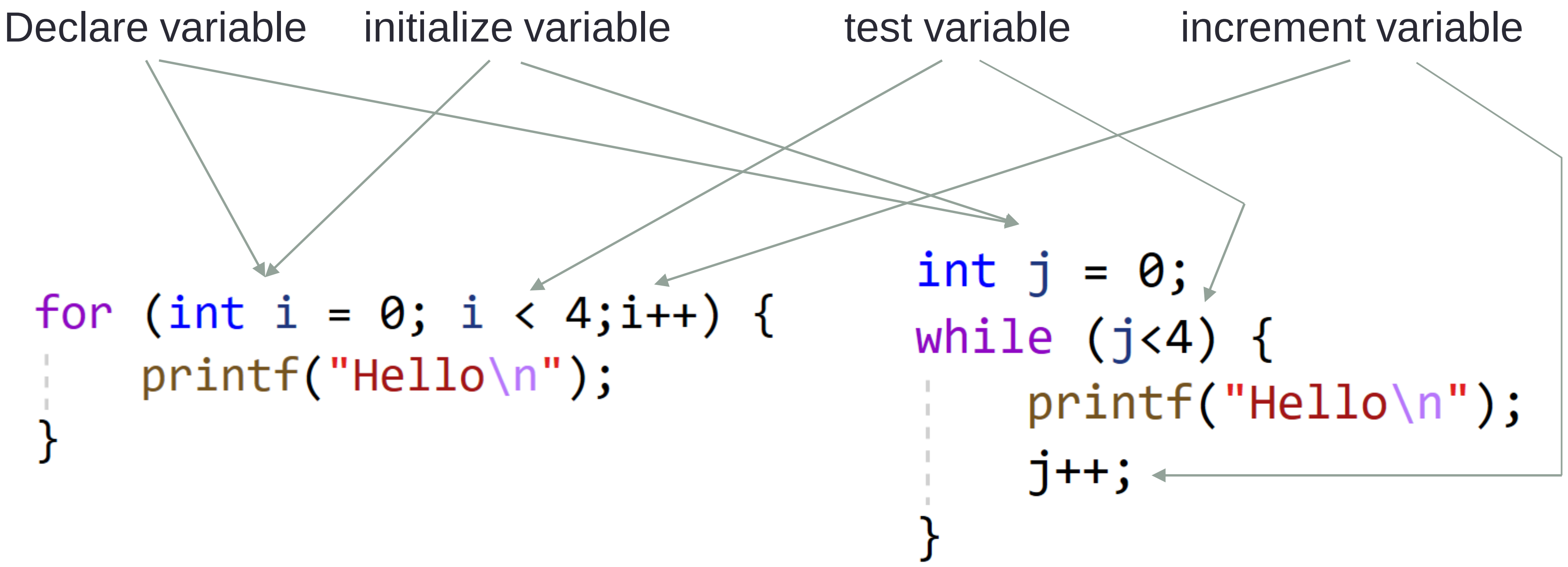


```
int j = 0;
while (j<4) {
    printf("Hello\n");
    j++;
}
```

While Loop

For Loop vs While Loop

- Lets compare the structure of for loops and while loops.



```
for (int i = 0; i < 4;i++) {
    printf("Hello\n");
}
```

```
int j = 0;
while (j<4) {
    printf("Hello\n");
    j++;
}
```

For Loop

While Loop

Which Loop Should I Use?

- Your choice of loop depends on what you want to do and how you want to end the loop.

- If you want to repeat a task “x” number of times, you can use either a **for loop** or a **while loop**.
  - E.g. If I want to do a calculation 5 times, use a for loop.
  - When I want to end the loop is determined by the number of calculations.

- If you do not know how many cycles the loop will run for, use a **while loop**.
  - E.g. If my program does BMI calculations, I don’t know how many calculations the user will want to do.

EXAMPLE PROBLEMS

ATM Problem

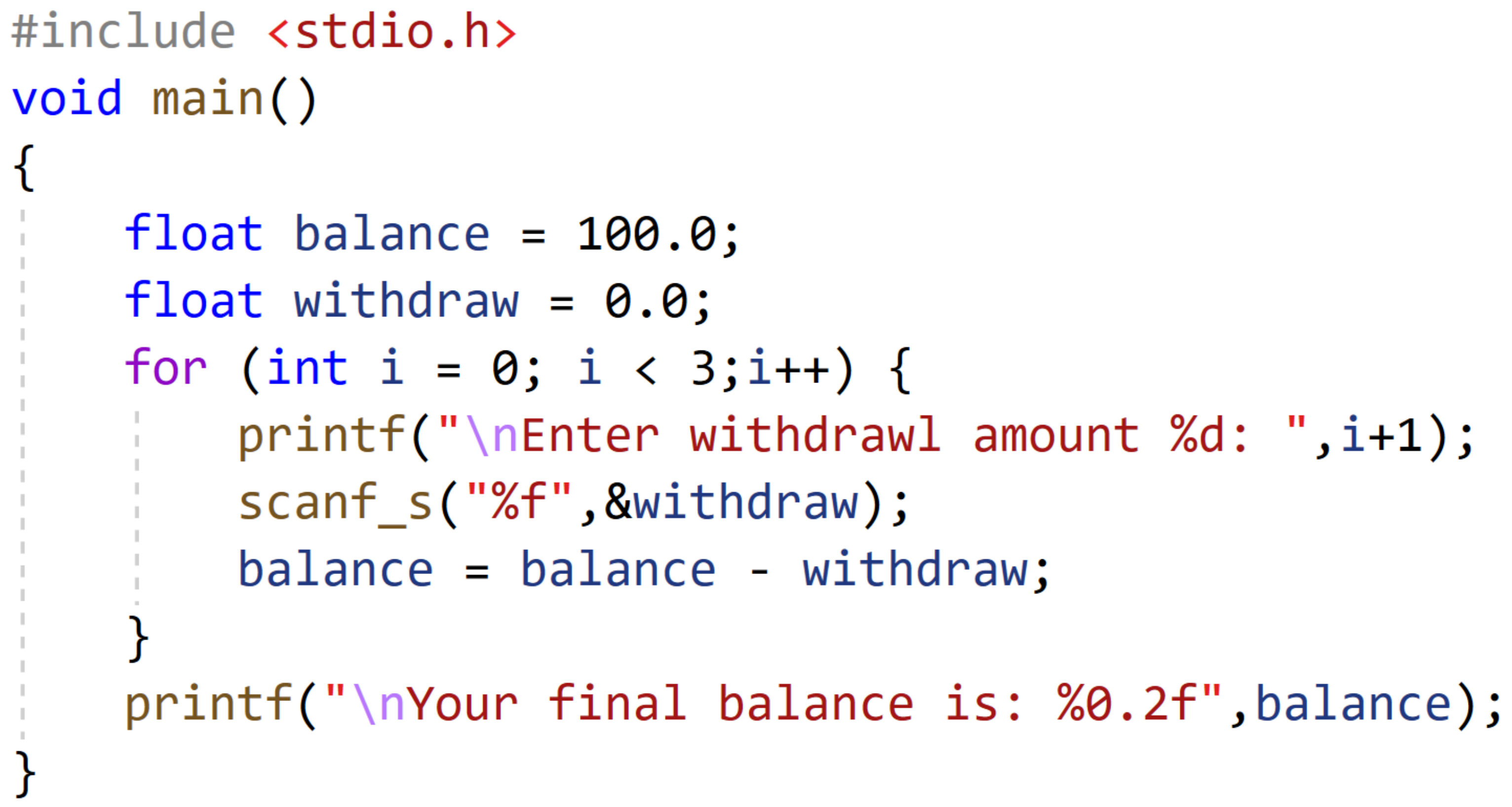
- You are working for a bank.
- You must write a program that:
  - Create a new bank account with a balance of €100.
  - Use a for loop to make 3 ATM withdrawals.
  - Update the bank account balance for each withdrawal.

ATM Problem

- Go to C program solution.

ATM Problem

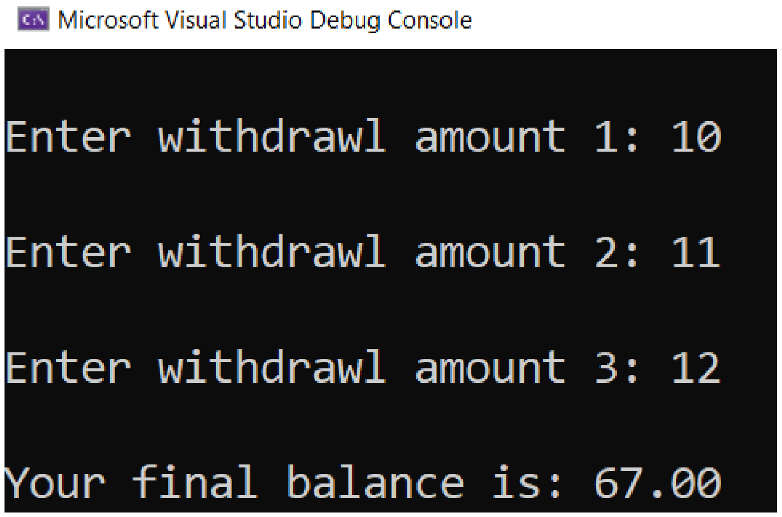
- The following code will work:



```c
#include <stdio.h>
void main()
{
    float balance = 100.0;
    float withdraw = 0.0;
    for (int i = 0; i < 3;i++) {
        printf("\nEnter withdrawl amount %d: ",i+1);
        scanf_s("%f",&withdraw);
        balance = balance - withdraw;
    }
    printf("\nYour final balance is: %0.2f",balance);
}
```

ATM C Program Output

- The code will produce the following output:


Microsoft Visual Studio Debug Console

```
Enter withdrawl amount 1: 10
Enter withdrawl amount 2: 11
Enter withdrawl amount 3: 12
Your final balance is: 67.00
```

Airlines Problem

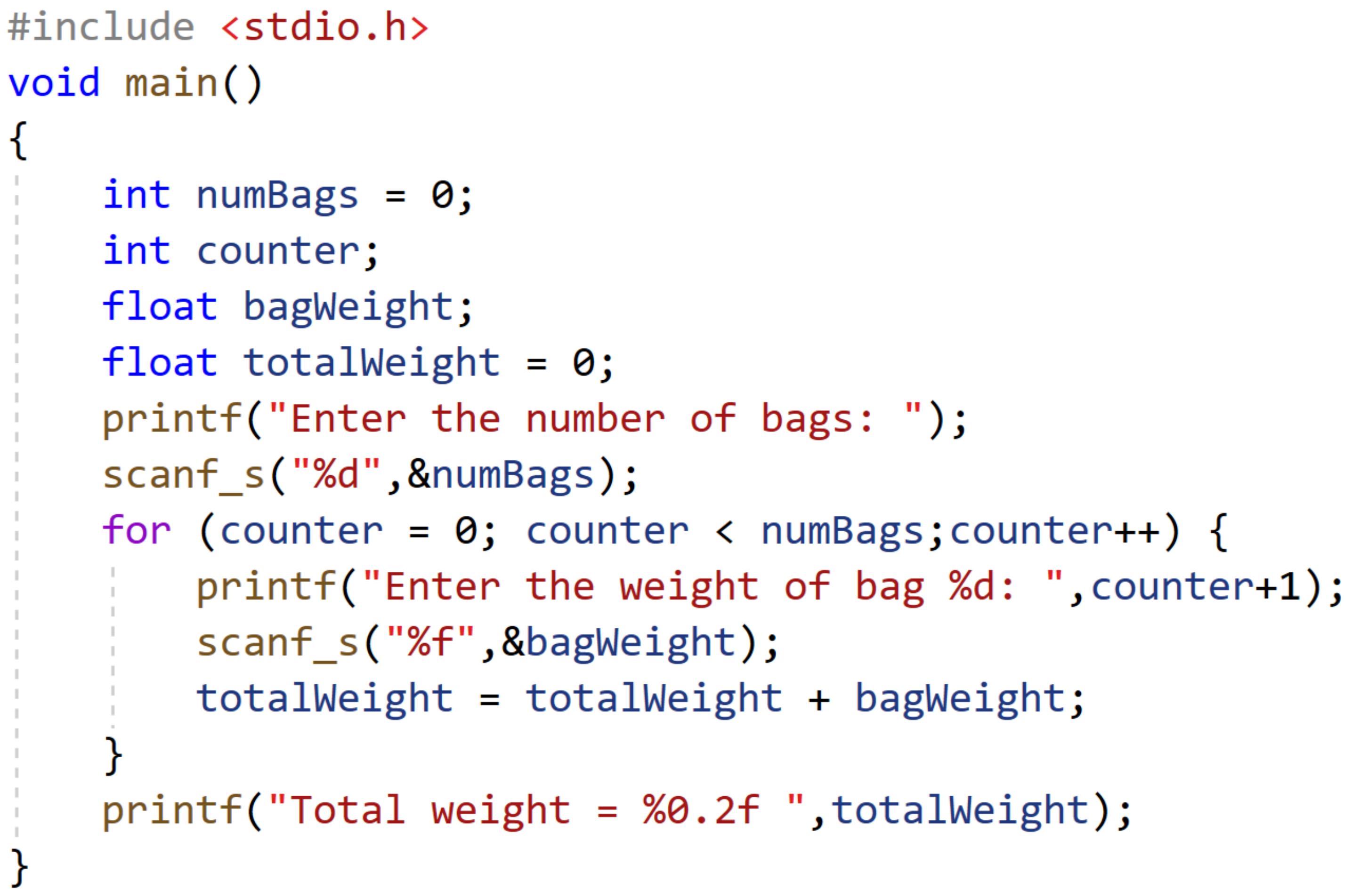
- You are working for a major airline “Brianair”.
- You must write a program that:
  - Reads in the number of bags to be checked in as input from the user.
  - Use a for loop to read in the weight of each individual bag.
  - Sum up the total weight of the bags and print it to the screen.
  - Rewrite the same program as outlined above, now using a while loop.

Airlines Problem

- Go to C program solution.

Airlines Problem

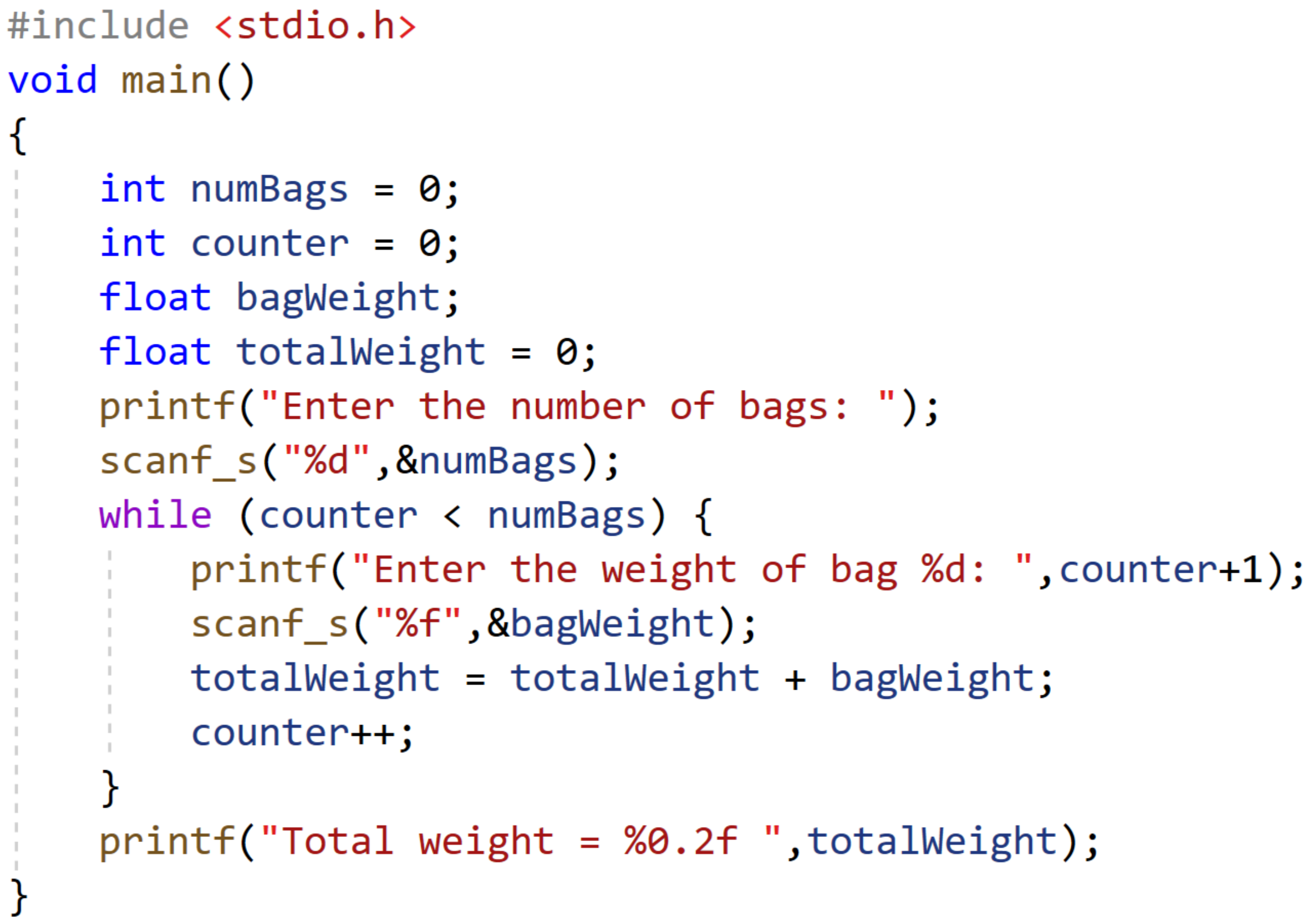
- The following code will work:



```c
#include <stdio.h>
void main()
{
    int numBags = 0;
    int counter;
    float bagWeight;
    float totalWeight = 0;
    printf("Enter the number of bags: ");
    scanf_s("%d",&numBags);
    for (counter = 0; counter < numBags;counter++) {
        printf("Enter the weight of bag %d: ",counter+1);
        scanf_s("%f",&bagWeight);
        totalWeight = totalWeight + bagWeight;
    }
    printf("Total weight = %0.2f ",totalWeight);
}
```

Airlines Problem

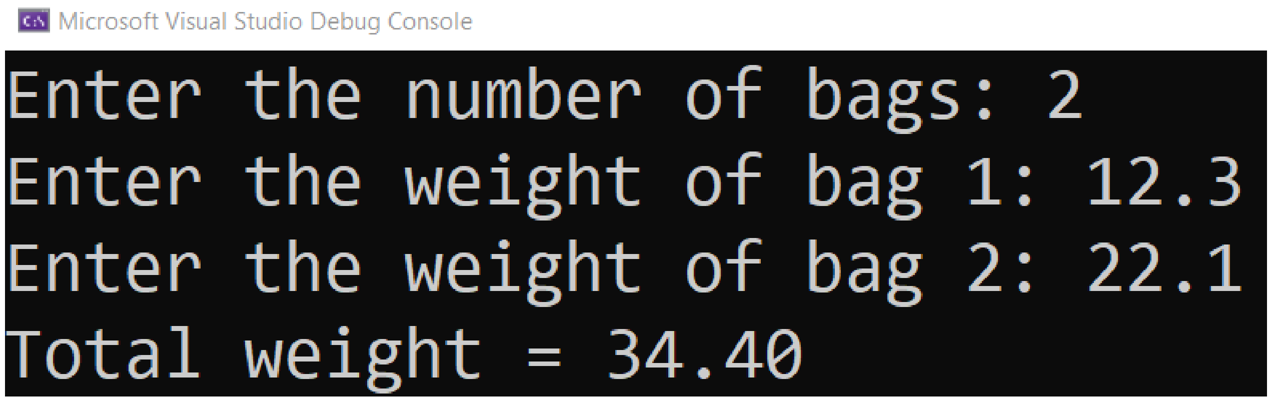
- The following code will also work, now with a while loop:



```c
#include <stdio.h>
void main()
{
    int numBags = 0;
    int counter = 0;
    float bagWeight;
    float totalWeight = 0;
    printf("Enter the number of bags: ");
    scanf_s("%d",&numBags);
    while (counter < numBags) {
        printf("Enter the weight of bag %d: ",counter+1);
        scanf_s("%f",&bagWeight);
        totalWeight = totalWeight + bagWeight;
        counter++;
    }
    printf("Total weight = %0.2f ",totalWeight);
}
```

Airline C Program Output

- The code will produce the following output:



```
Enter the number of bags: 2
Enter the weight of bag 1: 12.3
Enter the weight of bag 2: 22.1
Total weight = 34.40
```

PROGRAMMING

CT103

Week 4b

Sign in on Blackboard

- Please sign in on blackboard for CT103.
- I will leave the sign in option open for an hour after the lecture.

Lecture Content

- Last lecture (Week 4a):
  - Recap on while loops.
  - For loops.
  - Example C program.

- Today's lecture (Week 4b):
  - Arrays
  - Arrays and loops
  - Arrays example C program

ARRAYS

Array Definition

- What is an **array**?

- **Definition**: An array is a data structure consisting of a collection of elements. Each element can be identified by an index.

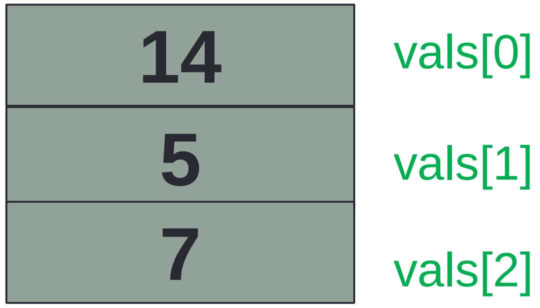
Array Definition

- What are **arrays** used for?
- An array is used to store a collection of data.
- You can think of an array as a collection of variables of the same type.

Arrays in C

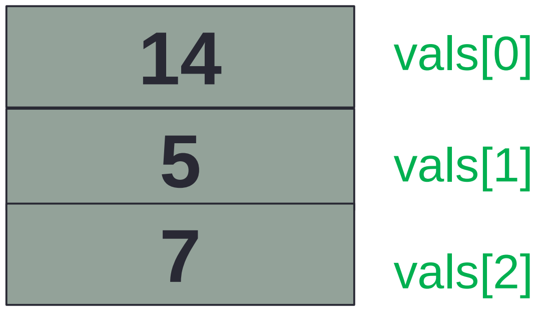
- You can define arrays of any type as follows:
  - E.g. int vals [3];
  - E.g. char initials[3];

- You can initialise like this if you want to:
  - int vals[3] = {14,5,7};



Arrays in C

- If we have the following array called **vals**.
- The size of vals is 3.
- The 1st element is at position 0 of the array.
- The 2nd element is at position 1 of the array.
- The 3rd element is at position 2 of the array.



Array Terminology

- Array – a collection of data.
- Element – one of the “*items*” in the array.
- Index – the position of the element in the array.
- Array size – how many elements in the array.

Initializing an Array

- float prices[3] = {65.56, 45.63, 7.90};
- double salary[2] = {45000.00, 33500.00};
- int grades[5] = {44, 55, 66, 33, 88};

Initializing an Array

- int ages[5] = {6,8,9,11,14}; /* Correct */
- int ages[]; /* Incorrect */
- int ages[] = {6,8,9,11,14}; /* Correct */

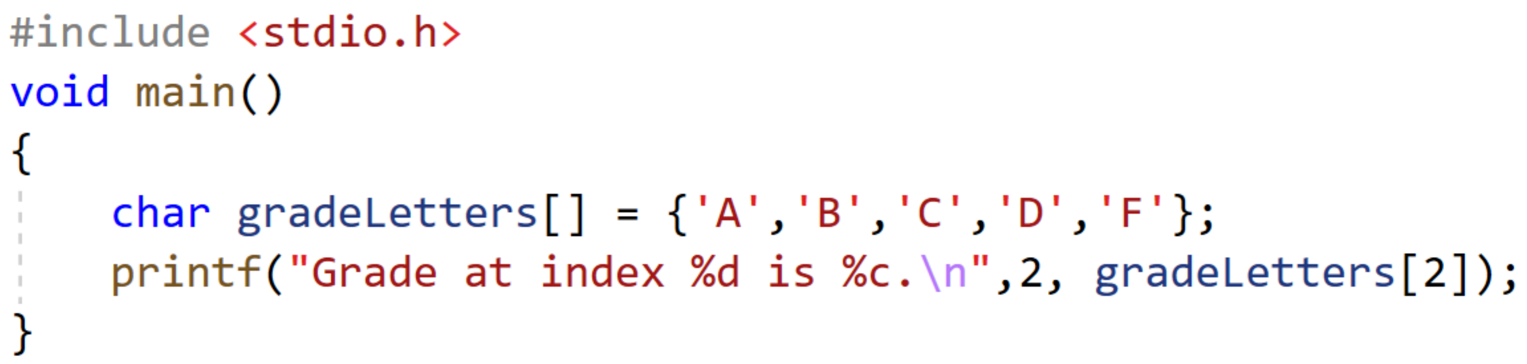
- Remember, you must initialise you array properly.

Simple Array Problem Example

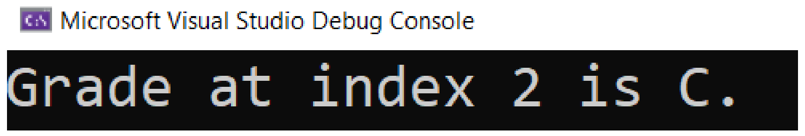
- Write a program that does the following:
- Store all of the possible letter grades that a student can get in an array.
- Print the grade at index 2 to the screen.

Simple Array Problem Example

- The following program creates at array for grade letters.



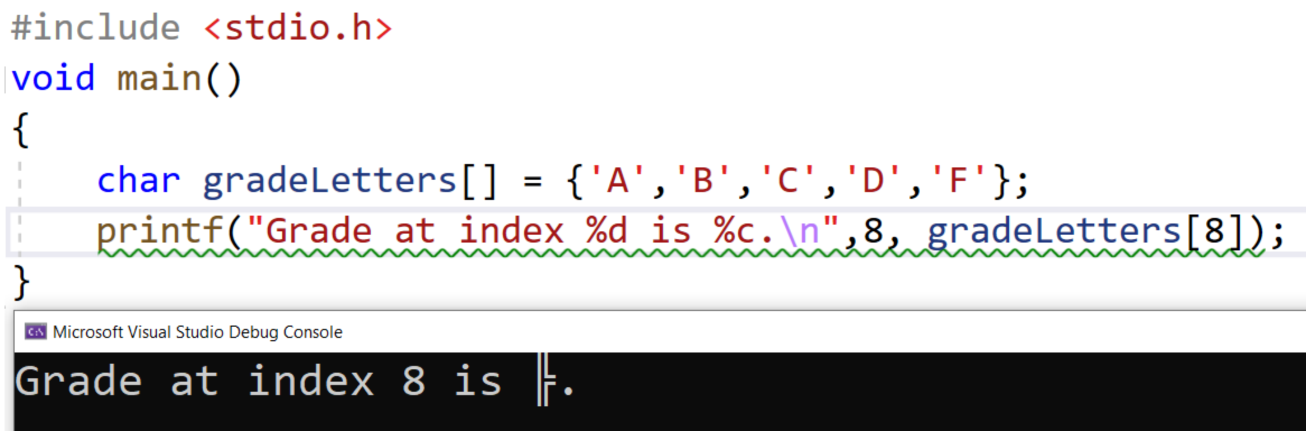
```c
#include <stdio.h>
void main()
{
    char gradeLetters[] = {'A','B','C','D','F'};
    printf("Grade at index %d is %c.\n",2, gradeLetters[2]);
}
```



```
Grade at index 2 is C.
```

Common Array Mistakes

- Accessing an index that is equal to or larger than the size of the array.
- Don’t do this.



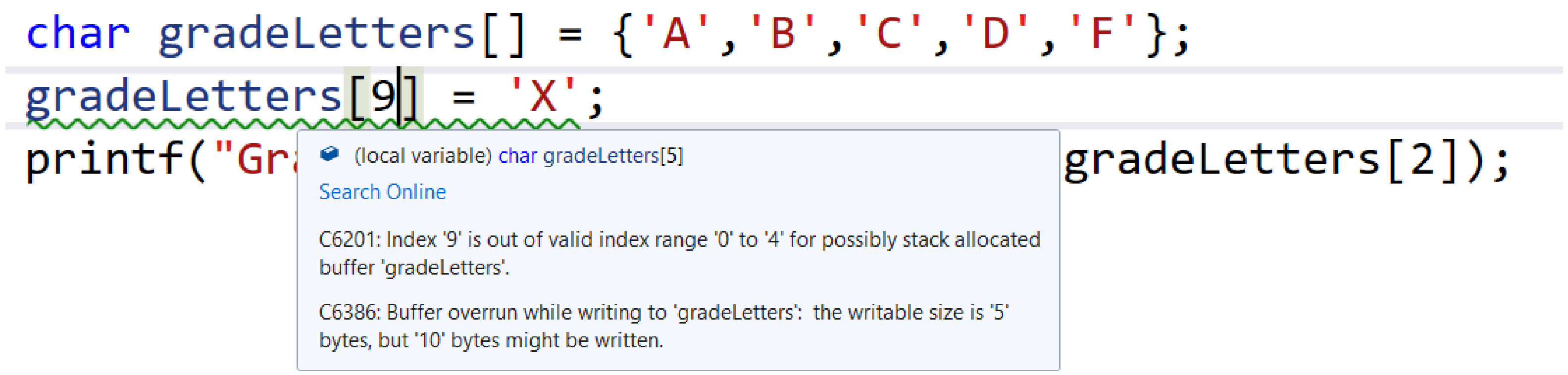
```
#include <stdio.h>
void main()
{
    char gradeLetters[] = {'A','B','C','D','F'};
    printf("Grade at index %d is %c.\n",8, gradeLetters[8]);
}
```

Microsoft Visual Studio Debug Console

```
Grade at index 8 is ╟.
```

Common Array Mistakes

- Setting the value of an array element who's index is equal to or larger than the size of the array.
- Don’t do this either.



```
char gradeLetters[] = {'A','B','C','D','F'};
gradeLetters[9] = 'X';
printf("Gr                                          gradeLetters[2]);
```

(local variable) char gradeLetters[5]

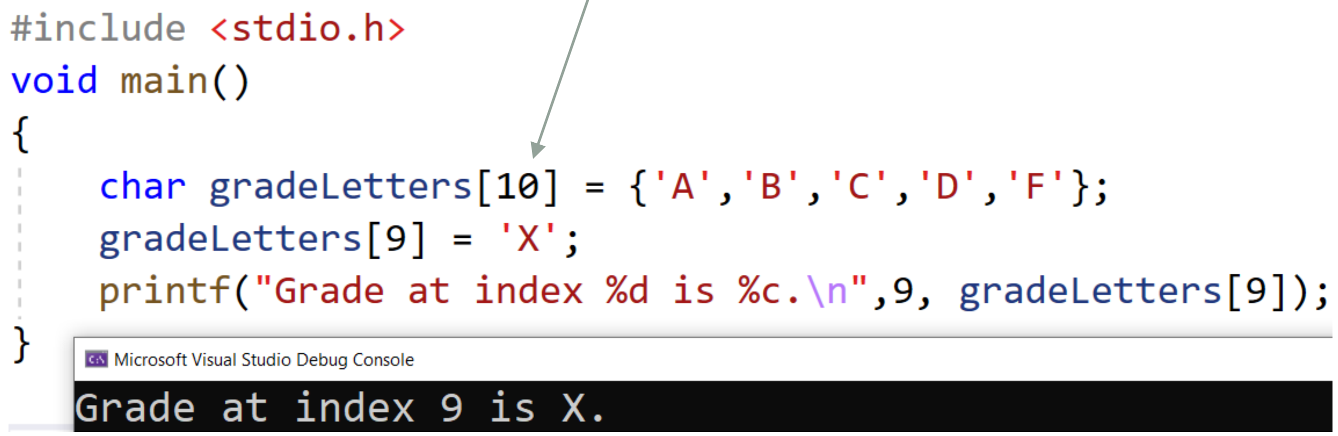
Search Online

C6201: Index '9' is out of valid index range '0' to '4' for possibly stack allocated buffer 'gradeLetters'.

C6386: Buffer overrun while writing to 'gradeLetters': the writable size is '5' bytes, but '10' bytes might be written.

Common Array Mistakes

- I would need to create a larger array.



```c
#include <stdio.h>
void main()
{
    char gradeLetters[10] = {'A','B','C','D','F'};
    gradeLetters[9] = 'X';
    printf("Grade at index %d is %c.\n",9, gradeLetters[9]);
}
```

Microsoft Visual Studio Debug Console

```
Grade at index 9 is X.
```

ARRAYS AND LOOPS

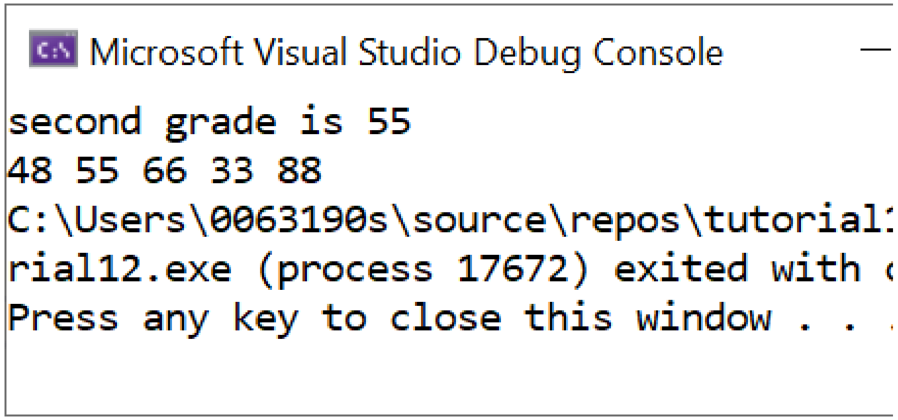
Remember

- The index of the array members always starts with 0, for example:
  - grades[0];

- For an array of length/size n (called myArray):
  - The indices range from 0 to n-1.
  - The elements range from myArray[0] to myArray[n-1].

Accessing array members

```
                 [0]  [1]  [2]  [3]  [4]
int grades[5] = { 44, 55, 66, 33, 88 };

grades[0] = 48; // easy to access/change any member of an array

printf("second grade is %d\n", grades[1]);

for (int i = 0;i < 5;i++)
{
  printf("%d ",grades[i]);
}
```



```
second grade is 55
48 55 66 33 88
C:\Users\0063190s\source\repos\tutorial1
rial12.exe (process 17672) exited with
Press any key to close this window . . .
```

EXAMPLE PROBLEMS

Exercise Tracker App

- You are designing an fitness app. The app allows the user to track their 5km running times.
- You must write a program that:
  - Reads in the number of 5km running times as input from the user.
  - Use loop to read in each 5km running time from the user.
  - Store these times in an array.
  - Print the running times out to the user so they can view them.

Exercise Tracker App

- Go to C program solution.

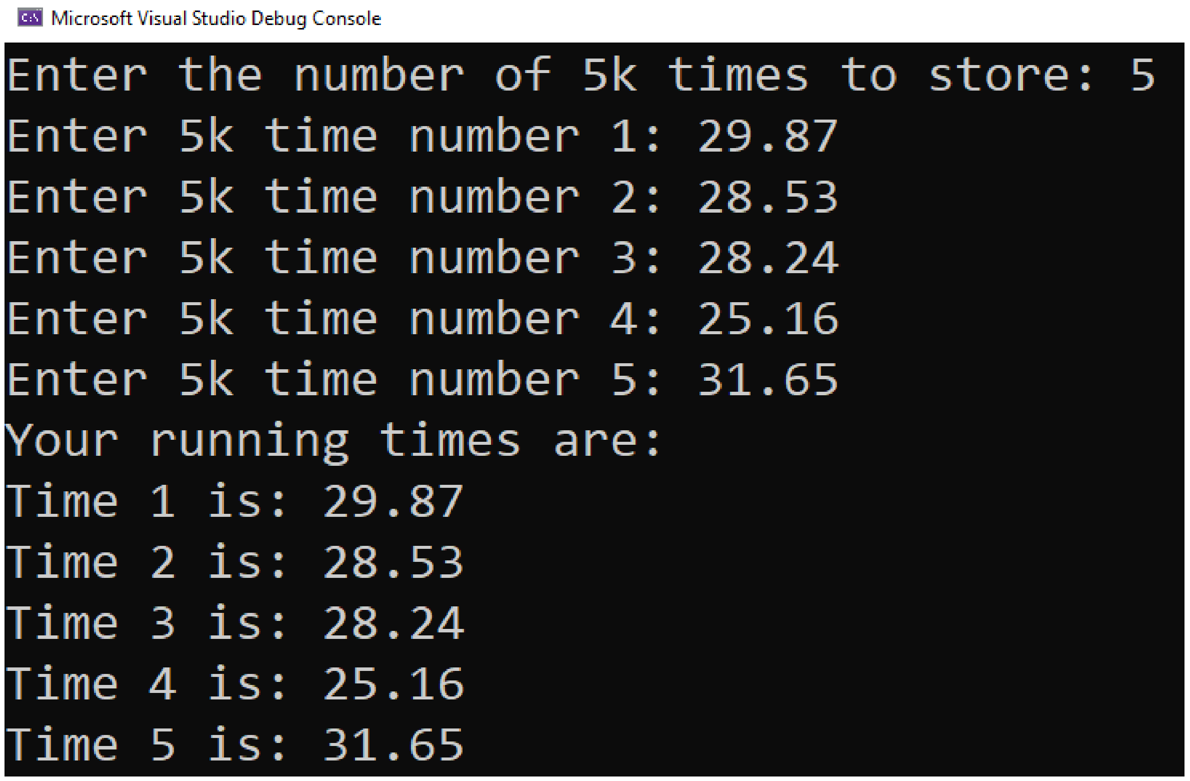
Exercise Tracker App

- The following code will work:

```c
#include <stdio.h>
void main()
{
    float runTimes[1000];
    int num;

    printf("Enter the number of 5k times to store: ");
    scanf_s("%d",&num);

    for (int i = 0; i < num;i++) {
        printf("Enter 5k time number %d: ",i+1);
        scanf_s("%f",&runTimes[i]);
    }

    printf("Your running times are: \n");
    for (int i = 0; i < num; i++) {
        printf("Time %d is: %0.2f\n", i+1, runTimes[i]);
    }
}
```

Exercise Tracker App

- The code will produce the following output:



Microsoft Visual Studio Debug Console

```
Enter the number of 5k times to store: 5
Enter 5k time number 1: 29.87
Enter 5k time number 2: 28.53
Enter 5k time number 3: 28.24
Enter 5k time number 4: 25.16
Enter 5k time number 5: 31.65
Your running times are:
Time 1 is: 29.87
Time 2 is: 28.53
Time 3 is: 28.24
Time 4 is: 25.16
Time 5 is: 31.65
```

Cinema Problem

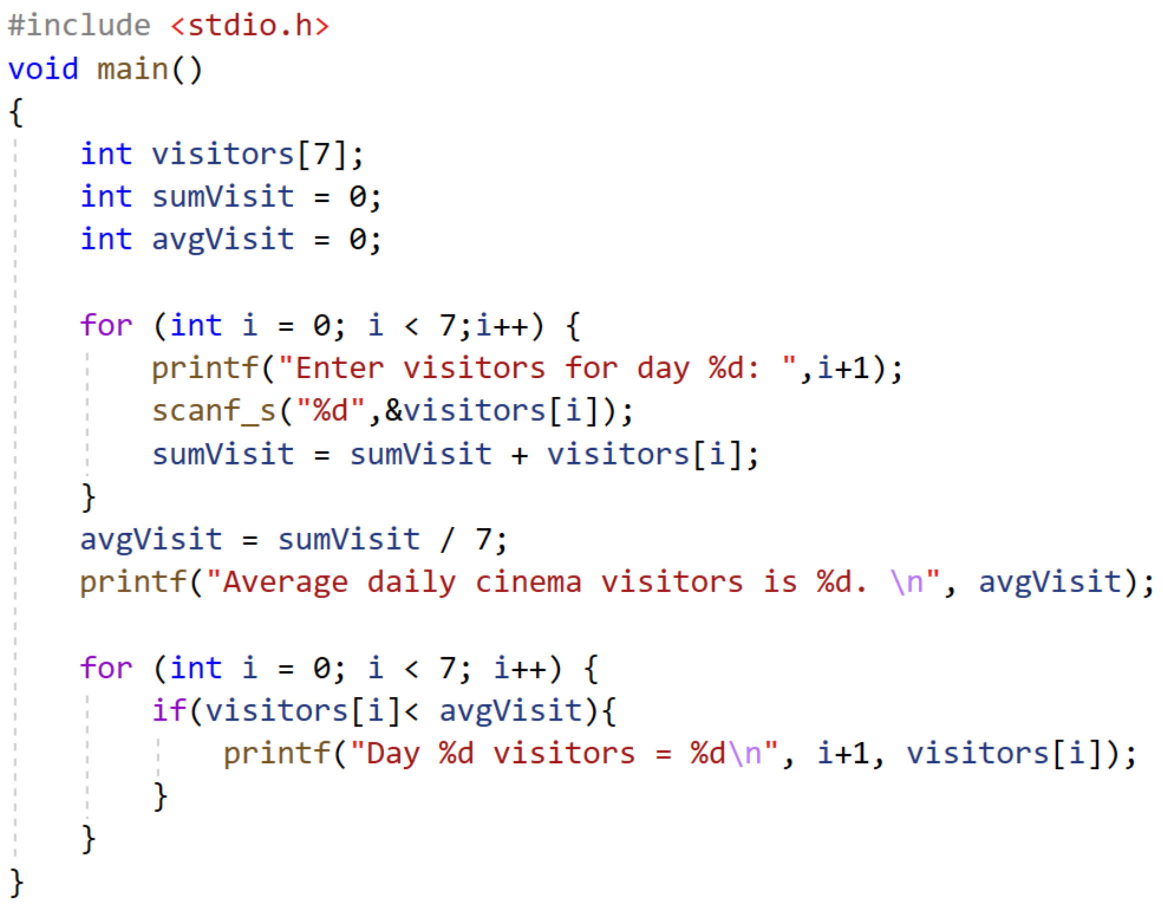
- You are designing software for a cinema. The cinema wants to record the daily visitors to the cinema over 7 days.
- You must write a program that:
  - Reads the number daily cinema goers as input from the user for 7 days.
  - Stores the cinema visitor numbers in an array.
  - Calculate and print the average number of daily cinema goers.
  - Prints out the daily visitor numbers that are below average.

Cinema Problem

- Go to C program solution.

Cinema Problem

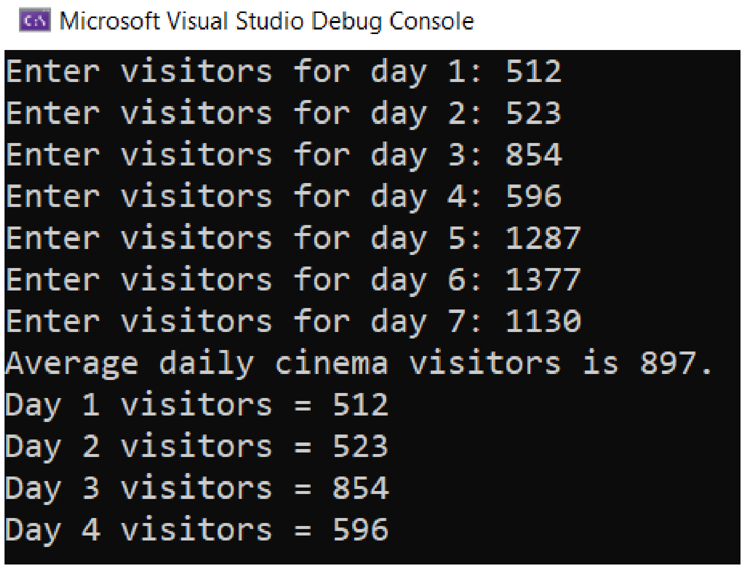
- The following code will work:



```
#include <stdio.h>
void main()
{
    int visitors[7];
    int sumVisit = 0;
    int avgVisit = 0;

    for (int i = 0; i < 7;i++) {
        printf("Enter visitors for day %d: ",i+1);
        scanf_s("%d",&visitors[i]);
        sumVisit = sumVisit + visitors[i];
    }
    avgVisit = sumVisit / 7;
    printf("Average daily cinema visitors is %d. \n", avgVisit);

    for (int i = 0; i < 7; i++) {
        if(visitors[i]< avgVisit){
            printf("Day %d visitors = %d\n", i+1, visitors[i]);
        }
    }
}
```

Cinema Problem

- The code will produce the following output:
- Note: Be careful of rounding. We use integers when calculating the average here.



```
Enter visitors for day 1: 512
Enter visitors for day 2: 523
Enter visitors for day 3: 854
Enter visitors for day 4: 596
Enter visitors for day 5: 1287
Enter visitors for day 6: 1377
Enter visitors for day 7: 1130
Average daily cinema visitors is 897.
Day 1 visitors = 512
Day 2 visitors = 523
Day 3 visitors = 854
Day 4 visitors = 596
```

PROGRAMMING

CT103

Week 5b

Sign in on Blackboard

- Please sign in on blackboard for CT103.
- I will leave the sign in option open for an hour after the lecture.

Lecture Content

- Last lecture (Week 4b):
  - Arrays
  - Arrays and loops
  - Arrays example C program

- Today’s lecture (Week 5b):
  - Arrays recap
  - Arrays in memory
  - 2D Arrays
  - Example 2D arrays C program

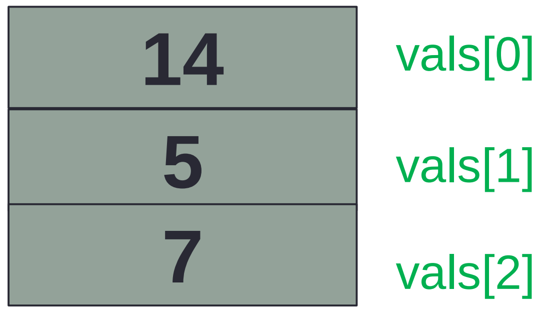
ARRAYS RECAP

Definitions

- **Definition**: An **array** is a data structure consisting of a collection of elements. Each element can be identified by an index.
- **Element** – one of the "*items*" in the array.
- **Index** – the position of the element in the array.

Arrays in C

- You can initialise like this:
  - int vals[3] = {14,5,7};



Declaring an Array

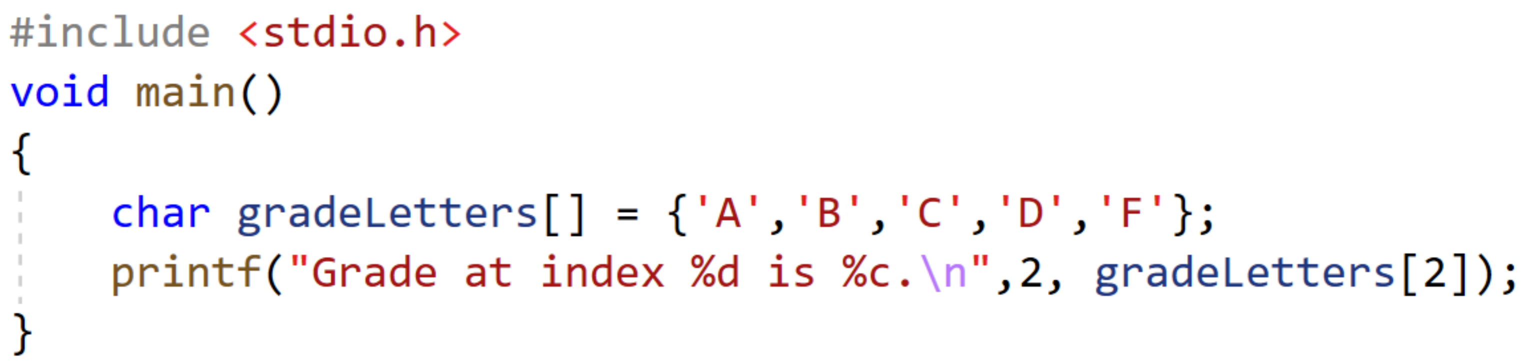
- Very straightforward – you just need to specify variable (array) type, name and size, e.g.:
  - `int grades[5];`

- To initialise, you can do like so:
  - `int grades[5] = { 44, 55, 66, 33, 88 };`

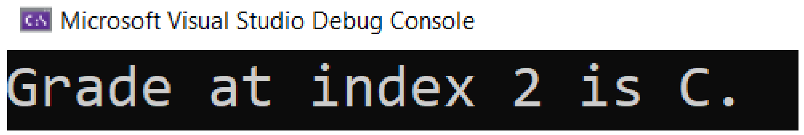
- You can implicitly dictate the size of the array:
  - `int grades[] = { 44, 55, 66, 33, 88 }; // size = 5`

Simple Array Problem Example

- The following program creates at array for grade letters.



```c
#include <stdio.h>
void main()
{
    char gradeLetters[] = {'A','B','C','D','F'};
    printf("Grade at index %d is %c.\n",2, gradeLetters[2]);
}
```

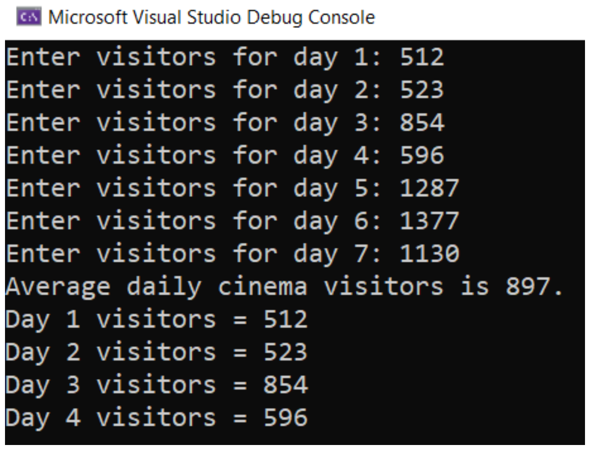


```
Grade at index 2 is C.
```

Cinema Problem

- This code from last week will read in daily cinema visitors, calculate the average and return the days < average.

```c
#include <stdio.h>
void main()
{
    int visitors[7];
    int sumVisit = 0;
    int avgVisit = 0;

    for (int i = 0; i < 7;i++) {
        printf("Enter visitors for day %d: ",i+1);
        scanf_s("%d",&visitors[i]);
        sumVisit = sumVisit + visitors[i];
    }
    avgVisit = sumVisit / 7;
    printf("Average daily cinema visitors is %d. \n", avgVisit);

    for (int i = 0; i < 7; i++) {
        if(visitors[i]< avgVisit){
            printf("Day %d visitors = %d\n", i+1, visitors[i]);
        }
    }
}
```



```
Enter visitors for day 1: 512
Enter visitors for day 2: 523
Enter visitors for day 3: 854
Enter visitors for day 4: 596
Enter visitors for day 5: 1287
Enter visitors for day 6: 1377
Enter visitors for day 7: 1130
Average daily cinema visitors is 897.
Day 1 visitors = 512
Day 2 visitors = 523
Day 3 visitors = 854
Day 4 visitors = 596
```

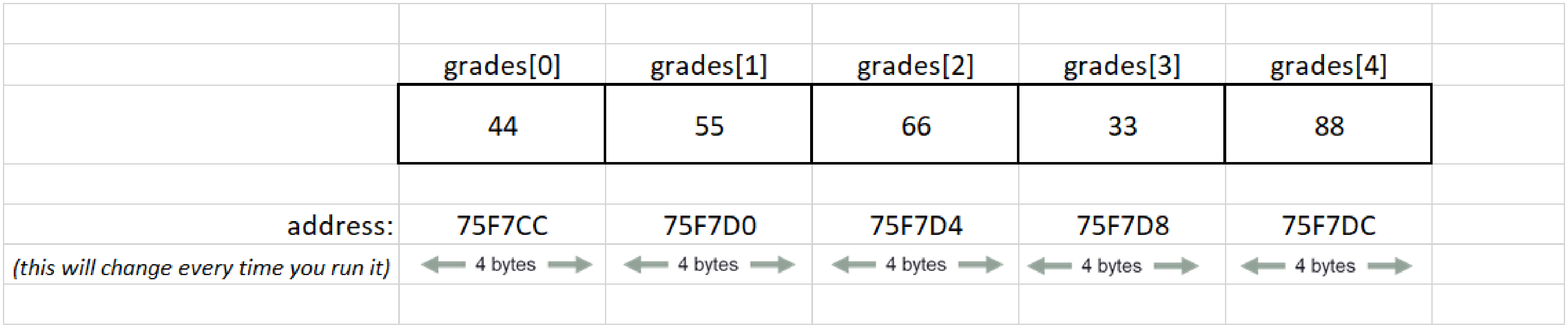
ARRAYS IN MEMORY

Where/how are arrays stored?

- An array is normally stored in sequential blocks of memory, i.e. RAM.
- Block size depends on the number of bytes required to store that type of variable.
- For example, an integer usually requires 4 bytes.

Where/how are arrays stored?

- An array is normally stored in sequential blocks of memory.

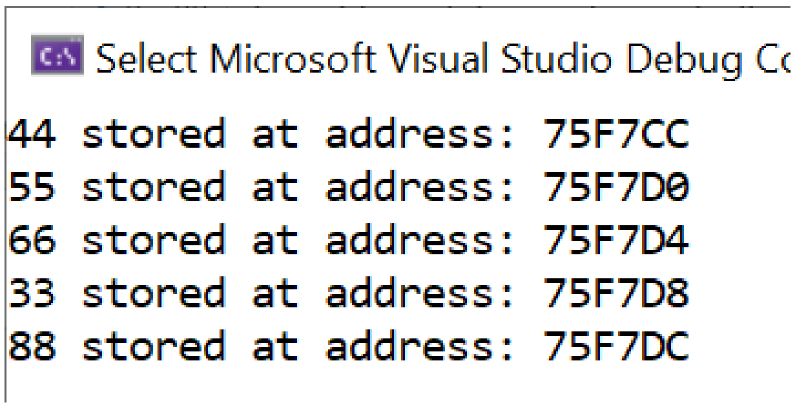


	grades[0]	grades[1]	grades[2]	grades[3]	grades[4]
	44	55	66	33	88
address:	75F7CC	75F7D0	75F7D4	75F7D8	75F7DC
(this will change every time you run it)	← 4 bytes →	← 4 bytes →	← 4 bytes →	← 4 bytes →	← 4 bytes →

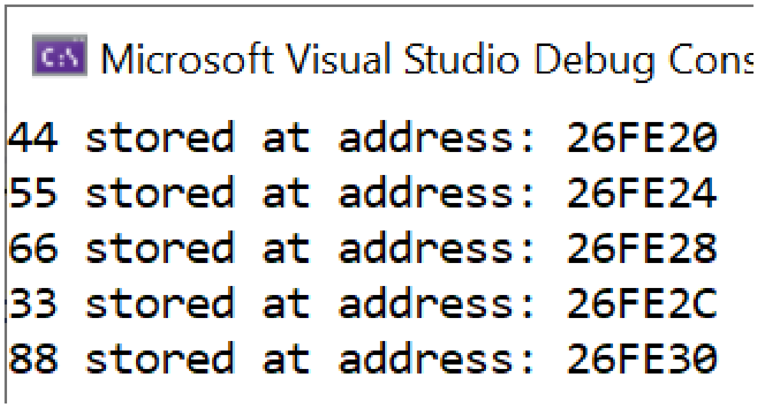
- Functions like *scanf()* need the address of a variable so that it can store new values there
- This is why you put & in front of the variable name, which gives *scanf()* the variable address rather than the variable's current value

Try this out

```
void main()
{
  int grades[5] = { 44, 55, 66, 33, 88 };

  for (int i = 0; i < 5; i++){
    printf("%d stored at address: %X \n", grades[i], &grades[i]);
  }

  printf("\n\n");
}
```



```
44 stored at address: 75F7CC
55 stored at address: 75F7D0
66 stored at address: 75F7D4
33 stored at address: 75F7D8
88 stored at address: 75F7DC
```



```
44 stored at address: 26FE20
55 stored at address: 26FE24
66 stored at address: 26FE28
33 stored at address: 26FE2C
88 stored at address: 26FE30
```

Copy an array into another

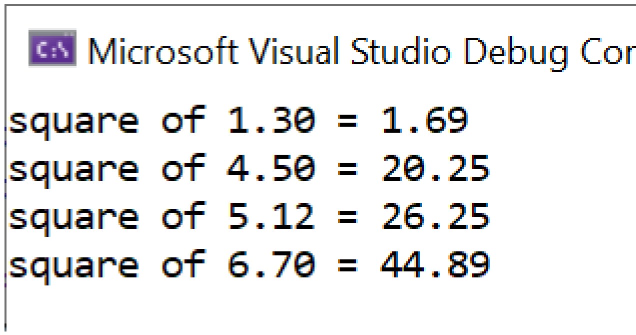
- Easy to do – just use the same index for the source array and the target array. Try this out:

```c
#include <stdio.h>

void main()
{
  int grades[5] = { 44, 55, 66, 33, 88 };
  int marks[5];

  for (int i = 0; i < 5; i++){
    marks[i] = grades[i];
  }

}
```

Create an array based on another array

- Easy to run through an array with a for loop and also set the values of another array of the same size

```
#include <stdio.h>

void main()
{
  double nums[4] = { 1.3, 4.5, 5.123, 6.7002 };
  double squares[4];

  for (int i = 0; i < 4; i++){
    squares[i] = nums[i] * nums[i];
    printf("square of %.2lf = %.2lf \n", nums[i], squares[i]);
  }
}
```



```
square of 1.30 = 1.69
square of 4.50 = 20.25
square of 5.12 = 26.25
square of 6.70 = 44.89
```

2D ARRAYS

2 Dimensional Arrays

- Up until now, we have only considered a 1 dimensional (1D) array.
  - E.g. `int vals[3] = {14,5,7};`

- What if we have 2 dimensional (2D) data that we need to use in our program?

- We use 2D arrays!

2 Dimensional Arrays

- What do 2D arrays look like?
- The following will create a 2-dimensional array of integers:
- int var[2][2];

var[0][0]	var[0][1]
var[1][0]	var[1][1]

- The **first index** is the **row** number, the **second index** is the **column** number.

Initialise 2D array

- Each row is an individual 1D array

- int var[2][2] = {{11,12},{21,22}};

11	12
21	22

Change element

- How do I change an element in a 2D array?
- var[1][0] = 55;

11	12
55	22

Loop over elements in 2D array

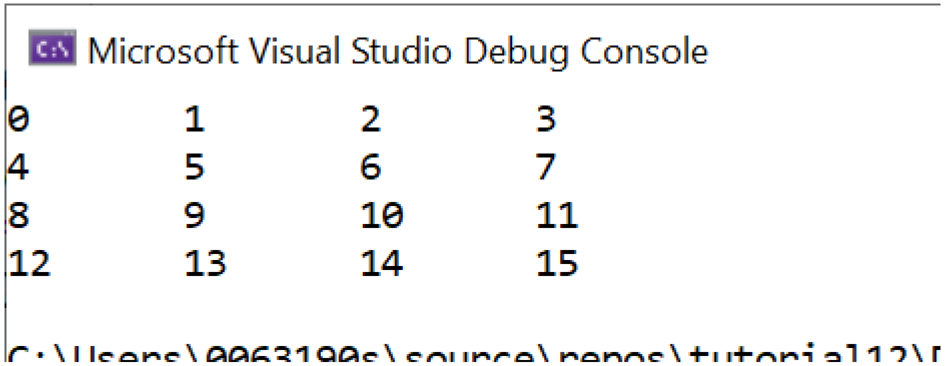
- How do I loop over elements in a 2D array?
- You need 2 loops:
  - Outer loop for the rows
  - Inner loop for the columns
  - In the first part of this example, we use two loops to set the values in a 4x4 array. We use a separate variable (val) for the values in the array.

```
int x[4][4];
int r, c, val = 0;

// set array values
for (r = 0; r < 4; r++){
  for (c = 0; c < 4; c++){
    x[r][c] = val;
    val++;
  }
}
```

Output the 2D array

- In the second part of the example we use the same approach to print out the array, using tabs (\t) to space out the values better

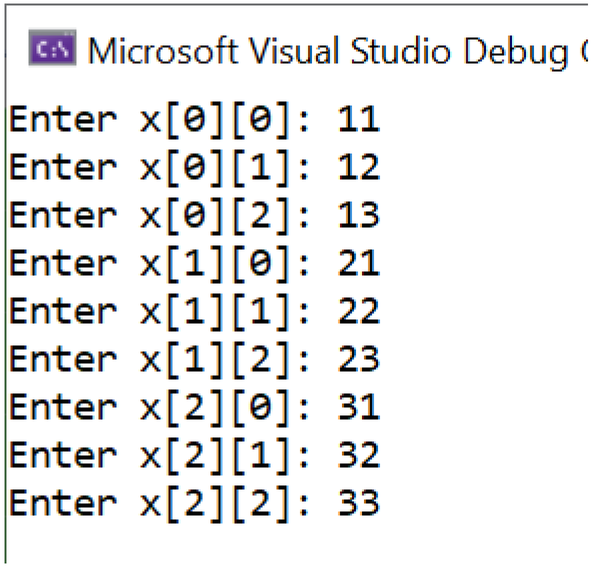
```
// output array
for (r = 0; r < 4; r++){
  for (c = 0; c < 4; c++){
    printf("%d\t", x[r][c]);
  }

  printf("\n");
}
```



```
0       1       2       3
4       5       6       7
8       9       10      11
12      13      14      15

C:\Users\00631900s\source\repos\tutorial12\
```

Input an array

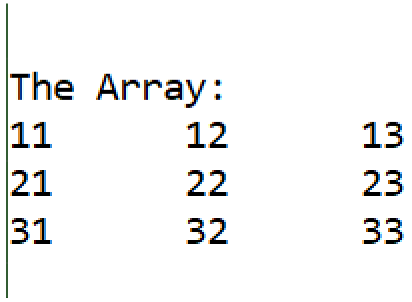
```c
int x[3][3];
int r, c;

// set array values
for (r = 0; r < 3; r++){
  for (c = 0; c < 3; c++){
    printf("Enter x[%d][%d]: ", r, c);
    scanf("%d", &x[r][c]);
  }
}
```



```
Enter x[0][0]: 11
Enter x[0][1]: 12
Enter x[0][2]: 13
Enter x[1][0]: 21
Enter x[1][1]: 22
Enter x[1][2]: 23
Enter x[2][0]: 31
Enter x[2][1]: 32
Enter x[2][2]: 33
```

And then output the array

```
printf("\n\nThe Array:\n");

// output array
for (r = 0; r < 3; r++){
  for (c = 0; c < 3; c++){
    printf("%d\t", x[r][c]);
  }
  printf("\n");
}
```



```
The Array:
11      12      13
21      22      23
31      32      33
```

EXAMPLE PROBLEMS

Grades Processing Problem

- You are writing software to process student grades for a small class with 5 students. Write a program that:
  - Reads and stores the **semester 1** grades of a subject for 2019 and 2020 classes. Use a 2D array to store these grades. It should look like the following:

	Students				
	1	2	3	4	5
2019	64	81	57	92	41
2020	52	76	42	90	61

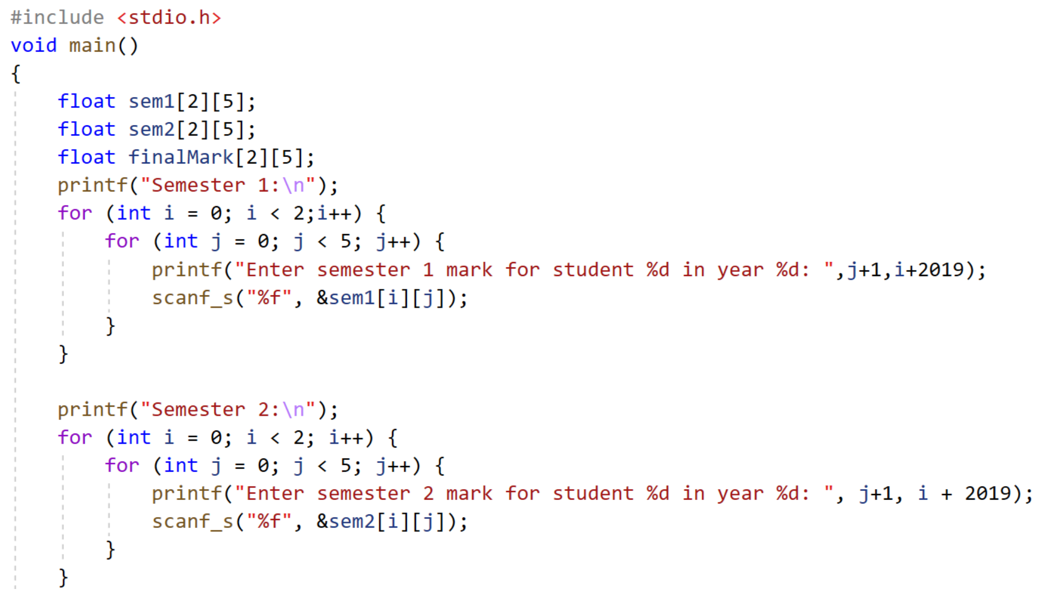
  - Create a similar 2D array to store the grades for **semester 2**. Read in the grades from the user.
  - Create a 3rd 2D array to store the final grade calculated as (semester 1 + semester 2)/2. Print this final 2D array to the screen.

Grades Processing Problem

- Go to C program solution.

Grades Processing Problem

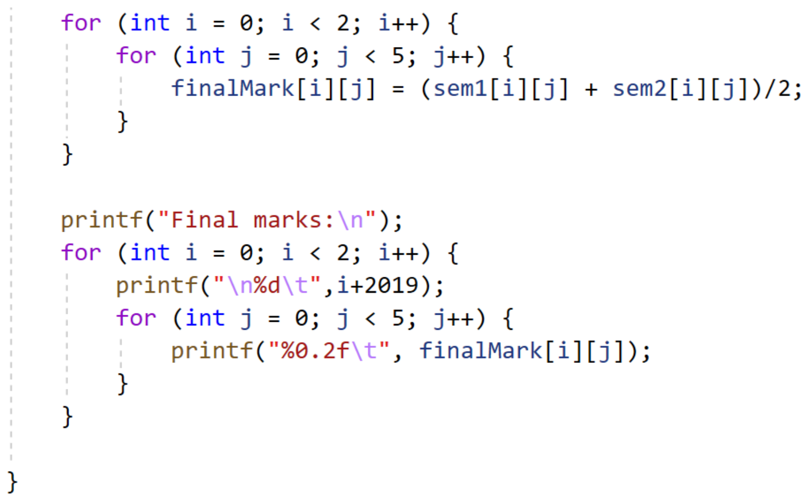
- The following code will work:



```c
#include <stdio.h>
void main()
{
    float sem1[2][5];
    float sem2[2][5];
    float finalMark[2][5];
    printf("Semester 1:\n");
    for (int i = 0; i < 2;i++) {
        for (int j = 0; j < 5; j++) {
            printf("Enter semester 1 mark for student %d in year %d: ",j+1,i+2019);
            scanf_s("%f", &sem1[i][j]);
        }
    }

    printf("Semester 2:\n");
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 5; j++) {
            printf("Enter semester 2 mark for student %d in year %d: ", j+1, i + 2019);
            scanf_s("%f", &sem2[i][j]);
        }
    }
```

…continue

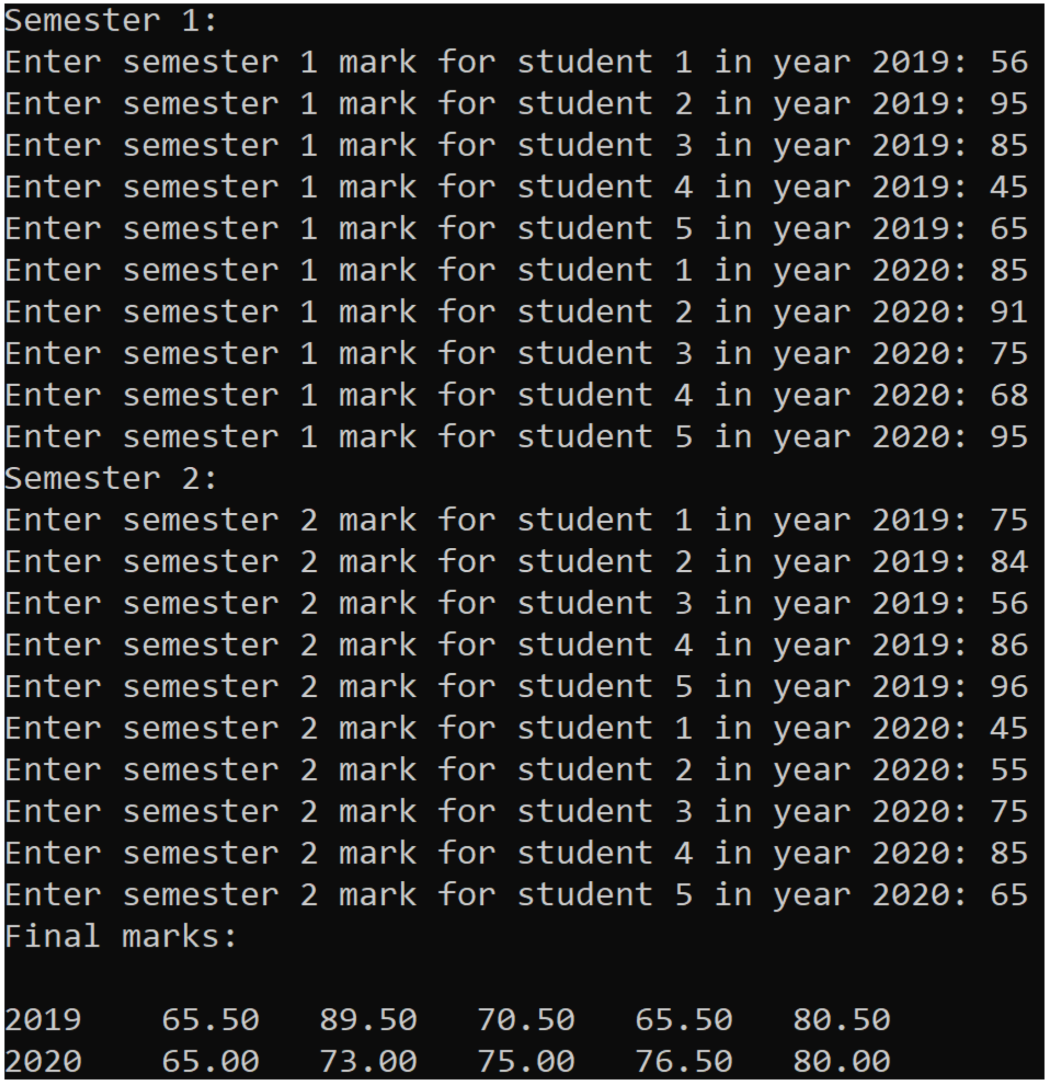
…continue



```c
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 5; j++) {
            finalMark[i][j] = (sem1[i][j] + sem2[i][j])/2;
        }
    }

    printf("Final marks:\n");
    for (int i = 0; i < 2; i++) {
        printf("\n%d\t",i+2019);
        for (int j = 0; j < 5; j++) {
            printf("%0.2f\t", finalMark[i][j]);
        }
    }
}
```

Grades Processing Problem

- C Program Output:



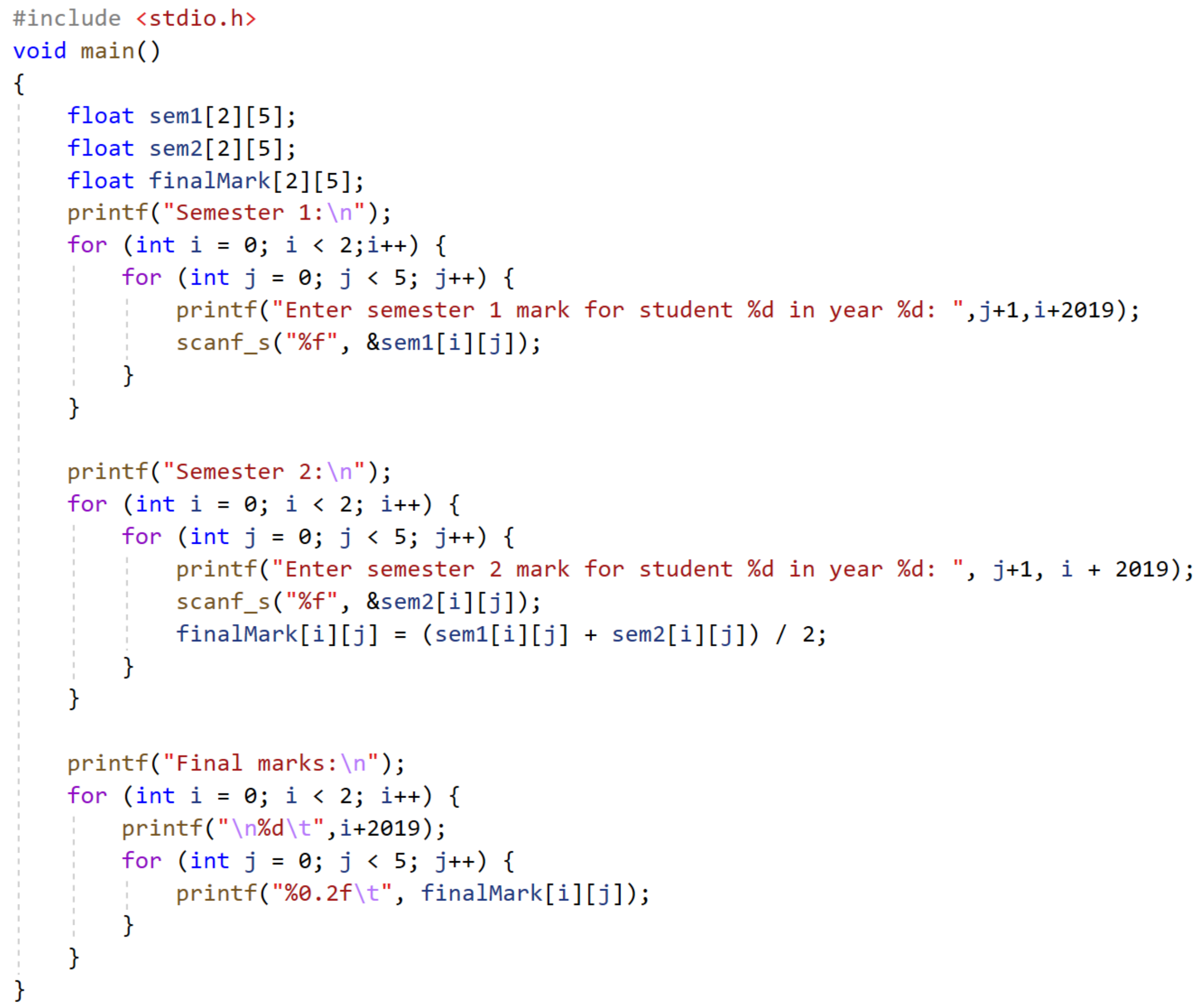
```
Semester 1:
Enter semester 1 mark for student 1 in year 2019: 56
Enter semester 1 mark for student 2 in year 2019: 95
Enter semester 1 mark for student 3 in year 2019: 85
Enter semester 1 mark for student 4 in year 2019: 45
Enter semester 1 mark for student 5 in year 2019: 65
Enter semester 1 mark for student 1 in year 2020: 85
Enter semester 1 mark for student 2 in year 2020: 91
Enter semester 1 mark for student 3 in year 2020: 75
Enter semester 1 mark for student 4 in year 2020: 68
Enter semester 1 mark for student 5 in year 2020: 95
Semester 2:
Enter semester 2 mark for student 1 in year 2019: 75
Enter semester 2 mark for student 2 in year 2019: 84
Enter semester 2 mark for student 3 in year 2019: 56
Enter semester 2 mark for student 4 in year 2019: 86
Enter semester 2 mark for student 5 in year 2019: 96
Enter semester 2 mark for student 1 in year 2020: 45
Enter semester 2 mark for student 2 in year 2020: 55
Enter semester 2 mark for student 3 in year 2020: 75
Enter semester 2 mark for student 4 in year 2020: 85
Enter semester 2 mark for student 5 in year 2020: 65
Final marks:

2019    65.50   89.50   70.50   65.50   80.50
2020    65.00   73.00   75.00   76.50   80.00
```

Grades Processing Problem

- The previous solution had 4 for loops, can we make our program shorter?

- Yes!
- This will produce the same output.

- Could we make our code shorter again?



```c
#include <stdio.h>
void main()
{
    float sem1[2][5];
    float sem2[2][5];
    float finalMark[2][5];
    printf("Semester 1:\n");
    for (int i = 0; i < 2;i++) {
        for (int j = 0; j < 5; j++) {
            printf("Enter semester 1 mark for student %d in year %d: ",j+1,i+2019);
            scanf_s("%f", &sem1[i][j]);
        }
    }

    printf("Semester 2:\n");
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 5; j++) {
            printf("Enter semester 2 mark for student %d in year %d: ", j+1, i + 2019);
            scanf_s("%f", &sem2[i][j]);
            finalMark[i][j] = (sem1[i][j] + sem2[i][j]) / 2;
        }
    }

    printf("Final marks:\n");
    for (int i = 0; i < 2; i++) {
        printf("\n%d\t",i+2019);
        for (int j = 0; j < 5; j++) {
            printf("%0.2f\t", finalMark[i][j]);
        }
    }
}
```

PROGRAMMING

CT103

Week 6a

Sign in on Blackboard

- Please sign in on blackboard for CT103.
- I will leave the sign in option open for an hour after the lecture.

Lecture Content

- Last lecture (Week 5b):
  - Arrays recap
  - Arrays in memory
  - 2D Arrays
  - Example 2D arrays C program

- Today’s lecture (Week 6a):
  - What is a string in C
  - How to initialise a string
  - Printing strings
  - Scanning strings
  - Example C program

STRINGS

String Definition

- A string is a collection of characters, i.e. text.

- Specifically, in C strings are defined as an array of characters.

- Examples of strings include:
  - “It’s a nice day!”
  - “My name is Fred”
  - “The temperature outside is 24 degrees”

Strings

- Creating a string:

- If you wanted to create a string in C, you would do something like this:

```c
char name[] = "Alex";
```

Why Use Strings?

- Up until now we have only considered numeric, Boolean and single character data.
- Strings are necessary because you will often be manipulating data that consists of text, e.g. names, addresses, etc.

Start with character arrays

- In C there is no variable type “String”.
- This is the case for many other higher level languages.
- We therefore use an array of characters to store a string.

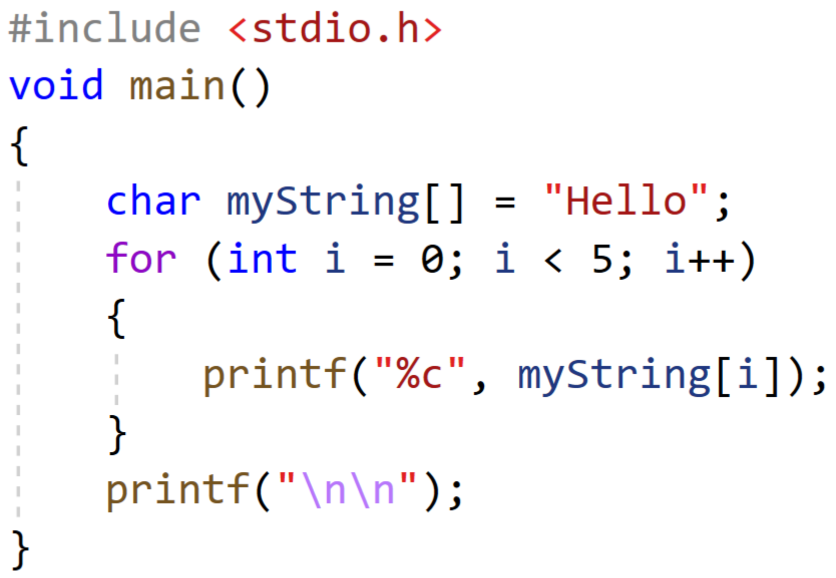
- char string1[100] = "Hello";

	string1[0]	string1[1]	string1[2]	string1[3]	string1[4]
string1[100]	‘H'	‘e'	‘l'	‘l'	‘o'
address:	75F7CC	75F7D0	75F7D4	75F7D8	75F7DC

PRINTING STRINGS

Printing Strings

- You could print strings using a for loop as shown below:

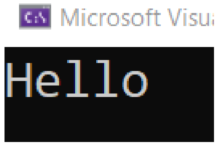


```c
#include <stdio.h>
void main()
{
    char myString[] = "Hello";
    for (int i = 0; i < 5; i++)
    {
        printf("%c", myString[i]);
    }
    printf("\n\n");
}
```

Printing Strings

- This will work but it is **not recommended**.

```
#include <stdio.h>
void main()
{
    char myString[] = "Hello";
    for (int i = 0; i < 5; i++)
    {
        printf("%c", myString[i]);
    }
    printf("\n\n");
}
```



```
Hello
```

Strings and Character Arrays

- What is the difference between strings and character arrays?
- A string is terminated with a special character ‘\0’.
- When you create a string, the character ‘\0’ is automatically put at the end.

Strings in Memory

- char string1[100] = "Hello";

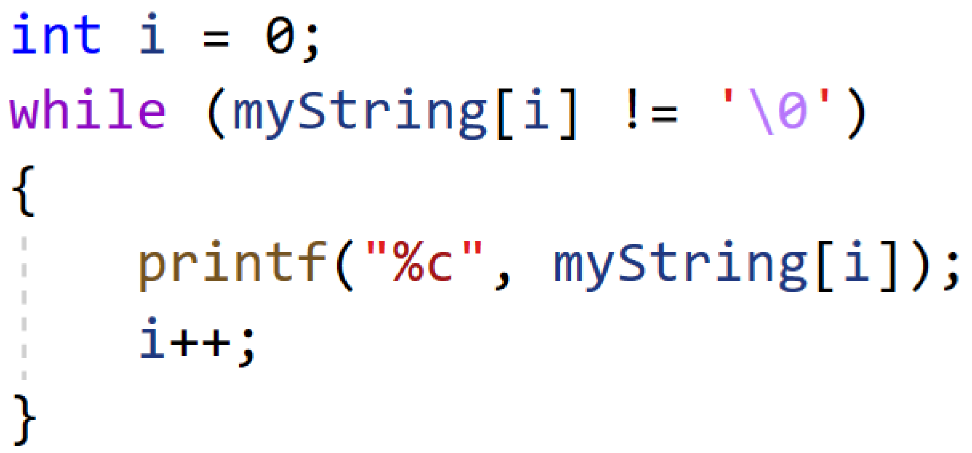
- Actually results in:

	string1[0]	string1[1]	string1[2]	string1[3]	string1[4]	string1[5]
string1[100]	'H'	'e'	'l'	'l'	'o'	'\0'
address:	75F7CC	75F7D0	75F7D4	75F7D8	75F7DC	75F7E1

So string[5] will contain ‘\0’ – used to stop processing by any function that processes this string

Printing Strings

- Since all strings end with ‘\0’, you could also print the string using:

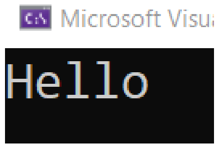


```
int i = 0;
while (myString[i] != '\0')
{
    printf("%c", myString[i]);
    i++;
}
```

Printing Strings

- You should simply use %s to print strings.

```c
#include <stdio.h>
void main()
{
    char myString[] = "Hello";
    printf("%s",myString);
    printf("\n\n");
}
```



```
Hello
```

SCANNING STRINGS

Scanning Strings

- Up until now, you needed to use ‘&’ when scanning in data.

- For example, you would type something like the following for characters (chars):

```
char c;
scanf("%c", &c);
```

Scanning Strings

- This is not the case with strings.
- You do not need to use ‘&’ when scanning in strings.
- The reason for this is a bit technical:
  - Char array names decay to pointers in C.
  - The string name already points to the address of the first element in the string.
  - Therefore we don’t need &.

Scanning Strings

- So how do we scan in strings in C?

- Use the following:

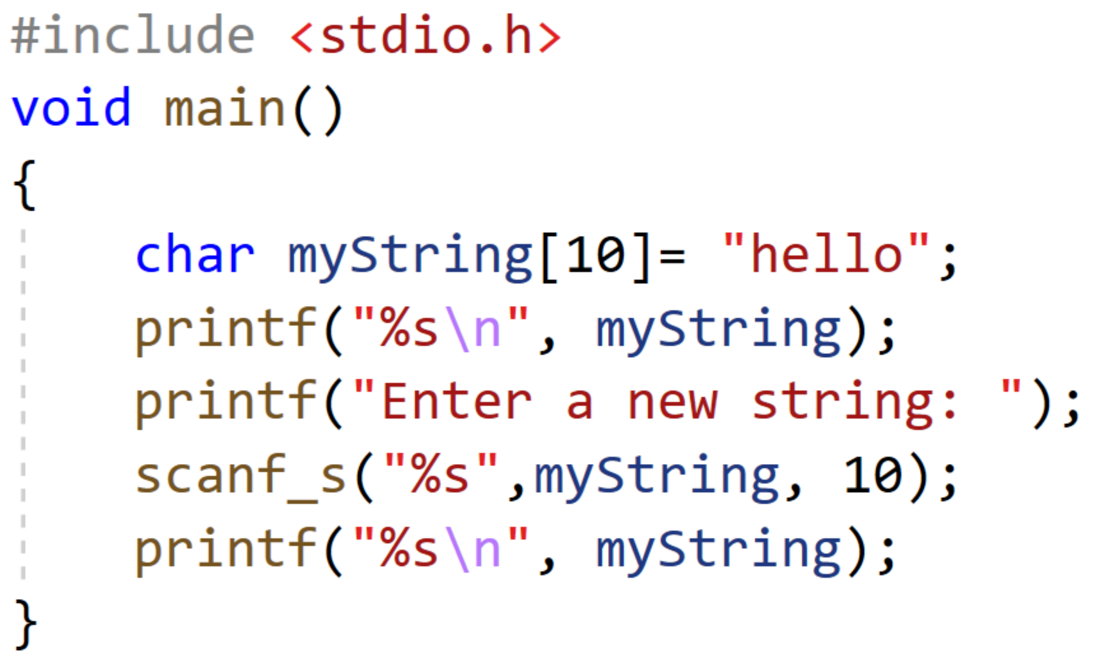
No & symbol!

```
char myString[10];
scanf_s("%s",myString, 10);
```

- Note how you need to specify the character array length!

Scanning Strings

- Lets look at the following example:

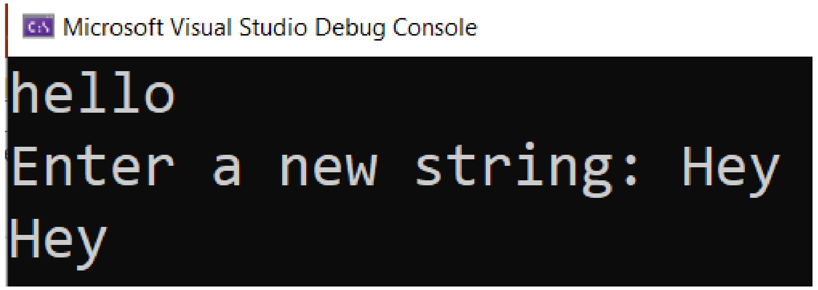


```c
#include <stdio.h>
void main()
{
    char myString[10]= "hello";
    printf("%s\n", myString);
    printf("Enter a new string: ");
    scanf_s("%s",myString, 10);
    printf("%s\n", myString);
}
```

Scanning Strings

- This outputs the following:

```c
#include <stdio.h>
void main()
{
    char myString[10]= "hello";
    printf("%s\n", myString);
    printf("Enter a new string: ");
    scanf_s("%s",myString, 10);
    printf("%s\n", myString);
}
```

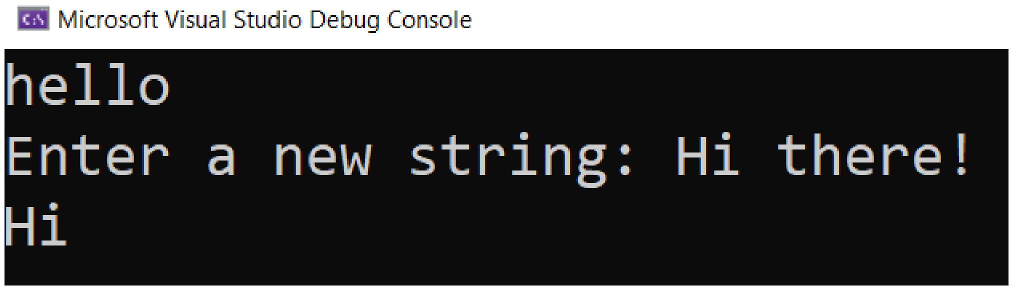


```
hello
Enter a new string: Hey
Hey
```

Scanning Strings

- Scanf_s is limited to one single word by default.
- If you enter a space, it will stop scanning.

```c
#include <stdio.h>
void main()
{
    char myString[10]= "hello";
    printf("%s\n", myString);
    printf("Enter a new string: ");
    scanf_s("%s",myString, 10);
    printf("%s\n", myString);
}
```



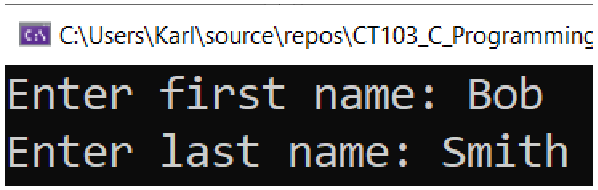
Microsoft Visual Studio Debug Console

```
hello
Enter a new string: Hi there!
Hi
```

Scanning Two Words

- You will can do the following if you want to scan two words.

```
char firstName[10];
char surname[10];

printf("Enter first name: ");
scanf_s("%s", firstName, 10);
printf("Enter last name: ");
scanf_s("%s", surname, 10);
```



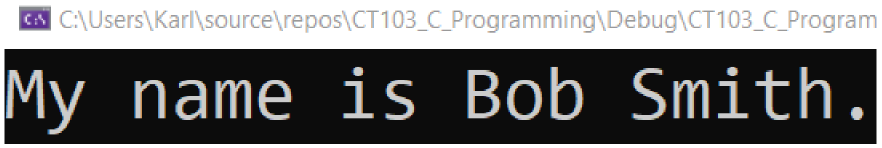
C:\Users\Karl\source\repos\CT103_C_Programming

```
Enter first name: Bob
Enter last name: Smith
```

Strings with Two Words

- This does not mean that you cannot have a string with spaces in it.

```c
char myName[10] = "Bob Smith";
printf("My name is %s.\n",myName);
```



C:\Users\Karl\source\repos\CT103_C_Programming\Debug\CT103_C_Program

```
My name is Bob Smith.
```

Scanning Two Words

- If you do want to scan two words into one string, you can do the following:

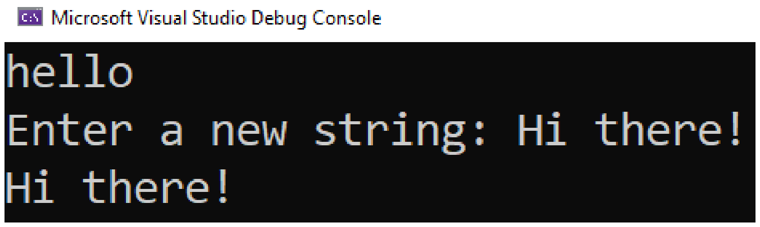
```
scanf_s("%[^\n]%*c",myString, 10);
```

- The **[^\n]** tells scanf to keep reading characters until a new line is entered (\n).
- The **%*c** remove the new line from the input buffer.

Scanning Two Words

- Lets see this work in a C program:

```c
char myString[10]= "hello";
printf("%s\n", myString);
printf("Enter a new string: ");
scanf_s("%[^\n]%*c",myString, 10);
printf("%s\n", myString);
```



Microsoft Visual Studio Debug Console

```
hello
Enter a new string: Hi there!
Hi there!
```

EXAMPLE PROBLEMS

Employee Name Scanner

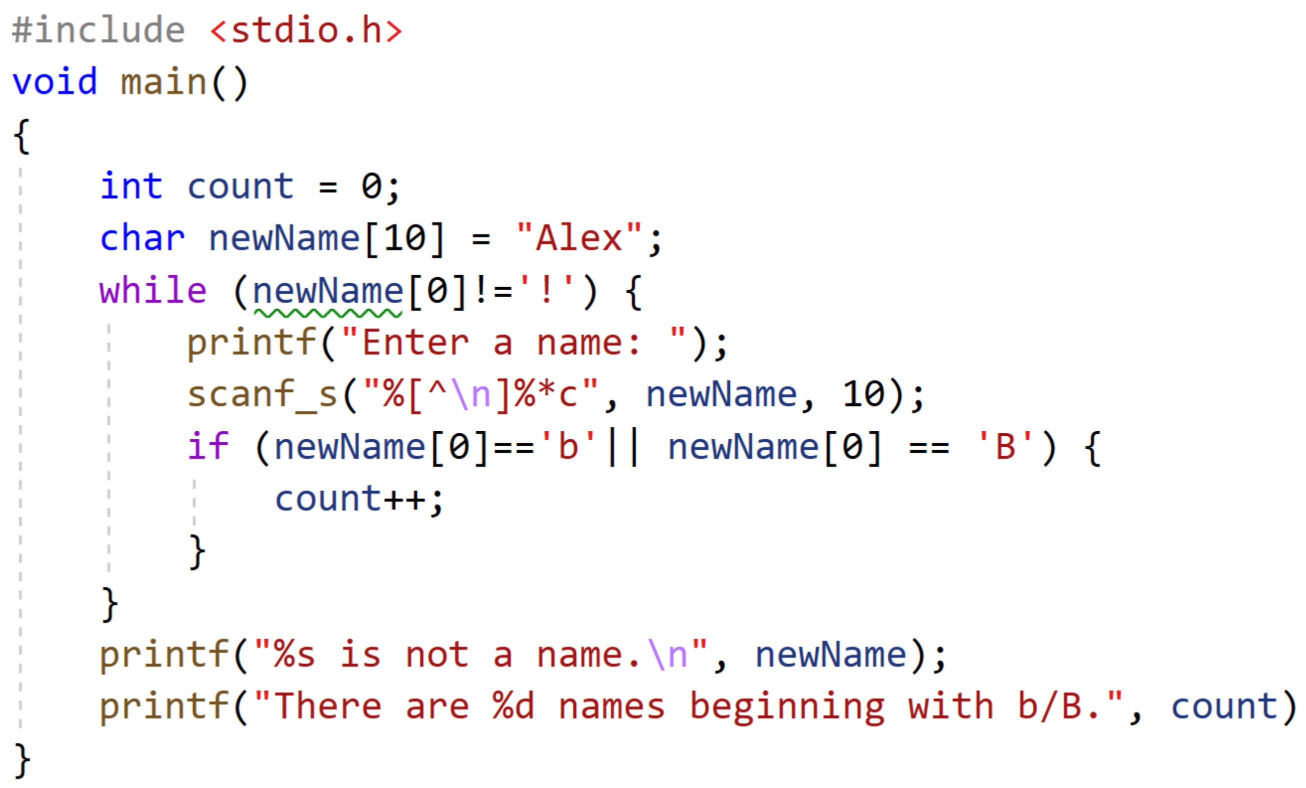
- You are writing software to read in employee names. Write a program that:
  - Reads in employee names as strings.
  - The program should stop reading names if the character ‘!’ is entered.
  - Count how many employees have names beginning with ‘b’ or ‘B’.
  - Print the answer to the screen.

Employee Name Scanner

- Go to C program solution.

Employee Name Scanner

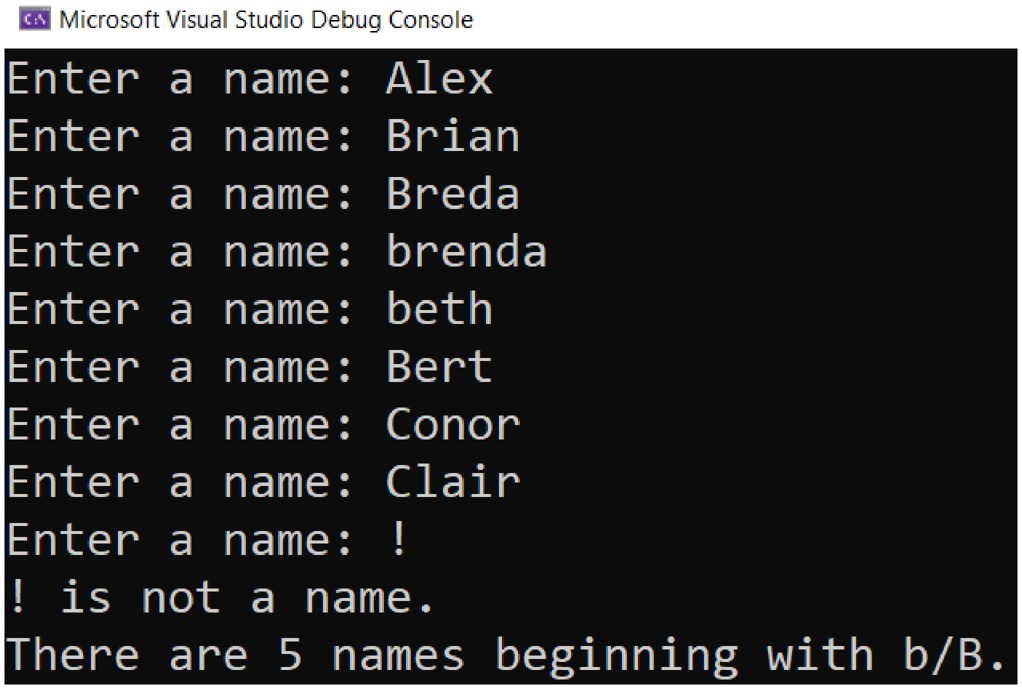
- The following code will work:



```c
#include <stdio.h>
void main()
{
    int count = 0;
    char newName[10] = "Alex";
    while (newName[0]!='!') {
        printf("Enter a name: ");
        scanf_s("%[^\n]%*c", newName, 10);
        if (newName[0]=='b'|| newName[0] == 'B') {
            count++;
        }
    }
    printf("%s is not a name.\n", newName);
    printf("There are %d names beginning with b/B.", count)
}
```

Employee Name Scanner

- C Program Output:



```
Enter a name: Alex
Enter a name: Brian
Enter a name: Breda
Enter a name: brenda
Enter a name: beth
Enter a name: Bert
Enter a name: Conor
Enter a name: Clair
Enter a name: !
! is not a name.
There are 5 names beginning with b/B.
```

PROGRAMMING

CT103

Week 6b

Sign in on Blackboard

- Please sign in on blackboard for CT103.
- I will leave the sign in option open for an hour after the lecture.

Lecture Content

- Last lecture (Week 6a):
  - What is a string in C
  - How to initialise a string
  - Printing strings
  - Scanning strings
  - Example C program

- Today’s lecture (Week 6b):
  - Length of a string
  - Insert data in a string
  - String functions
  - Example C program

STRING LENGTH

String Recap

- A string is a collection of characters, i.e. text.
- Specifically, in C strings are defined as an array of characters.
- You can create a string in C as follows:

```c
char name[] = "Alex";
```

String Length

- The **length** of a string is always the number of characters up to, but <u>not including</u>, the string terminator.

- By string length we really mean the number of characters actually used.

- The length of the following string is **9**:
    - “August 10\0”

Size of String

- Any string should be big enough to hold the text you need to put into it PLUS 1 more for the null character

```
// can hold up to 3 characters + null character
// size determined by [4]
char str1[4] = "One";

// can hold up to 3 characters + null character
// size determined when it is initialised with “Two”
char str2[] = "Two";

// can hold up to 99 characters + null character
// even though we only use 6 at initialisation
char str3[100] = "Three";
```

Get Length of String

- We could count the length of the string ourselves:

```
char string1[100] = "This is some random text";

int len = 0;

while (string1[len] != '\0')
{
        len++;
}

printf("Length of string = %d \n", len);
```

- Try out this code yourself!

Get Length of String

- It is much faster if we use the strlen() function to get the length of the string.
- strlen() does what the previous example does.
- In order to use strlen(), we need to include the “string.h” library at the beginning of the program.
- This is a library of string functions.

Get Length of String using strlen()

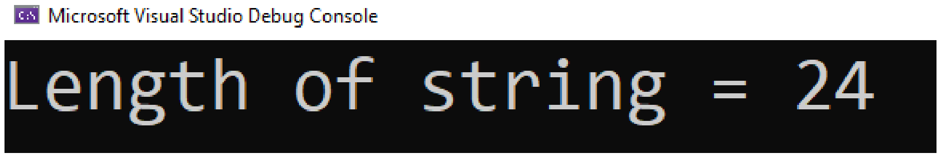
```
#include <stdio.h>
#include "string.h"

void main()
{
        char string1[100] = "This is some random text";

        int len = strlen(string1);

        printf("Length of string = %d \n", len);

}
```

Don’t forget this!

Get Length of String using strlen()

```
#include <stdio.h>
#include "string.h"

void main()
{
        char string1[100] = "This is some random text";

        int len = strlen(string1);

        printf("Length of string = %d \n", len);

}
```

Don’t forget this!



```
Length of string = 24
```

Length of a String Summary

- A string is just an array of characters.
- To use a string it must be ***terminated*** properly – this means the last character in the array must be the null character ‘\0’.
- Functions that return the length of a string don’t count the null character (even though they return it), so you always have to allocate an array of size 1 more than the number of characters you want to store.
- The length of the following string is 9:
  - “August 10”
- However, you would need to allocate an array of characters of size 10 to hold it!
- Usually you just allocate **plenty !**

DATA INTO STRINGS

Putting Data into Strings

- In lecture 6a, we talked about setting strings using scanf_s, e.g.

```
scanf_s("%[^\n]%*c",myString, 10);
```

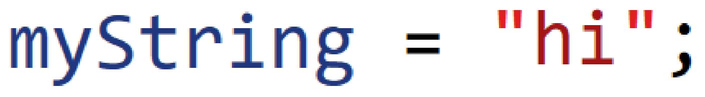
- How would we set a string without scanning in text?

- Can I simply write the following?

```
myString = "hi";
```

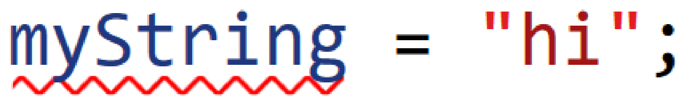
Putting Data into Strings

- Can I simply write the following?



```
myString = "hi";
```

- No, this won’t work.

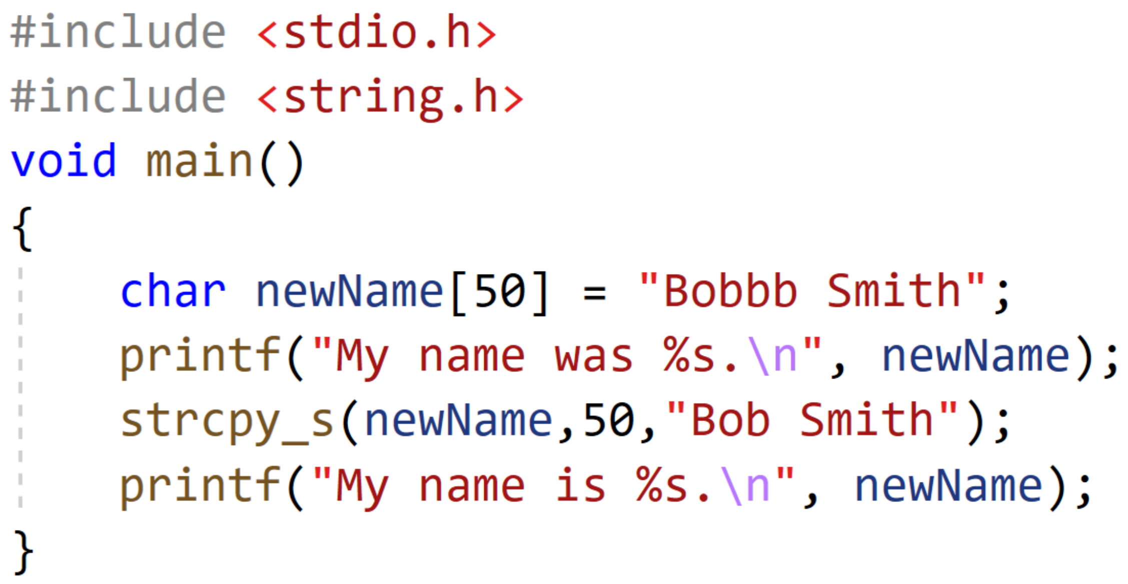


```
myString = "hi";
```

- You need to use strcpy_s() from the string.h library that we mentioned before.

Strcpy_s()

- See the following example that uses strcpy_s()

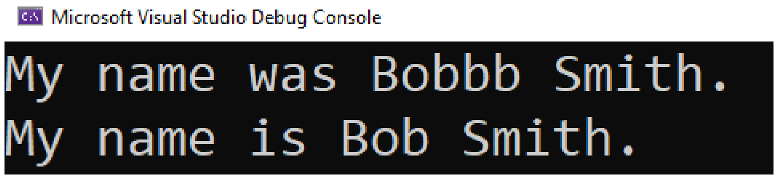


```c
#include <stdio.h>
#include <string.h>
void main()
{
    char newName[50] = "Bobbb Smith";
    printf("My name was %s.\n", newName);
    strcpy_s(newName,50,"Bob Smith");
    printf("My name is %s.\n", newName);
}
```

Strcpy_s()

- Produces the following output:

```c
#include <stdio.h>
#include <string.h>
void main()
{
    char newName[50] = "Bobbb Smith";
    printf("My name was %s.\n", newName);
    strcpy_s(newName,50,"Bob Smith");
    printf("My name is %s.\n", newName);
}
```



```
My name was Bobbb Smith.
My name is Bob Smith.
```

STRING FUNCTIONS

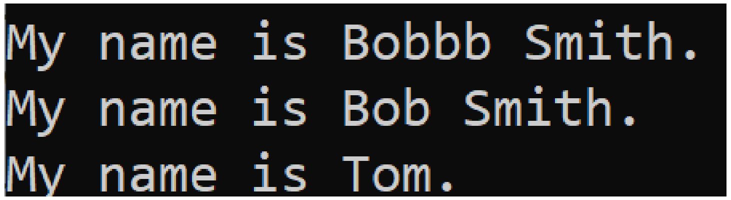
Common String functions

- Strcpy_s() Copy one string to another (**seen already**)
  - Strncpy_s() Copy n characters from one string to another
- Strcat_s() Link together (concatenate) two strings
  - Strncat_s() concatenate n characters from two strings
- strcmp() Compare two strings
  - strncmp() Compare n characters from two strings

Strncpy_s()

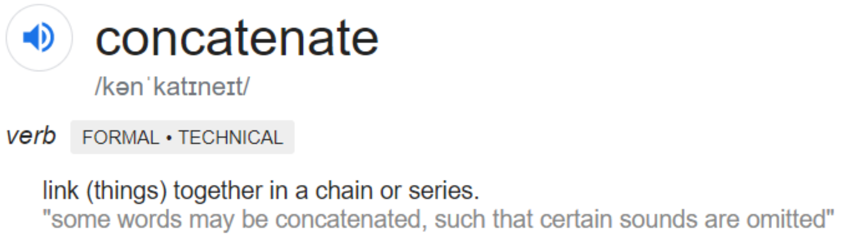
- Strncpy_s()
- Copy n characters from one string to another.

```
char tName[] = "Tommy";
char newName[50] = "Bobbb Smith";
printf("My name is %s.\n", newName);
strcpy_s(newName,50,"Bob Smith");
printf("My name is %s.\n", newName);
strncpy_s(newName,50, tName,3);
printf("My name is %s.\n", newName);
```

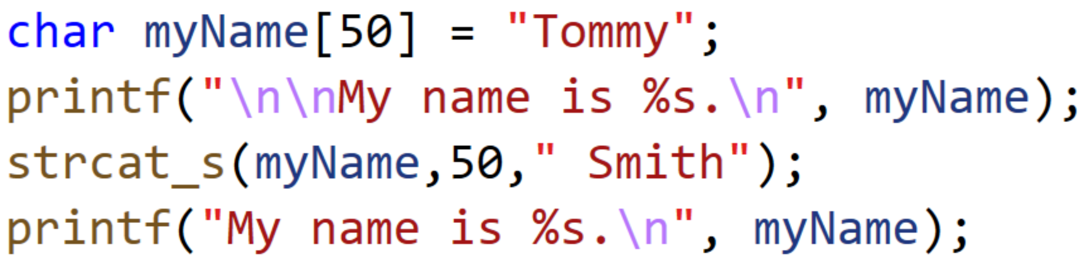


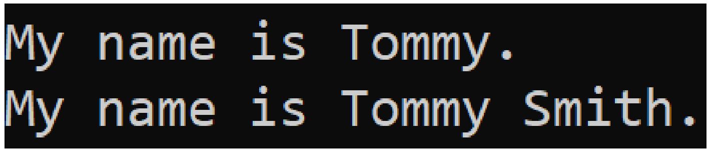
```
My name is Bobbb Smith.
My name is Bob Smith.
My name is Tom.
```

Strcat_s()


concatenate
/kənˈkatɪneɪt/
verb FORMAL • TECHNICAL
link (things) together in a chain or series.
"some words may be concatenated, such that certain sounds are omitted"

- Strcat_s()
- Strcat_s() Link together (concatenate) two strings

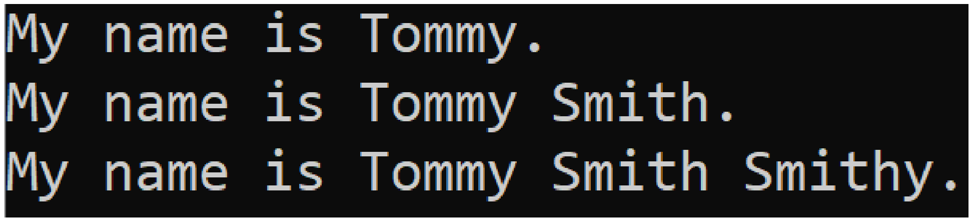

```c
char myName[50] = "Tommy";
printf("\n\nMy name is %s.\n", myName);
strcat_s(myName,50," Smith");
printf("My name is %s.\n", myName);
```


```
My name is Tommy.
My name is Tommy Smith.
```

Strncat_s()

- Strncat_s()
- Strncat_s() concatenate n characters from two strings

```c
char myName[50] = "Tommy";
printf("\n\nMy name is %s.\n", myName);
strcat_s(myName,50," Smith");
printf("My name is %s.\n", myName);
strncat_s(myName, 50, " Smithyyyy",7);
printf("My name is %s.\n", myName);
```

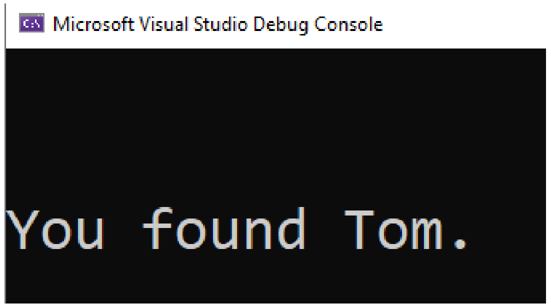


```
My name is Tommy.
My name is Tommy Smith.
My name is Tommy Smith Smithy.
```

Strcmp()

- Compare two strings
- Strcmp() will return 0 if both strings are the same.

```
char fName1[] = "Tom";
char fName2[] = "Tim";
if (strcmp(fName1, "Tom") == 0) {
    printf("\n\nYou found Tom.\n");
}
else {
    printf("\n\nKeep looking.\n");
}
```

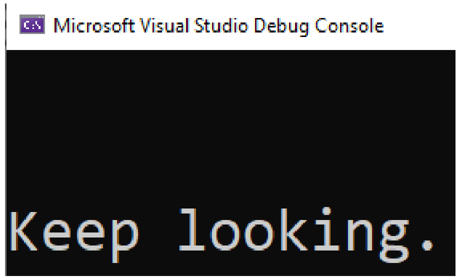


```
You found Tom.
```

Strcmp()

- Compare two strings
- Strcmp() will return 0 if both strings are the same.

```c
char fName1[] = "Tom";
char fName2[] = "Tim";
if (strcmp(fName2, "Tom") == 0) {
    printf("\n\nYou found Tom.\n");
}
else {
    printf("\n\nKeep looking.\n");
}
```

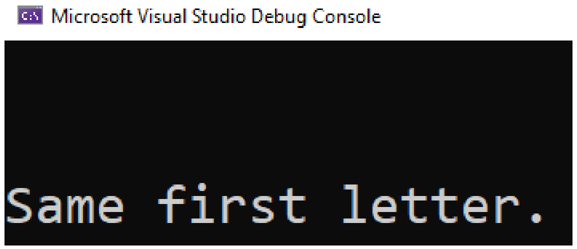


```
Keep looking.
```

Strncmp()

- Compare n characters from two strings
- Strncmp() will return 0 if first n chars of both strings are the same.

```c
char fName1[] = "Tom";
char fName2[] = "Tim";
if (strncmp(fName1, fName2,1) == 0) {
    printf("\n\nSame first letter.\n");
}
else {
    printf("\n\nDifferent first letter.\n");
}
```

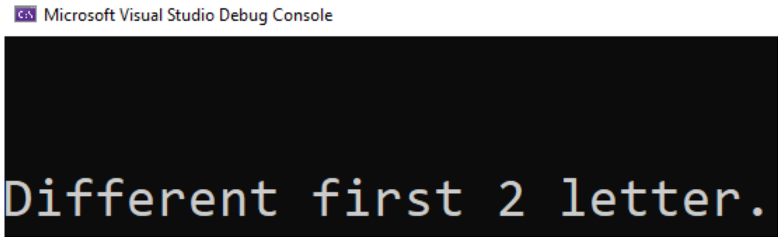

Microsoft Visual Studio Debug Console

```


Same first letter.
```

Strncmp()

- Compare n characters from two strings
- Strncmp() will return 0 if first n chars of both strings are the same.

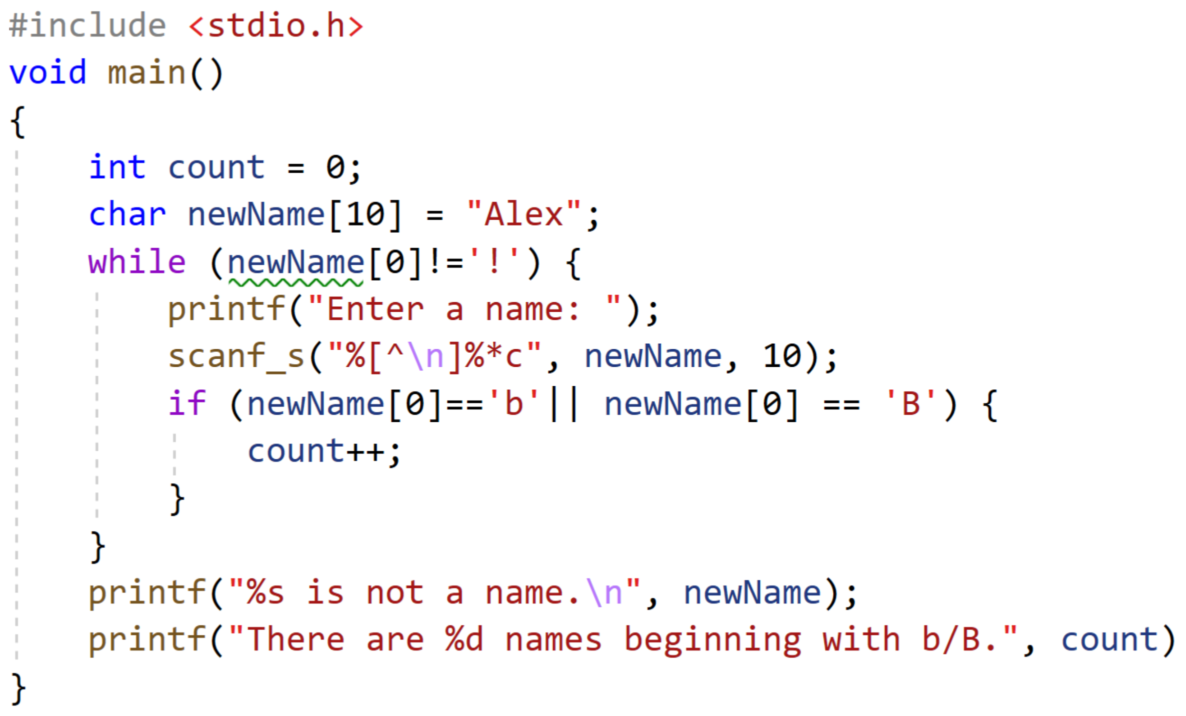
```c
char fName1[] = "Tom";
char fName2[] = "Tim";
if (strncmp(fName1, fName2,2) == 0) {
    printf("\n\nSame first 2 letter.\n");
}
else {
    printf("\n\nDifferent first 2 letter.\n");
}
```



```
Different first 2 letter.
```

Note on last weeks example

- We used **newName[0]!=‘!’**
- We could also use strncmp()



```c
#include <stdio.h>
void main()
{
    int count = 0;
    char newName[10] = "Alex";
    while (newName[0]!='!') {
        printf("Enter a name:  ");
        scanf_s("%[^\n]%*c", newName, 10);
        if (newName[0]=='b'|| newName[0] == 'B') {
            count++;
        }
    }
    printf("%s is not a name.\n", newName);
    printf("There are %d names beginning with b/B.", count)
}
```

Note on last weeks example

- See strncmp()

```
#include <stdio.h>
void main()
{
    int count = 0;
    char newName[10] = "Alex";
    while (newName[0]!='!') {
        printf("Enter a name: ");
        scanf_s("%[^\n]%*c", newName, 10);
        if (newName[0]=='b'|| newName[0] == 'B') {
            count++;
        }
    }
    printf("%s is not a name.\n", newName);
    printf("There are %d names beginning with b/B.", count)
}
```

```
#include <stdio.h>
#include <string.h>
void main()
{
    int count = 0;
    char newName[10] = "Alex";
    while (!strncmp(newName, "!", 1) == 0) {
        printf("Enter a name: ");
        scanf_s("%[^\n]%*c", newName, 10);
        if (newName[0]=='b'|| newName[0] == 'B') {
            count++;
        }
    }
    printf("%s is not a name.\n", newName);
    printf("There are %d names beginning with b/B.", count);
}
```

EXAMPLE PROBLEMS

Employee Name Comparison

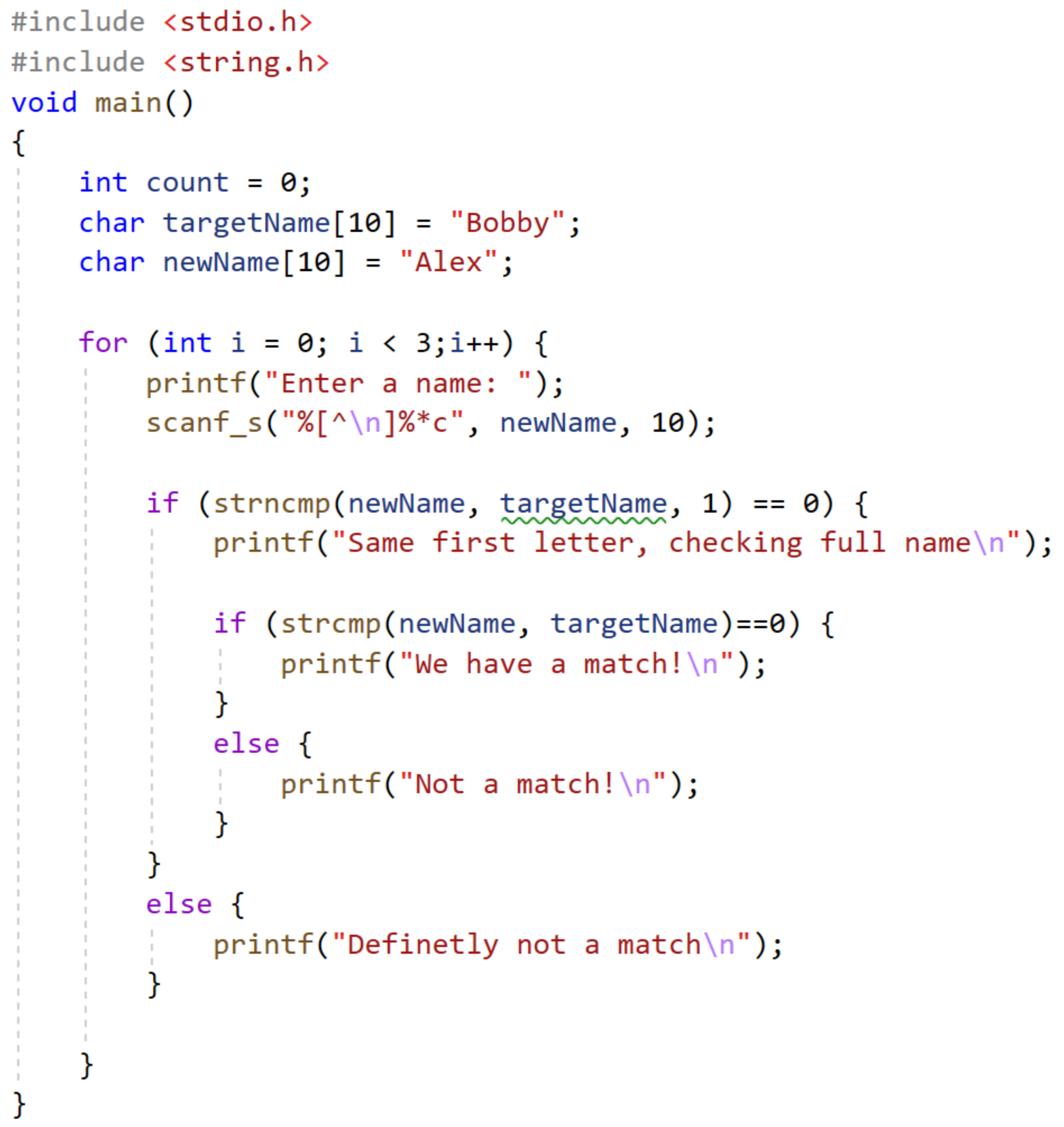
- You are writing more software to read in employee names. Write a program that:
  - Reads in 3 employee names as strings.
  - Check the first letter against the target name “Bobby”.
  - If the name also begins with the letter ‘B’, check and see if the full names are the same.

Employee Name Comparison

- Go to C program solution.

Employee Name Comparison

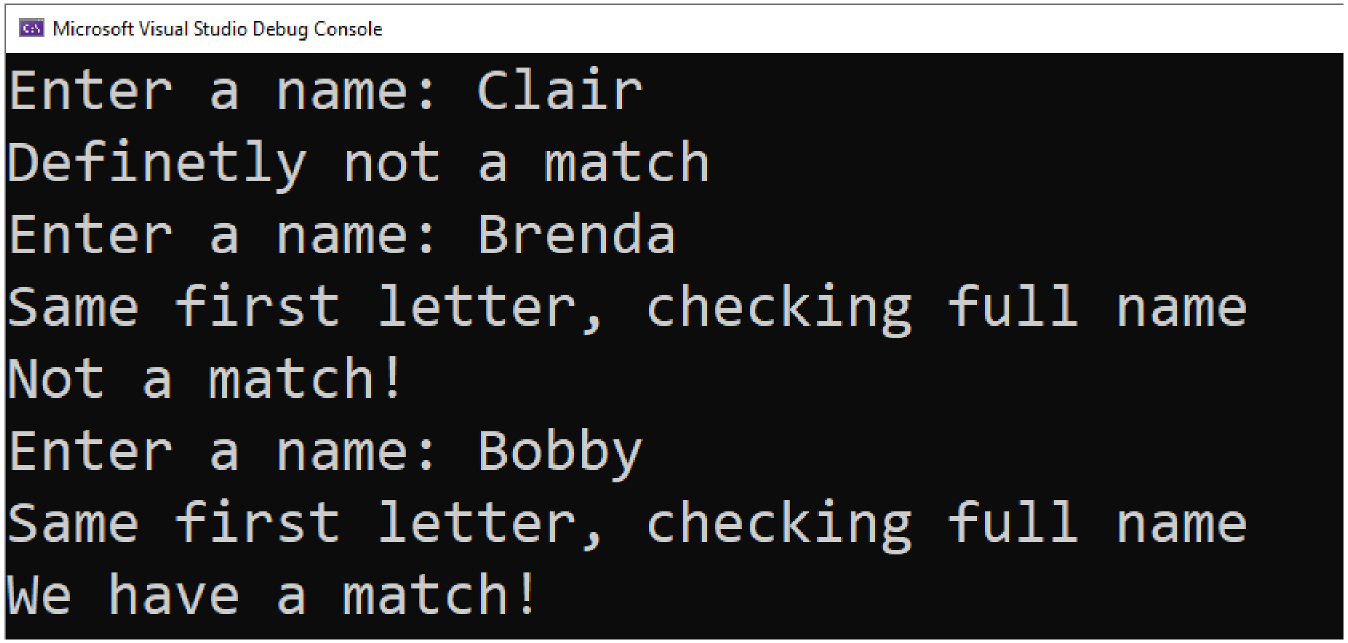
- The following code will work:



```c
#include <stdio.h>
#include <string.h>
void main()
{
    int count = 0;
    char targetName[10] = "Bobby";
    char newName[10] = "Alex";

    for (int i = 0; i < 3;i++) {
        printf("Enter a name: ");
        scanf_s("%[^\n]%*c", newName, 10);

        if (strncmp(newName, targetName, 1) == 0) {
            printf("Same first letter, checking full name\n");

            if (strcmp(newName, targetName)==0) {
                printf("We have a match!\n");
            }
            else {
                printf("Not a match!\n");
            }
        }
        else {
            printf("Definetly not a match\n");
        }

    }
}
```

Employee Name Comparison

- C Program Output:



```
Enter a name: Clair
Definetly not a match
Enter a name: Brenda
Same first letter, checking full name
Not a match!
Enter a name: Bobby
Same first letter, checking full name
We have a match!
```

PROGRAMMING

CT103

Week 7a

Sign in on Blackboard

- Please sign in on blackboard for CT103.
- I will leave the sign in option open for an hour after the lecture.

New Lab Groups

- Please note the **change in lab groups** for weeks 7-12:

- **2pm-4pm** lab session: Students with surnames **L to Z**.
- **4pm-6pm** lab session: Students with surnames **A to K**.

- Please make sure you attend the correct lab session.

Lecture Content

- Last lecture (Week 6b):
  - Length of a string
  - Insert data in a string
  - String functions
  - Example C program

- Today’s lecture (Week 7a):
  - Constants
  - Puts
  - Gets
  - Sscanf_s
  - Example C program

CONSTANTS

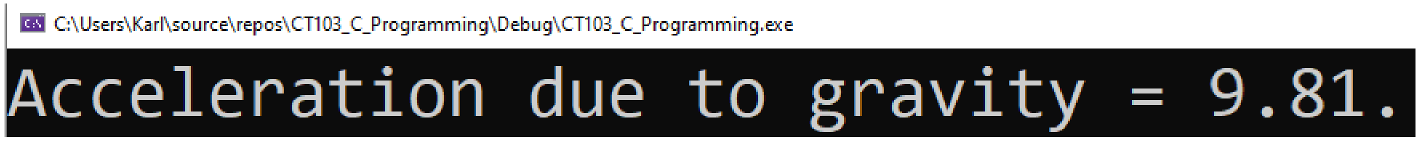
Constants

- We talked a lot about variables already and know what they are.
- E.g. `int age = 62;`
- We can also create **constants**.
- **Constants** refer to fixed values that the program cannot change during its execution. These are also often called literals.

Constants

- You would create a constant in C as follows:

```c
const float gravity = 9.81;
printf("Acceleration due to gravity = %0.2f.\n\n", gravity);
```

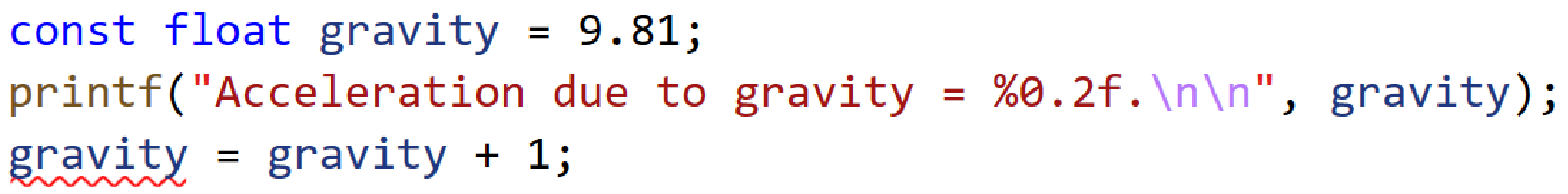


C:\Users\Karl\source\repos\CT103_C_Programming\Debug\CT103_C_Programming.exe

```
Acceleration due to gravity = 9.81.
```

Constants

- As the name suggests, you cannot change the value of a constant.



```c
const float gravity = 9.81;
printf("Acceleration due to gravity = %0.2f.\n\n", gravity);
gravity = gravity + 1;
```

Can't do this!

#Define in C

- You can also declare constants using #define.
- These need to be created outside of main.
- Using #define creates what is called a **macro**.

Macro

- What is a **macro**?

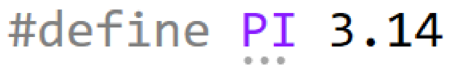
- A macro is a fragment of code which has been given a name. Whenever the name is used it is replaced by the contents of the macro.

- There are two types of macros:
  - Object like macros.
  - Function like macros (we will ignore these for now).

#Define in C

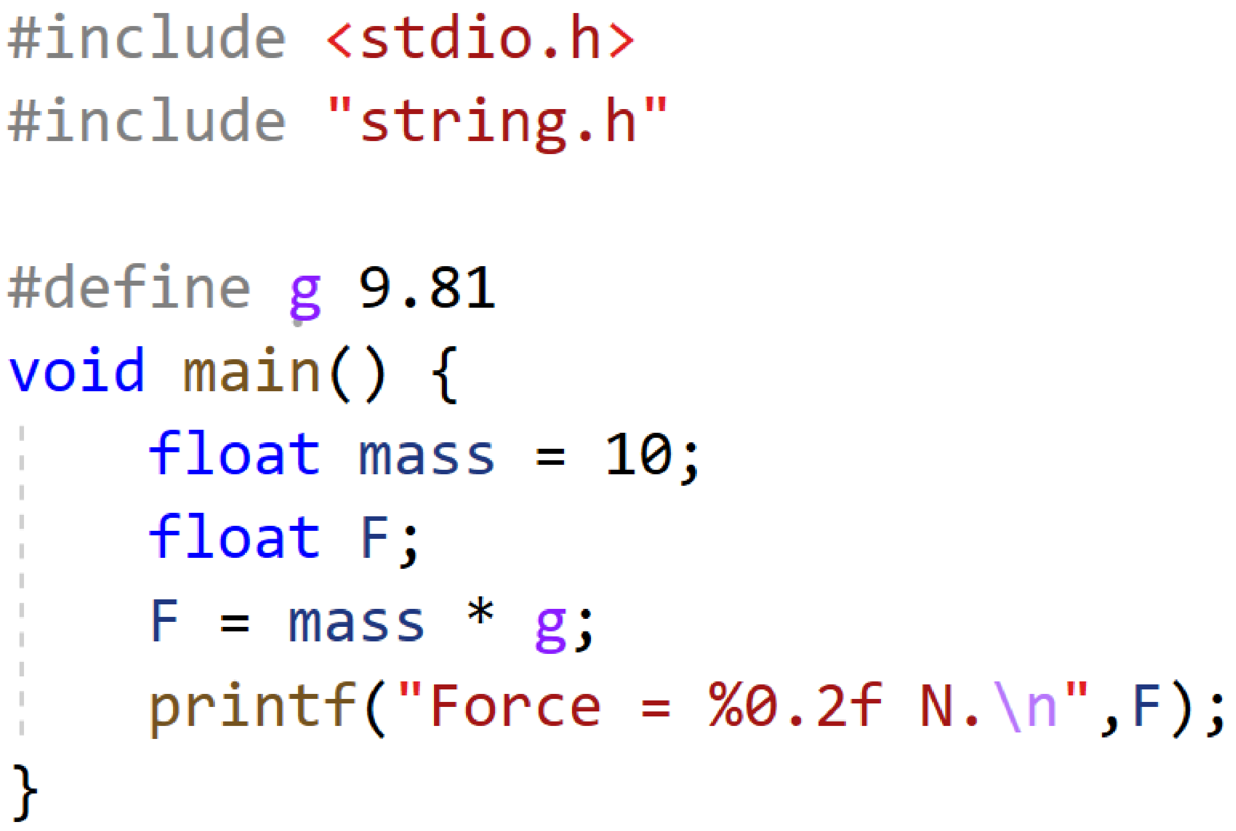
- What does #define look like in C?

- You can create an object like macro in C using the following:



```
#define PI 3.14
```

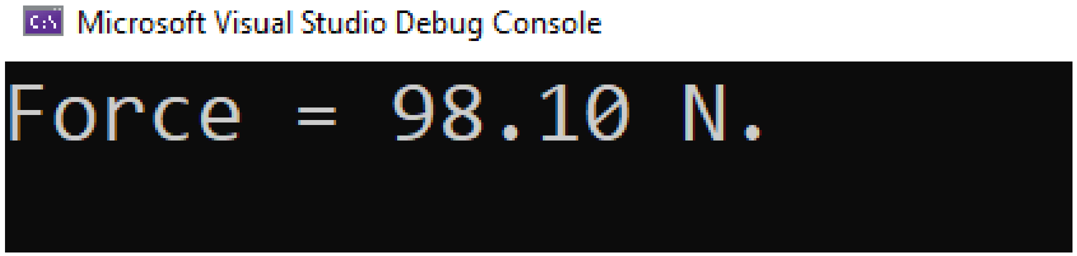
#Define in C Example



```c
#include <stdio.h>
#include "string.h"

#define g 9.81
void main() {
    float mass = 10;
    float F;
    F = mass * g;
    printf("Force = %0.2f N.\n",F);
}
```

#Define in C Example

```c
#include <stdio.h>
#include "string.h"

#define g 9.81
void main() {
    float mass = 10;
    float F;
    F = mass * g;
    printf("Force = %0.2f N.\n",F);
}
```



```
Force = 98.10 N.
```

PUTS

Puts

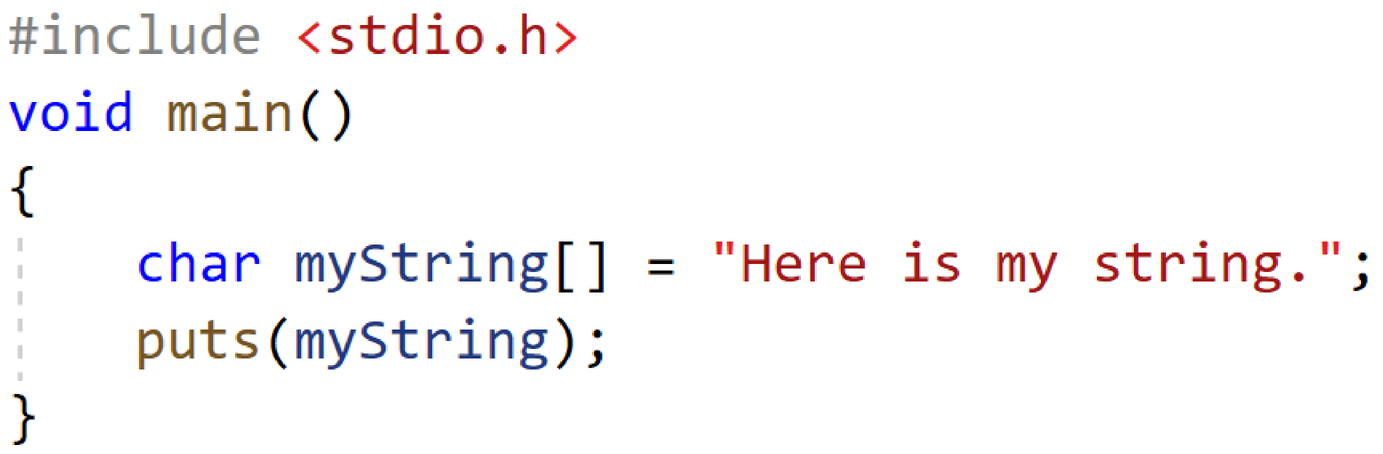
- What is puts?
- Puts is a function for printing strings to the screen.
- You need to include the <stdio.h> library to call puts.

Why Puts Over Printf?

- Puts is simple.
- Puts is less expensive than printf.
- Puts is more secure.

Puts in C

- You would use Puts as follows in C:

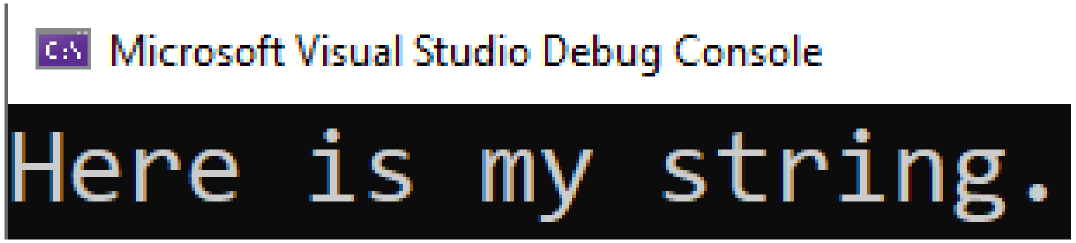


```c
#include <stdio.h>
void main()
{
    char myString[] = "Here is my string.";
    puts(myString);
}
```

Puts in C

- This gives the following output:

```c
#include <stdio.h>
void main()
{
    char myString[] = "Here is my string.";
    puts(myString);
}
```



```
Here is my string.
```

GETS

Gets

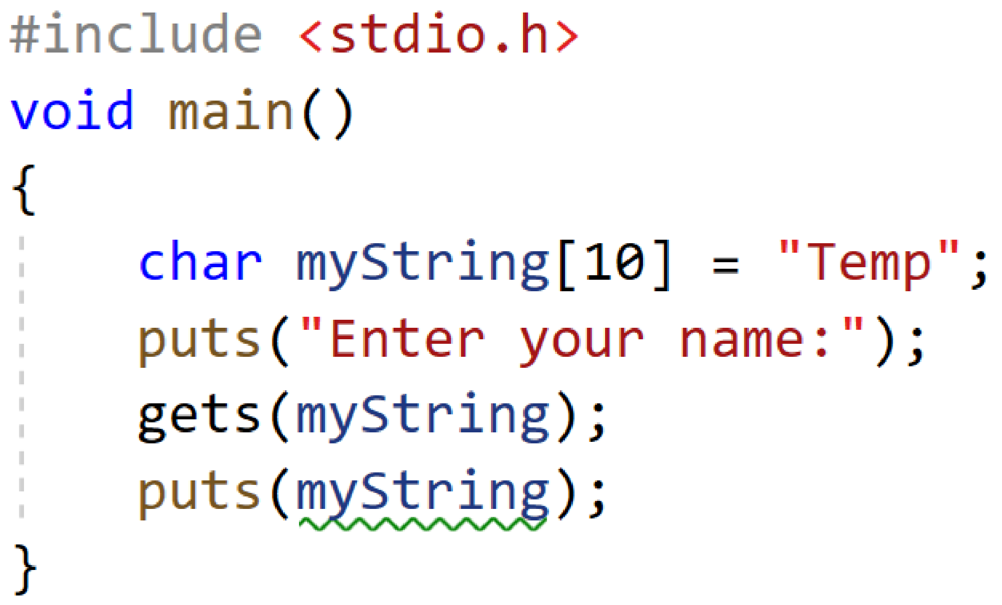
- What is gets?
- Gets is a function for reading input from the keyboard.
- You also need to include the <stdio.h> library to call gets.

Gets vs Scanf?

- Gets is only used for strings.
- Gets will not stop reading characters, even with whitespace, until it reaches a newline.
- Gets is easy to use.

Gets in C

- You would use Gets as follows in C:

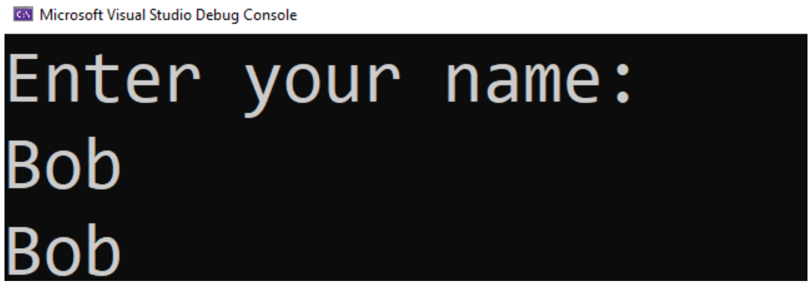


```c
#include <stdio.h>
void main()
{
	char myString[10] = "Temp";
	puts("Enter your name:");
	gets(myString);
	puts(myString);
}
```

Gets in C

- This gives the following output:

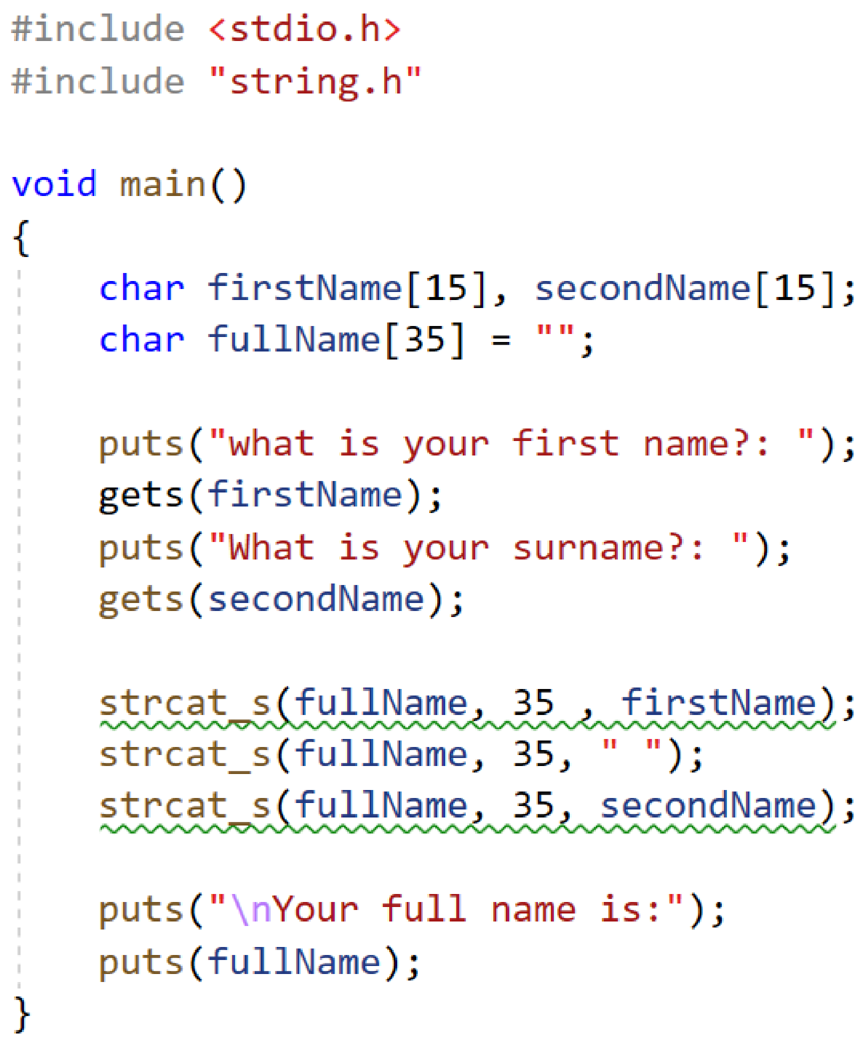
```c
#include <stdio.h>
void main()
{
	char myString[10] = "Temp";
	puts("Enter your name:");
	gets(myString);
	puts(myString);
}
```


Microsoft Visual Studio Debug Console

```
Enter your name:
Bob
Bob
```

USING PUTS AND GETS

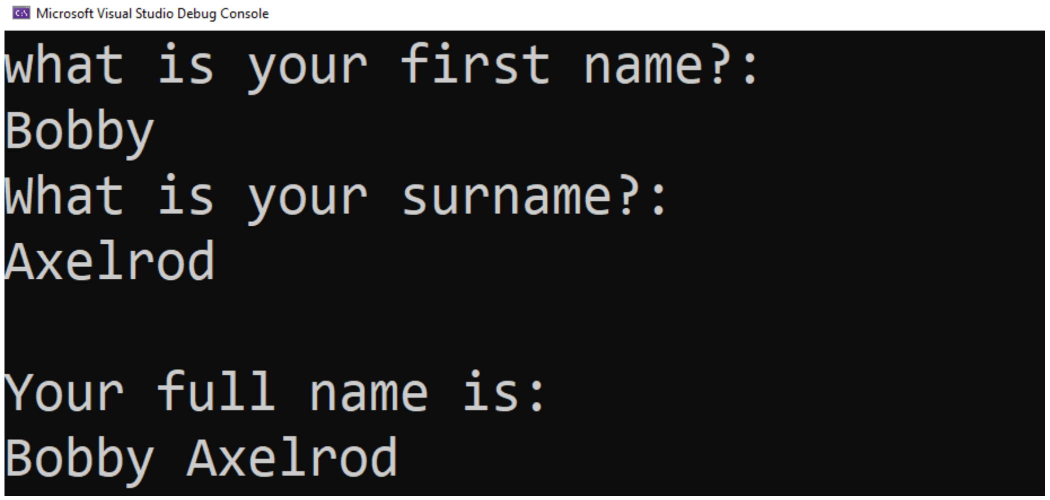
Puts and Gets



```
#include <stdio.h>
#include "string.h"

void main()
{
    char firstName[15], secondName[15];
    char fullName[35] = "";

    puts("what is your first name?: ");
    gets(firstName);
    puts("What is your surname?: ");
    gets(secondName);

    strcat_s(fullName, 35 , firstName);
    strcat_s(fullName, 35, " ");
    strcat_s(fullName, 35, secondName);

    puts("\nYour full name is:");
    puts(fullName);
}
```

Puts and Gets

```c
#include <stdio.h>
#include "string.h"

void main()
{
    char firstName[15], secondName[15];
    char fullName[35] = "";

    puts("what is your first name?: ");
    gets(firstName);
    puts("What is your surname?: ");
    gets(secondName);

    strcat_s(fullName, 35 , firstName);
    strcat_s(fullName, 35, " ");
    strcat_s(fullName, 35, secondName);

    puts("\nYour full name is:");
    puts(fullName);
}
```



```
what is your first name?:
Bobby
What is your surname?:
Axelrod

Your full name is:
Bobby Axelrod
```

SSCANF_S

Sscanf_s

- You may get strings from anywhere, such as files or databases.

- These may then need to be parsed to extract data.

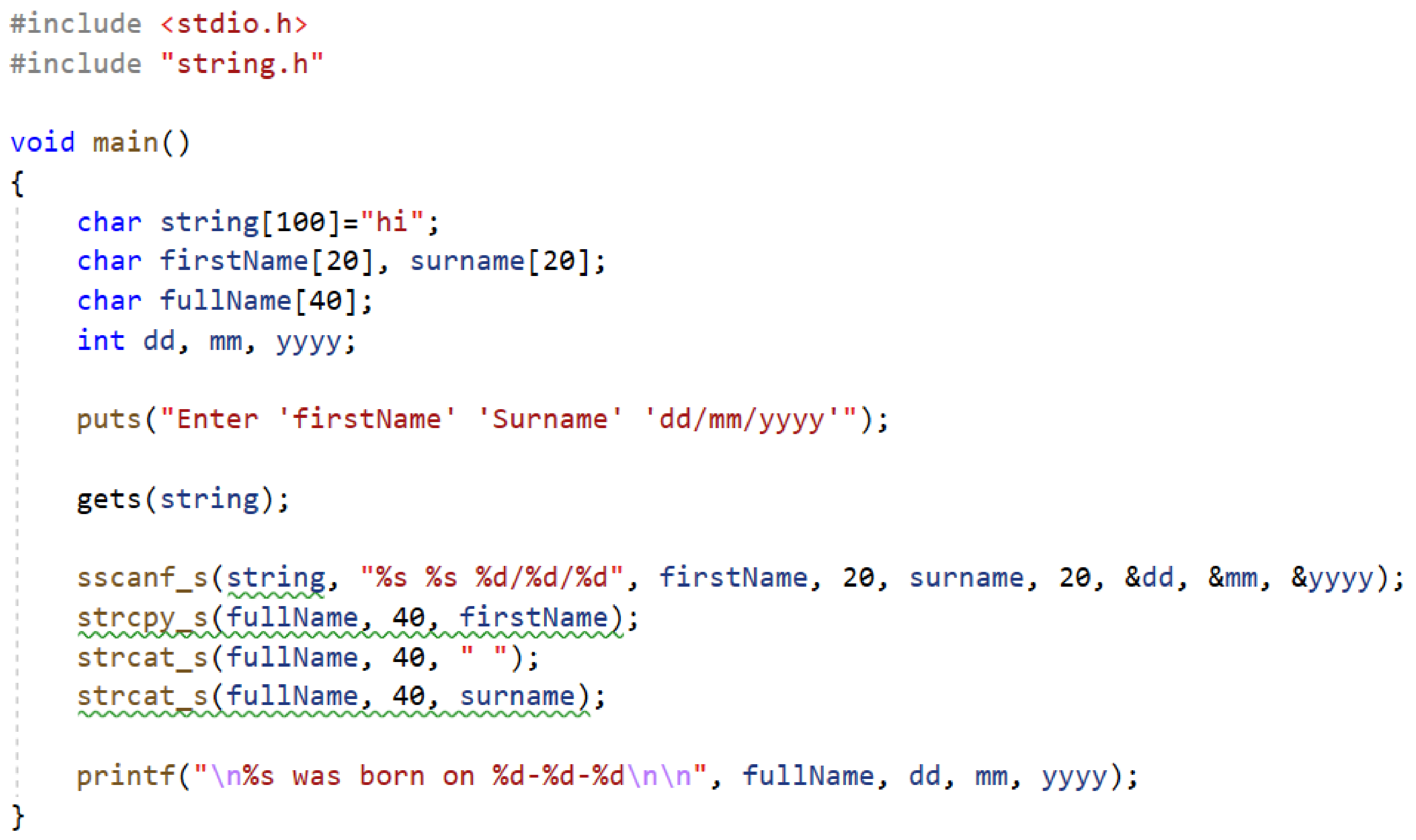
- What does **parse** mean?
  - Transforming a steam of text into some other form of information.

Sscanf_s

- Sscanf_s is useful for **scanning formatted data from a string**.

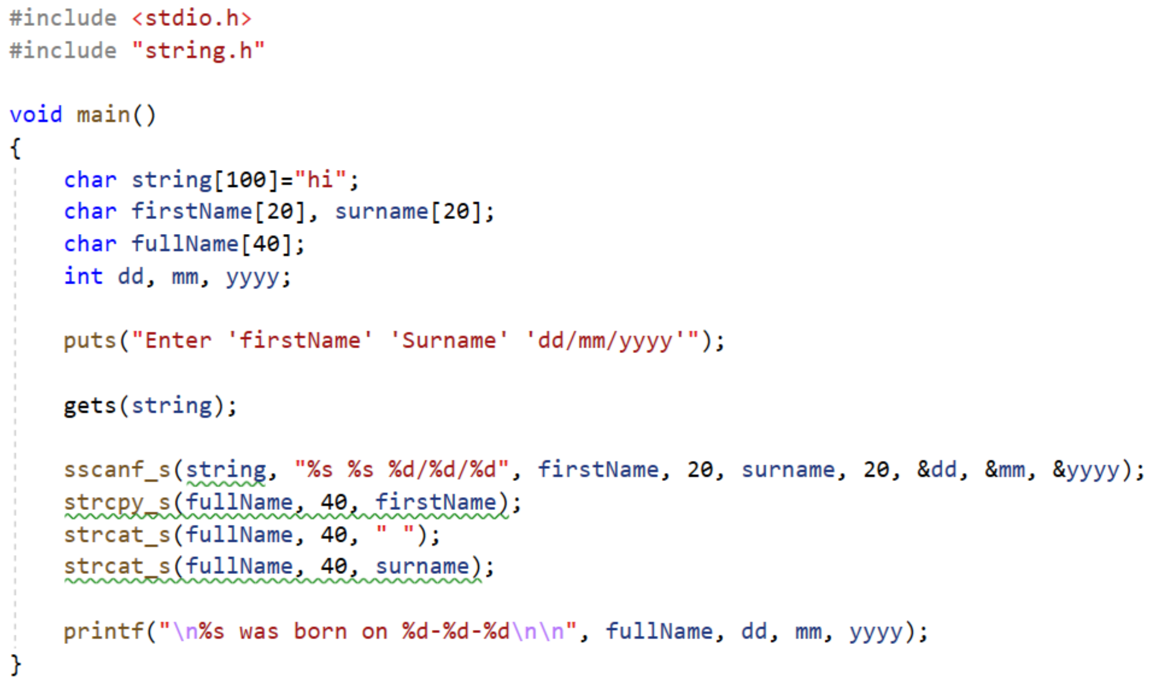
- **IF** you know the exact format of the string (and it won’t be changed), you can read it the same way as you would read the console input using scanf.

Sscanf_s Example

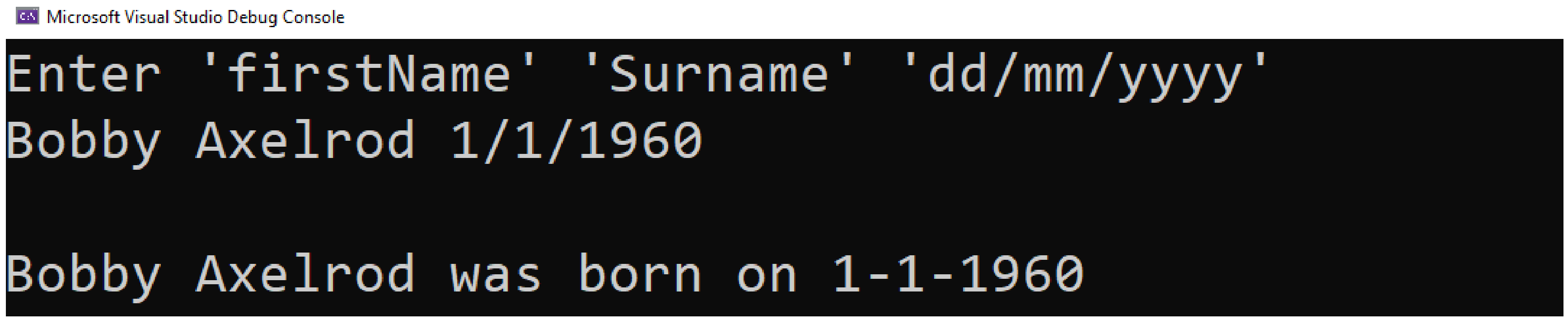


```c
#include <stdio.h>
#include "string.h"

void main()
{
    char string[100]="hi";
    char firstName[20], surname[20];
    char fullName[40];
    int dd, mm, yyyy;

    puts("Enter 'firstName' 'Surname' 'dd/mm/yyyy'");

    gets(string);

    sscanf_s(string, "%s %s %d/%d/%d", firstName, 20, surname, 20, &dd, &mm, &yyyy);
    strcpy_s(fullName, 40, firstName);
    strcat_s(fullName, 40, " ");
    strcat_s(fullName, 40, surname);

    printf("\n%s was born on %d-%d-%d\n\n", fullName, dd, mm, yyyy);
}
```

Sscanf_s Example



```c
#include <stdio.h>
#include "string.h"

void main()
{
    char string[100]="hi";
    char firstName[20], surname[20];
    char fullName[40];
    int dd, mm, yyyy;

    puts("Enter 'firstName' 'Surname' 'dd/mm/yyyy'");

    gets(string);

    sscanf_s(string, "%s %s %d/%d/%d", firstName, 20, surname, 20, &dd, &mm, &yyyy);
    strcpy_s(fullName, 40, firstName);
    strcat_s(fullName, 40, " ");
    strcat_s(fullName, 40, surname);

    printf("\n%s was born on %d-%d-%d\n\n", fullName, dd, mm, yyyy);
}
```



Microsoft Visual Studio Debug Console

```
Enter 'firstName' 'Surname' 'dd/mm/yyyy'
Bobby Axelrod 1/1/1960

Bobby Axelrod was born on 1-1-1960
```

EXAMPLE PROBLEMS

Physics Energy Calculator

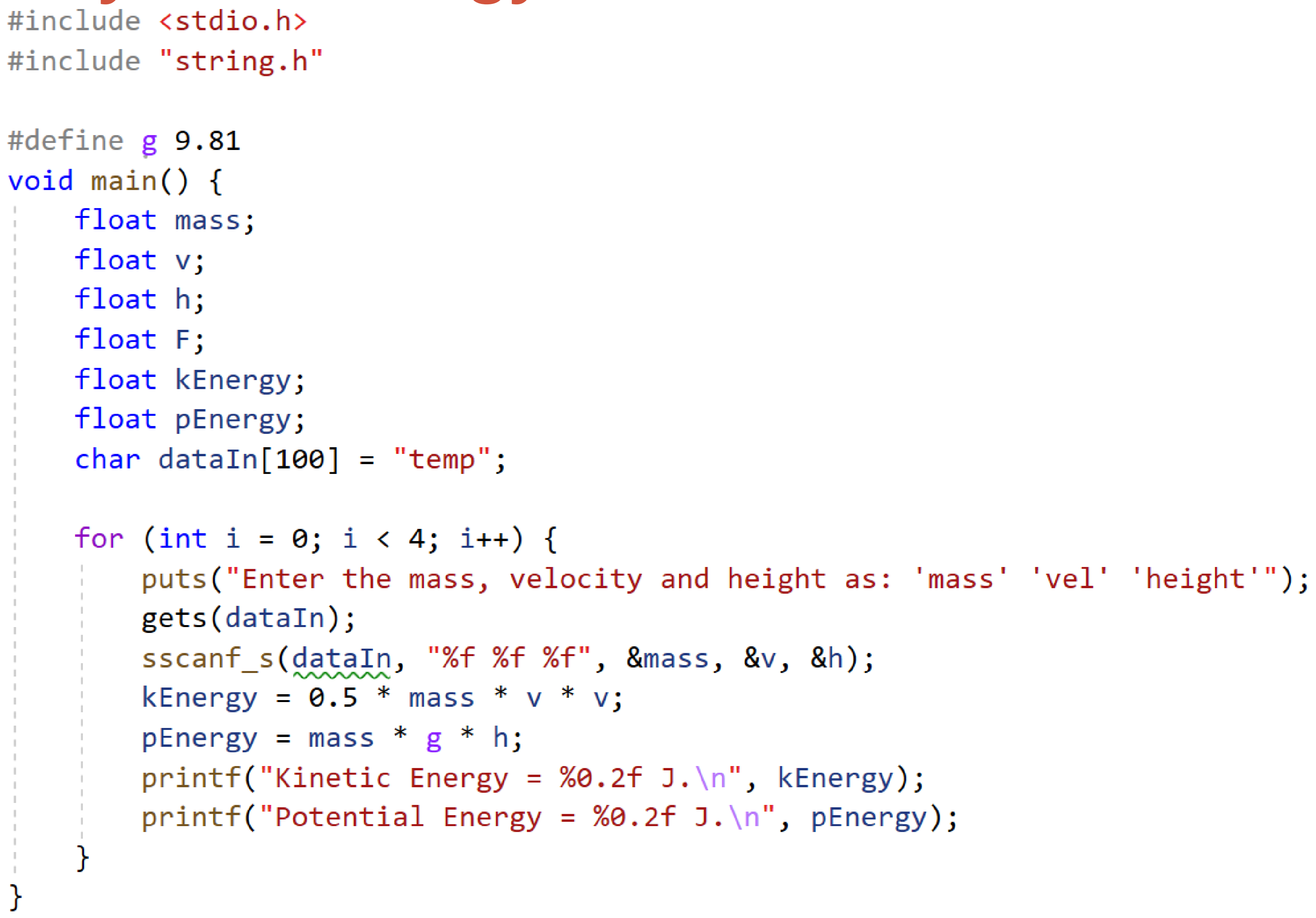
- You are writing software to calculate physics equations. Write a program that:
  - Defines acceleration due to gravity as a constant.
  - Reads in 4 measurements of the following:
    - Mass (m), velocity (v) and height (h).
  - Calculate the kinetic (KE) and potential (PE) energy of the 4 objects.

  - **Note**:
    - $KE = 0.5\ m\ v^2$
    - $PE = m\ g\ h$
    - $g = 9.81\ m/s^2$

Physics Energy Calculator

- Go to C program solution.

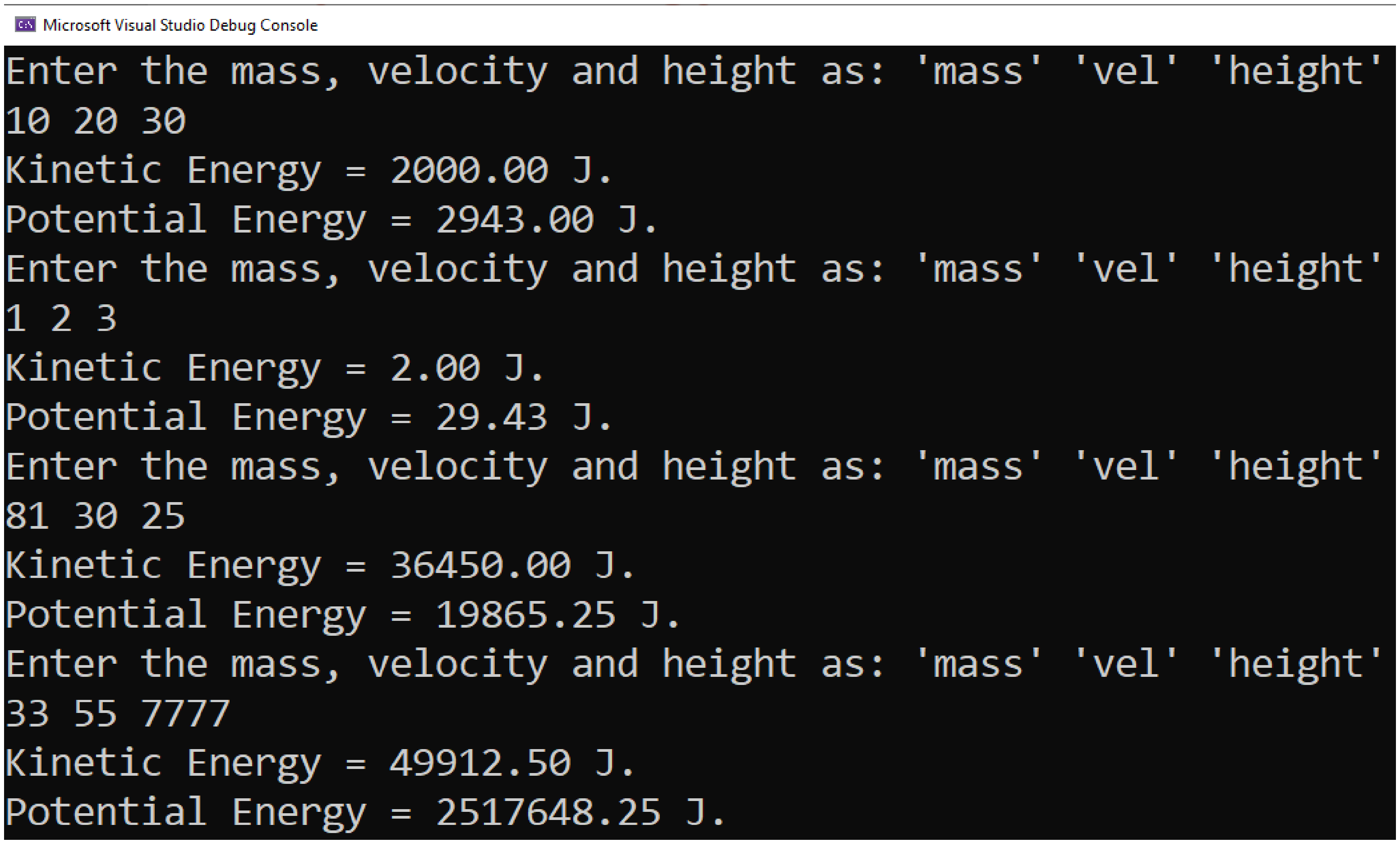
Physics Energy Calculator



```c
#include <stdio.h>
#include "string.h"

#define g 9.81
void main() {
    float mass;
    float v;
    float h;
    float F;
    float kEnergy;
    float pEnergy;
    char dataIn[100] = "temp";

    for (int i = 0; i < 4; i++) {
        puts("Enter the mass, velocity and height as: 'mass' 'vel' 'height'");
        gets(dataIn);
        sscanf_s(dataIn, "%f %f %f", &mass, &v, &h);
        kEnergy = 0.5 * mass * v * v;
        pEnergy = mass * g * h;
        printf("Kinetic Energy = %0.2f J.\n", kEnergy);
        printf("Potential Energy = %0.2f J.\n", pEnergy);
    }
}
```

Physics Energy Calculator

- C Program Output:


Microsoft Visual Studio Debug Console

```
Enter the mass, velocity and height as: 'mass' 'vel' 'height'
10 20 30
Kinetic Energy = 2000.00 J.
Potential Energy = 2943.00 J.
Enter the mass, velocity and height as: 'mass' 'vel' 'height'
1 2 3
Kinetic Energy = 2.00 J.
Potential Energy = 29.43 J.
Enter the mass, velocity and height as: 'mass' 'vel' 'height'
81 30 25
Kinetic Energy = 36450.00 J.
Potential Energy = 19865.25 J.
Enter the mass, velocity and height as: 'mass' 'vel' 'height'
33 55 7777
Kinetic Energy = 49912.50 J.
Potential Energy = 2517648.25 J.
```

PROGRAMMING

CT103

Week 7b

Sign in on Blackboard

- Please sign in on blackboard for CT103.
- I will leave the sign in option open for an hour after the lecture.

Lab Solution Submissions

- Internet access on Eduroam is available again.

- From **week 8** lab onwards, you will have to submit your lab assignments by the end of the lab.
  - i.e. 6pm Tuesday.

- You will not have until midnight to finish your assignment.

- If it is submitted after 6pm, it will be marked as late and you may receive a penalty.

Lecture Content

- Last lecture (Week 7a):
  - Constants
  - Puts
  - Gets
  - Sscanf_s
  - Example C program

- Today’s lecture (Week 7b):
  - Testing characters
  - Character mapping
  - Arrays of strings
  - Example C program

TESTING CHARACTERS

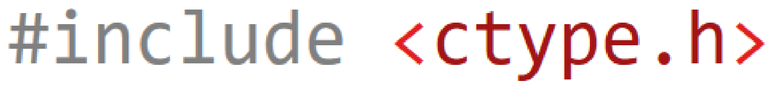
Character Tests

- There are a number of character tests that we can use that are useful for analyzing characters.
- We already saw that upper and lower case letters are different in C.
- There are tests that we can do to check for upper/lower case letters.

Ctype.h Library

- We will first need to use the ctype.h library.

- This is a library with functions that are useful for testing and mapping characters.



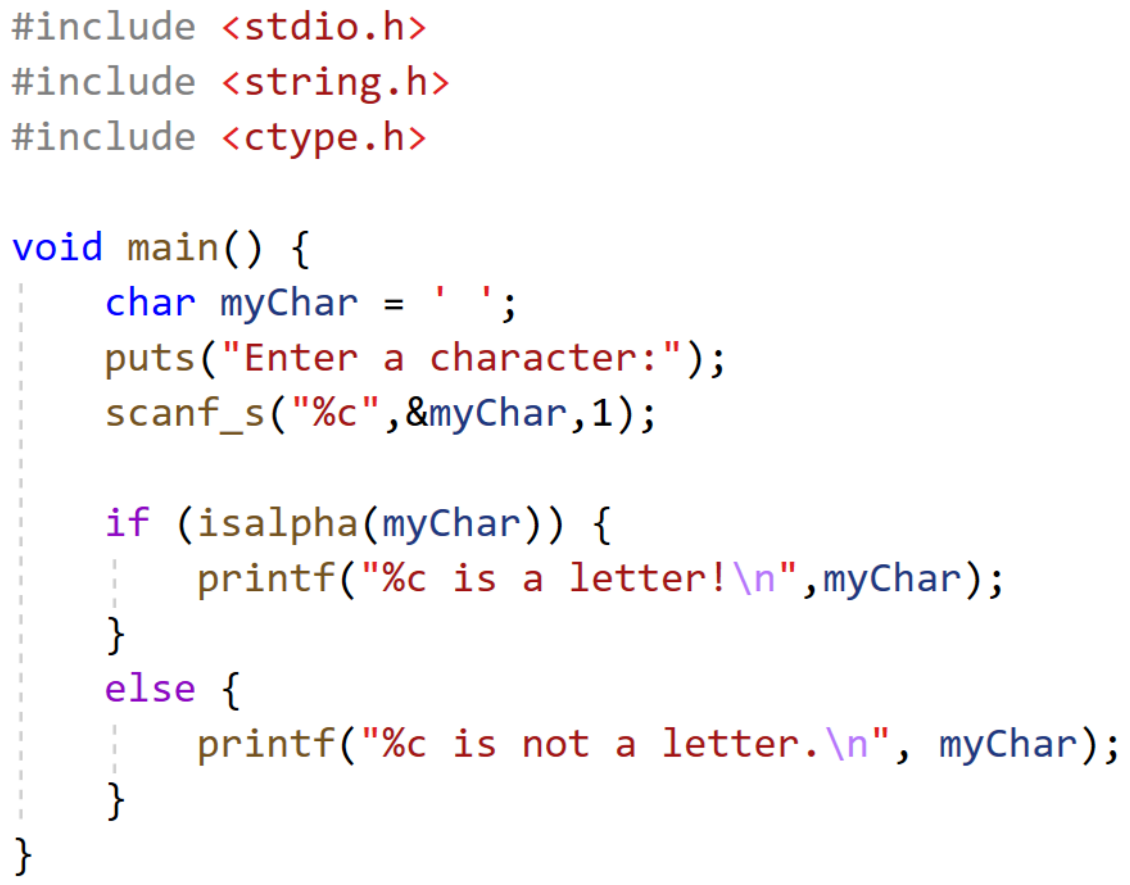
```c
#include <ctype.h>
```

Character Tests

- Some useful character testing functions:
  - isalpha
  - isdigit
  - isupper
  - islower
  - isspace

isalpha

- The isalpha function is useful for checking if the character is alphabetic.

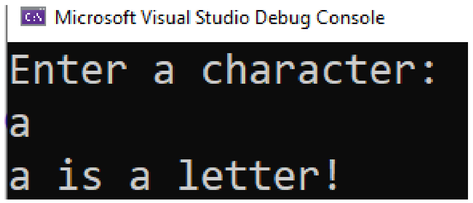


```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void main() {
    char myChar = ' ';
    puts("Enter a character:");
    scanf_s("%c",&myChar,1);

    if (isalpha(myChar)) {
        printf("%c is a letter!\n",myChar);
    }
    else {
        printf("%c is not a letter.\n", myChar);
    }
}
```

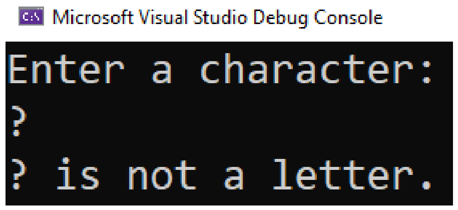
isalpha

- The isalpha function is useful for checking if the character is alphabetic.

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void main() {
    char myChar = ' ';
    puts("Enter a character:");
    scanf_s("%c",&myChar,1);

    if (isalpha(myChar)) {
        printf("%c is a letter!\n",myChar);
    }
    else {
        printf("%c is not a letter.\n", myChar);
    }
}
```


Microsoft Visual Studio Debug Console

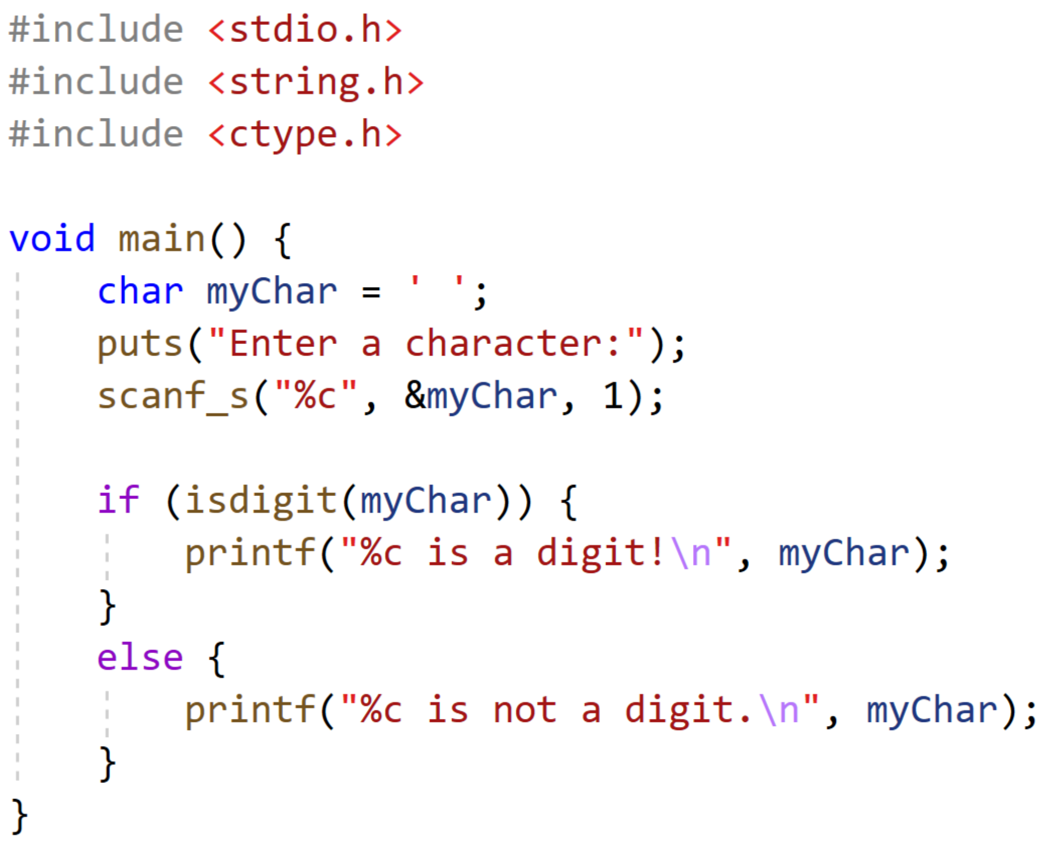
```
Enter a character:
a
a is a letter!
```


Microsoft Visual Studio Debug Console

```
Enter a character:
?
? is not a letter.
```

isdigit

- The isdigit function is useful for checking if the character is a digit.

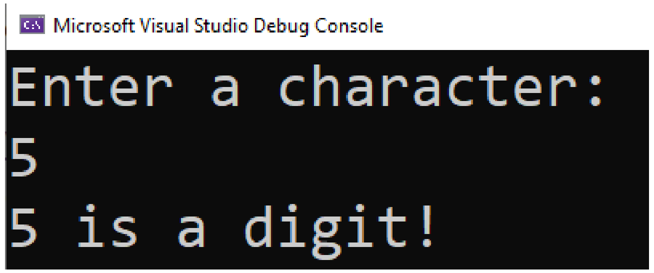


```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void main() {
    char myChar = ' ';
    puts("Enter a character:");
    scanf_s("%c", &myChar, 1);

    if (isdigit(myChar)) {
        printf("%c is a digit!\n", myChar);
    }
    else {
        printf("%c is not a digit.\n", myChar);
    }
}
```

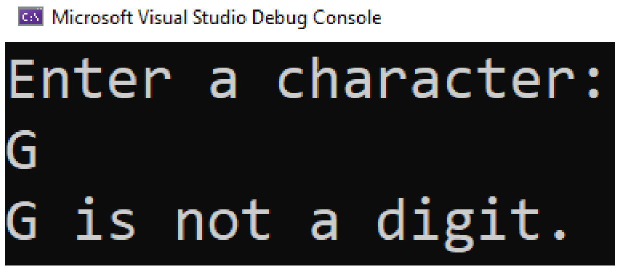
isdigit

- The isdigt function is useful for checking if the character is a digit.

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void main() {
    char myChar = ' ';
    puts("Enter a character:");
    scanf_s("%c", &myChar, 1);

    if (isdigit(myChar)) {
        printf("%c is a digit!\n", myChar);
    }
    else {
        printf("%c is not a digit.\n", myChar);
    }
}
```



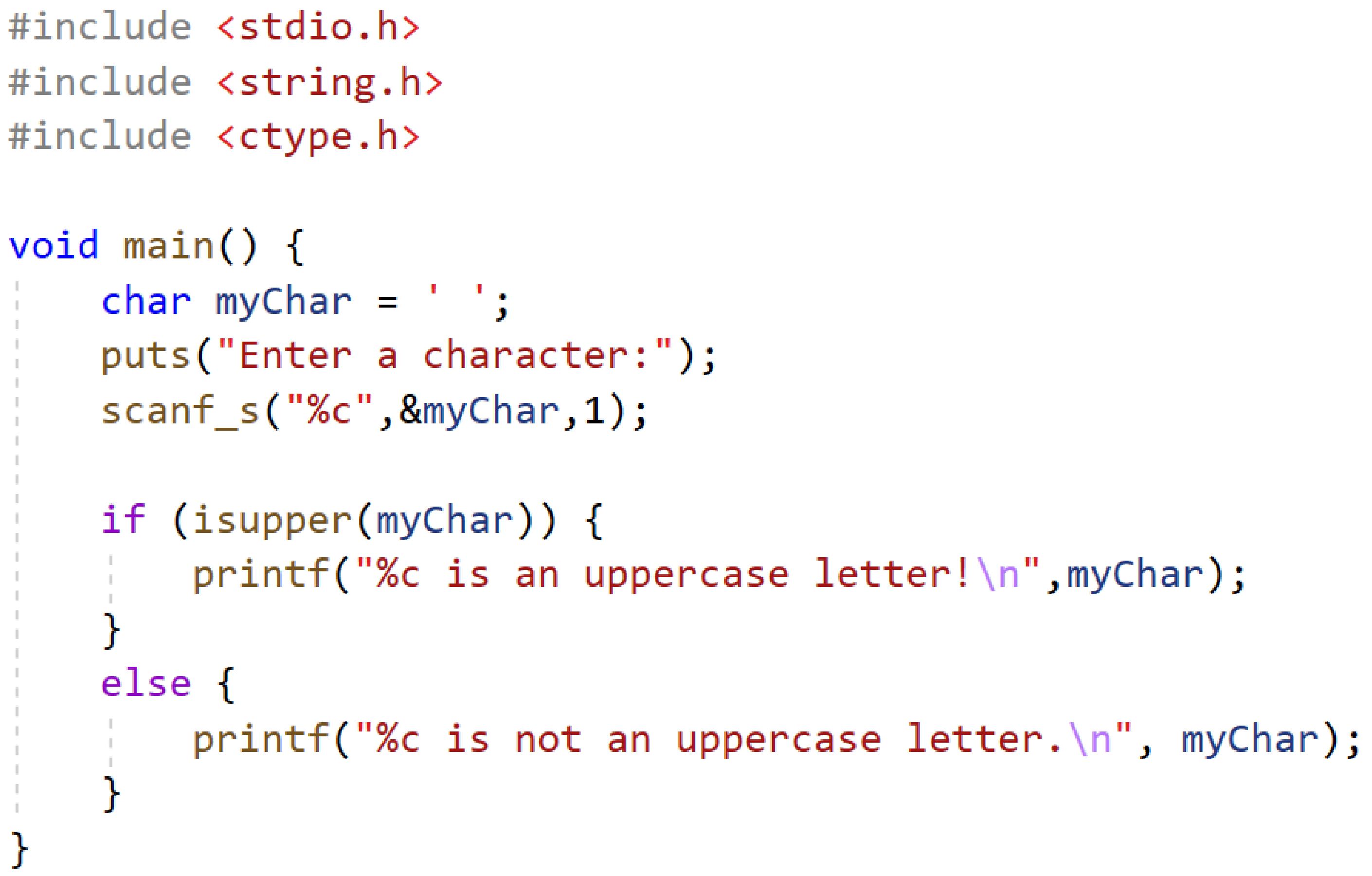
```
Enter a character:
5
5 is a digit!
```



```
Enter a character:
G
G is not a digit.
```

isupper

- The isupper function is useful for checking if the character is an uppercase letter.

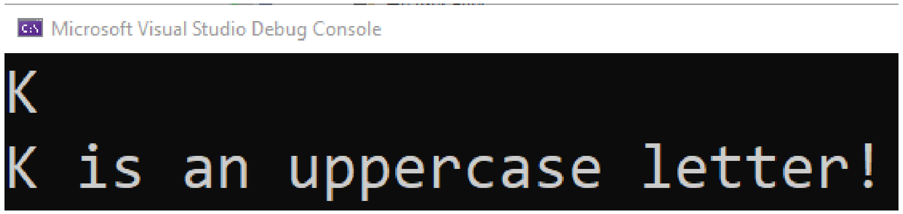


```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void main() {
    char myChar = ' ';
    puts("Enter a character:");
    scanf_s("%c",&myChar,1);

    if (isupper(myChar)) {
        printf("%c is an uppercase letter!\n",myChar);
    }
    else {
        printf("%c is not an uppercase letter.\n", myChar);
    }
}
```

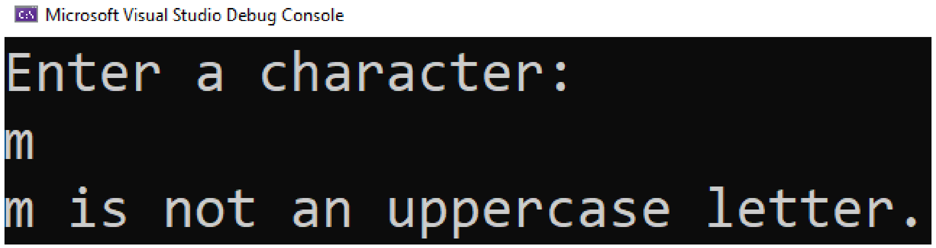
isupper

- The isupper function is useful for checking if the character is an uppercase letter.

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void main() {
    char myChar = ' ';
    puts("Enter a character:");
    scanf_s("%c",&myChar,1);

    if (isupper(myChar)) {
        printf("%c is an uppercase letter!\n",myChar);
    }
    else {
        printf("%c is not an uppercase letter.\n", myChar);
    }
}
```



```
K
K is an uppercase letter!
```



```
Enter a character:
m
m is not an uppercase letter.
```

islower

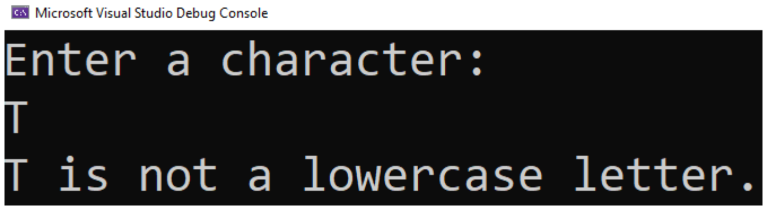
- The islower function is useful for checking if the character is a lowercase letter.

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void main() {
    char myChar = ' ';
    puts("Enter a character:");
    scanf_s("%c",&myChar,1);

    if (islower(myChar)) {
        printf("%c is a lowercase letter!\n",myChar);
    }
    else {
        printf("%c is not a lowercase letter.\n", myChar);
    }
}
```

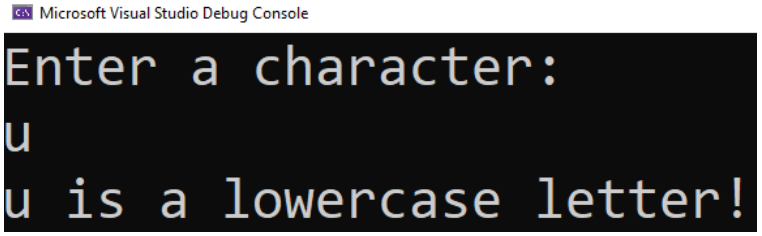
islower

- The islower function is useful for checking if the character is a lowercase letter.

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void main() {
    char myChar = ' ';
    puts("Enter a character:");
    scanf_s("%c",&myChar,1);

    if (islower(myChar)) {
        printf("%c is a lowercase letter!\n",myChar);
    }
    else {
        printf("%c is not a lowercase letter.\n", myChar);
    }
}
```



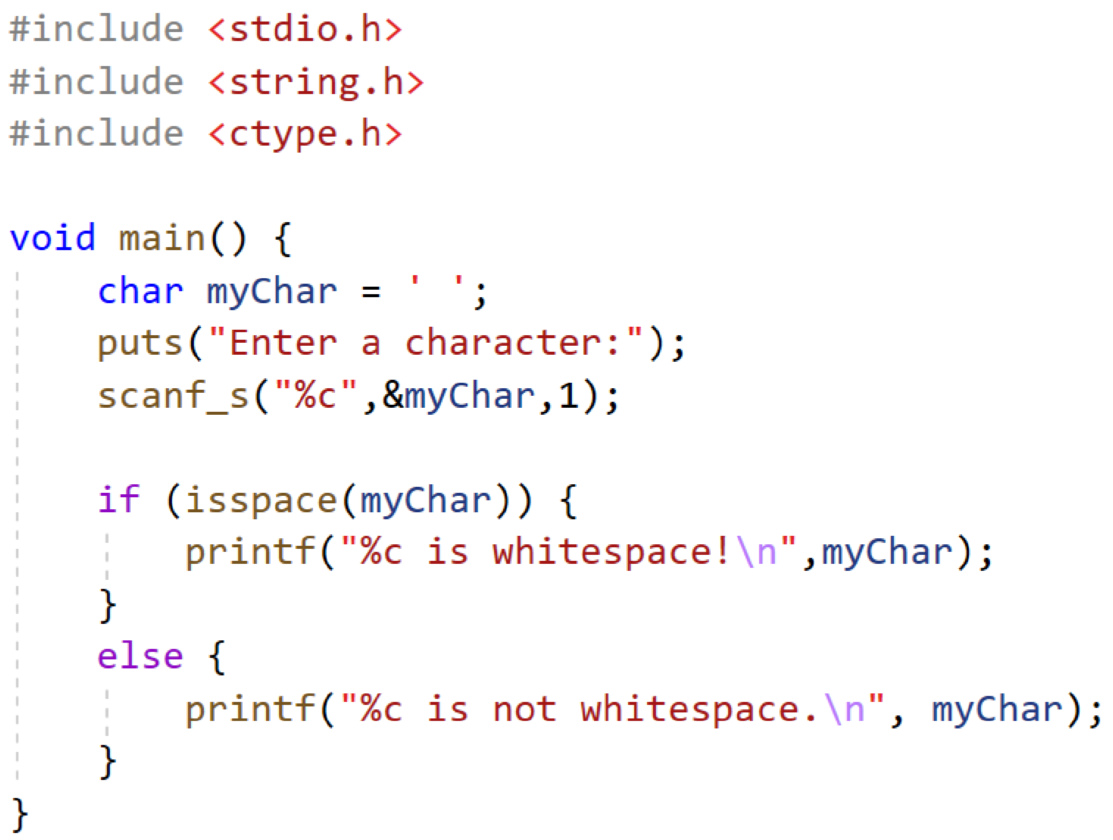
```
Enter a character:
T
T is not a lowercase letter.
```



```
Enter a character:
u
u is a lowercase letter!
```

isspace

- The isspace function is useful for checking if the character is whitespace.

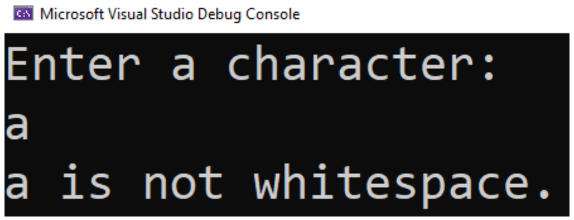


```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void main() {
    char myChar = ' ';
    puts("Enter a character:");
    scanf_s("%c",&myChar,1);

    if (isspace(myChar)) {
        printf("%c is whitespace!\n",myChar);
    }
    else {
        printf("%c is not whitespace.\n", myChar);
    }
}
```

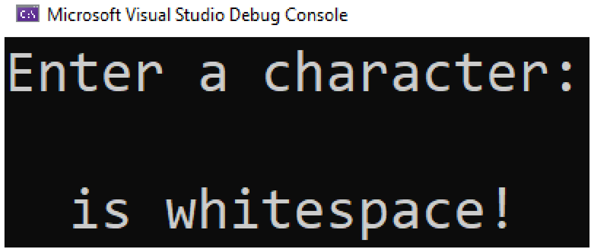
isspace

- The isspace function is useful for checking if the character is whitespace.

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void main() {
    char myChar = ' ';
    puts("Enter a character:");
    scanf_s("%c",&myChar,1);

    if (isspace(myChar)) {
        printf("%c is whitespace!\n",myChar);
    }
    else {
        printf("%c is not whitespace.\n", myChar);
    }
}
```



```
Enter a character:
a
a is not whitespace.
```



```
Enter a character:

  is whitespace!
```

CHARACTER MAPPING

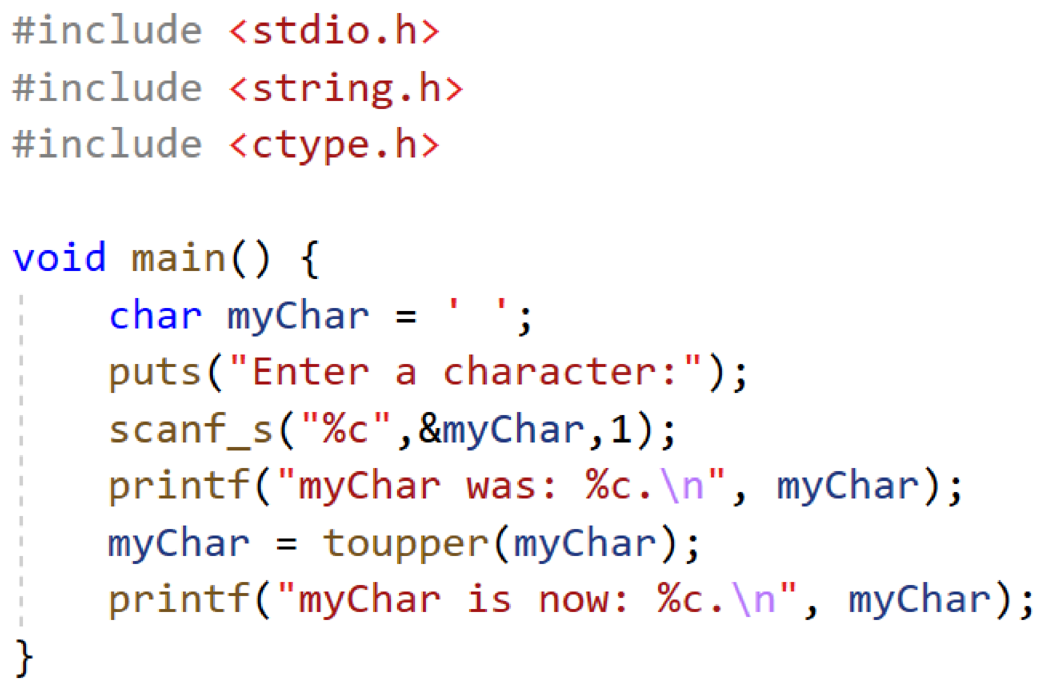
Character Mapping

- We have looked at useful character testing functions:
  - isalpha
  - isdigit
  - isupper
  - Islower
  - Isspace

- Very useful functions to convert character case:
  - **toupper**
  - **tolower**

toupper

- The toupper function is very useful as it allows us to convert letter to uppercase.

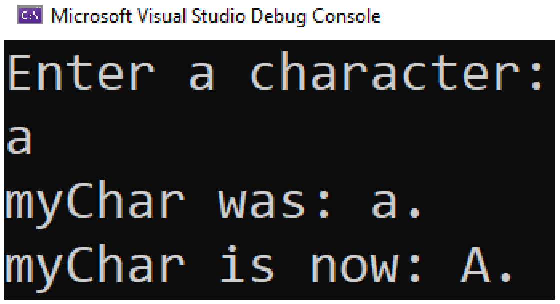


```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void main() {
    char myChar = ' ';
    puts("Enter a character:");
    scanf_s("%c",&myChar,1);
    printf("myChar was: %c.\n", myChar);
    myChar = toupper(myChar);
    printf("myChar is now: %c.\n", myChar);
}
```

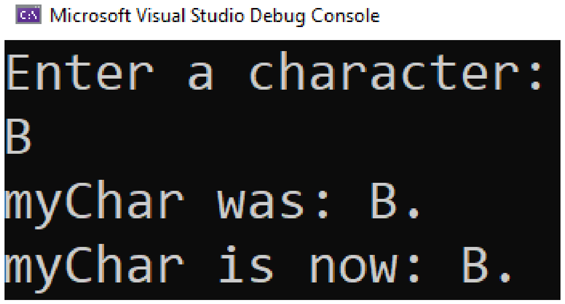
toupper

- The toupper function is very useful as it allows us to convert letter to uppercase.

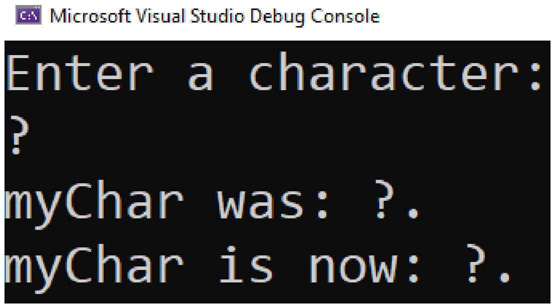
```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void main() {
    char myChar = ' ';
    puts("Enter a character:");
    scanf_s("%c",&myChar,1);
    printf("myChar was: %c.\n", myChar);
    myChar = toupper(myChar);
    printf("myChar is now: %c.\n", myChar);
}
```


Microsoft Visual Studio Debug Console

```
Enter a character:
a
myChar was: a.
myChar is now: A.
```


Microsoft Visual Studio Debug Console

```
Enter a character:
B
myChar was: B.
myChar is now: B.
```

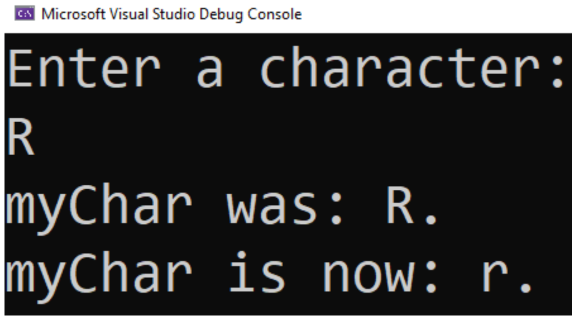

Microsoft Visual Studio Debug Console

```
Enter a character:
?
myChar was: ?.
myChar is now: ?.
```

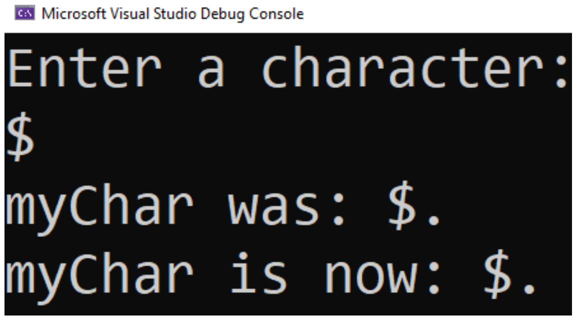
tolower

- The tolower function is very useful as it allows us to convert letter to lowercase.

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void main() {
    char myChar = ' ';
    puts("Enter a character:");
    scanf_s("%c",&myChar,1);
    printf("myChar was: %c.\n", myChar);
    myChar = tolower(myChar);
    printf("myChar is now: %c.\n", myChar);
}
```



```
Enter a character:
R
myChar was: R.
myChar is now: r.
```

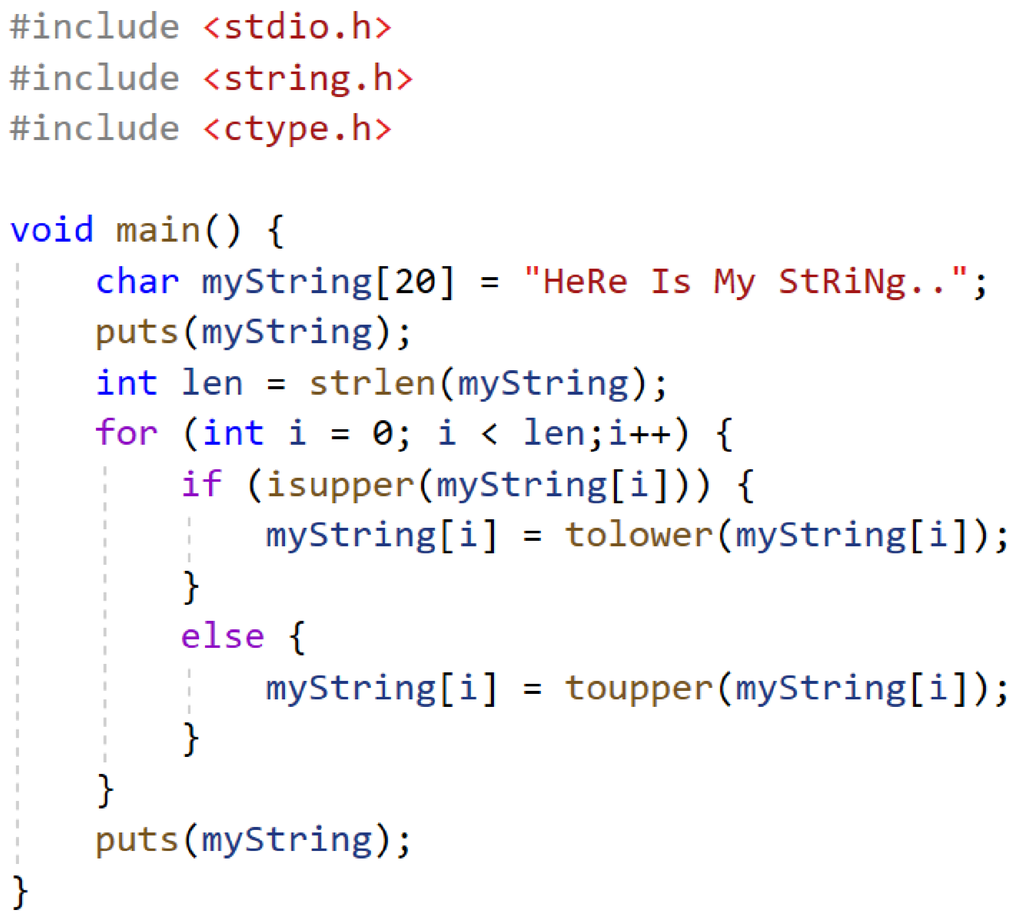


```
Enter a character:
$
myChar was: $.
myChar is now: $.
```

USING CHARACTER MAPPING

Example C Program

- See C program using character testing and mapping:

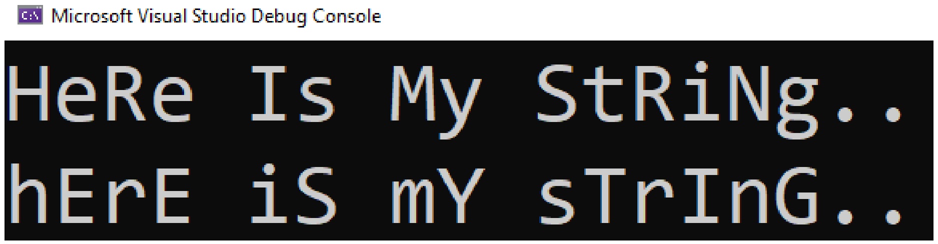


```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void main() {
    char myString[20] = "HeRe Is My StRiNg..";
    puts(myString);
    int len = strlen(myString);
    for (int i = 0; i < len;i++) {
        if (isupper(myString[i])) {
            myString[i] = tolower(myString[i]);
        }
        else {
            myString[i] = toupper(myString[i]);
        }
    }
    puts(myString);
}
```

Example C Program

- We can swap upper and lower case characters:

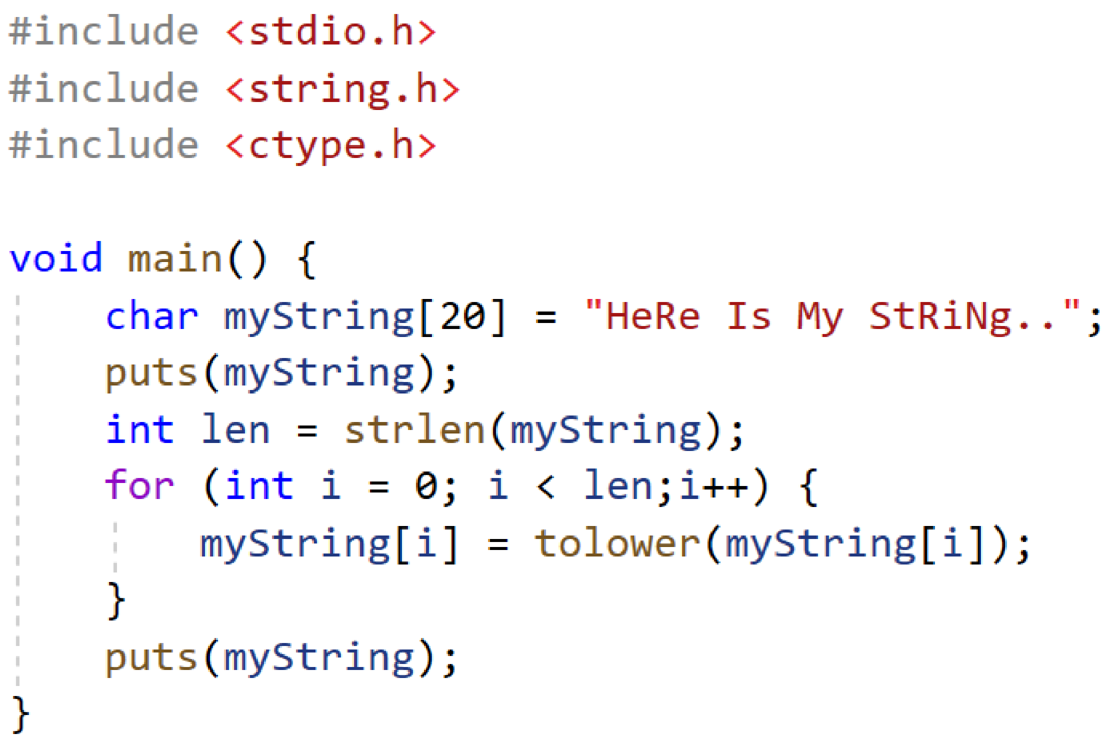
```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void main() {
    char myString[20] = "HeRe Is My StRiNg..";
    puts(myString);
    int len = strlen(myString);
    for (int i = 0; i < len;i++) {
        if (isupper(myString[i])) {
            myString[i] = tolower(myString[i]);
        }
        else {
            myString[i] = toupper(myString[i]);
        }
    }
    puts(myString);
}
```



```
HeRe Is My StRiNg..
hErE iS mY sTrInG..
```

Example C Program

- See C program using character testing and mapping:

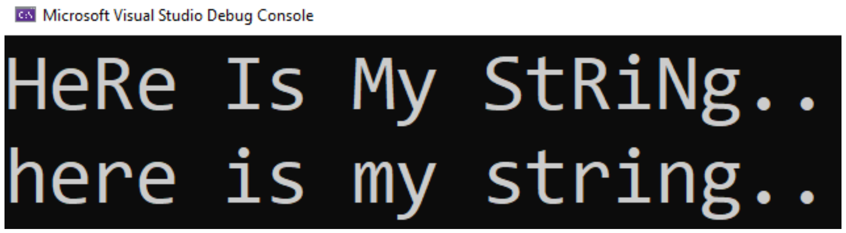


```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void main() {
    char myString[20] = "HeRe Is My StRiNg..";
    puts(myString);
    int len = strlen(myString);
    for (int i = 0; i < len;i++) {
        myString[i] = tolower(myString[i]);
    }
    puts(myString);
}
```

Example C Program

- Convert all characters to lower case:

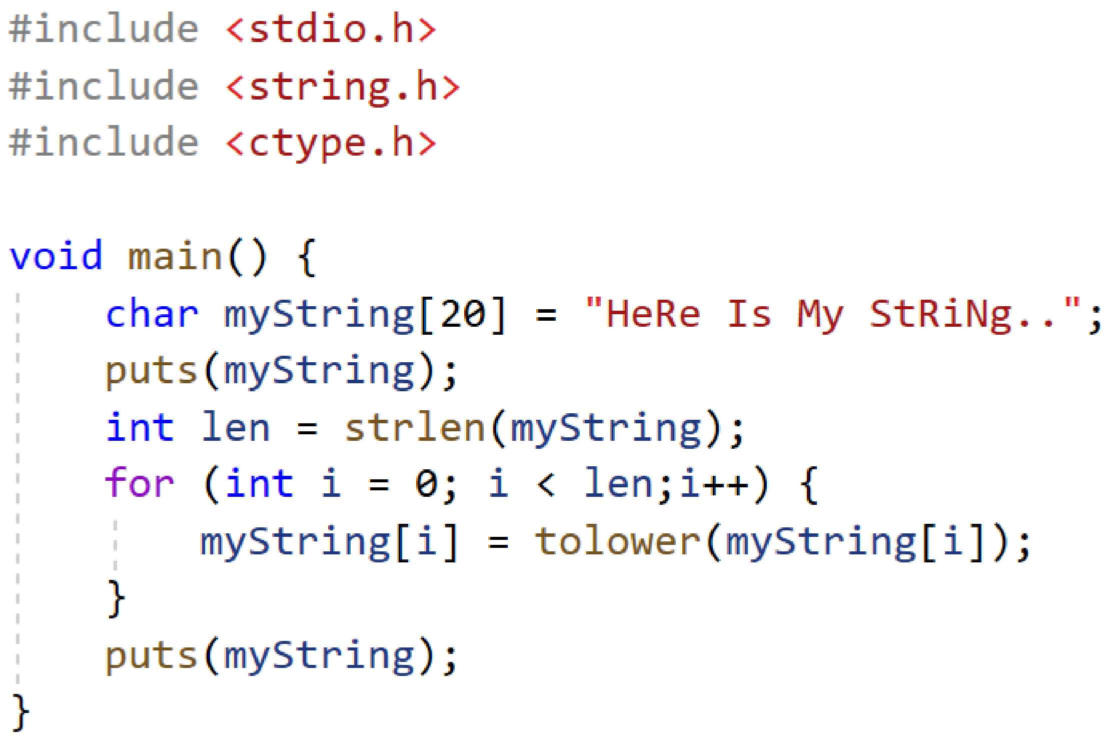
```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void main() {
    char myString[20] = "HeRe Is My StRiNg..";
    puts(myString);
    int len = strlen(myString);
    for (int i = 0; i < len;i++) {
        myString[i] = tolower(myString[i]);
    }
    puts(myString);
}
```


Microsoft Visual Studio Debug Console

```
HeRe Is My StRiNg..
here is my string..
```

Try it yourself

- Try change the program below to convert all characters to uppercase.

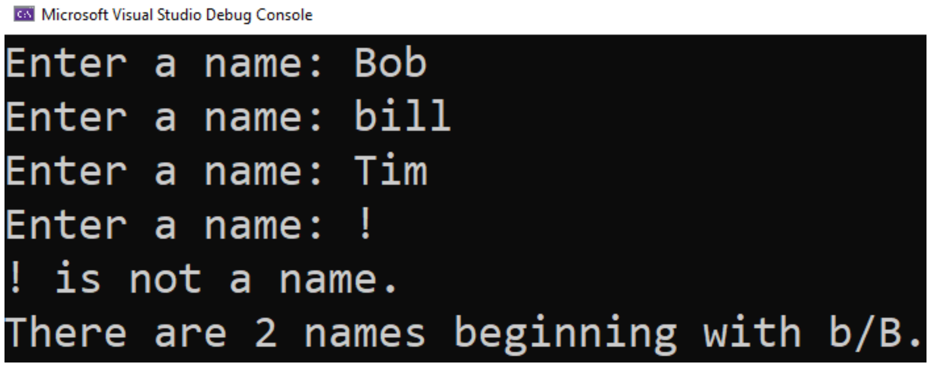


```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void main() {
    char myString[20] = "HeRe Is My StRiNg..";
    puts(myString);
    int len = strlen(myString);
    for (int i = 0; i < len;i++) {
        myString[i] = tolower(myString[i]);
    }
    puts(myString);
}
```

Name Scanner Program

- Remember our name scanner program from a few weeks ago?

```c
#include <stdio.h>
#include <string.h>
void main()
{
    int count = 0;
    char newName[10] = "Alex";
    while (!strncmp(newName, "!", 1) == 0) {
        printf("Enter a name: ");
        scanf_s("%[^\n]%*c", newName, 10);
        if (newName[0]=='b'|| newName[0] == 'B') {
            count++;
        }
    }
    printf("%s is not a name.\n", newName);
    printf("There are %d names beginning with b/B.", count);
}
```

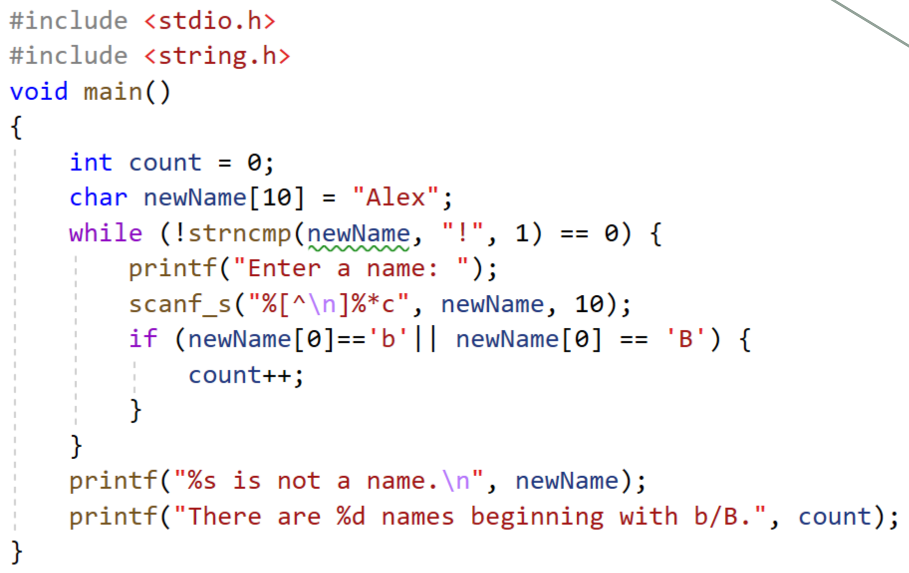

Microsoft Visual Studio Debug Console

```
Enter a name: Bob
Enter a name: bill
Enter a name: Tim
Enter a name: !
! is not a name.
There are 2 names beginning with b/B.
```

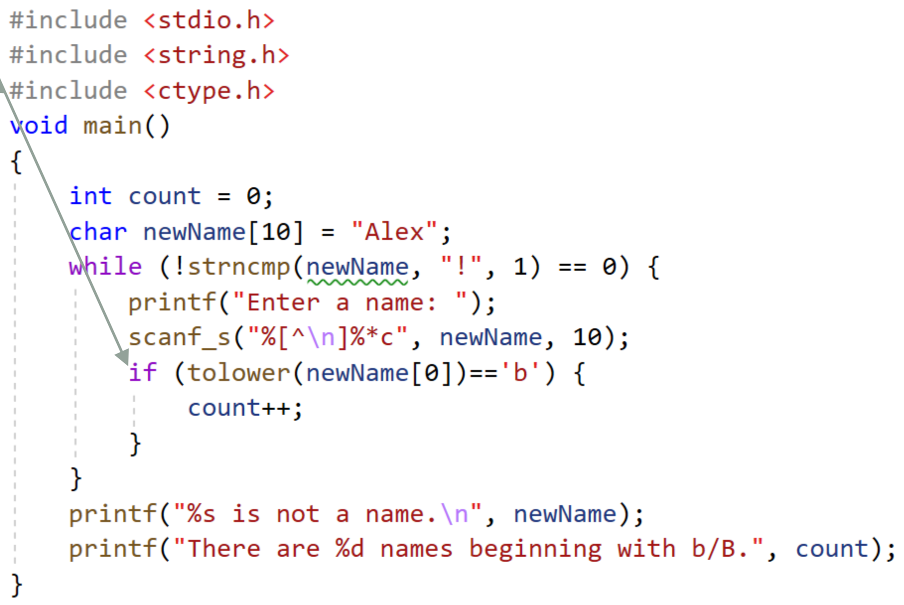
Name Scanner Program

- Could have used tolower when checking for the letter b.

Using ctype.h



```
#include <stdio.h>
#include <string.h>
void main()
{
    int count = 0;
    char newName[10] = "Alex";
    while (!strncmp(newName, "!", 1) == 0) {
        printf("Enter a name: ");
        scanf_s("%[^\n]%*c", newName, 10);
        if (newName[0]=='b'|| newName[0] == 'B') {
            count++;
        }
    }
    printf("%s is not a name.\n", newName);
    printf("There are %d names beginning with b/B.", count);
}
```



```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
void main()
{
    int count = 0;
    char newName[10] = "Alex";
    while (!strncmp(newName, "!", 1) == 0) {
        printf("Enter a name: ");
        scanf_s("%[^\n]%*c", newName, 10);
        if (tolower(newName[0])=='b') {
            count++;
        }
    }
    printf("%s is not a name.\n", newName);
    printf("There are %d names beginning with b/B.", count);
}
```

ARRAYS OF STRINGS

Arrays of Strings

- We have talked about arrays already.
- In C, Strings are arrays of characters.
- We also covered 2D arrays!
- Next we will discuss arrays of strings.

Arrays of Strings

- Often we need to process lists of strings, such as names.
- As with the other 2D arrays we have seen, we can create a 2D array of characters.
- Each row (the first index) is a different string.
- Each column is a character.

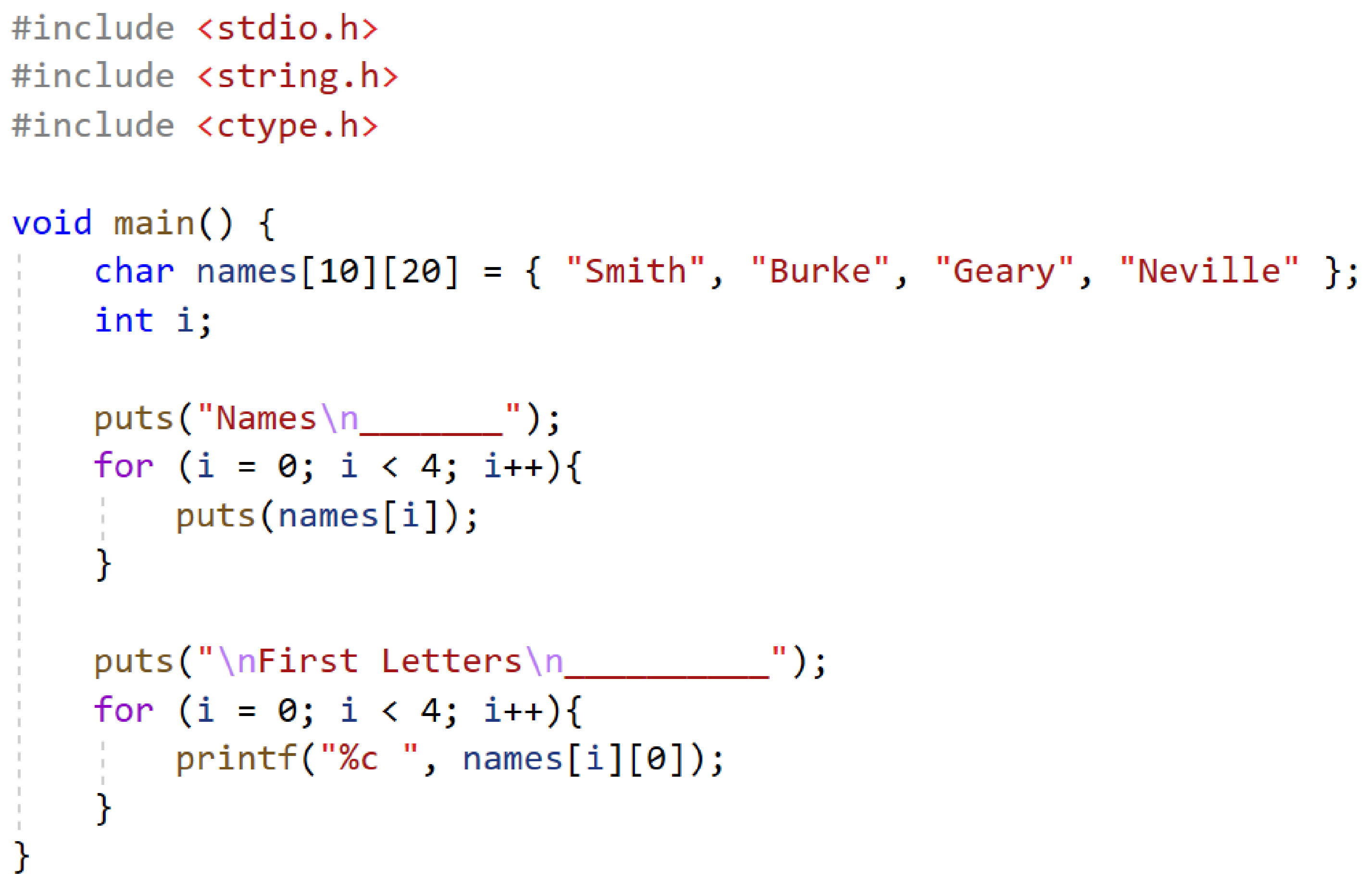
char names[][20]

- In the following example we create a list of names, called “names”!
- We can refer to each string using the first index
- So for example names[2] is “Geary”

	names[i][0]	names[i][1]	names[i][2]	names[i][3]	names[i][4]	names[i][5]	names[i][6]	names[i][7]
names[0]	S	m	i	t	h	\0		
names[1]	B	u	r	k	e	\0		
names[2]	G	e	a	r	y	\0		
names[3]	N	e	v	i	l	l	e	\0

Arrays of Strings Example

- Array of Strings in C:

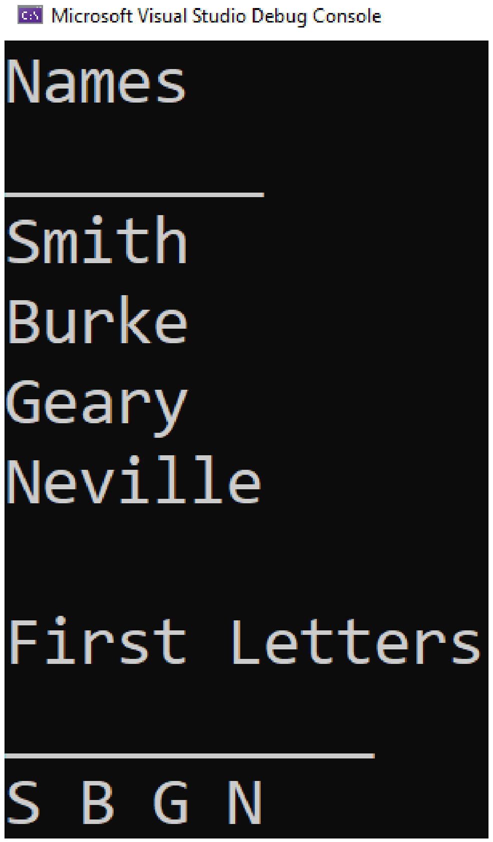


```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void main() {
    char names[10][20] = { "Smith", "Burke", "Geary", "Neville" };
    int i;

    puts("Names\n_______");
    for (i = 0; i < 4; i++){
        puts(names[i]);
    }

    puts("\nFirst Letters\n__________");
    for (i = 0; i < 4; i++){
        printf("%c ", names[i][0]);
    }
}
```

Arrays of Strings Example

- Array of Strings in C:

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void main() {
    char names[10][20] = { "Smith", "Burke", "Geary", "Neville" };
    int i;

    puts("Names\n_______");
    for (i = 0; i < 4; i++){
        puts(names[i]);
    }

    puts("\nFirst Letters\n__________");
    for (i = 0; i < 4; i++){
        printf("%c ", names[i][0]);
    }
}
```



Microsoft Visual Studio Debug Console

```
Names
_______
Smith
Burke
Geary
Neville

First Letters
__________
S B G N
```

EXAMPLE PROBLEMS

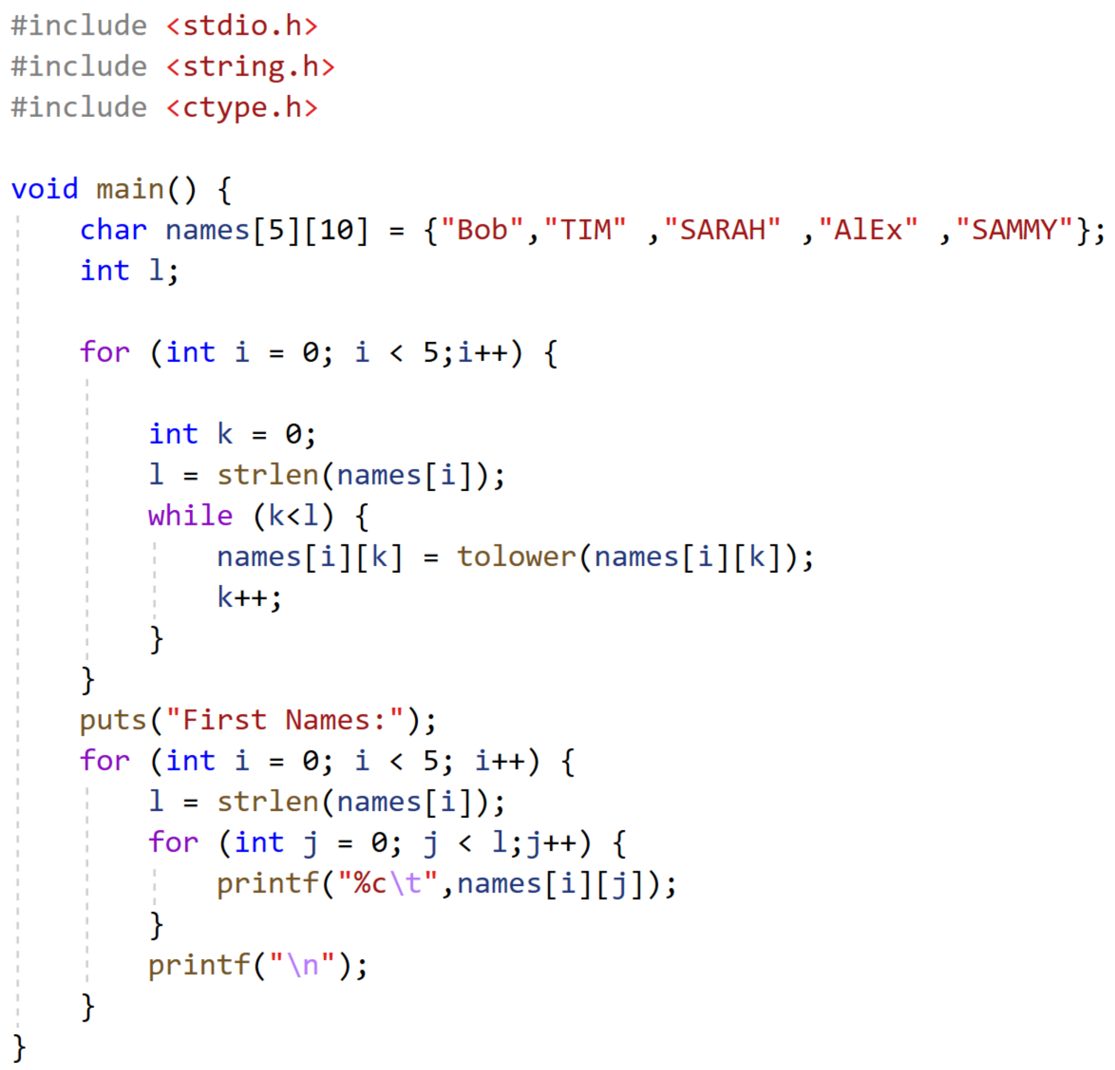
Names Processor

- You are writing software to process a list of names:
  - You have an array of names (Strings): "Bob","TIM" ,"SARAH" ,"AlEx" ,"SAMMY"
  - Loop through these names and convert all characters to lower case.
  - Display the new array of strings to the screen. Separate each character by a tab when printing each string.

Names Processor

- Go to C program solution.

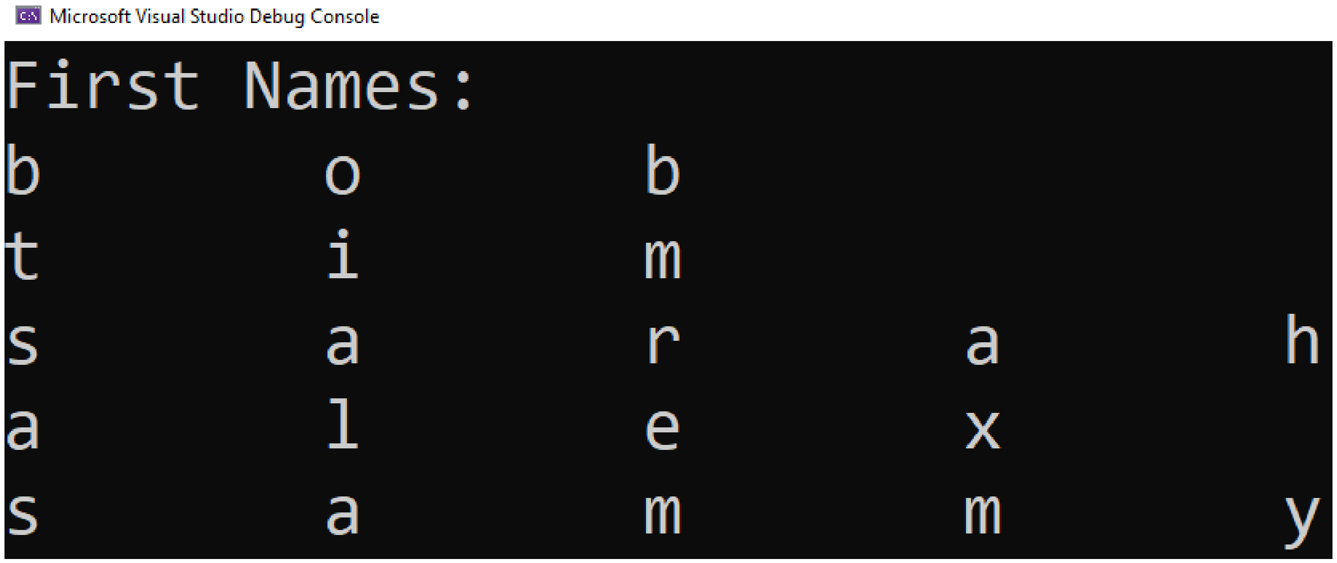
Names Processor



```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void main() {
    char names[5][10] = {"Bob","TIM" ,"SARAH" ,"AlEx" ,"SAMMY"};
    int l;

    for (int i = 0; i < 5;i++) {

        int k = 0;
        l = strlen(names[i]);
        while (k<l) {
            names[i][k] = tolower(names[i][k]);
            k++;
        }
    }
    puts("First Names:");
    for (int i = 0; i < 5; i++) {
        l = strlen(names[i]);
        for (int j = 0; j < l;j++) {
            printf("%c\t",names[i][j]);
        }
        printf("\n");
    }
}
```

Names Processor

- C Program Output:



```
First Names:
b       o       b
t       i       m
s       a       r       a       h
a       l       e       x
s       a       m       m       y
```

PROGRAMMING

CT103

Week 8a

Sign in on Blackboard

- Please sign in on blackboard for CT103.
- I will leave the sign in option open for an hour after the lecture.

Lecture Content

- Last lecture (Week 7b):
  - Testing characters
  - Character mapping
  - Arrays of strings
  - Example C program

- Today’s lecture (Week 8a):
  - Functions
  - Writing functions
  - Functions in C
  - Example C program

FUNCTIONS

Functions

- What is a function?

- **Definition**: A function is a piece of code that can be called whenever we need to execute that code.

Functions

- What functions have we seen so far?

- We have seen many functions in C:
  - strlen()
  - isalpha()
  - isdigit()
  - isupper()
  - islower()
  - isspace()

- These are all examples of pieces of code that we can call in our program to achieve some task.

Functions

- What is the point of functions?

- Benefits:
  - Functions allow us to reuse code, therefore avoid repetition.
  - More readable programs.
  - Enables us to divide complex problems into simpler ones.
  - Easier to make changes to program.

WRITING FUNCTIONS

Function Template

- All functions have the following template:

```
type name (parameters){
        return;
}
```

- Type = data type returned by the function (can be void).
- Name = function name.
- Parameters = data we are giving to the function (can be empty).
- Return = what data is returned by the function (can also return nothing).

Function Type

- Like a variable, a function must have a ***type***
- It can be one of the standard variable types (char, double, float, int) or it can be void
- The type tells the compiler what *type* of variable the function returns
  - For example
  - getchar() returns a char
  - strcmp() returns an int

Why return anything?

- Functions can return a value or answer to some calculation or query
  - E.g. int getEmployeeAge(int employeeID);

- When we have the answer to the calculation or query, we will likely want to use this somewhere else in our program.

- In order to do this, we need to return that value from the function.

Naming functions

- Function names can’t contain spaces.

- You should give your function a helpful name that reflects what it does.

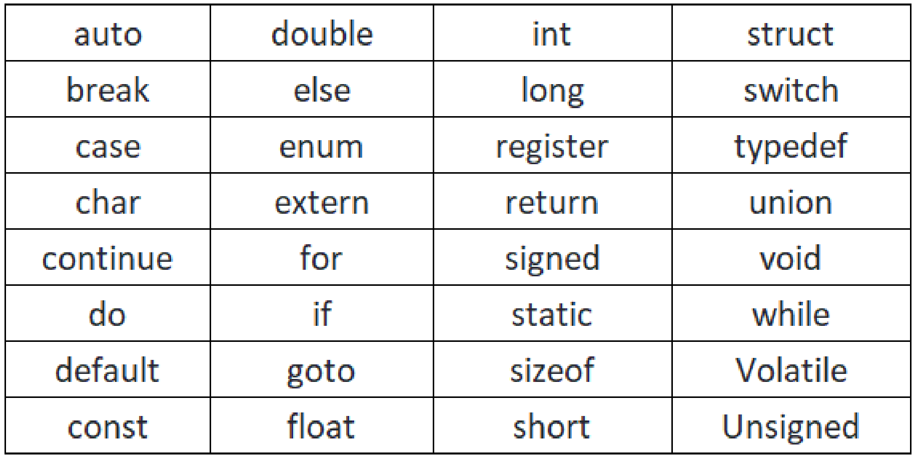
- Each functions is declared with parentheses “**()**” after the function name (even if it doesn’t don’t take any parameters), e.g. `void main().`

- You can’t name your function using a “reserved word”.

Reserved Words

- You can’t name your function using a “reserved word”.

- What is a reserved word?
  - There are 32 reserved words that have predefined meaning in C. You therefore can’t use these as variable names.



auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
continue	for	signed	void
do	if	static	while
default	goto	sizeof	Volatile
const	float	short	Unsigned

Function Prototypes

- Before your compiler will let you use a function, you have to give it a prototype.
- We have to do this before we call it, normally before the main() function, and after any #include or #define directive.
- The .h files (header files) contain the prototypes for C library functions we call.

Writing a Function

- We have already actually written a function:

```
void main() {
}
```

- Every time we wrote our C programs, we wrote our code inside of a function **main()**.

main()

- main() is the first function called when a program is executed
- When is finished the program exits
- Main() can return nothing or an integer

```
int main()
{
  return 0;
}

void main()
{
  return;
}
```

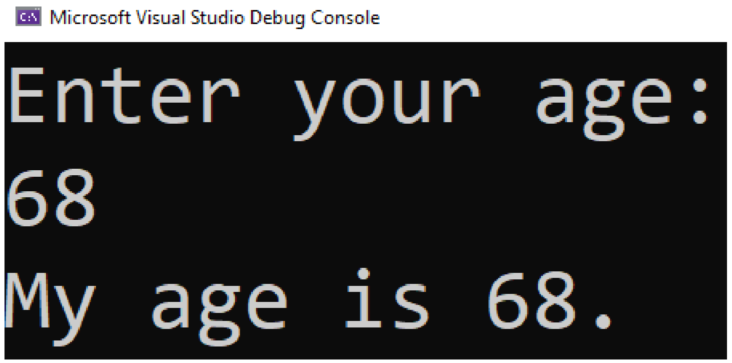
- so… the “type” of a function specifies what it returns (*void* if nothing)

FUNCTIONS IN C

C Program without Function EG1

- Simple C program that reads in an age and prints it to the screen.

```c
void main() {
    int age;
    puts("Enter your age:");
    scanf_s("%d", &age);
    printf("My age is %d.\n", age);
}
```

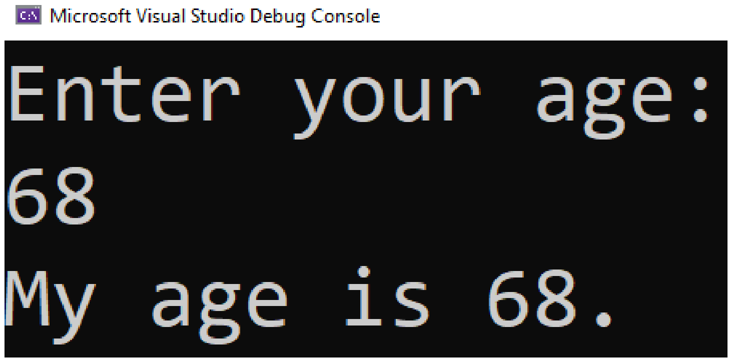


Microsoft Visual Studio Debug Console

```
Enter your age:
68
My age is 68.
```

C Program with Function EG1

- C program that creates a function to read in an age.
- Notice how this function does not read in any parameters.


Microsoft Visual Studio Debug Console

```
Enter your age:
68
My age is 68.
```

```c
#include <string.h>
#include <ctype.h>
#include <stdio.h>

int readAge();

void main() {
    int myAge = readAge();
    printf("My age is %d.\n", myAge);
}

int readAge() {
    int age;
    puts("Enter your age:");
    scanf_s("%d",&age);
    return age;
}
```

Function prototype

Main (we should be familiar with this one)

Function itself

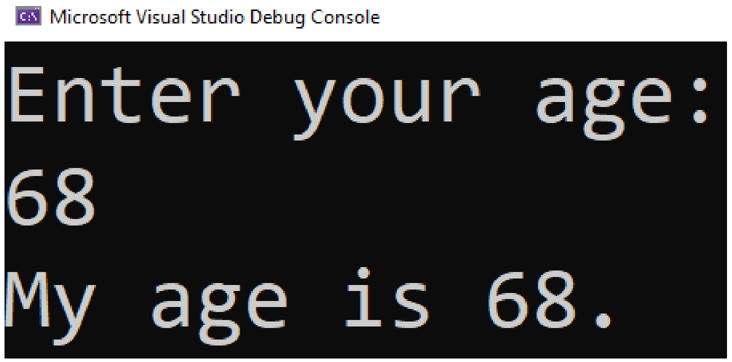
C Program EG1 Comparison

- These programs do the same thing.

No Function

Don't forget header files here.

```c
void main() {
    int age;
    puts("Enter your age:");
    scanf_s("%d", &age);
    printf("My age is %d.\n", age);
}
```



Microsoft Visual Studio Debug Console

```
Enter your age:
68
My age is 68.
```

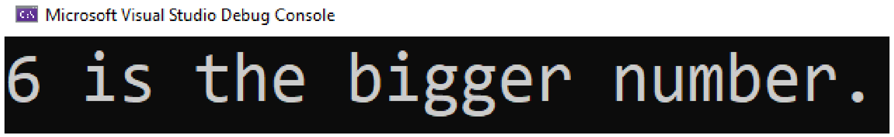
Using a Function

```c
#include <string.h>
#include <ctype.h>
#include <stdio.h>

int readAge();

void main() {
    int myAge = readAge();
    printf("My age is %d.\n", myAge);
}

int readAge() {
    int age;
    puts("Enter your age:");
    scanf_s("%d",&age);
    return age;
}
```

C Program without Function EG2

- Get the max number out of 2 numbers:

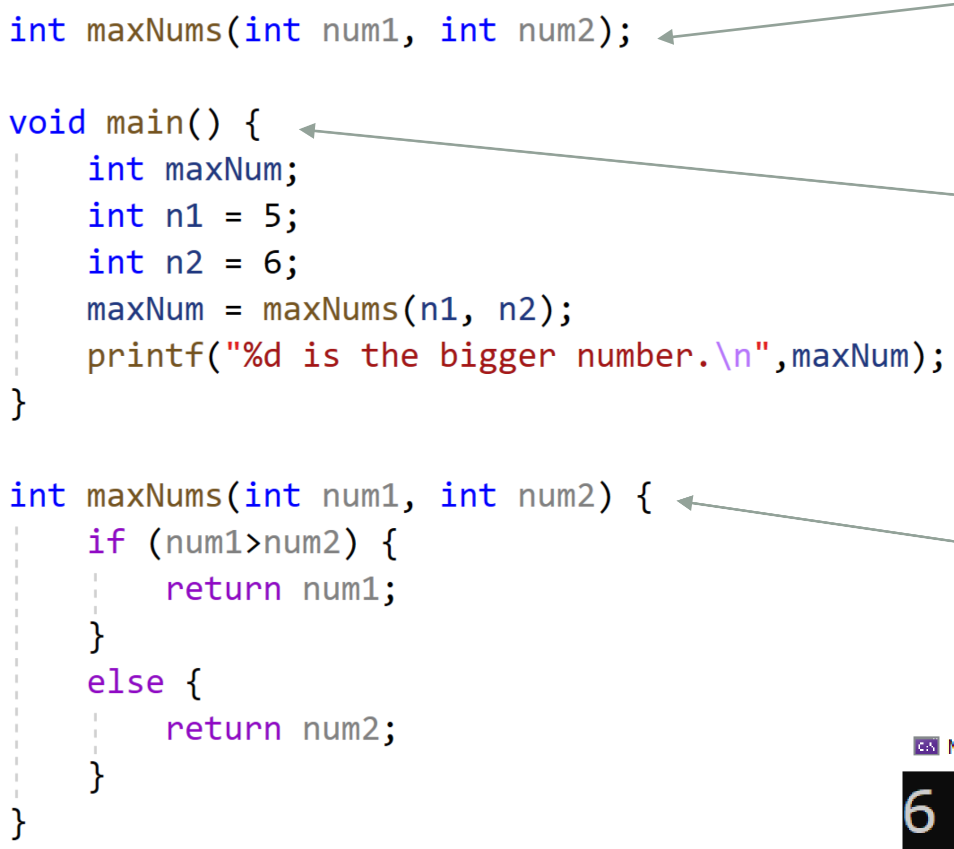
```c
#include <stdio.h>

void main() {
    int n1 = 1;
    int n2 = 6;
    int maxNum;
    if (n1 > n2) {
        maxNum = n1;
    }
    else {
        maxNum = n2;
    }
    printf("%d is the bigger number.\n",maxNum);
}
```


Microsoft Visual Studio Debug Console

```
6 is the bigger number.
```

C Program with Function EG2

- Get the max number out of 2 numbers:

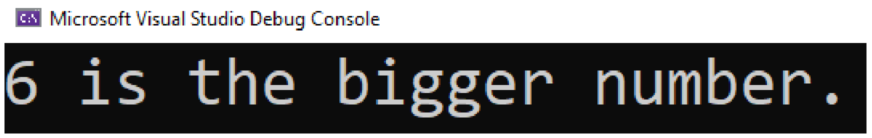


```c
int maxNums(int num1, int num2);

void main() {
    int maxNum;
    int n1 = 5;
    int n2 = 6;
    maxNum = maxNums(n1, n2);
    printf("%d is the bigger number.\n",maxNum);
}

int maxNums(int num1, int num2) {
    if (num1>num2) {
        return num1;
    }
    else {
        return num2;
    }
}
```

Function prototype

Main (we should be familiar with this one)

Function itself



Microsoft Visual Studio Debug Console

```
6 is the bigger number.
```

C Program EG2 Comparison

- If these programs do the same thing, why would you use functions? This program is much longer…

No Function

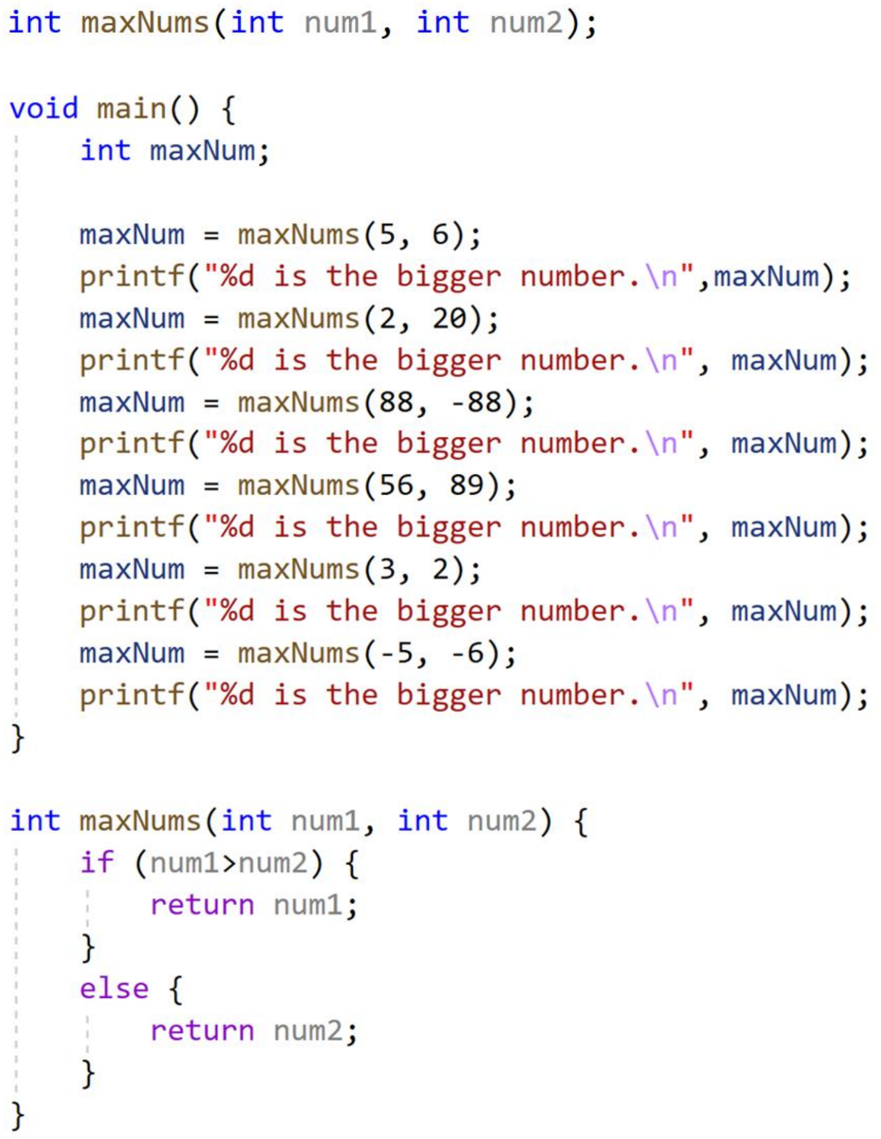
```c
#include <stdio.h>

void main() {
    int n1 = 1;
    int n2 = 6;
    int maxNum;
    if (n1 > n2) {
        maxNum = n1;
    }
    else {
        maxNum = n2;
    }
    printf("%d is the bigger number.\n",maxNum);
}
```

Using a Function

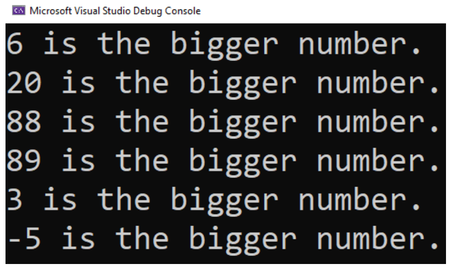
```c
int maxNums(int num1, int num2);

void main() {
    int maxNum;
    int n1 = 5;
    int n2 = 6;
    maxNum = maxNums(n1, n2);
    printf("%d is the bigger number.\n",maxNum);
}

int maxNums(int num1, int num2) {
    if (num1>num2) {
        return num1;
    }
    else {
        return num2;
    }
}
```

C Program with Function Cont.

- Well what if I wanted to do more than 1 comparison?
- If I use a function, I can simply call the function again.
- This is much more scalable than not using a function.



```c
int maxNums(int num1, int num2);

void main() {
    int maxNum;

    maxNum = maxNums(5, 6);
    printf("%d is the bigger number.\n",maxNum);
    maxNum = maxNums(2, 20);
    printf("%d is the bigger number.\n", maxNum);
    maxNum = maxNums(88, -88);
    printf("%d is the bigger number.\n", maxNum);
    maxNum = maxNums(56, 89);
    printf("%d is the bigger number.\n", maxNum);
    maxNum = maxNums(3, 2);
    printf("%d is the bigger number.\n", maxNum);
    maxNum = maxNums(-5, -6);
    printf("%d is the bigger number.\n", maxNum);
}

int maxNums(int num1, int num2) {
    if (num1>num2) {
        return num1;
    }
    else {
        return num2;
    }
}
```



```
6 is the bigger number.
20 is the bigger number.
88 is the bigger number.
89 is the bigger number.
3 is the bigger number.
-5 is the bigger number.
```

EXAMPLE PROBLEM

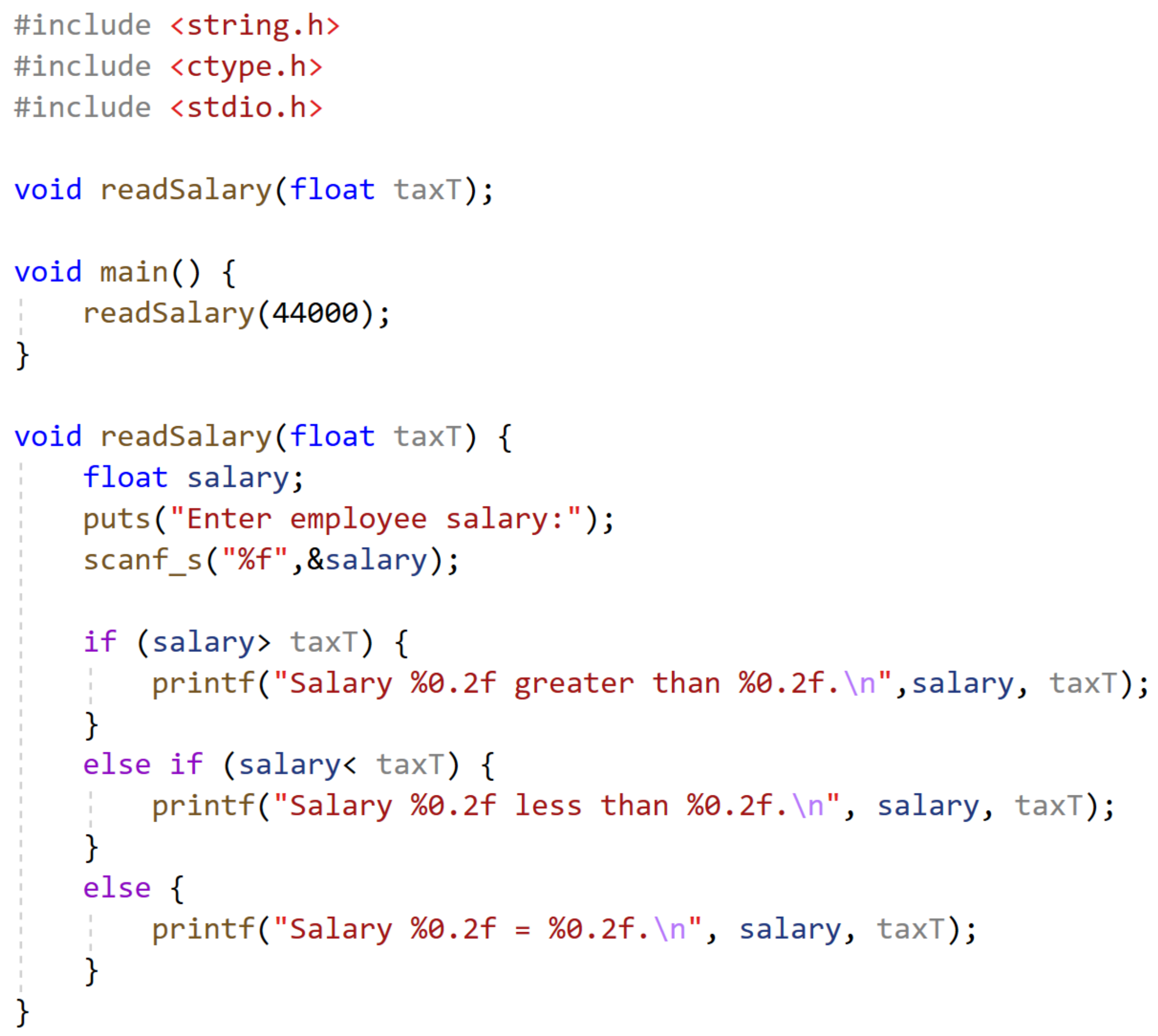
Salary Tax Function Problem

- You are writing software to process employees salaries:
  - Write a function called “readSalary”.
  - readSalary does not return anything.
  - This function should read in a tax threshold in Euro as a parameter.
  - The function should ask the user to enter the employee salary.
  - The function should then check if the salary is >, <, or = the tax threshold.
  - You should print a message to the console saying which of these is the situation.
  - Test your function by passing in the value of €44,000 as a tax threshold when you call the readSalary function in main.

Salary Tax Function Problem

- Go to C program solution.

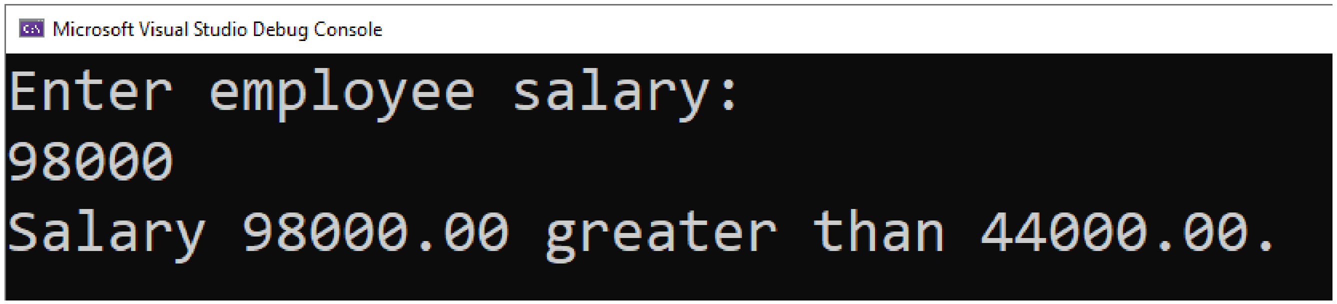
Salary Tax Function Problem



```c
#include <string.h>
#include <ctype.h>
#include <stdio.h>

void readSalary(float taxT);

void main() {
    readSalary(44000);
}

void readSalary(float taxT) {
    float salary;
    puts("Enter employee salary:");
    scanf_s("%f",&salary);

    if (salary> taxT) {
        printf("Salary %0.2f greater than %0.2f.\n",salary, taxT);
    }
    else if (salary< taxT) {
        printf("Salary %0.2f less than %0.2f.\n", salary, taxT);
    }
    else {
        printf("Salary %0.2f = %0.2f.\n", salary, taxT);
    }
}
```

Salary Tax Function Problem

- C Program Output:



```
Enter employee salary:
98000
Salary 98000.00 greater than 44000.00.
```

PROGRAMMING

CT103

Week 8b

Sign in on Blackboard

- Please sign in on blackboard for CT103.
- I will leave the sign in option open for an hour after the lecture.

Lecture Content

- Last lecture (Week 8a):
  - Functions
  - Writing functions
  - Functions in C
  - Example C program

- Today’s lecture (Week 8b):
  - Functions recap
  - Local variables
  - Global variables
  - Functions and strings
  - Example C program

FUNCTIONS RECAP

Functions

- A function is a piece of code that can be called whenever we need to execute that code.
- We have seen many functions in C, e.g. strlen().
- Benefits:
  - Functions allow us to reuse code, therefore avoid repetition.
  - More readable programs.
  - Enables us to divide complex problems into simpler ones.
  - Easier to make changes to program.

Function Template

- All functions have the following template:

```
        type name (parameters){
                return;
        }
```

- Type = data type returned by the function (can be void).
- Name = function name.
- Parameters = data we are giving to the function (can be empty).
- Return = what data is returned by the function (can also return nothing).

C Program with Functions

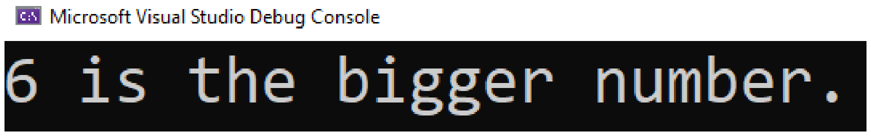
- Example from Monday: Get max of 2 numbers:

```c
int maxNums(int num1, int num2);

void main() {
    int maxNum;
    int n1 = 5;
    int n2 = 6;
    maxNum = maxNums(n1, n2);
    printf("%d is the bigger number.\n",maxNum);
}

int maxNums(int num1, int num2) {
    if (num1>num2) {
        return num1;
    }
    else {
        return num2;
    }
}
```

Function prototype

Main (we should be familiar with this one)

Function itself


Microsoft Visual Studio Debug Console

```
6 is the bigger number.
```

LOCAL VARIABLES

Variable Scope

- What do we mean by variable scope?

- **Definition**: The scope of the variable defines the region of the program where the variable is visible.

- A variables scope can be either **local** or **global**.

Local Variables

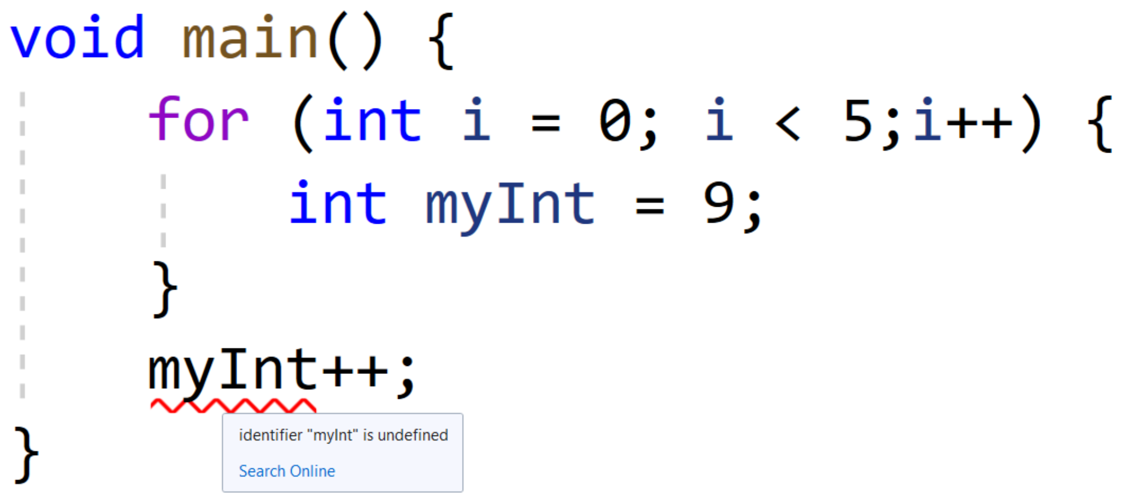
- Local variables are variables that are declared inside a function or code block.

- We will see in the coming slides how the visibility of a variable is important!

Local Variables

- Here the variable “myInt” is local to the for loop.

- “myInt” is not visible outside of the for loop as this is outside of its scope.



```
void main() {
    for (int i = 0; i < 5;i++) {
        int myInt = 9;
    }
    myInt++;
}
```

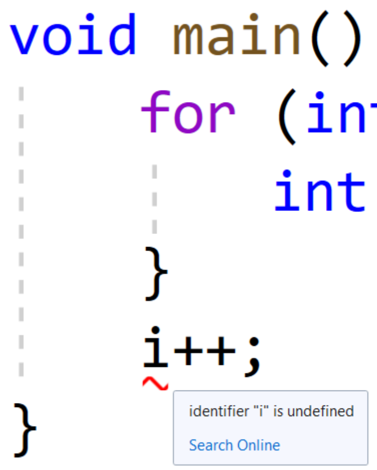
identifier "myInt" is undefined

Search Online

Local Variables

- Similarily, the variable “i” is also local to the for loop.

- “i” is not visible outside of the for loop as this is outside of its scope.



```
void main() {
    for (int i = 0; i < 5;i++) {
        int myInt = 9;
    }
    i++;
}
```

identifier "i" is undefined

Local Variables

- We can change the scope of “i” and “myInt” by declaring these variables outside of the for loop.

```
void main() {
    int i=0;
    int myInt=0;
    for (i = 0; i < 5;i++) {
        myInt = 9;
    }
    i++;
    myInt++;
}
```

No problems accessing these variables outside for loop now

Local Variables

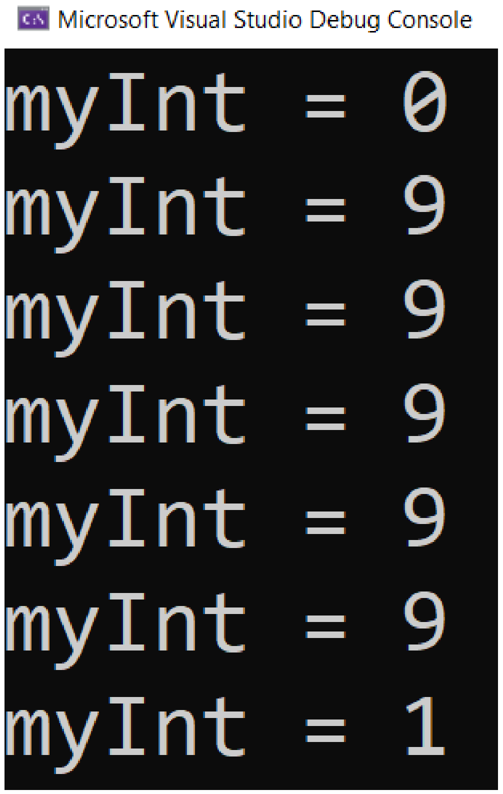
- What will the following code output?

```
void main() {
    int i=0;
    int myInt=0;
    printf("myInt = %d\n",myInt);
    for (i = 0; i < 5;i++) {
        int myInt = 9;
        printf("myInt = %d\n", myInt);
    }
    i++;
    myInt++;
    printf("myInt = %d\n", myInt);
}
```

Local Variables

- Why does the code output the following?

```
void main() {
    int i=0;
    int myInt=0;
    printf("myInt = %d\n",myInt);
    for (i = 0; i < 5;i++) {
        int myInt = 9;
        printf("myInt = %d\n", myInt);
    }
    i++;
    myInt++;
    printf("myInt = %d\n", myInt);
}
```

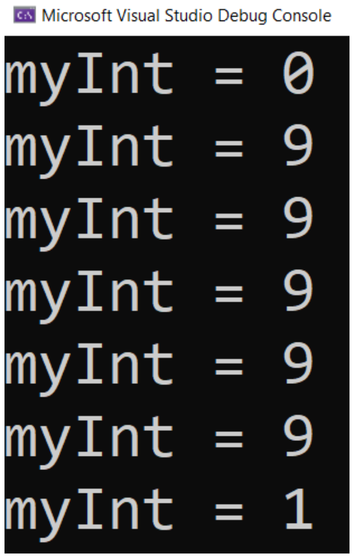

Microsoft Visual Studio Debug Console

```
myInt = 0
myInt = 9
myInt = 9
myInt = 9
myInt = 9
myInt = 9
myInt = 1
```

Local Variables

- Why does the code output the following?
  - Two variables called “myInt” are being created.
  - One is local to the main function, the other is local to the for loop.
  - This is **bad practice**.
  - Any time you are creating variables, give them a unique name.

```
void main() {
    int i=0;
    int myInt=0;
    printf("myInt = %d\n",myInt);
    for (i = 0; i < 5;i++) {
        int myInt = 9;
        printf("myInt = %d\n", myInt);
    }
    i++;
    myInt++;
    printf("myInt = %d\n", myInt);
}
```


Microsoft Visual Studio Debug Console

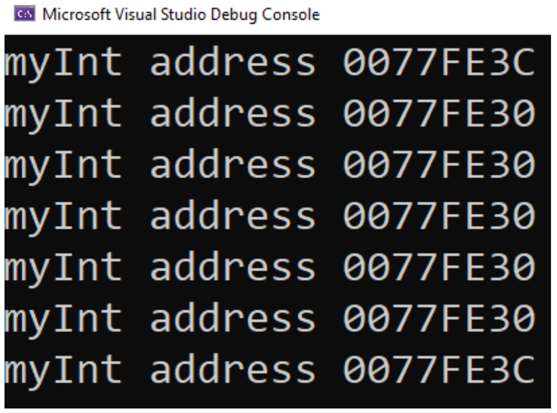
```
myInt = 0
myInt = 9
myInt = 9
myInt = 9
myInt = 9
myInt = 9
myInt = 1
```

Local Variables

- We can see the both “myInt” variables are stored separately in memory.

%p is for pointers.
You can simply use %X if you wish.
We will not discuss pointers yet.

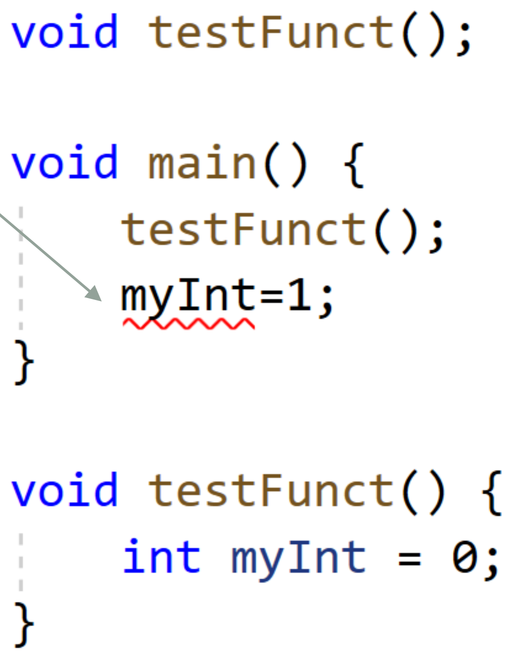
```c
void main() {
    int i=0;
    int myInt=0;
    printf("myInt address %p\n",&myInt);
    for (i = 0; i < 5;i++) {
        int myInt = 9;
        printf("myInt address %p\n", &myInt);
    }
    i++;
    myInt++;
    printf("myInt address %p\n", &myInt);
}
```



```
myInt address 0077FE3C
myInt address 0077FE30
myInt address 0077FE30
myInt address 0077FE30
myInt address 0077FE30
myInt address 0077FE30
myInt address 0077FE3C
```

Local Variables

- Similarly we can create variables local to specific functions.
- "myInt" is visible in testFunct() but not in main().

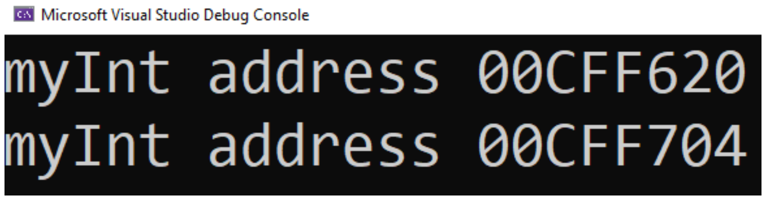


```
void testFunct();

void main() {
    testFunct();
    myInt=1;
}

void testFunct() {
    int myInt = 0;
}
```

Local Variables

- Similar to before, we see in the example below the different addresses for both “myInt” variables in testFunct() and main().

```
void testFunct();

void main() {
    testFunct();
    int myInt=1;
    printf("myInt address %p\n", &myInt);
}

void testFunct() {
    int myInt = 0;
    printf("myInt address %p\n", &myInt);
}
```


Microsoft Visual Studio Debug Console

```
myInt address 00CFF620
myInt address 00CFF704
```

GLOBAL VARIABLES

Global Variables

- Global variables are variables that are created outside of a function.

- These variables can be used anywhere in the program after it is declared.

- To set up a global variable, simply declare it outside of any function. It can then be accessed by any function. We normally declare it before main().

Global Variables

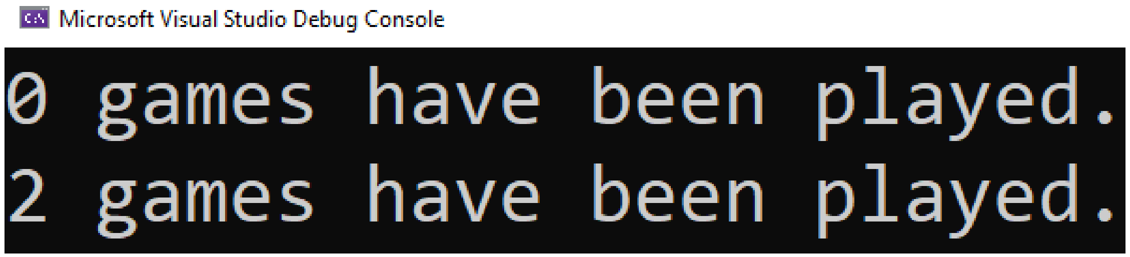
- Lets look at the following example where we declare a global variable:

```
void playGame();
void displayGames();
int gamesPlayed = 0;

void main() {
    displayGames();
    playGame();
    playGame();
    playGame();
    gamesPlayed--;
    displayGames();
}

void playGame() {
    gamesPlayed++;
}

void displayGames() {
    printf("%d games have been played.\n", gamesPlayed);
}
```

Global Variables

- This code outputs the following:

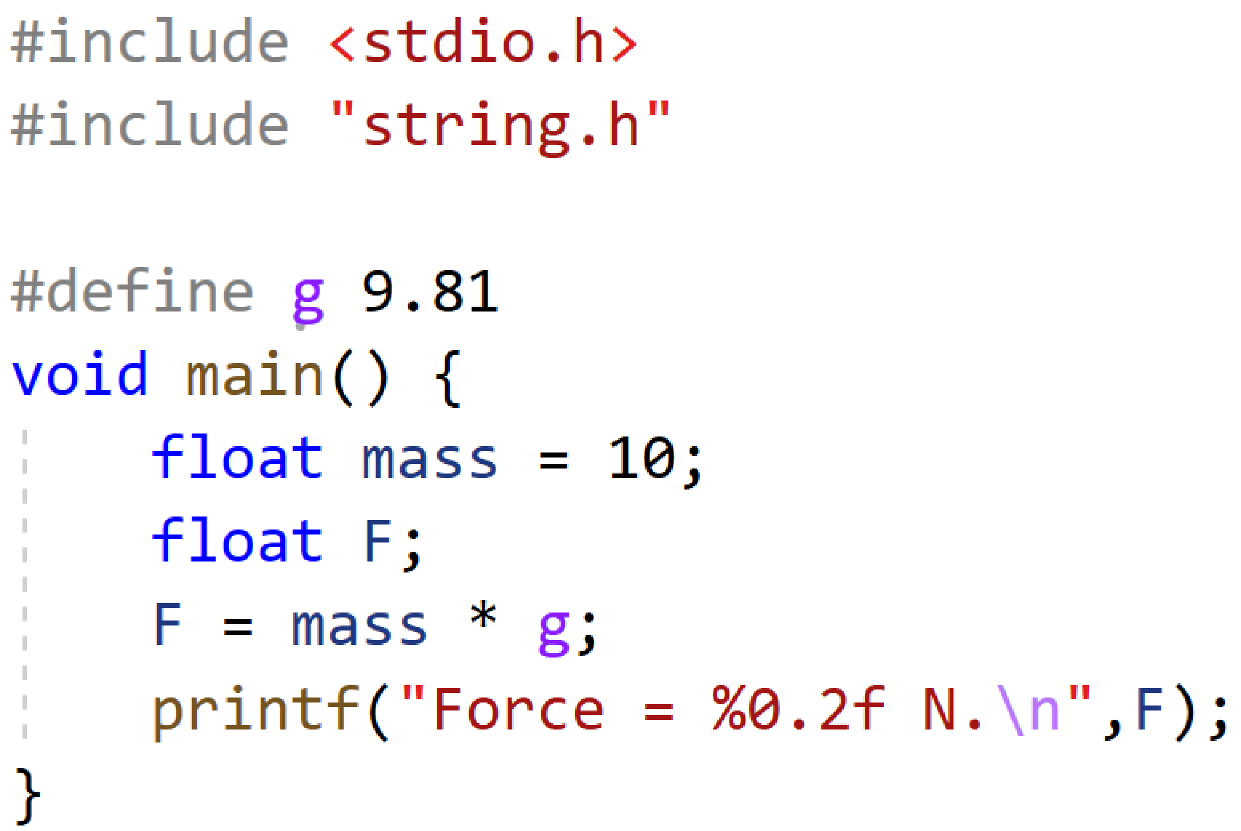
```c
void playGame();
void displayGames();
int gamesPlayed = 0;

void main() {
    displayGames();
    playGame();
    playGame();
    playGame();
    gamesPlayed--;
    displayGames();
}

void playGame() {
    gamesPlayed++;
}

void displayGames() {
    printf("%d games have been played.\n", gamesPlayed);
}
```



```
0 games have been played.
2 games have been played.
```

Global Scope

- We have already seen macros that have global scope in previous lectures (week 7a):



```
#include <stdio.h>
#include "string.h"

#define g 9.81
void main() {
    float mass = 10;
    float F;
    F = mass * g;
    printf("Force = %0.2f N.\n",F);
}
```

FUNCTIONS AND STRINGS

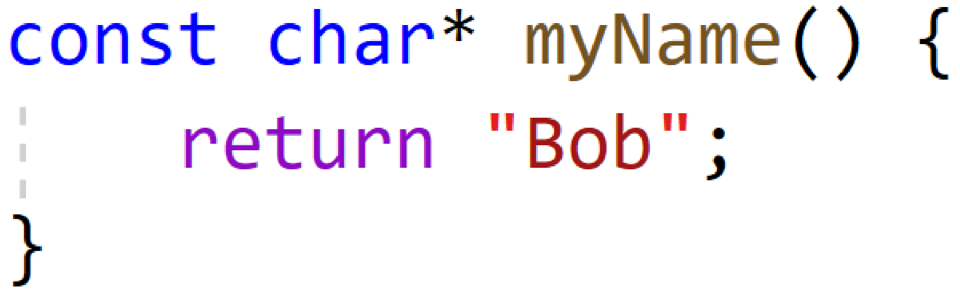
Strings and Functions

- We have not yet talked about how to pass a string to and from a function.

- Unfortunately, it is not as straightforward with strings since they are arrays of characters…

Returning Strings

- If we want to return a string from a function, we need to declare the return type as “**const char***”:

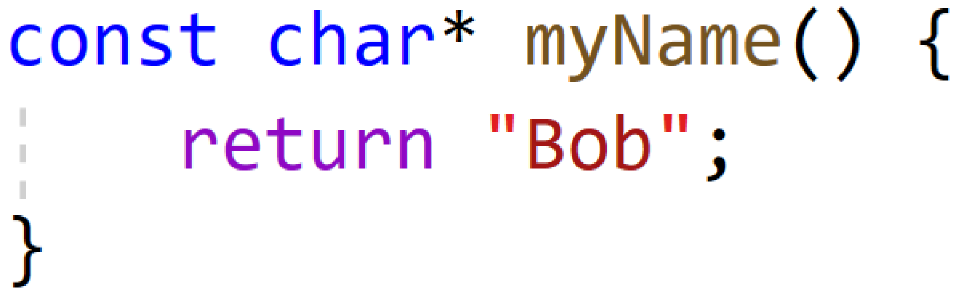


```
const char* myName() {
    return "Bob";
}
```

Returning Strings

- If we want to return a string from a function, we need to declare the return type as “**const char***”.

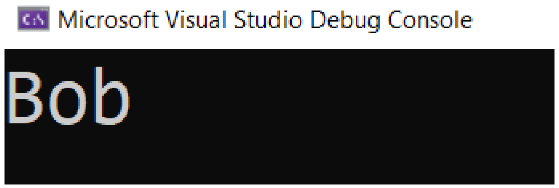
- What we are actually doing here is returning a pointer (*) to the first element of the string.



```
const char* myName() {
    return "Bob";
}
```

Returning Strings

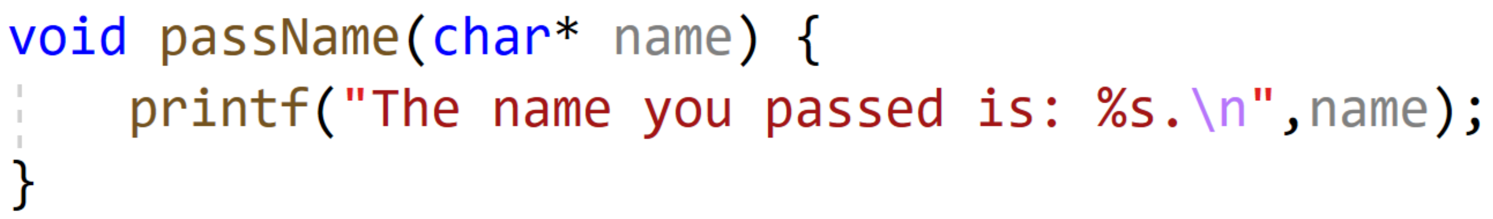
```c
#include <stdio.h>
#include "string.h"

const char* myName();

void main() {
    char newName[10];
    strcpy_s(newName,10, myName());
    puts(newName);
}

const char* myName() {
    return "Bob";
}
```


Microsoft Visual Studio Debug Console

```
Bob
```

Passing Strings

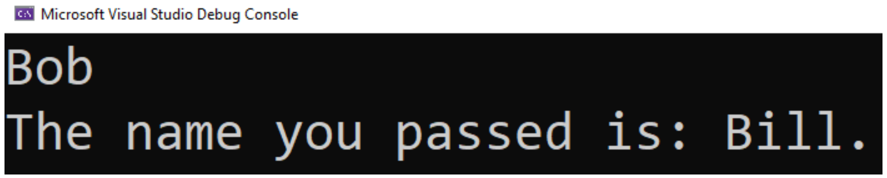
- If you want to pass a string to a function, you need to use "**char***".
- Here we are passing a pointer (*) to the first character of the string.
- Again, don't worry about pointers yet.



```
void passName(char* name) {
    printf("The name you passed is: %s.\n",name);
}
```

Passing Strings

```c
#include <stdio.h>
#include "string.h"

const char* myName();
void passName(char* name);

void main() {
    char newName[10];
    strcpy_s(newName,10, myName());
    puts(newName);
    passName("Bill");
}

void passName(char* name) {
    printf("The name you passed is: %s.\n",name);
}

const char* myName() {
    return "Bob";
}
```


Microsoft Visual Studio Debug Console

```
Bob
The name you passed is: Bill.
```

EXAMPLE PROBLEM

Bank Account Problem

- You are writing software to process bank accounts:
  - Create a global variable that represents the bank account balance.
  - Write a function that initializes the balance to €50.
  - Write a function that allows the user to make a withdrawal and update the bank balance.
  - Write a function that allows the user to make a deposit and update the bank balance.
  - Write a function that displays the bank balance.
  - Test the software by:
    - Creating a bank acc.
    - Withdraw €10.
    - Deposit €60.
    - Withdraw €30.
    - Display the balance between each transaction.

Bank Account Problem

- Go to C program solution.

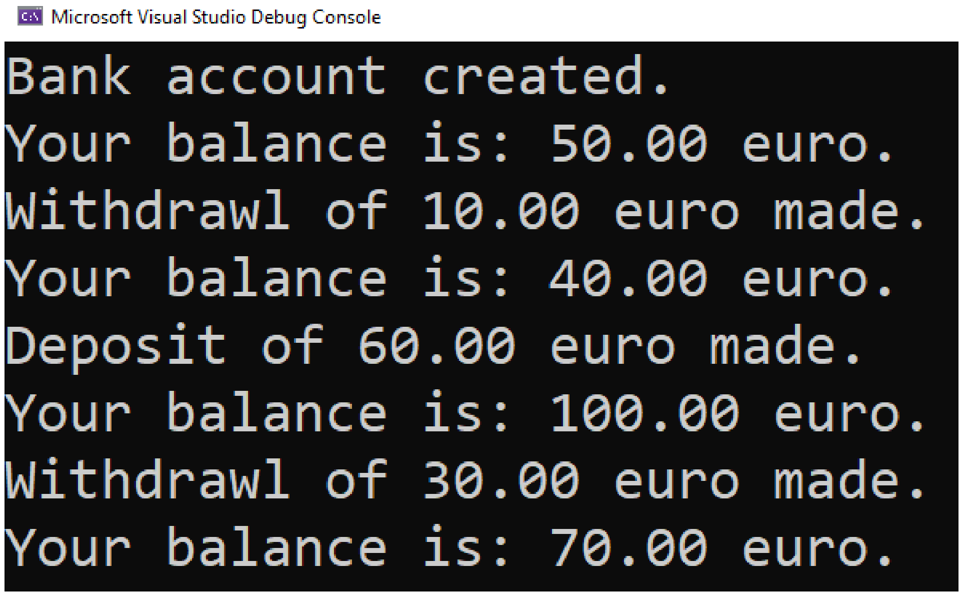
Bank Account Problem

```c
#include <string.h>
#include <ctype.h>
#include <stdio.h>



void initBankAC();
void withdrawal(float amnt);
void deposit(float amnt);
void displayBal();

float bankBalance;

void main() {
    initBankAC();
    displayBal();
    withdrawal(10.0);
    displayBal();
    deposit(60.0);
    displayBal();
    withdrawal(30.0);
    displayBal();
}
```

```c
void initBankAC() {
    bankBalance = 50.0;
    puts("Bank account created.");
}

void withdrawal(float amnt) {
    bankBalance -= amnt;
    printf("Withdrawl of %0.2f euro made.\n",amnt);
}

void deposit(float amnt){
    bankBalance += amnt;
    printf("Deposit of %0.2f euro made.\n", amnt);
}

void displayBal() {
    printf("Your balance is: %0.2f euro.\n", bankBalance);
}
```

Bank Account Problem

- C Program Output:



```
Bank account created.
Your balance is: 50.00 euro.
Withdrawl of 10.00 euro made.
Your balance is: 40.00 euro.
Deposit of 60.00 euro made.
Your balance is: 100.00 euro.
Withdrawl of 30.00 euro made.
Your balance is: 70.00 euro.
```

PROGRAMMING

CT103

Week 9a

Sign in on Blackboard

- Please sign in on blackboard for CT103.
- I will leave the sign in option open for an hour after the lecture.

Lecture Content

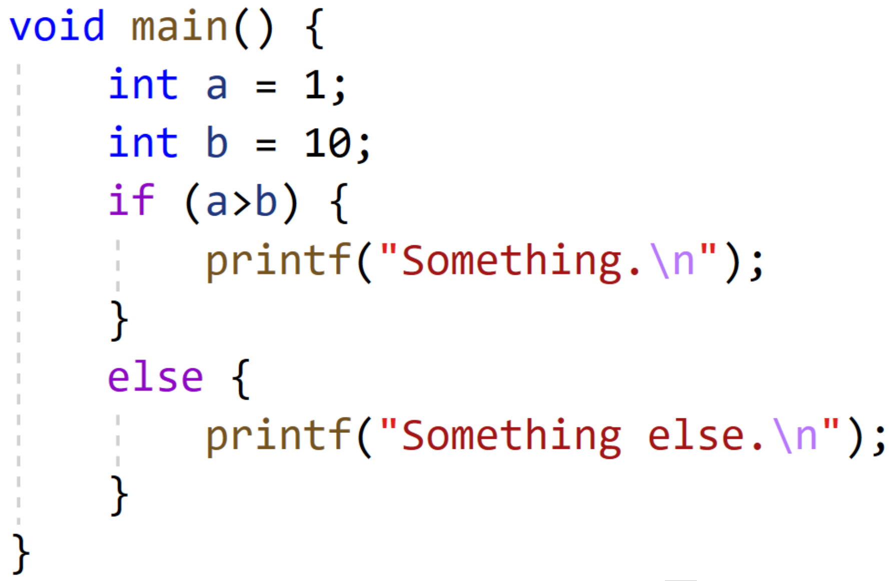
- Last lecture (Week 8b):
  - Functions recap
  - Local variables
  - Global variables
  - Functions and strings
  - Example C program

- Today’s lecture (Week 9a):
  - Conditional operator
  - Maths functions
  - Example C program

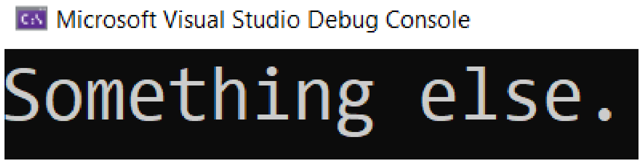
CONDITIONAL OPERATOR

Conditions

- Up until now, if we wanted to check if a condition is true, we would use an **if else** statement like this:



```c
void main() {
    int a = 1;
    int b = 10;
    if (a>b) {
        printf("Something.\n");
    }
    else {
        printf("Something else.\n");
    }
}
```



```
Something else.
```

Conditional Operator

- Today we will introduce the conditional operator in C!
- This involves using two symbols: ‘**?**’ and ‘**:**’
- It is a compact way of representing decision making statements.

Conditional Operator

- The conditional operator looks like:

  Variable = (test) ? ‘value if true’ : ‘value if false’ ;

- The test will yield a value of true or false which determines the value which will be put into the variable on the LHS of the ‘=‘

- The RHS evaluates to either value, depending on whether the condition is true or false

Conditional Operator

- The conditional operator in C looks like this:

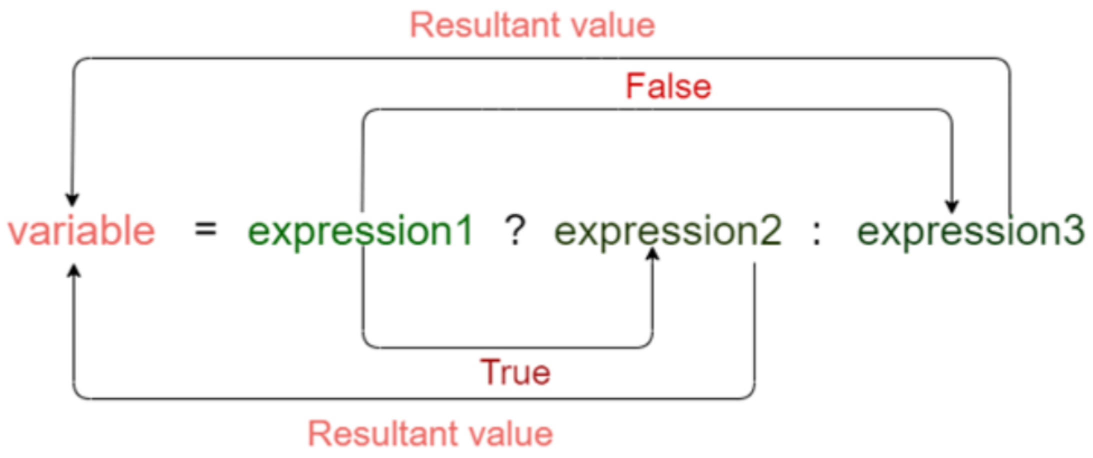


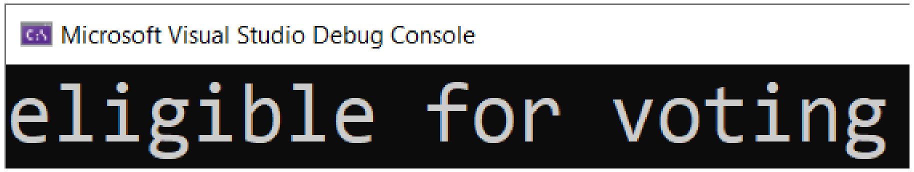
Image from: Javapoint.com

- Expression 1 is a Boolean condition.
- Expression 2 will execute if expression 1 is true.
- Expression 3 will execute if expression 1 is false.

Conditional Operator Example

- Conditional operator in C example:

```c
int age = 45;
(age >= 18) ? (printf("eligible for voting")) : (printf("not eligible for voting"));
```


Microsoft Visual Studio Debug Console

```
eligible for voting
```

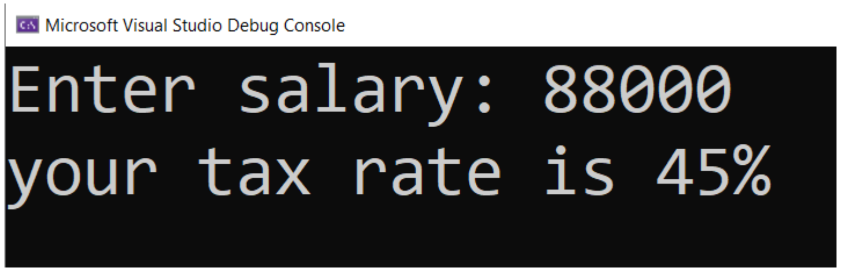
Tax Rate Calculator

- Take the following example:

```c
void main()
{
    double taxThreshold = 30000.00;
    int lowRate = 25, highRate = 45;
    double salary;
    int rate;

    printf("Enter salary: ");
    scanf_s("%lf", &salary);

    if (salary >= taxThreshold){
        rate = highRate;
    }
    else{
        rate = lowRate;
    }
    printf("your tax rate is %d%% \n", rate);
}
```

Tax Rate Calculator

```c
void main()
{
    double taxThreshold = 30000.00;
    int lowRate = 25, highRate = 45;
    double salary;
    int rate;

    printf("Enter salary: ");
    scanf_s("%lf", &salary);

    if (salary >= taxThreshold){
        rate = highRate;
    }
    else{
        rate = lowRate;
    }
    printf("your tax rate is %d%% \n", rate);
}
```


Microsoft Visual Studio Debug Console

```
Enter salary: 88000
your tax rate is 45%
```

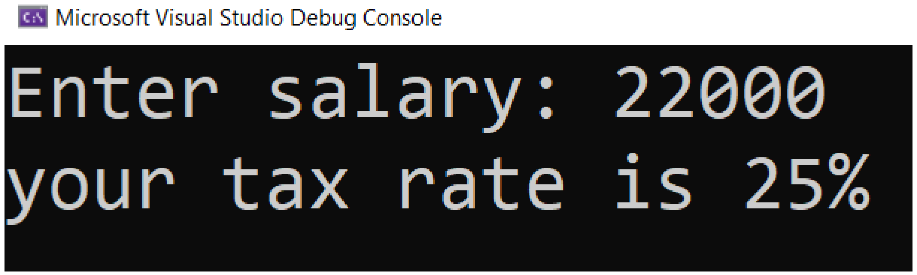
Tax Rate with Conditional Operator

- Take the following example:

```c
void main()
{
    double taxThreshold = 30000.00;
    int lowRate = 25, highRate = 45;
    double salary;
    int rate;

    printf("Enter salary: ");
    scanf_s("%lf", &salary);

    rate = (salary >= taxThreshold) ? highRate : lowRate;

    printf("your tax rate is %d%% \n", rate);
}
```

Tax Rate with Conditional Operator

```c
void main()
{
    double taxThreshold = 30000.00;
    int lowRate = 25, highRate = 45;
    double salary;
    int rate;

    printf("Enter salary: ");
    scanf_s("%lf", &salary);

    rate = (salary >= taxThreshold) ? highRate : lowRate;

    printf("your tax rate is %d%% \n", rate);
}
```


Microsoft Visual Studio Debug Console

```
Enter salary: 22000
your tax rate is 25%
```

Tax Rate with Conditional Operator

- We are replacing this:

```
if (salary >= taxThreshold) {
    rate = highRate;
}
else {
    rate = lowRate;
}
```

- With this:

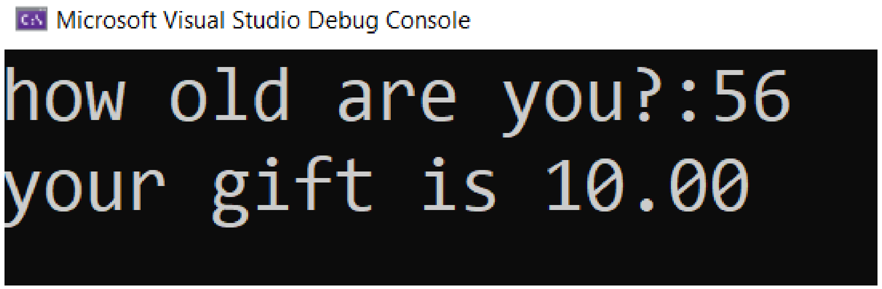
```
rate = (salary >= taxThreshold) ? highRate : lowRate;
```

- Much shorter!

Another example

```
void main()
{
    int age;
    float gift;
    printf("how old are you?:");
    scanf_s("%d", &age);

    if (age < 18)
    {
        gift = 5.0;
    }
    else
    {
        gift = 10.0;
    }

    printf("your gift is %.2f\n", gift);
}
```

Can be written ..

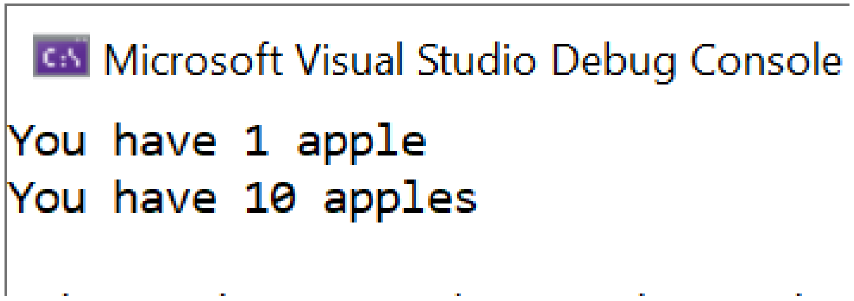

Microsoft Visual Studio Debug Console

```
how old are you?:56
your gift is 10.00
```

```c
void main()
{
    int age;
    float gift;
    printf("how old are you?:");
    scanf_s("%d", &age);

    gift = (age < 18) ? 5.0 : 10.0;

    printf("your gift is %.2f\n", gift);
}
```

Nice example

```
void main()
{
    int num;
    char s;

    num = 1;
    s = (num == 1) ? ' ' : 's';
    printf("You have %d apple%c \n", num, s);

    num = 10;
    s = (num == 1) ? ' ' : 's';
    printf("You have %d apple%c \n", num, s);
}
```



```
You have 1 apple
You have 10 apples
```

Going too far ?

- This is even shorter, but perhaps a bit too hard to read!
- This works because C evaluates the conditional operator *BEFORE* sending the result to printf

```c
void main()
{
    int num;

    num = 1;
    printf("You have %d apple%c \n", num, (num == 1) ? ' ' : 's');

    num = 10;
    printf("You have %d apple%c \n", num, (num == 1) ? ' ' : 's');
}
```

MATHS

Maths in C

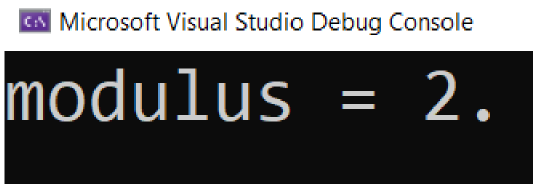
- We have done plenty of maths in C until now.
- We have done:
  - Addition: ‘+’
  - Subtraction: ‘-’
  - Multiplication: ‘*’
  - Division: ‘/’

  - Also modulus (remainder): ‘%’

Modulus Recap

- A quick reminder about modulus (%).
- This allows us to get the remainder when dividing a number by another number.
- See example:

```
void main()
{
    int num = 12%5;
    printf("modulus = %d.\n",num);
}
```


Microsoft Visual Studio Debug Console

```
modulus = 2.
```

Maths in C

- There is also the maths library in C that contains lots of very useful functions for doing mathematical operations!

- You will need to import **math.h** to use these:

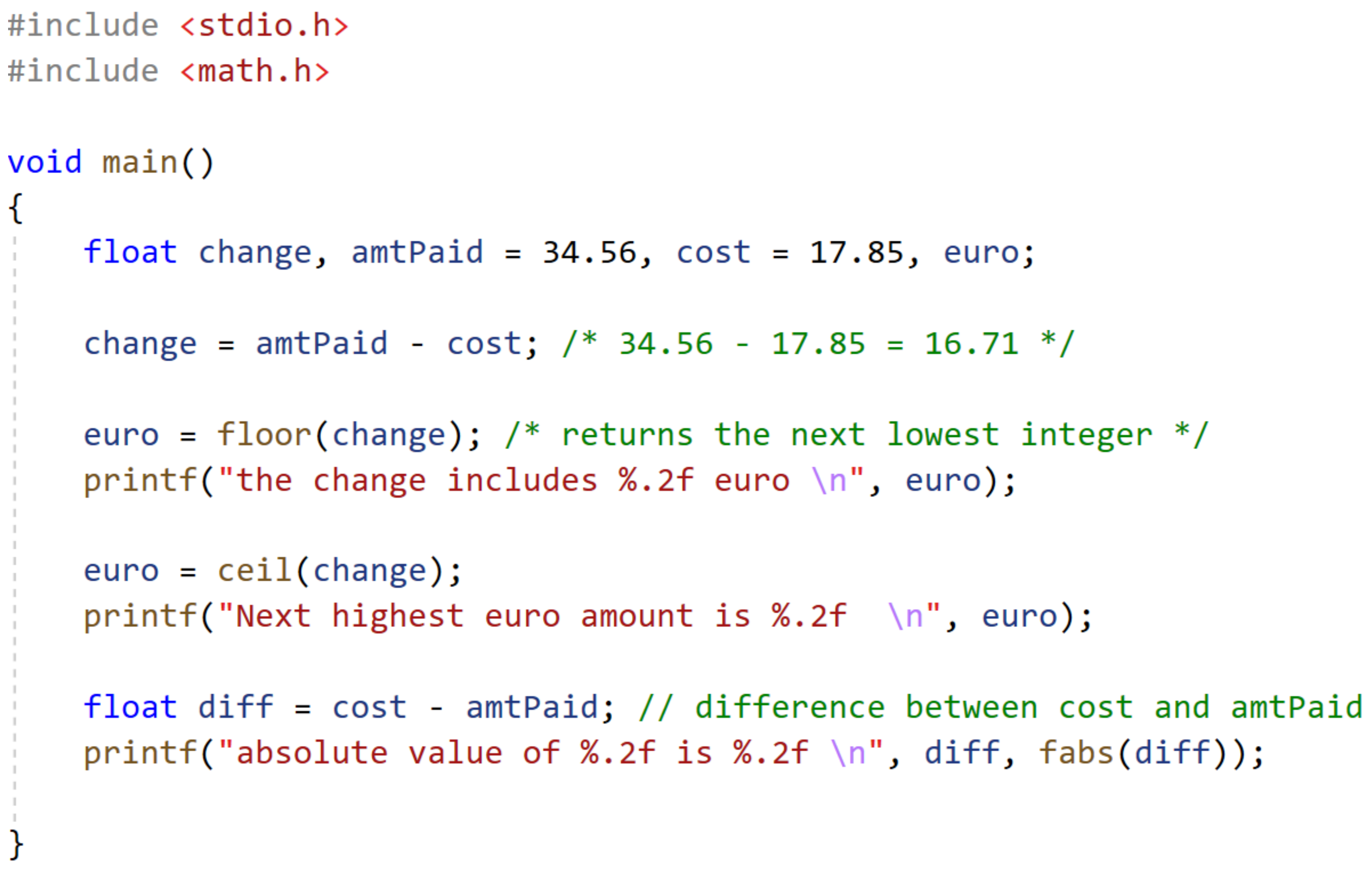
```
#include <math.h>
```

Maths in C

- Math.h has many useful functions:

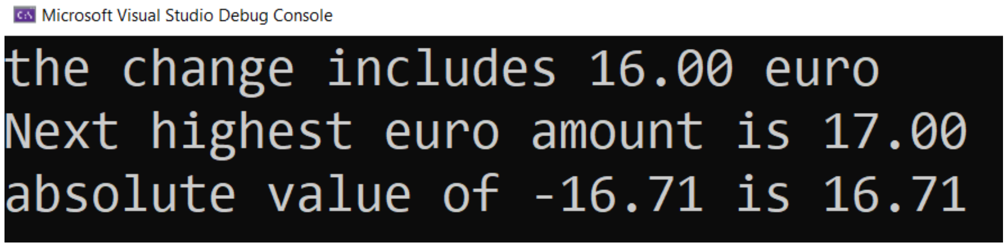
- A small selection of these that we will talk about today include:
  - floor() – returns the next lowest whole number
  - ceil() – returns the next highest whole number
  - fabs() – returns the absolute value

Sample Program



```c
#include <stdio.h>
#include <math.h>

void main()
{
    float change, amtPaid = 34.56, cost = 17.85, euro;

    change = amtPaid - cost; /* 34.56 - 17.85 = 16.71 */

    euro = floor(change); /* returns the next lowest integer */
    printf("the change includes %.2f euro \n", euro);

    euro = ceil(change);
    printf("Next highest euro amount is %.2f  \n", euro);

    float diff = cost - amtPaid; // difference between cost and amtPaid
    printf("absolute value of %.2f is %.2f \n", diff, fabs(diff));

}
```

Output


Microsoft Visual Studio Debug Console

```
the change includes 16.00 euro
Next highest euro amount is 17.00
absolute value of -16.71 is 16.71
```

```c
#include <stdio.h>
#include <math.h>

void main()
{
    float change, amtPaid = 34.56, cost = 17.85, euro;

    change = amtPaid - cost; /* 34.56 - 17.85 = 16.71 */

    euro = floor(change); /* returns the next lowest integer */
    printf("the change includes %.2f euro \n", euro);

    euro = ceil(change);
    printf("Next highest euro amount is %.2f  \n", euro);

    float diff = cost - amtPaid; // difference between cost and amtPaid
    printf("absolute value of %.2f is %.2f \n", diff, fabs(diff));

}
```

EXAMPLE PROBLEM

Payment Processing Problem

- You are writing software to process online shop payments:
  - Read in:
    - The number of past customer purchases from the user.
    - The current purchase amount in euro.
    - The customer payment amount in euro.
  - If the customer has made more that 5 previous purchases, round their bill down to the nearest euro.
  - Write a function to check if the customer has over/under paid. Print the absolute value of the amount to the screen to let the user know.

Payment Processing Problem

- Go to C program solution.

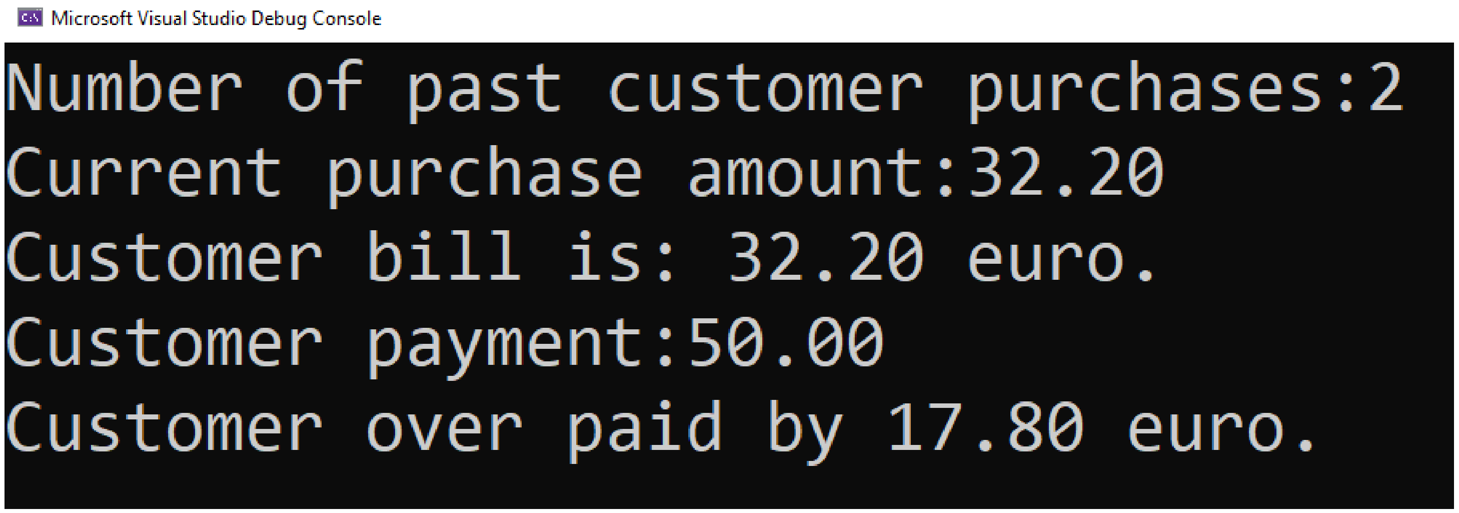
Payment Processing Problem

```c
#include <stdio.h>
#include <math.h>

void checkOverUnderPay(float payment, float bill);

void main()
{
    int numCustPurchases;
    float curPurchaseAmnt, custBill, custPayment;

    printf("Number of past customer purchases:");
    scanf_s("%d",&numCustPurchases);

    printf("Current purchase amount:");
    scanf_s("%f", &curPurchaseAmnt);

    custBill = (numCustPurchases>5)?floor(curPurchaseAmnt):curPurchaseAmnt;
    printf("Customer bill is: %0.2f euro.\n",custBill);

    printf("Customer payment:");
    scanf_s("%f", &custPayment);
```

```c
    checkOverUnderPay(custPayment, custBill);

}

void checkOverUnderPay(float payment, float bill) {
    float diff = payment - bill;
    if (payment > bill) {
        printf("Customer over paid by %0.2f euro.\n", fabs(diff));
    }
    else if (payment <bill) {
        printf("Customer still needs to pay extra of %0.2f euro.\n", fabs(diff));
    }
    else {
        printf("Payment processed.\n");
    }
}
```

Payment Processing Problem

- C Program Output:



```
Number of past customer purchases:2
Current purchase amount:32.20
Customer bill is: 32.20 euro.
Customer payment:50.00
Customer over paid by 17.80 euro.
```

PROGRAMMING

CT103

Week 9b

Sign in on Blackboard

- Please sign in on blackboard for CT103.
- I will leave the sign in option open for an hour after the lecture.

Lecture Content

- Last lecture (Week 9a):
  - Conditional operator
  - Maths functions
  - Example C program

- Today’s lecture (Week 9b):
  - More maths functions
  - Macros
  - Example C program

MATHS

Maths in C Recap

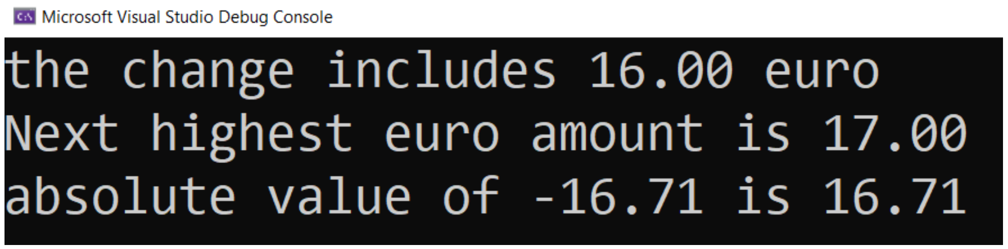
- We can use the maths library in C to do mathematical operations using their functions!

```
#include <math.h>
```

- On Monday we explored:
  - floor() – returns the next lowest whole number
  - ceil() – returns the next highest whole number
  - fabs() – returns the absolute value

Maths in C Recap

```c
#include <stdio.h>
#include <math.h>

void main()
{
    float change, amtPaid = 34.56, cost = 17.85, euro;

    change = amtPaid - cost; /* 34.56 - 17.85 = 16.71 */

    euro = floor(change); /* returns the next lowest integer */
    printf("the change includes %.2f euro \n", euro);

    euro = ceil(change);
    printf("Next highest euro amount is %.2f  \n", euro);

    float diff = cost - amtPaid; // difference between cost and amtPaid
    printf("absolute value of %.2f is %.2f \n", diff, fabs(diff));

}
```



```
the change includes 16.00 euro
Next highest euro amount is 17.00
absolute value of -16.71 is 16.71
```

MORE MATHS FUNCTIONS

Maths in C

- Math.h has many other use functions other than the ones we just created:
  - Pow() – raise one number to the power of another.
  - Sqrt() – Square root of a number.
  - Sin() – returns the sine of an angle in radians.
  - Cos() – returns the cosine of an angle in radians.
  - Tan() – returns the tangent of an angle in radians.
  - Exp() – returns the exponent on a value.
  - Log() – returns the natural log of a value.

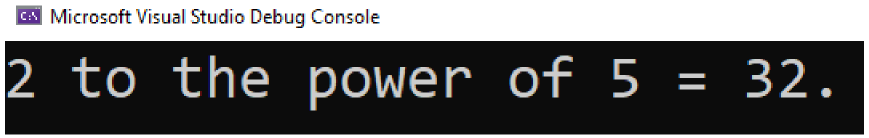
Pow()

- Pow() – raise one number to the power of another.
- E.g. $2^5 = 32$
- We say here that we are raising the number 2 to the power of 5.

Pow()

- Pow() – raise one number to the power of another.

```c
#include <stdio.h>
#include <math.h>

void main() {
    int n = pow(2,5);
    printf("%d to the power of %d = %d.\n",2,5,n);
}
```

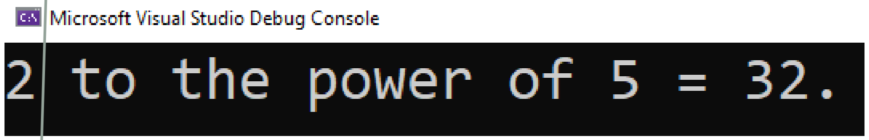


```
2 to the power of 5 = 32.
```

Not using Pow()

- The alternative would be to do this...

```
int n = 2*2*2*2*2;
printf("%d to the power of %d = %d.\n", 2, 5, n);
```


Microsoft Visual Studio Debug Console

```
2 to the power of 5 = 32.
```

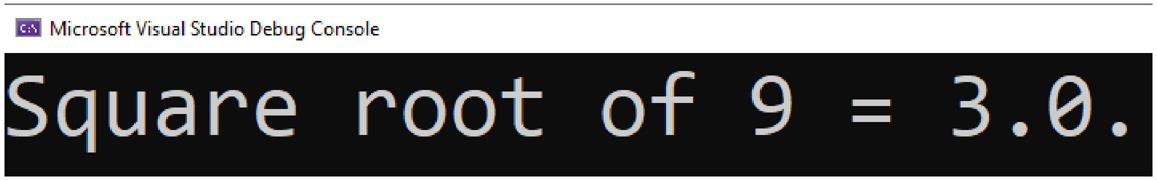
- Much easier to just use pow() instead of this.

Sqrt()

- Sqrt() – Square root of a number.

- E.g. Square root of 9 = 3.

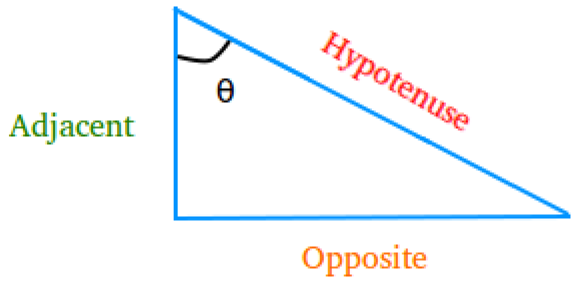
```c
#include <stdio.h>
#include <math.h>

void main() {
    float n = sqrt(9);
    printf("Square root of %d = %0.1f.\n",9,n);
}
```


Microsoft Visual Studio Debug Console

```
Square root of 9 = 3.0.
```

Trigonometry

- Quick refresher on trigonometry:



$$\sin\theta = \frac{\text{Opposite}}{\text{Hypotenuse}} \qquad \cos\theta = \frac{\text{Adjacent}}{\text{Hypotenuse}}$$

$$\tan\theta = \frac{\text{Opposite}}{\text{Adjacent}}$$

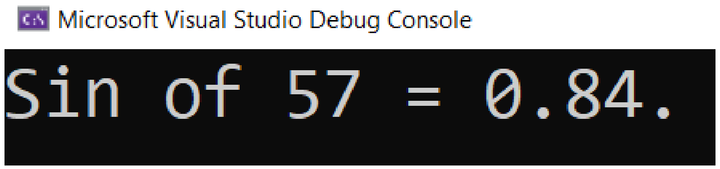
Sin()

- Sin() – returns the sine of an angle in **radians**.
- What are radians?
- Radians are another way of measuring an angle.
- To convert angle **D** from degrees to radians **R**, you use the following equation:
  - R = D * π / 180

Sin()

- Sin() – returns the sine of an angle in **radians**.

```c
#include <stdio.h>
#include <math.h>

void main() {
    float n = sin(1); // approx 57 degrees
    printf("Sin of %d = %0.2f.\n", 57, n);
}
```



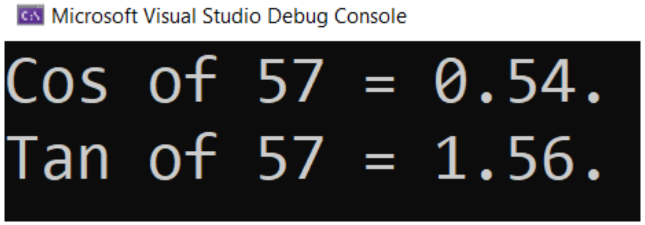
Microsoft Visual Studio Debug Console

```
Sin of 57 = 0.84.
```

Cos() and Tan ()

- Cos() – returns the cosine of an angle in radians.
- Tan() – returns the tangent of an angle in radians.

```c
#include <stdio.h>
#include <math.h>

void main() {
    float n = cos(1); // approx 57 degrees
    printf("Cos of %d = %0.2f.\n", 57, n);

    n = tan(1); // approx 57 degrees
    printf("Tan of %d = %0.2f.\n", 57, n);
}
```

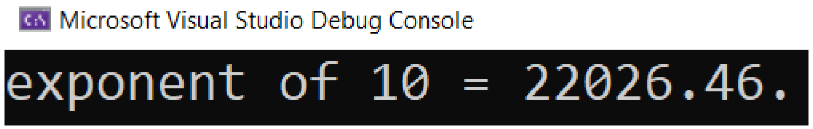


```
Cos of 57 = 0.54.
Tan of 57 = 1.56.
```

Exp()

- Exp() – returns the exponent on a value.
- E.g. exp(10) = e^{10} = 22026.
- Where e = 2.71828 (Euler's number).

```c
#include <stdio.h>
#include <math.h>

void main() {
    float n = exp(10);
    printf("exponent of %d = %0.2f.\n", 10, n);

}
```


Microsoft Visual Studio Debug Console

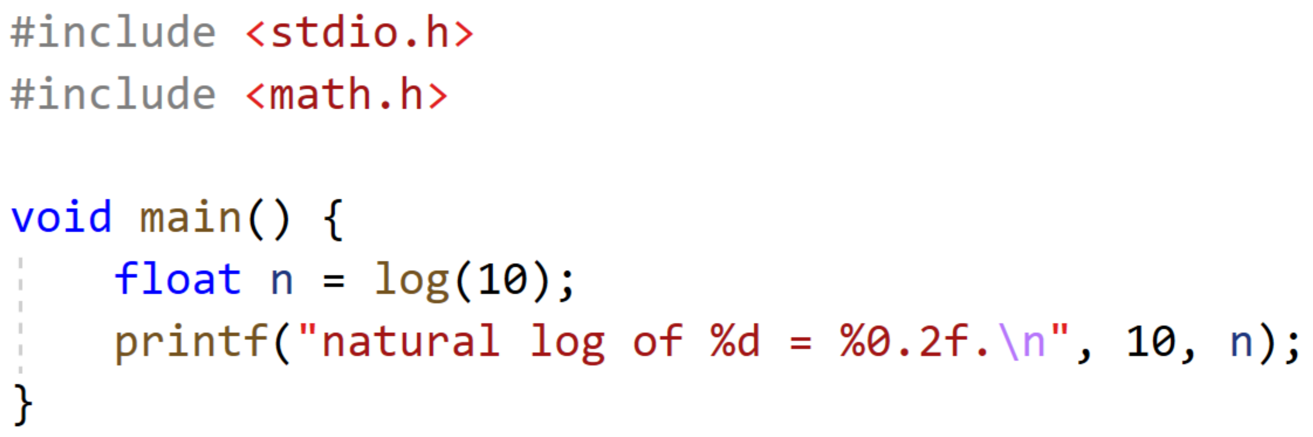
```
exponent of 10 = 22026.46.
```

Log()

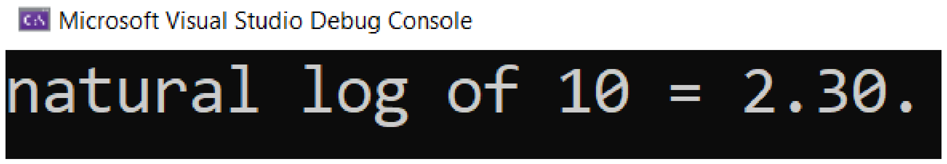
- Log() – returns the natural log of a value.
- E.g. log(10) = $\log_e 10$ = 2.303.
- Natural log of x is the power that e is raised to equal x.
- I.e. $e^{2.303} = 10$

Log()

- Log() – returns the natural log of a value.



```c
#include <stdio.h>
#include <math.h>

void main() {
    float n = log(10);
    printf("natural log of %d = %0.2f.\n", 10, n);
}
```



Microsoft Visual Studio Debug Console

```
natural log of 10 = 2.30.
```

MACROS

Macros

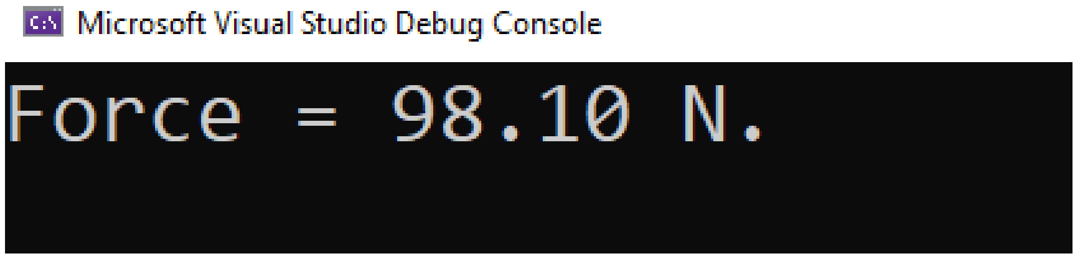
- We introduced macros in lecture 7a.

- A macro is a fragment of code which has been given a name. Whenever the name is used it is replaced by the contents of the macro.

- There are two types of macros:
  - Object like macros.
  - Function like macros (we skipped over these).

Object Like Macros in C

```c
#include <stdio.h>
#include "string.h"

#define g 9.81
void main() {
    float mass = 10;
    float F;
    F = mass * g;
    printf("Force = %0.2f N.\n",F);
}
```


Microsoft Visual Studio Debug Console

```
Force = 98.10 N.
```

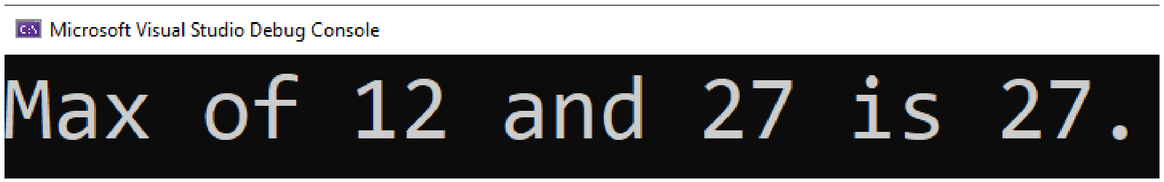
Function like Macros

- Today we will cover function like macros.
- Function like macros are pieces of code that are given a name.
- Unlike object like macros, function like macros contain a function.

Function like Macros in C

- Function like macro in C:

```c
#include <stdio.h>
#include <math.h>

#define MAX(x,y)(x>y?x:y)

void main() {
    int num = MAX(12,27);
    printf("Max of %d and %d is %d.\n", 12, 27,num);
}
```

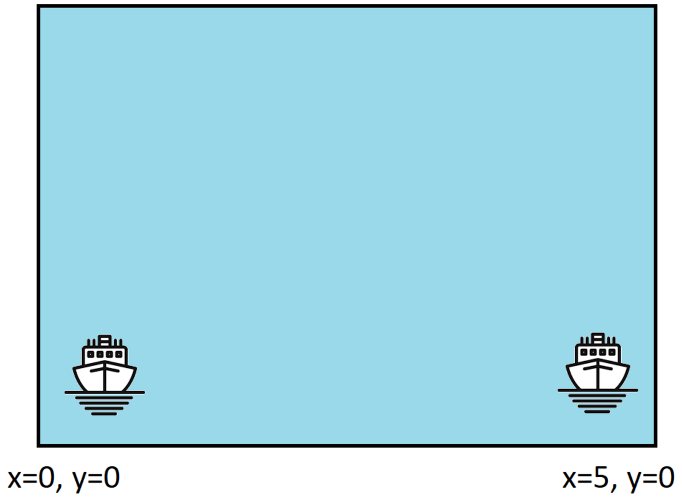


```
Max of 12 and 27 is 27.
```

EXAMPLE PROBLEM

Cost Guard Problem

- You are writing software for the coast guard to track ships in the ocean:
  - There are 2 ships in locations.
    - **Ship1** - location: x = 0, y = 0. velocity: v_x = 0.5 km/hr, v_y = 0.5 km/hr.
    - **Ship2** - location: x = 5, y = 0. velocity: v_x = -0.5 km/hr, v_y = 0.5 km/hr.



Cost Guard Problem

- You are writing software for the coast guard to track ships in the ocean:

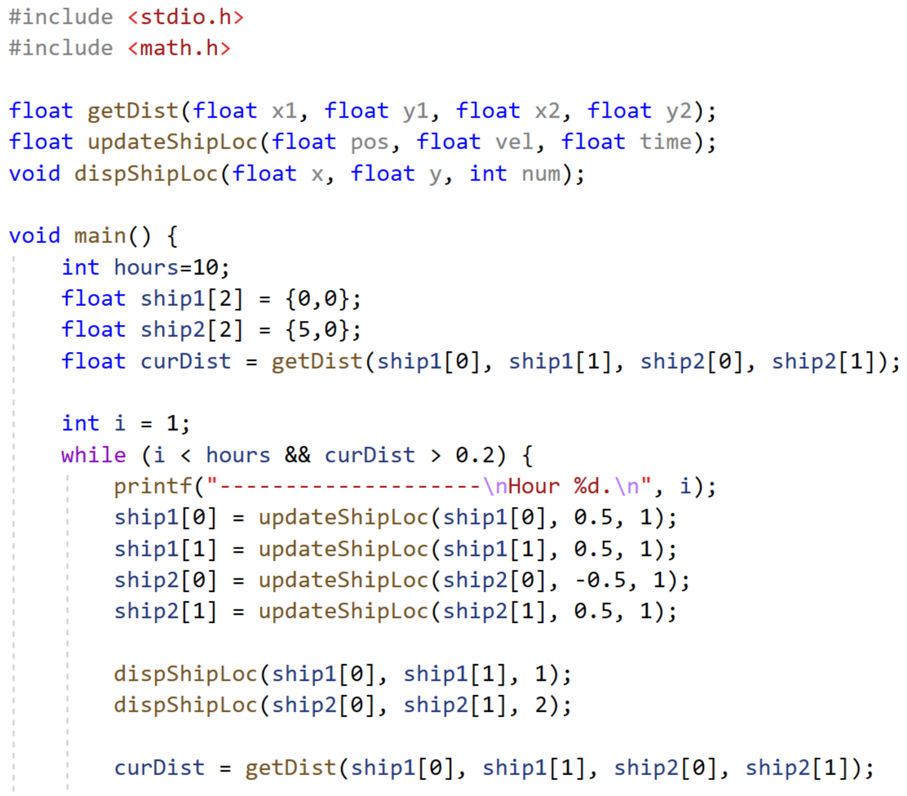
  - There are 2 ships in locations.
    - **Ship1** - location: x = 0, y = 0. velocity: v_x = 0.5 km/hr, v_y = 0.5 km/hr.
    - **Ship2** - location: x = 5, y = 0. velocity: v_x = -0.5 km/hr, v_y = 0.5 km/hr.

  - Write a C program that tracks their movements over 10 hours.
    - Write a function to return the Euclidean distance between each ship.
    - Write a function to update and return the location of a ship.
    - Write a function to display the location of each ship.
    - If the ships are within 1.5 km of one another, display a warning.
    - If the ships are within 200 meters of one another, the ships have collided. End the program.
    - Will the ships collide? If so, after how many hours?

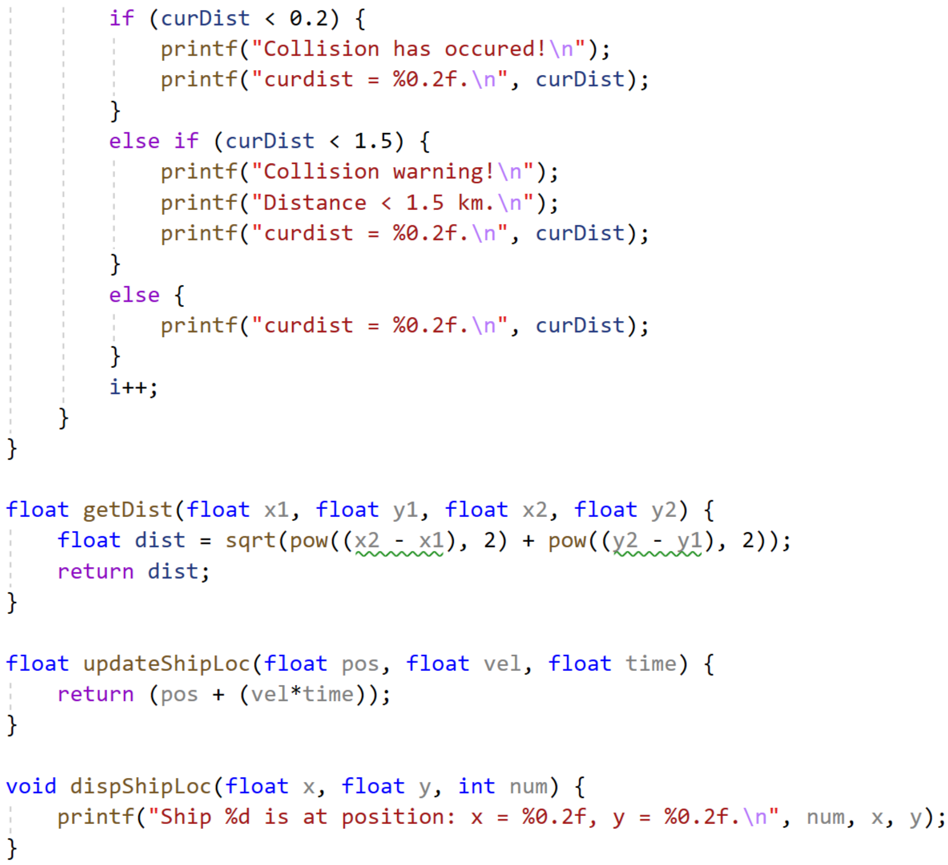
Cost Guard Problem

- Go to C program solution.

Cost Guard Problem



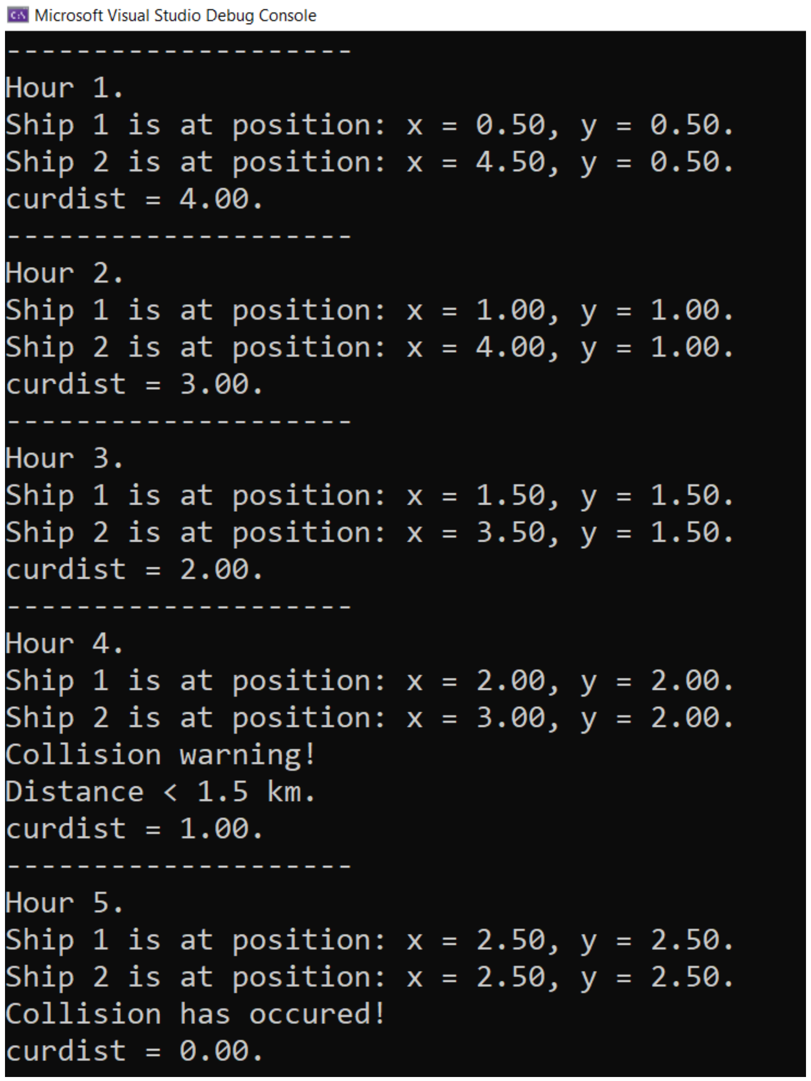
```c
#include <stdio.h>
#include <math.h>

float getDist(float x1, float y1, float x2, float y2);
float updateShipLoc(float pos, float vel, float time);
void dispShipLoc(float x, float y, int num);

void main() {
    int hours=10;
    float ship1[2] = {0,0};
    float ship2[2] = {5,0};
    float curDist = getDist(ship1[0], ship1[1], ship2[0], ship2[1]);

    int i = 1;
    while (i < hours && curDist > 0.2) {
        printf("--------------------\nHour %d.\n", i);
        ship1[0] = updateShipLoc(ship1[0], 0.5, 1);
        ship1[1] = updateShipLoc(ship1[1], 0.5, 1);
        ship2[0] = updateShipLoc(ship2[0], -0.5, 1);
        ship2[1] = updateShipLoc(ship2[1], 0.5, 1);

        dispShipLoc(ship1[0], ship1[1], 1);
        dispShipLoc(ship2[0], ship2[1], 2);

        curDist = getDist(ship1[0], ship1[1], ship2[0], ship2[1]);
```



```c
        if (curDist < 0.2) {
            printf("Collision has occured!\n");
            printf("curdist = %0.2f.\n", curDist);
        }
        else if (curDist < 1.5) {
            printf("Collision warning!\n");
            printf("Distance < 1.5 km.\n");
            printf("curdist = %0.2f.\n", curDist);
        }
        else {
            printf("curdist = %0.2f.\n", curDist);
        }
        i++;
    }
}

float getDist(float x1, float y1, float x2, float y2) {
    float dist = sqrt(pow((x2 - x1), 2) + pow((y2 - y1), 2));
    return dist;
}

float updateShipLoc(float pos, float vel, float time) {
    return (pos + (vel*time));
}

void dispShipLoc(float x, float y, int num) {
    printf("Ship %d is at position: x = %0.2f, y = %0.2f.\n", num, x, y);
}
```

Cost Guard Problem

- C Program Output:



Microsoft Visual Studio Debug Console

```
--------------------
Hour 1.
Ship 1 is at position: x = 0.50, y = 0.50.
Ship 2 is at position: x = 4.50, y = 0.50.
curdist = 4.00.
--------------------
Hour 2.
Ship 1 is at position: x = 1.00, y = 1.00.
Ship 2 is at position: x = 4.00, y = 1.00.
curdist = 3.00.
--------------------
Hour 3.
Ship 1 is at position: x = 1.50, y = 1.50.
Ship 2 is at position: x = 3.50, y = 1.50.
curdist = 2.00.
--------------------
Hour 4.
Ship 1 is at position: x = 2.00, y = 2.00.
Ship 2 is at position: x = 3.00, y = 2.00.
Collision warning!
Distance < 1.5 km.
curdist = 1.00.
--------------------
Hour 5.
Ship 1 is at position: x = 2.50, y = 2.50.
Ship 2 is at position: x = 2.50, y = 2.50.
Collision has occured!
curdist = 0.00.
```

PROGRAMMING

CT103

Week 10a

Sign in on Blackboard

- Please sign in on blackboard for CT103.
- I will leave the sign in option open for an hour after the lecture.

Lecture Content

- Last lecture (Week 9b):
  - More maths functions
  - Macros
  - Example C program

- Today’s lecture (Week 10a):
  - Passing arrays to functions
  - Generating random numbers
  - Breakpoints
  - Example C program

PASSING ARRAYS TO FUNCTIONS

Passing Arrays to Functions

- We already mentioned how we can use pointers to pass in strings, i.e. character arrays.
- We will see in the coming slides how to pass arrays of other variable types, e.g. ints and floats, to functions.
- We will also see how to pass strings to functions without using pointers.

Passing Arrays to Functions

- In order to accept arrays as function parameters, we have to specify its parameters as: type of array variables, an identifier and square brackets []. E.g:

  ***void* procedure (*int* arg[])**

  accepts a parameter of type "array of int" called arg. In order to pass to this function an array declared as:

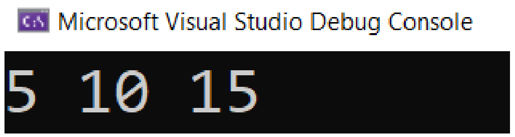
  int myarray [40];

  it would be enough to write a call like this:

  procedure (myarray);

Example – pass int array

```c
#include <stdio.h>
#include <stdlib.h>

void printarray(int arg[], int length);

void main()
{
    int firstarray[] = { 5, 10, 15 };
    printarray(firstarray, 3);
}

void printarray(int arg[], int length)
{
    for (int n = 0; n < length; n++){
        printf("%d ", arg[n]);
    }
}
```

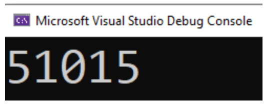


```
5 10 15
```

Formatting int display

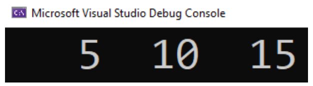
- We can format the output of ints using %**x**d, where ‘x’ is the number of characters to print out with the int.
- If int requires 2 characters, e.g. ‘10’, then %3d will print 1 space before 10, i.e. ‘ 10’.
- This works for floats too!

```c
printf("%d", arg[n]);
```


Microsoft Visual Studio Debug Console

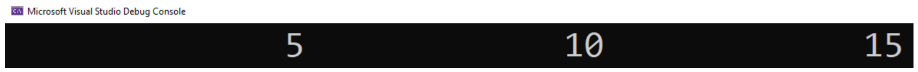
```
51015
```

```c
printf("%4d", arg[n]);
```


Microsoft Visual Studio Debug Console

```
   5  10  15
```

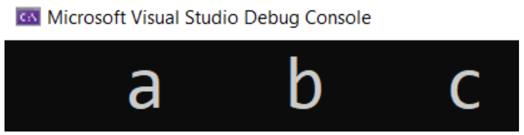
```c
printf("%15d", arg[n]);
```


Microsoft Visual Studio Debug Console

```
              5             10             15
```

Pass char array

```c
#include <stdio.h>
#include <stdlib.h>

void printarray(char arg[], int length);

void main()
{
    char firstarray[] = { 'a','b','c' };

    printarray(firstarray, 3);
}

void printarray(char arg[], int length)
{
    for (int n = 0; n < length; n++){
        printf("%4c", arg[n]);
    }
}
```



```
   a   b   c
```

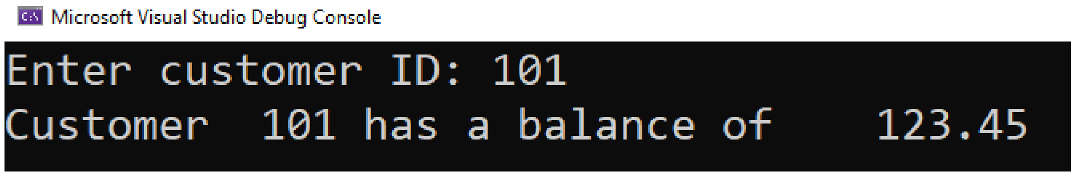
Passing Array Example

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

double printBalance(double bals[], int ids[], int len, int custID);

void main()
{
    int custID[10] = { 313,453,502,101,892 };
    double custBal[10] = { 0.00, 56.56, 34.56, 123.45, 9.05 };
    int numCust = 5;
    double balance;
    int ID;

    printf("Enter customer ID: ");
    scanf_s("%d", &ID);

    balance = printBalance(custBal, custID, numCust, ID);
    if (balance == -10000000.00){
        printf("Error - Customer not found \n");
    }
}
```

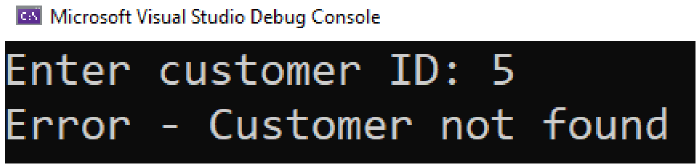
Passing Array Example (*continued*)

```
double printBalance(double bals[], int ids[], int numCust, int idSearch){
    for (int ctr = 0; ctr < numCust; ctr++){
        if (ids[ctr] == idSearch){
            printf("Customer %4d has a balance of %9.2f \n", idSearch, bals[ctr]);
            return bals[ctr];
        }
    }
    return -10000000.00;
}
```

Passing Array Example - Output



```
Enter customer ID: 101
Customer  101 has a balance of    123.45
```

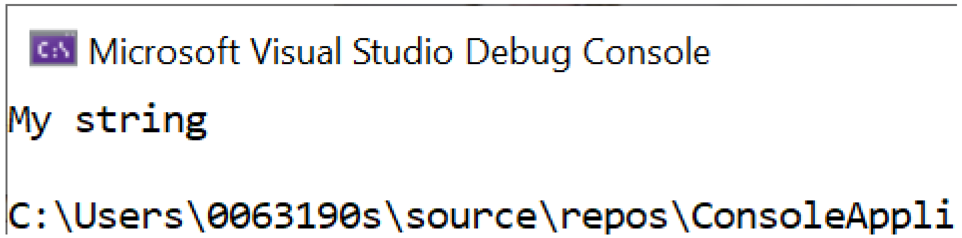


```
Enter customer ID: 5
Error - Customer not found
```

Pass string as parameter

- A string in C is just a char array
- Here we pass a string to the function puts()

```
#include <stdio.h>

void main()
{
    char string1[] = "My string";

    puts(string1);
}
```

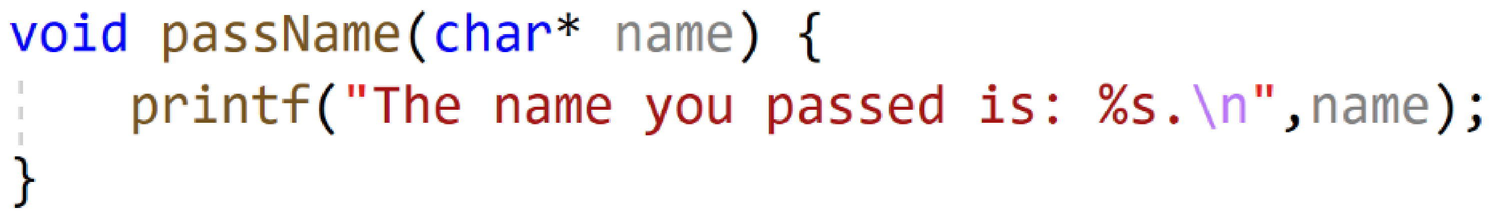


Microsoft Visual Studio Debug Console

```
My string

C:\Users\0063190s\source\repos\ConsoleAppli
```

Pass String Using Pointers

- We saw already how we can pass a string using pointers from week 8b.

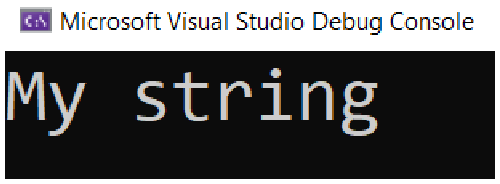


```c
void passName(char* name) {
    printf("The name you passed is: %s.\n",name);
}
```

- Here char* is a pointer to the first character in the string name.
- This is the accepted convention for passing strings to functions in C.
- You can do it without pointers as we will see next..

Passing Strings Without Pointers

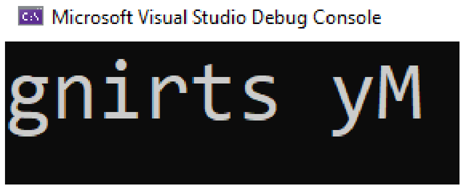
```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

void printString(char s[]);

void main(){
    char string1[] = "My string";
    printString(string1);
}

void printString(char str[])
{
    printf("%s\n",str);
}
```



```
My string
```

Passing Strings Example

```c
#include <stdio.h>

void printStringBackwards(char s[]);

void main(){
    char string1[] = "My string";
    printStringBackwards(string1);
}

void printStringBackwards(char str[]){
    int len = 0, i = 0;

    len = strlen(str);
    i = len - 1;

    while (i >= 0)
    {
        printf("%c", str[i]);
        i--;
    }
    printf("\n");
}
```

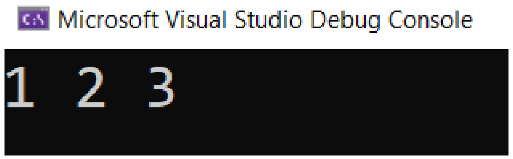


```
gnirts yM
```

Passing Arrays with Pointers

- Also, you can use pointers with arrays of ints too…

```c
void printIntArrPoint(int* myIntArray, int len);

void main(){
    int tempInts[3] = { 1, 2, 3 };
    printIntArrPoint(tempInts,3);
}

void printIntArrPoint(int* myIntArray, int len) {
    for (int i = 0; i < len;i++) {
        printf("%d ",myIntArray[i]);
    }
    printf("\n");
}
```



```
1 2 3
```

- Don’t worry about this for now.
- We have not yet covered pointers so this won’t make much sense until we do.

RANDOM NUMBERS

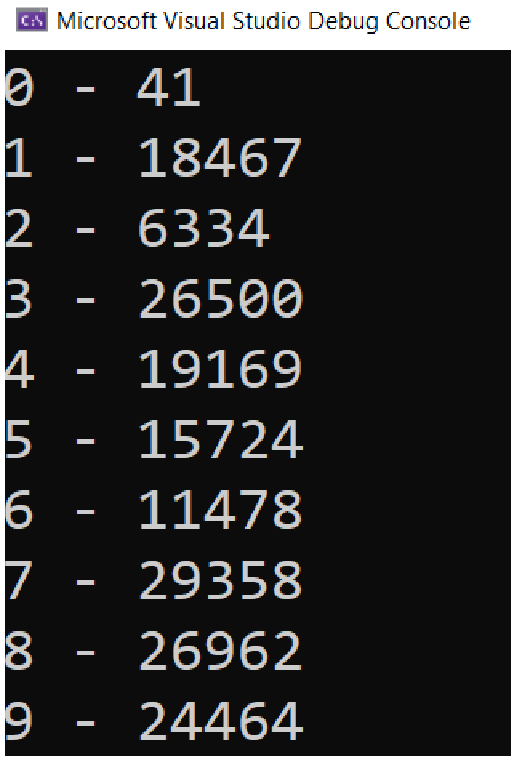
Random numbers

- Sometime we need to generate random numbers, e.g. for games.
- To do this we can use the rand() function, which returns a random number from 0 to 32767 (this upper limit will vary depending on the system and implementation of the rand function)
- You will need to include the stdlib.h header file

```
#include <stdlib.h>
```

Rand() Function

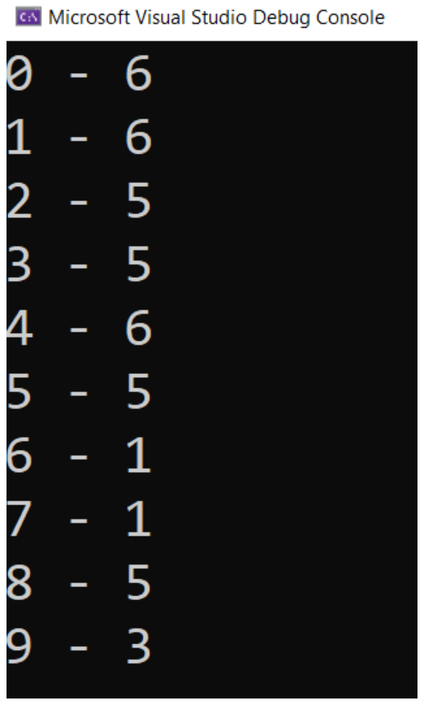
```c
#include <stdio.h>
#include <stdlib.h>

void main()
{
    int randNum;
    int i;

    for (i = 0; i < 10; i++)
    {
        randNum = rand();
        printf("%d - %d\n", i,
        randNum);
    }
}
```



```
0 - 41
1 - 18467
2 - 6334
3 - 26500
4 - 19169
5 - 15724
6 - 11478
7 - 29358
8 - 26962
9 - 24464
```

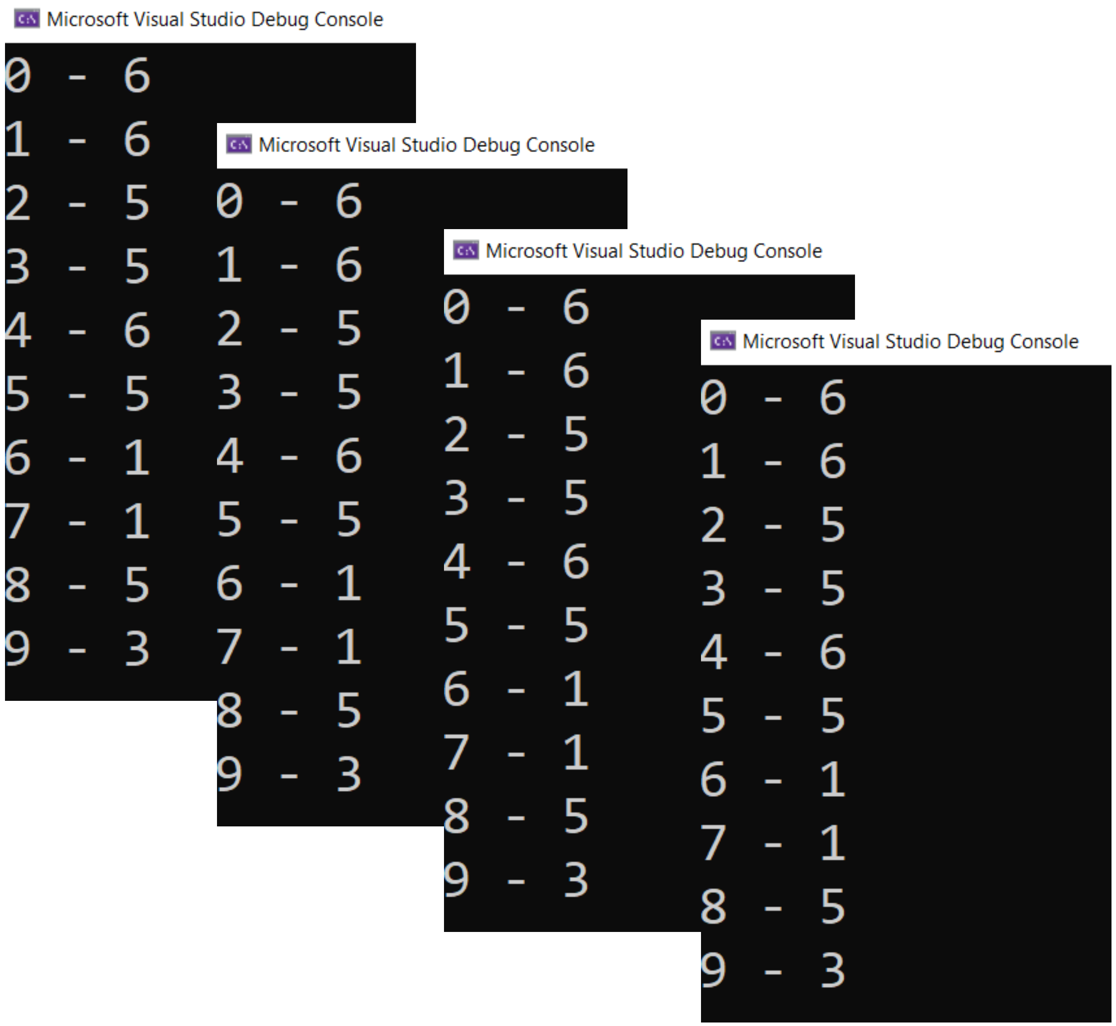
Dice – random numbers from 1 to 6

```c
#include <stdio.h>
#include <stdlib.h>

void main()
{
    int randNum;
    int i;

    for (i = 0; i < 10; i++)
    {
        // %6 gives numbers from 0 to 5, so add 1 to get
        // numbers from 1 to 6
        randNum = rand() % 6 + 1;
        printf("%d - %d\n", i, randNum);
    }
}
```



```
0 - 6
1 - 6
2 - 5
3 - 5
4 - 6
5 - 5
6 - 1
7 - 1
8 - 5
9 - 3
```

Trouble is…every time you run it – you get the same numbers!



Microsoft Visual Studio Debug Console

```
0 - 6
1 - 6
2 - 5
3 - 5
4 - 6
5 - 5
6 - 1
7 - 1
8 - 5
9 - 3
```

Microsoft Visual Studio Debug Console

```
0 - 6
1 - 6
2 - 5
3 - 5
4 - 6
5 - 5
6 - 1
7 - 1
8 - 5
9 - 3
```

Microsoft Visual Studio Debug Console

```
0 - 6
1 - 6
2 - 5
3 - 5
4 - 6
5 - 5
6 - 1
7 - 1
8 - 5
9 - 3
```

Microsoft Visual Studio Debug Console

```
0 - 6
1 - 6
2 - 5
3 - 5
4 - 6
5 - 5
6 - 1
7 - 1
8 - 5
9 - 3
```

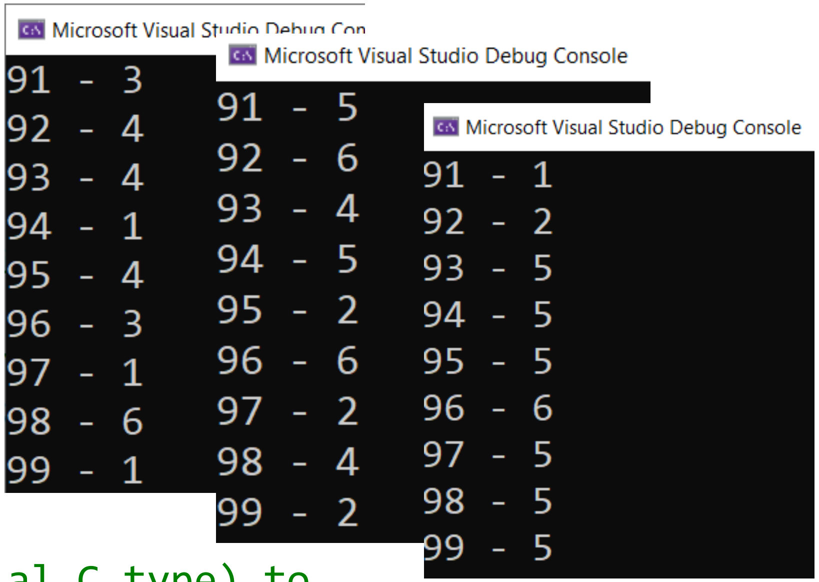
Increasing randomness

- rand() will repeat the same set of random numbers if repeated
    - Basically it's an algorithm (has a starting point)
- Programmers often use srand() to generate numbers that are *more* random
    - Giving the algorithm a different starting point!
    - This is called *seeding* the algorithm
    - The trick is to use a different seed every time
    - This is usually accomplished by using the current time as the seed

Increasing randomness

- Programmers often use srand() to generate numbers that are ***more*** random
- Generating truly random numbers is actually non-trivial and is an active research area.
  - Some researchers use physical phenomena, e.g. radioactive decay to generate ***truly*** random numbers.
    - Philosophical debate as to whether or not anything is random…
      - This is outside of the scope of this course ☺
- The algorithms used in computer programs are called pseudorandom number generators.
  - Not truly random but good enough for what they are needed for.

Random Seed


Microsoft Visual Studio Debug Con

```
91 - 3
92 - 4
93 - 4
94 - 1
95 - 4
96 - 3
97 - 1
98 - 6
99 - 1
```

Microsoft Visual Studio Debug Console

```
91 - 5
92 - 6
93 - 4
94 - 5
95 - 2
96 - 6
97 - 2
98 - 4
99 - 2
```

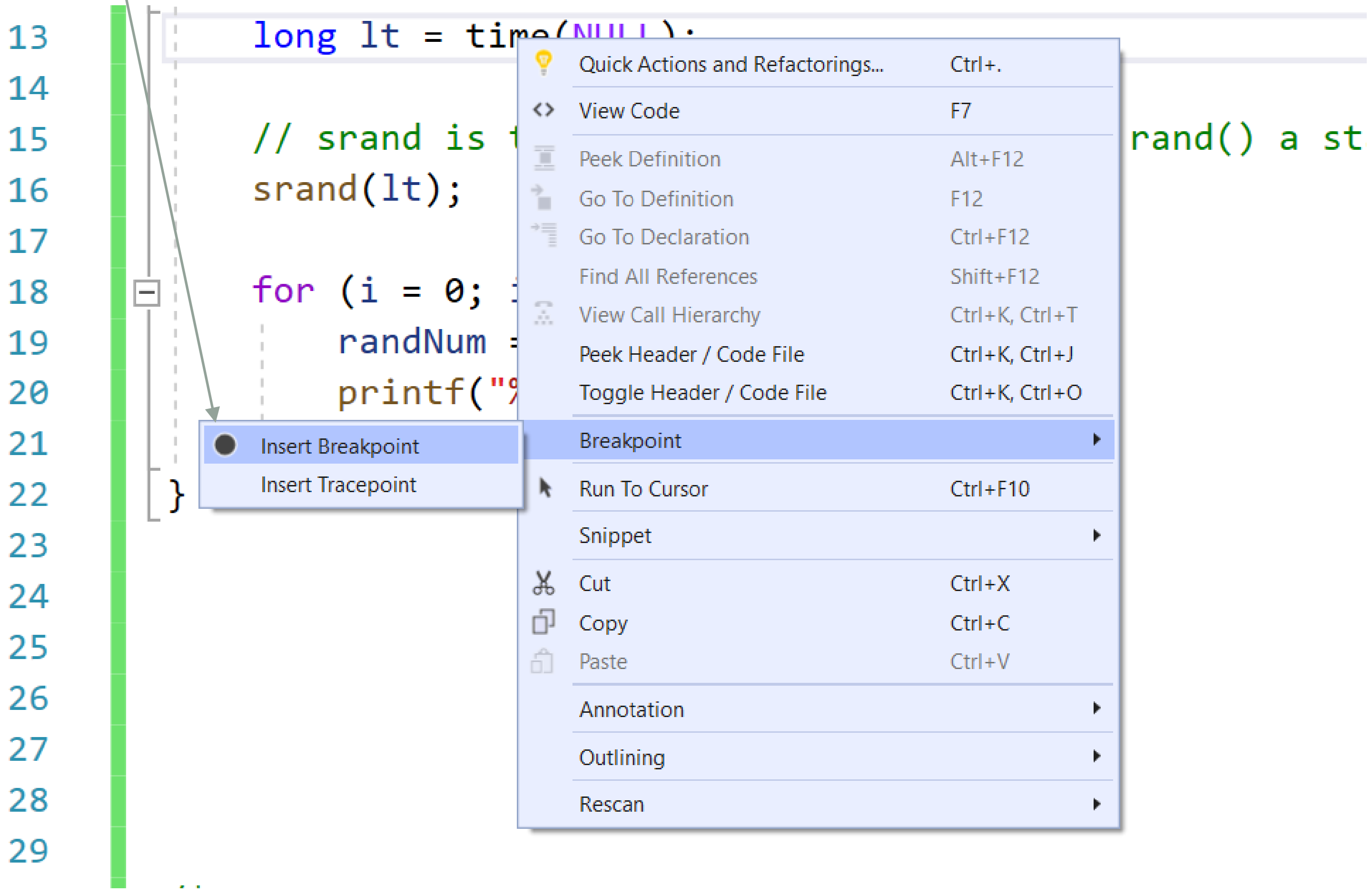
Microsoft Visual Studio Debug Console

```
91 - 1
92 - 2
93 - 5
94 - 5
95 - 5
96 - 6
97 - 5
98 - 5
99 - 5
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void main()
{
    int randNum;

    // this converts a time structure (special C type) to
    // a long integer
    // lt will contain the number of seconds since 1 Jan 1970, 00:00:00
    long lt = time(NULL);

    // srand is the seeding function - gives rand() a starting point
    srand(lt);

    for (int i = 0; i < 100; i++){
        randNum = rand() % 6 + 1;
        printf("%d - %d\n", i, randNum);
    }
}
```

Breakpoints

- Breakpoints are a useful way of seeing what is happening in your code:

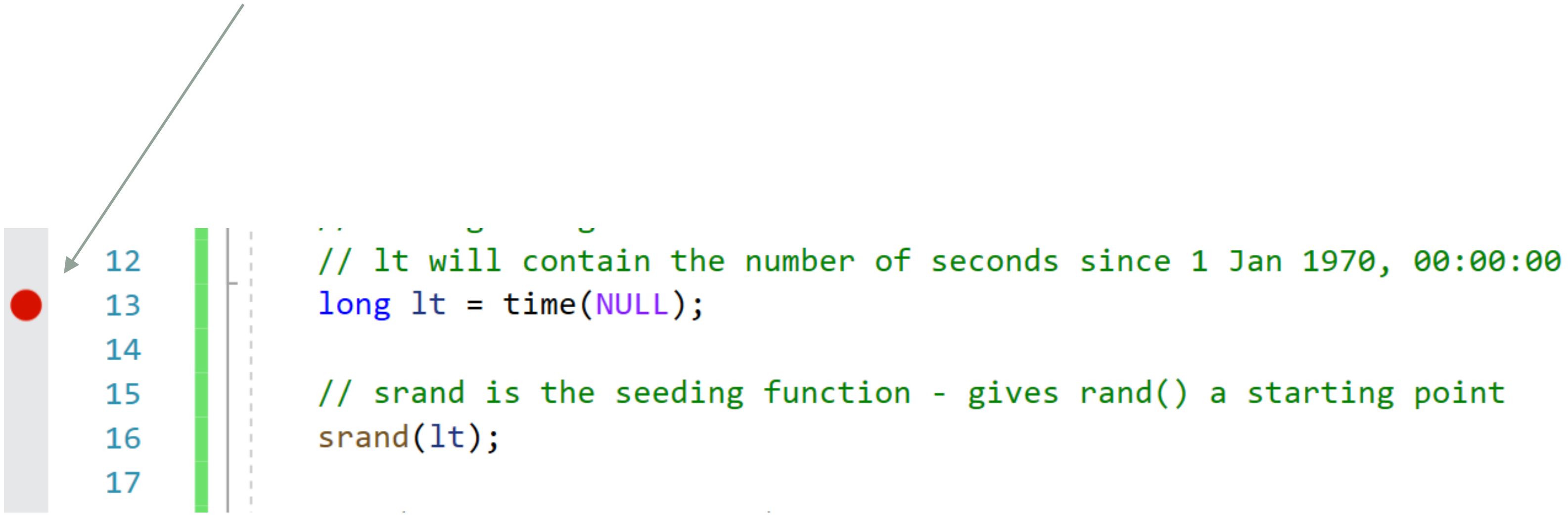


```
    long lt = time(NULL);

    // srand is t                                      rand() a st
    srand(lt);

    for (i = 0; i
        randNum =
        printf("%

}
```

Quick Actions and Refactorings... Ctrl+.
View Code F7
Peek Definition Alt+F12
Go To Definition F12
Go To Declaration Ctrl+F12
Find All References Shift+F12
View Call Hierarchy Ctrl+K, Ctrl+T
Peek Header / Code File Ctrl+K, Ctrl+J
Toggle Header / Code File Ctrl+K, Ctrl+O
Breakpoint
Insert Breakpoint
Insert Tracepoint
Run To Cursor Ctrl+F10
Snippet
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Annotation
Outlining
Rescan

Breakpoints

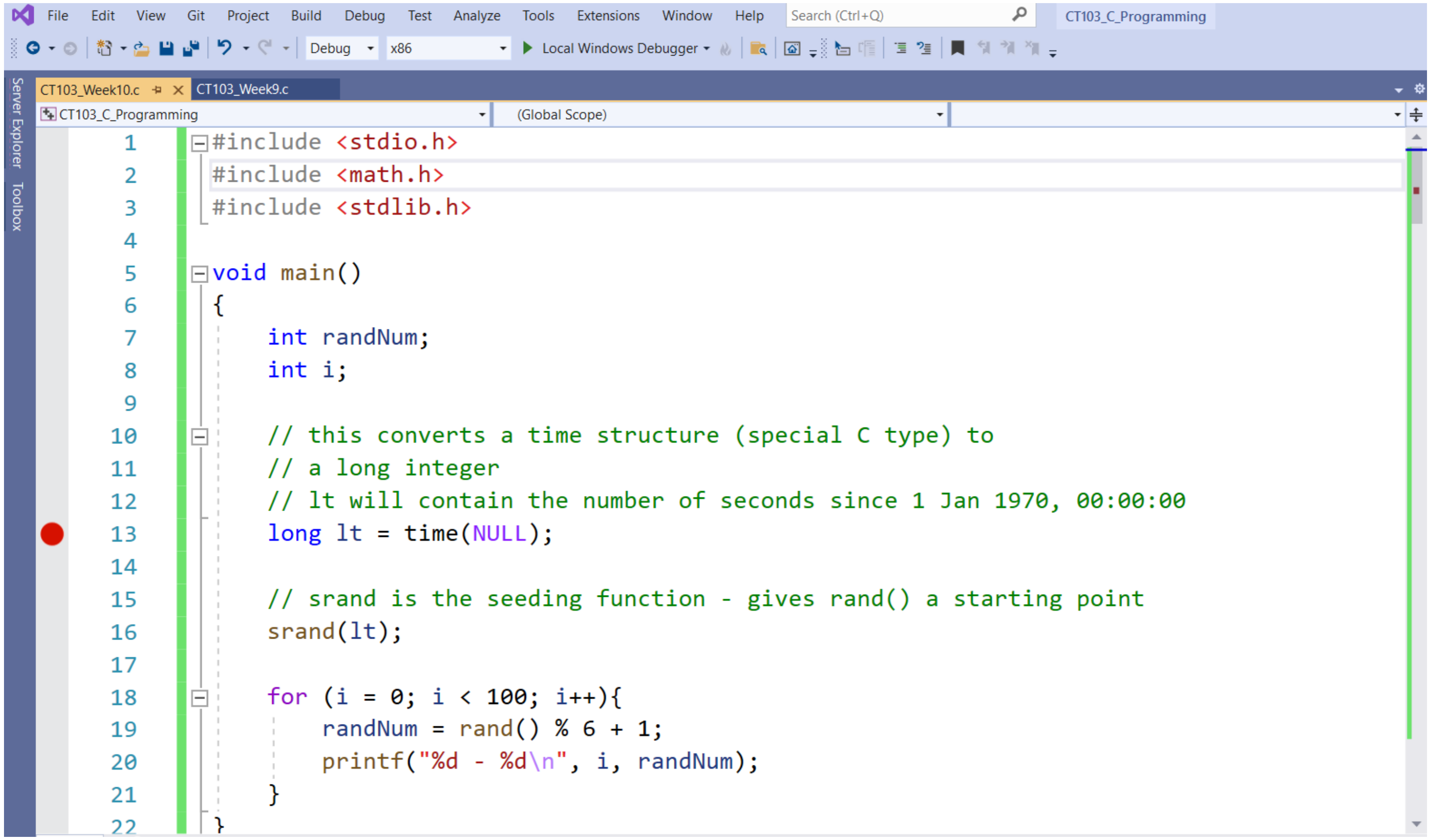
- The breakpoint will then look like this:



```
// lt will contain the number of seconds since 1 Jan 1970, 00:00:00
long lt = time(NULL);

// srand is the seeding function - gives rand() a starting point
srand(lt);
```

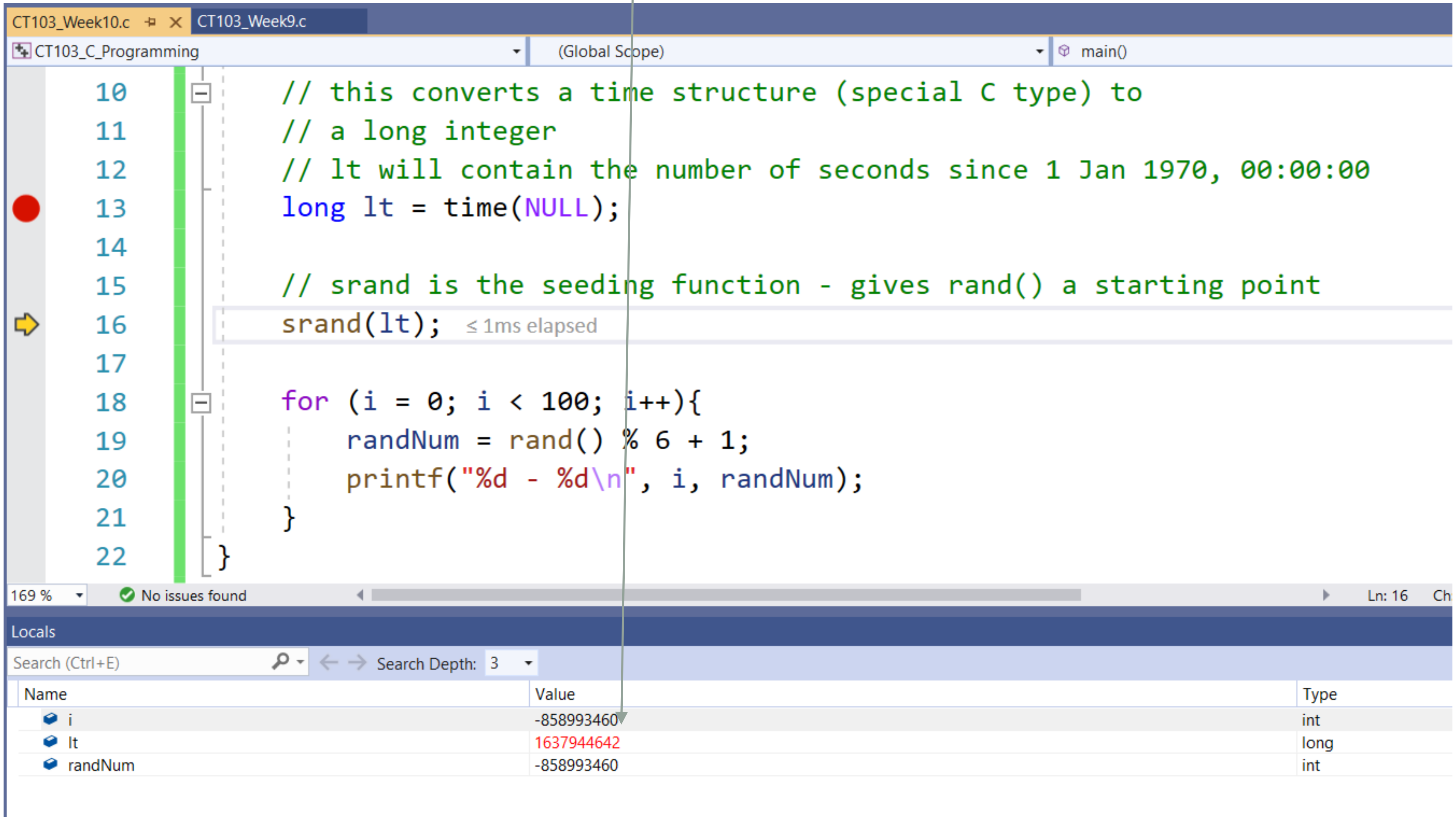
Use the debugger to check it out

- Use the debugger to stop the code



```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

void main()
{
    int randNum;
    int i;

    // this converts a time structure (special C type) to
    // a long integer
    // lt will contain the number of seconds since 1 Jan 1970, 00:00:00
    long lt = time(NULL);

    // srand is the seeding function - gives rand() a starting point
    srand(lt);

    for (i = 0; i < 100; i++){
        randNum = rand() % 6 + 1;
        printf("%d - %d\n", i, randNum);
    }
}
```

- Use f10 to step over line 13
- You can see the value of lt from the watch window



```
10      // this converts a time structure (special C type) to
11      // a long integer
12      // lt will contain the number of seconds since 1 Jan 1970, 00:00:00
13      long lt = time(NULL);
14
15      // srand is the seeding function - gives rand() a starting point
16      srand(lt);  ≤ 1ms elapsed
17
18      for (i = 0; i < 100; i++){
19          randNum = rand() % 6 + 1;
20          printf("%d - %d\n", i, randNum);
21      }
22  }
```

Name	Value	Type
i	-858993460	int
lt	1637944642	long
randNum	-858993460	int

EXAMPLE PROBLEM

Coin Toss Problem

- Write software to do 10 games of “coin toss”.
- This game consists of tossing a coin and seeing if it came up heads or tails.
- Each game should toss the coin 1000 times.
- Record the number of heads and tails in each game.
- What percentage of tosses were heads and what were tails for each game?
- Is this what you expect?

Coin Toss Problem

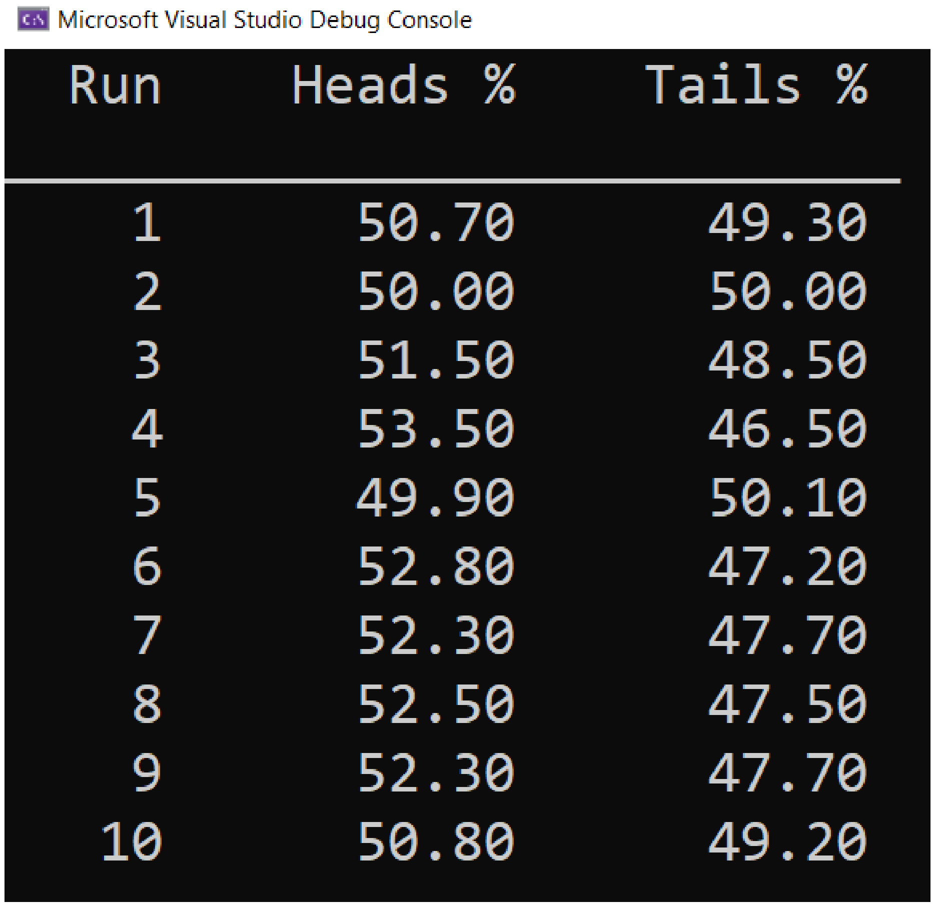
- Go to C program solution.

Coin Toss Problem

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define NUMTOSSES 1000

void main()
{
         int coin, i, j;
         int totalHeads;
         double headsPercent, tailsPercent;

         printf("%5s %10s %10s\n____________________________\n", "Run", "Heads %", "Tails
                  %");

         for (i = 1; i <= 10; i++){
                  srand(i * time(NULL));
                  totalHeads = 0;
                           for (j = 1; j <= NUMTOSSES; j++){
                                    /* coin: 1 is heads, 0 is tails */
                                    coin = rand() % 2;
                                    totalHeads += coin;
                           }

                  headsPercent = (double)totalHeads * 100.0 / NUMTOSSES;
                  tailsPercent = 100.0 - headsPercent;
                  printf("%5d %10.2lf %10.2lf\n", i, headsPercent, tailsPercent);
         }
}
```

Coin Toss Problem

- C Program Output:



```
   Run       Heads %      Tails %
----------------------------------
     1         50.70        49.30
     2         50.00        50.00
     3         51.50        48.50
     4         53.50        46.50
     5         49.90        50.10
     6         52.80        47.20
     7         52.30        47.70
     8         52.50        47.50
     9         52.30        47.70
    10         50.80        49.20
```

Notice anything ?

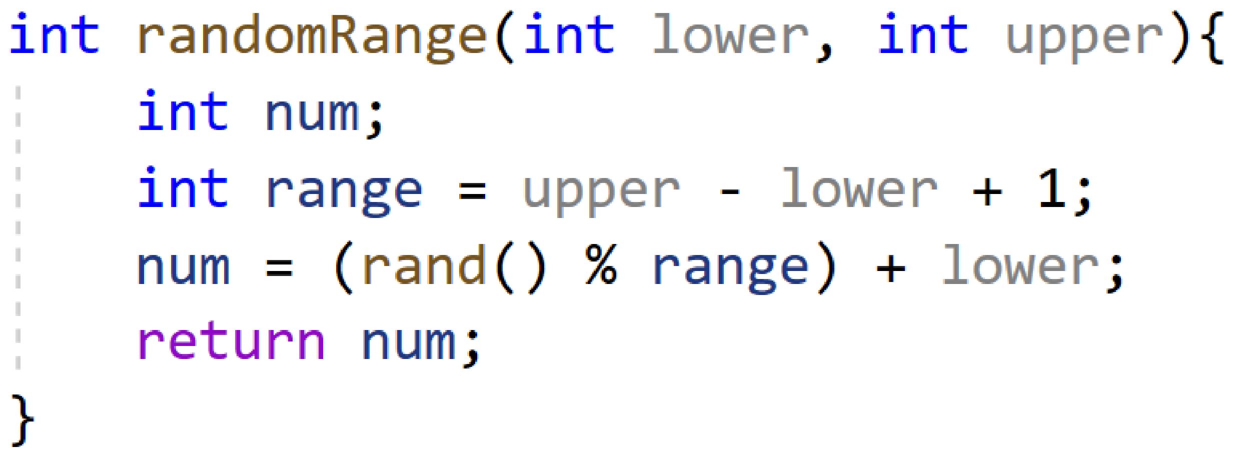
- What about this line:

- `srand(i * time(NULL));`

- Because the time take to do 1000 tosses might be less than 1 second, the time value would not change, so we multiple it by i to be sure that each round has a different seed!

Random numbers in a range

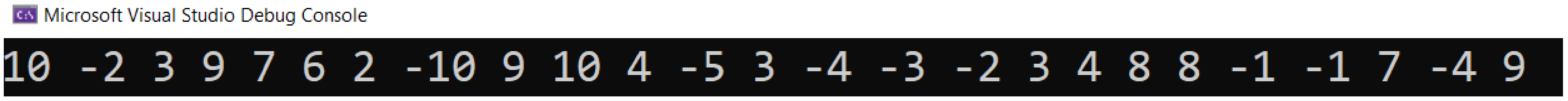
- This function will generate a random number between some lower and upper range (inclusive)



```
int randomRange(int lower, int upper){
    int num;
    int range = upper - lower + 1;
    num = (rand() % range) + lower;
    return num;
}
```

Using it

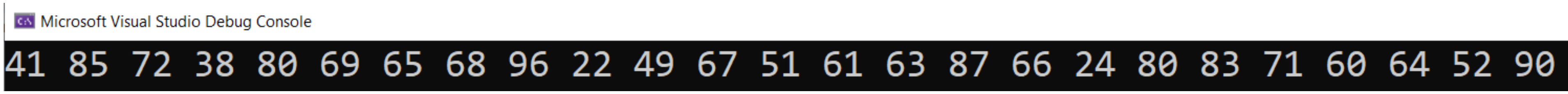
```c
for (int i = 0; i < 25; i++){
    printf("%d ", randomRange(-10, 10));
}
```



Microsoft Visual Studio Debug Console

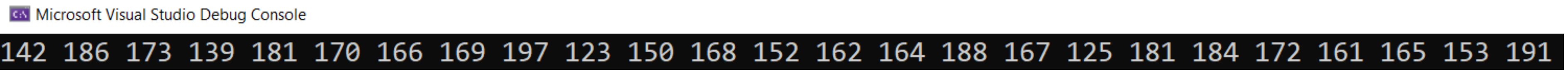
```
10 -2 3 9 7 6 2 -10 9 10 4 -5 3 -4 -3 -2 3 4 8 8 -1 -1 7 -4 9
```

```
// 0 to 100
for (int i = 0; i < 25; i++){
    printf("%d ", randomRange(0, 100));
}
```



```
41 85 72 38 80 69 65 68 96 22 49 67 51 61 63 87 66 24 80 83 71 60 64 52 90
```

```c
// 101 to 201
for (int i = 0; i < 25; i++){
    printf("%d ", randomRange(101, 201));
}
```



Microsoft Visual Studio Debug Console

```
142 186 173 139 181 170 166 169 197 123 150 168 152 162 164 188 167 125 181 184 172 161 165 153 191
```

PROGRAMMING

CT103

Week 10b

Sign in on Blackboard

- Please sign in on blackboard for CT103.
- I will leave the sign in option open for an hour after the lecture.

CT103 Lab Update

- There will be no more graded assignments in semester 1 for CT103 after the week 10 lab (30/11/21).
- The labs will continue running until and including week 12 (Tuesday 14/12/21). Attendance is optional at week 11 and 12 labs.
- These sessions will be a mix of:
  - Q&A where you can ask the instructors about anything you are unsure of.
  - A worksheet (not graded) so you can get more practice.

Lecture Content

- Last lecture (Week 9b):
  - Passing arrays to functions
  - Generating random numbers
  - Breakpoints
  - **Example C program – We will go through this today**.
- Today’s lecture (Week 10a):
  - Structures
  - Arrays of structures
  - Functions and structures
  - Example C program

STRUCTURES

Structures

- We can use arrays to hold multiple data items of the same type, e.g. ints, chars, etc.

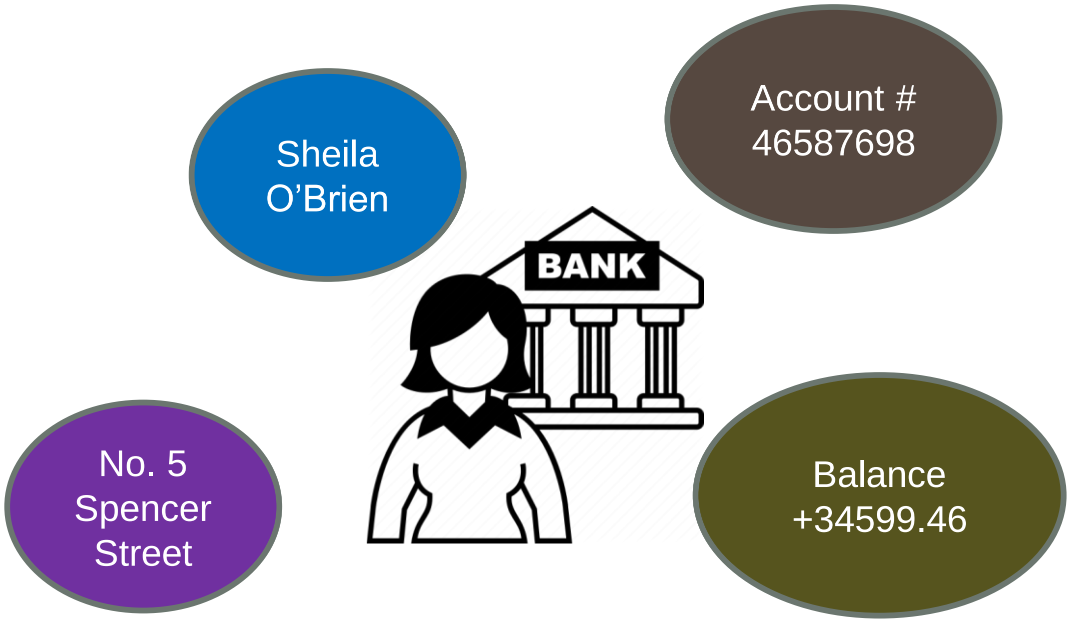
```
int myArrayInt[] = {1,2,3};
```

- What if we want to hold multiple data items that are of different types?

- We can use **structures**!

Structures

- Let’s say you keep several pieces of information about customers, for example:
  - Name
  - Account number
  - Balance
  - Address
  - ….



Storing Related Data

- If we only had a bank customer balance to record, we could use an array:

```
float bankCustBal[] = { 100.20,502.34,20.0 };
```

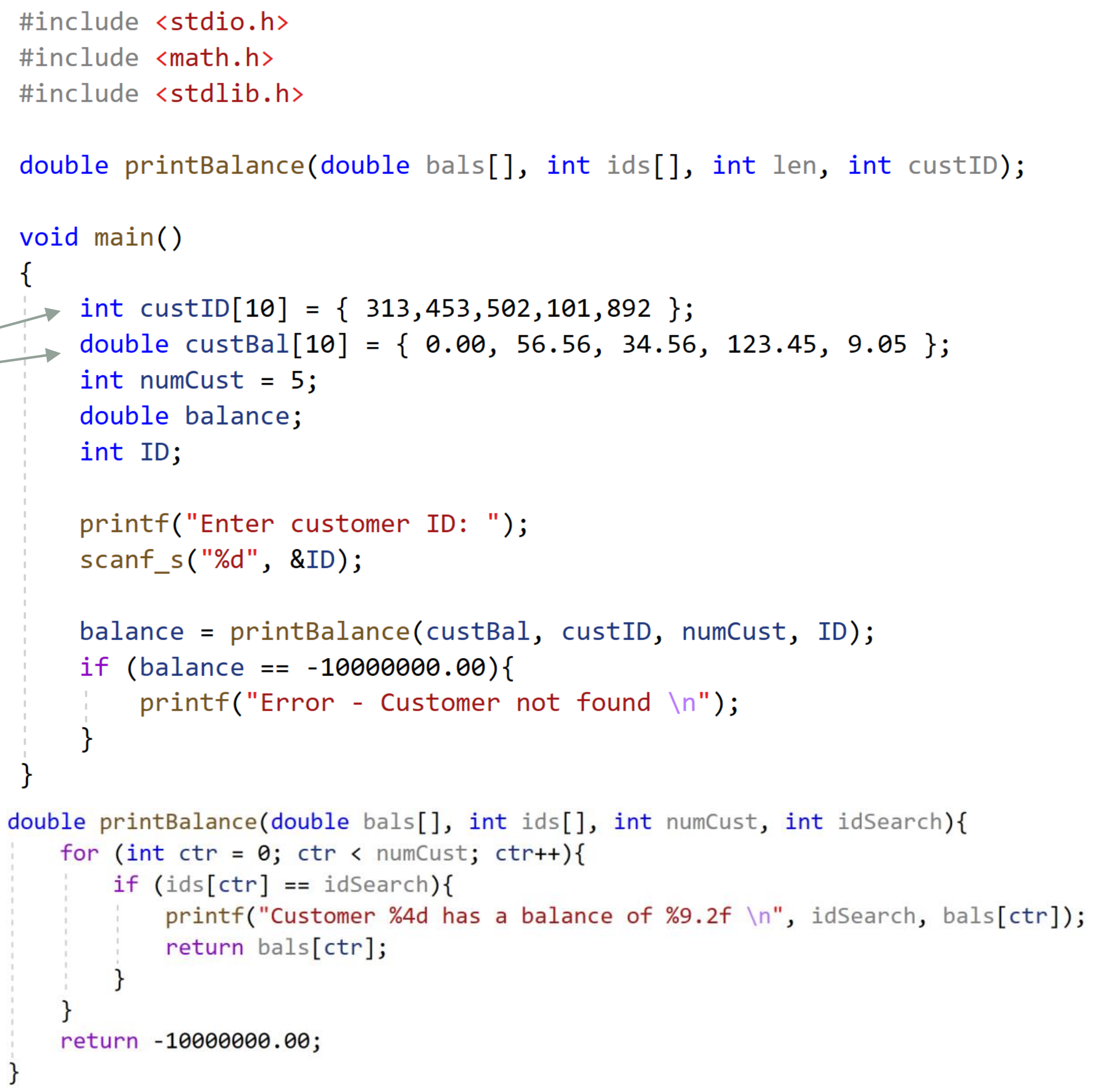
- We could also create a separate array to store bank customer IDs:

```
int bankCustID[] = { 33, 25, 98};
```

- This would work fine, we would just need to ensure that all arrays are ordered in the same way.

Week 10a Example

- The example from Monday did exactly this.
- We have 2 arrays storing related information about customers.



```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

double printBalance(double bals[], int ids[], int len, int custID);

void main()
{
    int custID[10] = { 313,453,502,101,892 };
    double custBal[10] = { 0.00, 56.56, 34.56, 123.45, 9.05 };
    int numCust = 5;
    double balance;
    int ID;

    printf("Enter customer ID: ");
    scanf_s("%d", &ID);

    balance = printBalance(custBal, custID, numCust, ID);
    if (balance == -10000000.00){
        printf("Error - Customer not found \n");
    }
}
double printBalance(double bals[], int ids[], int numCust, int idSearch){
    for (int ctr = 0; ctr < numCust; ctr++){
        if (ids[ctr] == idSearch){
            printf("Customer %4d has a balance of %9.2f \n", idSearch, bals[ctr]);
            return bals[ctr];
        }
    }
    return -10000000.00;
}
```

Structures

- We can use **structures** to store this information in a more organised manner.
- We declare a structure using the keyword **struct**, as follows:

```
struct customer
{
    char name[20];
    int accountNumber;
    float balance;
    char address[20];
};
```

Creating a struct variable

```
struct customer customer1;
```

- You reference the members of a struct using the . notation:

```
strcpy(customer1.name, "Richie Rich");

customer1.accountNumber = 101;
```

Structure Example

```
#include <stdio.h>
#include <string.h>

struct customer{
    char name[20];
    int accountNumber;
    float balance;
    char address[20];
};

void main(){
    struct customer customer1;

    strcpy_s(customer1.name,20, "Richie Rich");
    customer1.accountNumber = 101;
    customer1.balance = 9875234.00;
    Strcpy_s(customer1.address,20, "Millionaire Drive");
}
```

typedef

- This is a bit long-winded:
  - `struct customer customer1;`
- We can use ***typedef*** to shorten this.
- *typedef* is used to create user-defined types.

Structures using typedef

```
#include <stdio.h>
#include <string.h>
struct customer{
       char name[20];
       int accountNumber;
       float balance;
       char address[20];
};

typedef struct customer customer; // creating new type - customer

void main(){
       //struct customer customer1;
       customer customer1; // creating variable of type customer

       strcpy_s(customer1.name,20, "Richie Rich");
       customer1.accountNumber = 101;
       customer1.balance = 9875234.00;
       strcpy_s(customer1.address,20, "Millionaire Drive");
}
```

This also works

```
#include <stdio.h>
#include <string.h>

typedef struct customer
{
    char name[20];
    int accountNumber;
    float balance;
    char address[20];
} customer;

void main()
{
    customer customer1;

    strcpy_s(customer1.name,20, "Richie Rich");
    customer1.accountNumber = 101;
    customer1.balance = 9875234.00;
    strcpy_s(customer1.address,20, "Millionaire Drive");
}
```

As does this

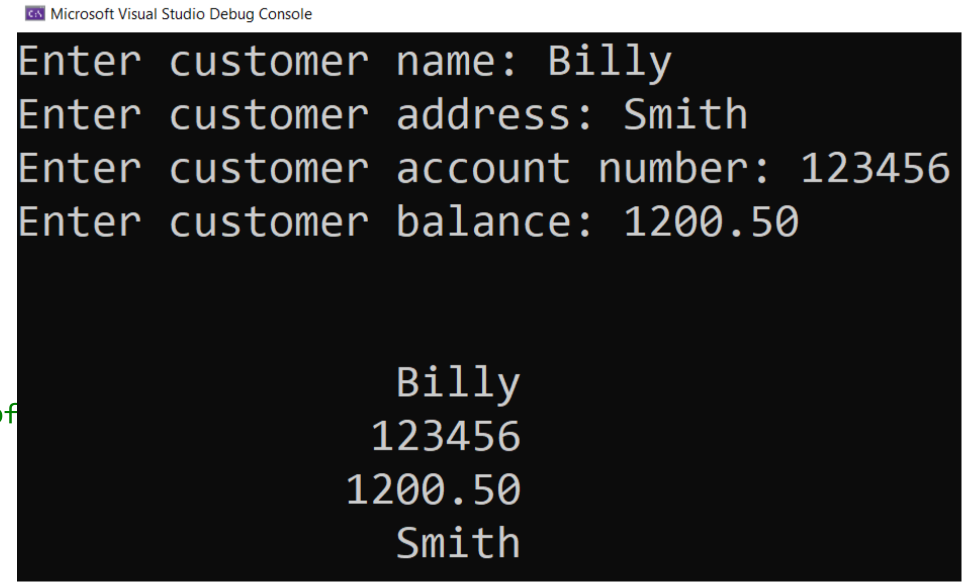
```
#include <stdio.h>
#include <string.h>

typedef struct
{
    char name[20];
    int accountNumber;
    float balance;
    char address[20];
} customer;

void main()
{
    customer customer1;

    strcpy_s(customer1.name,20, "Richie Rich");
    customer1.accountNumber = 101;
    customer1.balance = 9875234.00;
    strcpy_s(customer1.address,20, "Millionaire Drive");
}
```

Passing Data into Structures

```c
#include <stdio.h>
#include <string.h>

typedef struct {
      char name[20];
      int accountNumber;
      float balance;
      char address[20];
} customer;

void main(){
      customer customer1; // creating variable of type customer

      printf("Enter customer name: ");
      gets(customer1.name);

      printf("Enter customer address: ");
      gets(customer1.address);

      printf("Enter customer account number: ");
      scanf_s("%d", &customer1.accountNumber);

      printf("Enter customer balance: ");
      scanf_s("%f", &customer1.balance);

      printf("\n\n%20s\n%20d\n%20.2lf\n%20s\n", customer1.name,
      customer1.accountNumber,
      customer1.balance, customer1.address);
}
```

Passing Data into Structures

```c
#include <stdio.h>
#include <string.h>

typedef struct {
	char name[20];
	int accountNumber;
	float balance;
	char address[20];
} customer;

void main(){
	customer customer1; // creating variable of

	printf("Enter customer name: ");
	gets(customer1.name);

	printf("Enter customer address: ");
	gets(customer1.address);

	printf("Enter customer account number: ");
	scanf_s("%d", &customer1.accountNumber);

	printf("Enter customer balance: ");
	scanf_s("%f", &customer1.balance);

	printf("\n\n%20s\n%20d\n%20.2lf\n%20s\n", customer1.name,
	customer1.accountNumber,
	customer1.balance, customer1.address);
}
```



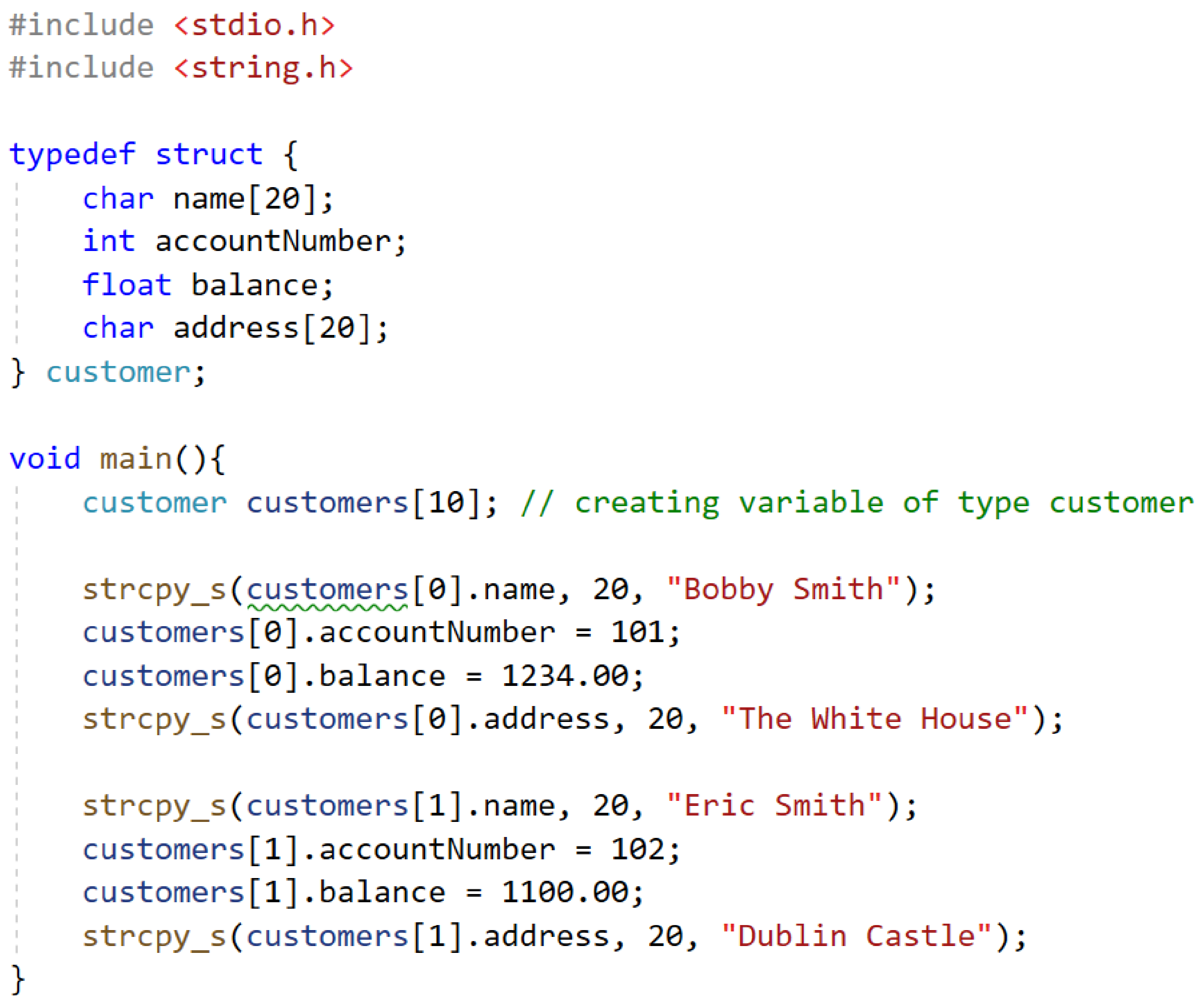
```
Enter customer name: Billy
Enter customer address: Smith
Enter customer account number: 123456
Enter customer balance: 1200.50


               Billy
              123456
             1200.50
               Smith
```

ARRAYS OF STRUCTS

Arrays of Structs

- Just like we can create arrays of integers, we can also create **arrays of structs**.
- Lets have a look at how we might do this using the example from before.

Arrays of Structs



```c
#include <stdio.h>
#include <string.h>

typedef struct {
    char name[20];
    int accountNumber;
    float balance;
    char address[20];
} customer;

void main(){
    customer customers[10]; // creating variable of type customer

    strcpy_s(customers[0].name, 20, "Bobby Smith");
    customers[0].accountNumber = 101;
    customers[0].balance = 1234.00;
    strcpy_s(customers[0].address, 20, "The White House");

    strcpy_s(customers[1].name, 20, "Eric Smith");
    customers[1].accountNumber = 102;
    customers[1].balance = 1100.00;
    strcpy_s(customers[1].address, 20, "Dublin Castle");
}
```

STRUCTS AND FUNCTIONS

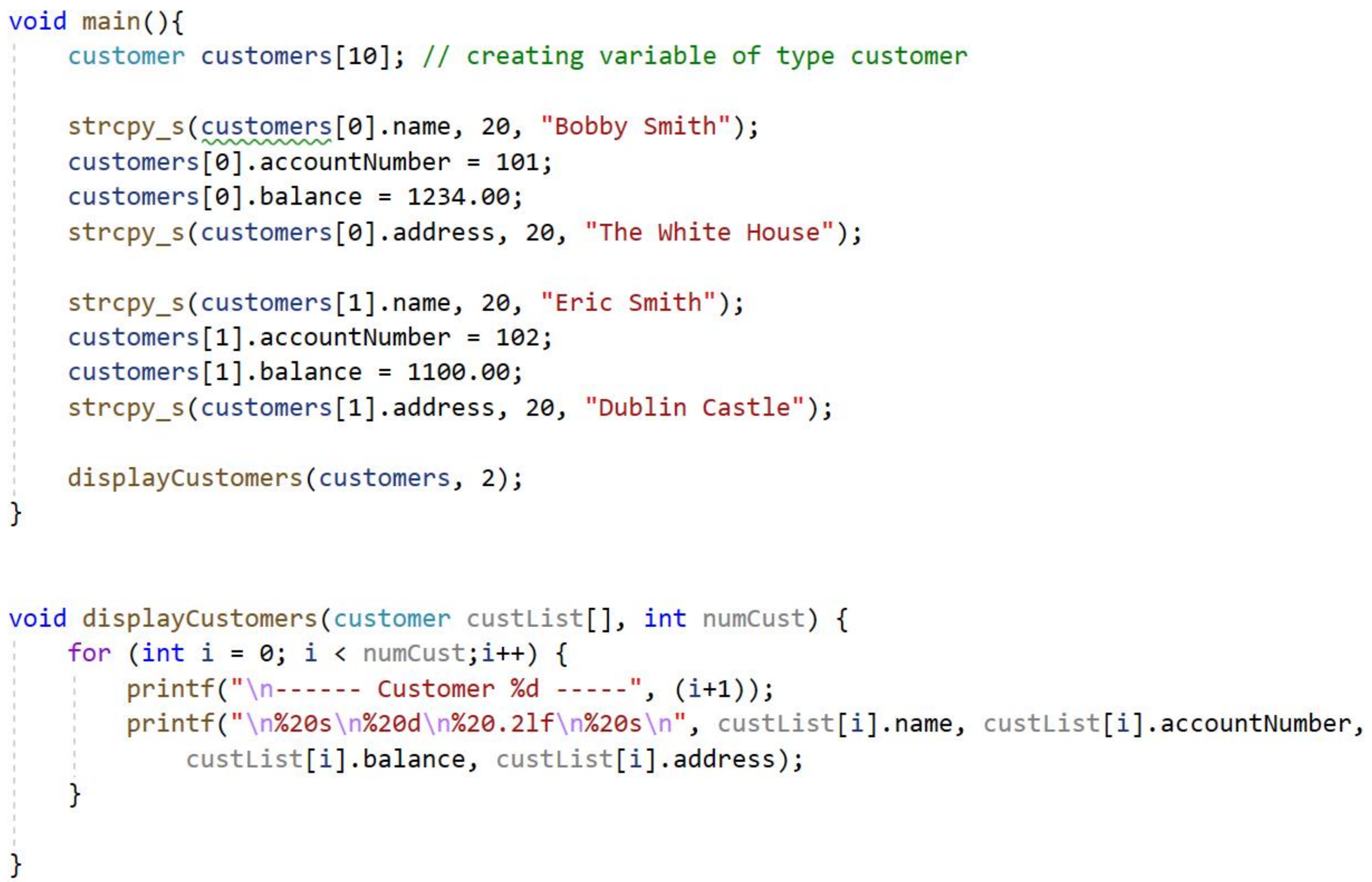
Structs and Functions

- How can I pass structs to functions?
- You can do this similar to how you would pass any other variable type!

Structs and Functions

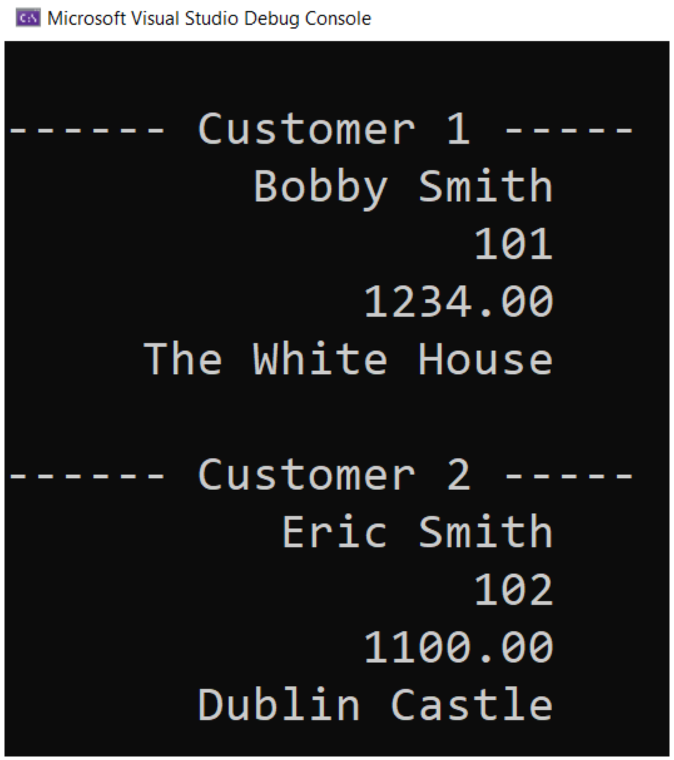
```
#include <stdio.h>
#include <string.h>

typedef struct {
    char name[20];
    int accountNumber;
    float balance;
    char address[20];
} customer;

void displayCustomers(customer custList[], int numCust);
```

Structs and Functions



```
void main(){
    customer customers[10]; // creating variable of type customer

    strcpy_s(customers[0].name, 20, "Bobby Smith");
    customers[0].accountNumber = 101;
    customers[0].balance = 1234.00;
    strcpy_s(customers[0].address, 20, "The White House");

    strcpy_s(customers[1].name, 20, "Eric Smith");
    customers[1].accountNumber = 102;
    customers[1].balance = 1100.00;
    strcpy_s(customers[1].address, 20, "Dublin Castle");

    displayCustomers(customers, 2);
}


void displayCustomers(customer custList[], int numCust) {
    for (int i = 0; i < numCust;i++) {
        printf("\n------ Customer %d -----", (i+1));
        printf("\n%20s\n%20d\n%20.2lf\n%20s\n", custList[i].name, custList[i].accountNumber,
            custList[i].balance, custList[i].address);
    }

}
```

Structs and Functions



```
------ Customer 1 -----
          Bobby Smith
                  101
              1234.00
      The White House

------ Customer 2 -----
           Eric Smith
                  102
              1100.00
        Dublin Castle
```

EXAMPLE PROBLEM

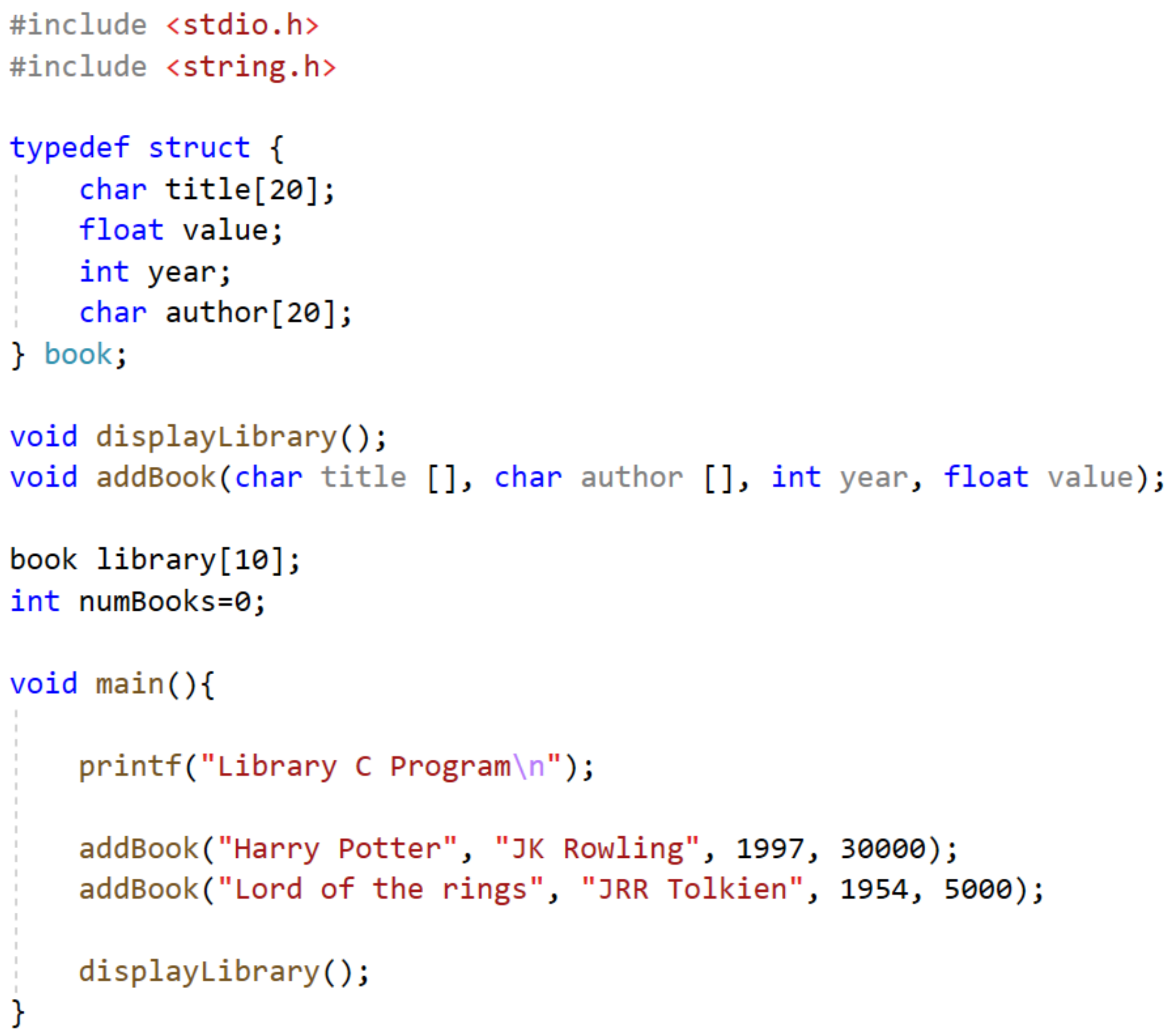
Library Software

- Write program to store books in a library:
- A book should be represented as a structure with:
  - Author.
  - Title.
  - Year.
  - Value.
- Create a global array to store each of the books in the library.
- Create a function to add a book to the library.
- Create another function to display the full library.

Library Software

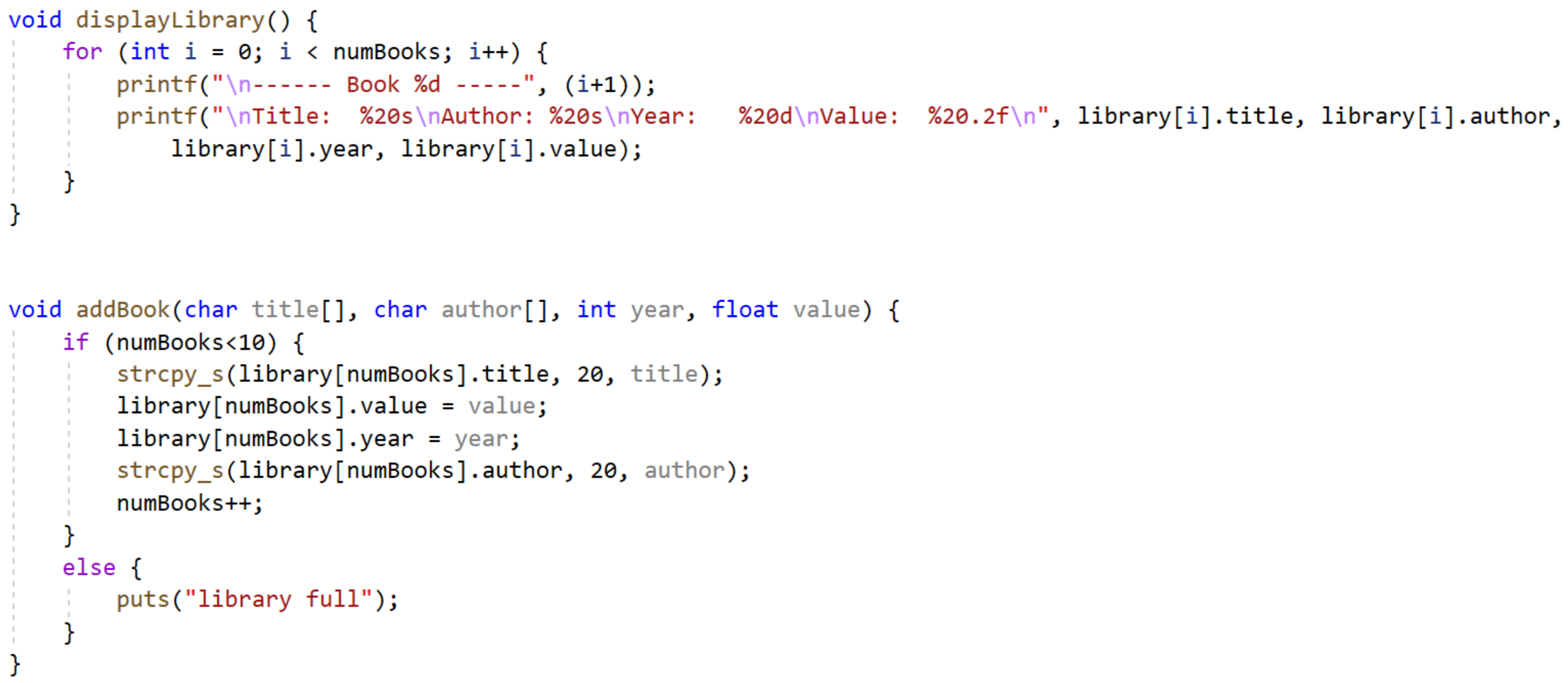
- Go to C program solution.

Library Software



```c
#include <stdio.h>
#include <string.h>

typedef struct {
    char title[20];
    float value;
    int year;
    char author[20];
} book;

void displayLibrary();
void addBook(char title [], char author [], int year, float value);

book library[10];
int numBooks=0;

void main(){

    printf("Library C Program\n");

    addBook("Harry Potter", "JK Rowling", 1997, 30000);
    addBook("Lord of the rings", "JRR Tolkien", 1954, 5000);

    displayLibrary();
}
```

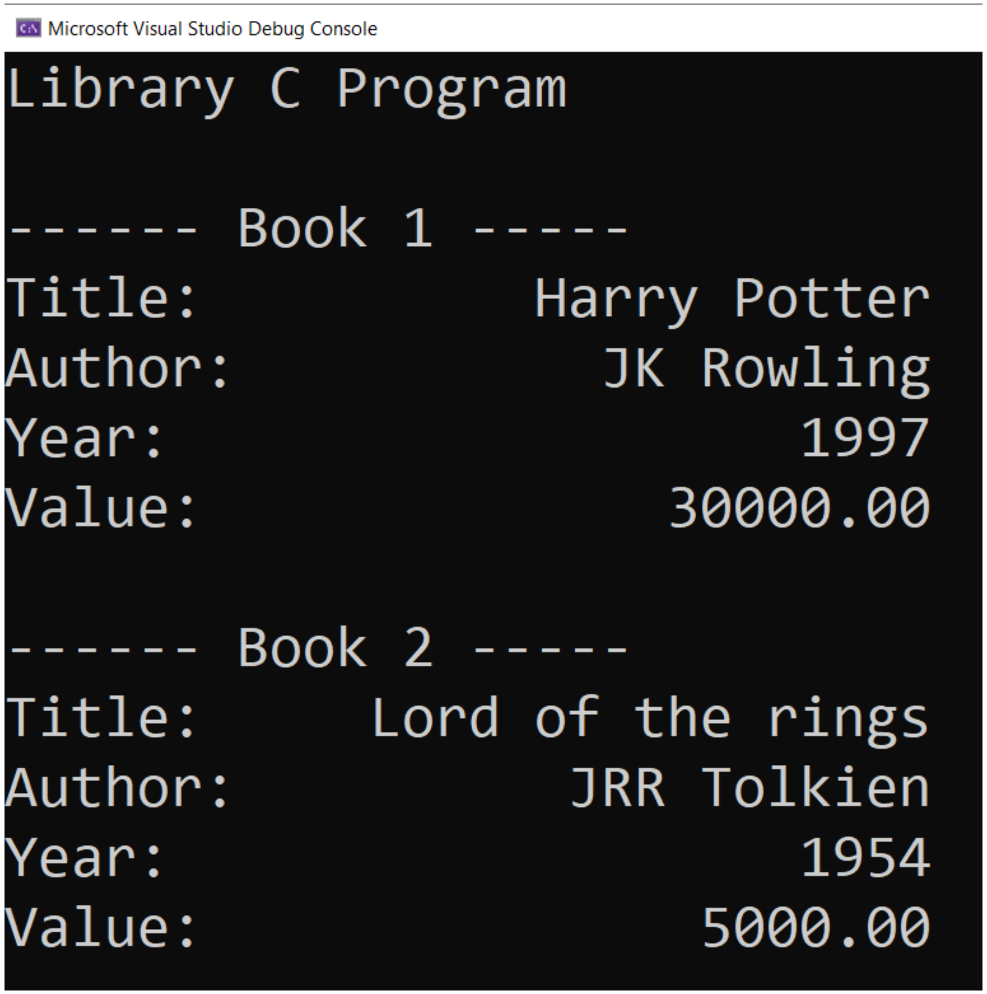
Library Software



```c
void displayLibrary() {
    for (int i = 0; i < numBooks; i++) {
        printf("\n------ Book %d -----", (i+1));
        printf("\nTitle:  %20s\nAuthor: %20s\nYear:   %20d\nValue:  %20.2f\n", library[i].title, library[i].author,
            library[i].year, library[i].value);
    }
}


void addBook(char title[], char author[], int year, float value) {
    if (numBooks<10) {
        strcpy_s(library[numBooks].title, 20, title);
        library[numBooks].value = value;
        library[numBooks].year = year;
        strcpy_s(library[numBooks].author, 20, author);
        numBooks++;
    }
    else {
        puts("library full");
    }
}
```

Library Software

- C Program Output:



```
Library C Program

------ Book 1 -----
Title:            Harry Potter
Author:             JK Rowling
Year:                     1997
Value:                30000.00

------ Book 2 -----
Title:       Lord of the rings
Author:            JRR Tolkien
Year:                     1954
Value:                 5000.00
```

PROGRAMMING

CT103

Week 11a

Sign in on Blackboard

- Please sign in on blackboard for CT103.
- I will leave the sign in option open for an hour after the lecture.

Lecture Content

- Last lecture (Week 10b):
  - Structures
  - Arrays of structures
  - Functions and structures
  - Example C program
- Today’s lecture (Week 11a):
  - Enumeration
  - Sorting
  - Example C program

ENUMERATION

Enumeration

- We often like to use names for common values rather than numeric values.

- **Definition**: Enumeration is a user defined datatype in the C programming language. It is used to assign names to values. This makes the program more readable.

Enumeration

- For example:
  - In an ordering system, status of an order could be:
    - Open
    - Closed
    - Delivered
    - Cancelled
  - In an alarm system, the system status could be:
    - Read
    - Asleep
    - Maintenance
  - In a traffic system, a light could be:
    - Red
    - Orange
    - Green

Enumeration

- Of course underlying those names would need to be actual values
- For example:
  - In an ordering system, status of an order could be:
    - Open = 101
    - Closed = 102
    - Delivered = 103
    - Cancelled = 104
  - In an alarm system, the system status could be:
    - Ready = 0
    - Asleep = 1
    - Maintenance = 2
  - In a traffic system, a light could be:
    - Red = 1
    - Orange = 2
    - Green = 3

Enumeration

- How do we use enumeration in C?

- We use the keyword: **enum**

Enumeration

- Illustration of enum:

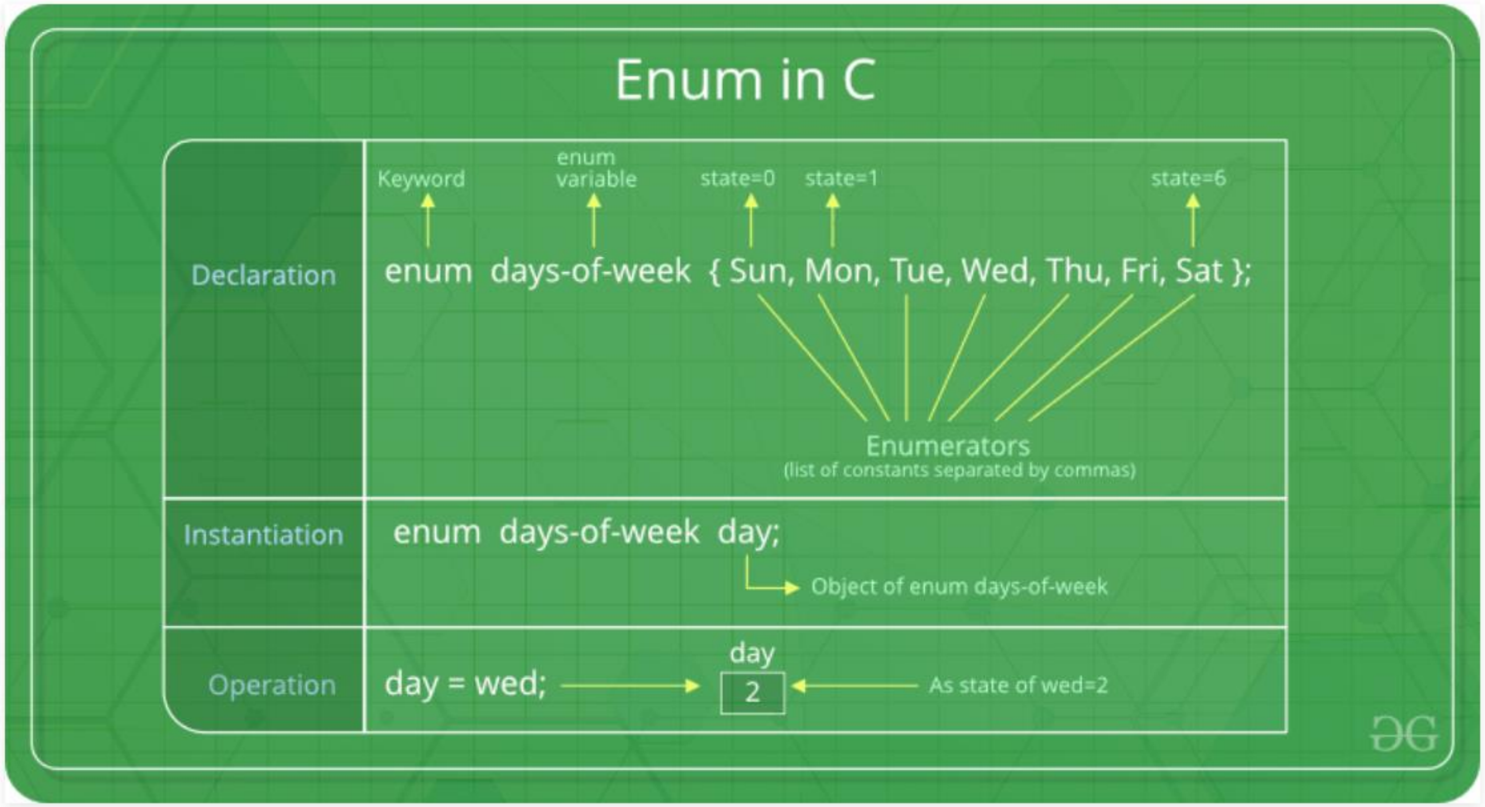


Image from: geeksforgeeks.org

Enumeration Example

- Let's say we want to know the status of an important system, and it can be in one of the following 4 states:
  - { IDLE, BUSY, ASLEEP, MAINTENANCE };
- We just define an enumeration that contains those values:
  - enum STATUS { IDLE, BUSY, ASLEEP, MAINTENANCE };
- Underneath, the enum values are associated with specific integer values, by default starting at 0
- So in fact IDLE = 0, BUSY = 1, ASLEEP = 2 and MAINTENANCE = 3
- Since they are actually integers we can use them in if statements, switch statements etc.

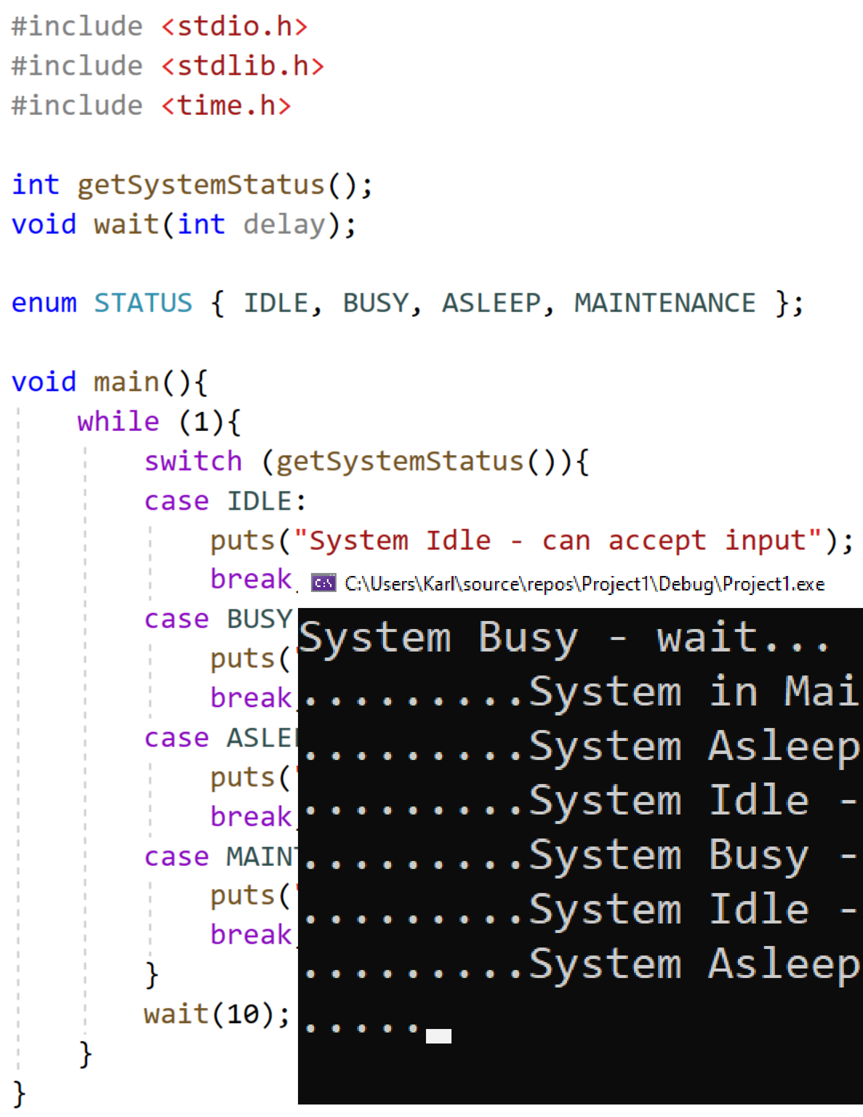
Enumeration C Example Part 1

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int getSystemStatus();
void wait(int delay);

enum STATUS { IDLE, BUSY, ASLEEP, MAINTENANCE };

void main(){
    while (1){
        switch (getSystemStatus()){
        case IDLE:
            puts("System Idle - can accept input");
            break;
        case BUSY:
            puts("System Busy - wait...");
            break;
        case ASLEEP:
            puts("System Asleep - initate wakeup procedure");
            break;
        case MAINTENANCE:
            puts("System in Maintenance mode - wait until finished");
            break;
        }
        wait(10);
    }
}
```

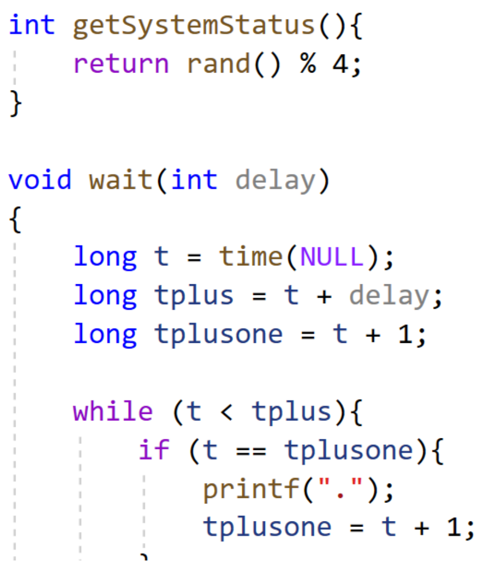
Enumeration C Example Part 2

```
int getSystemStatus(){
    return rand() % 4;
}

void wait(int delay)
{
    long t = time(NULL);
    long tplus = t + delay;
    long tplusone = t + 1;

    while (t < tplus){
        if (t == tplusone){
            printf(".");
            tplusone = t + 1;
        }
        t = time(NULL);
    }
}
```

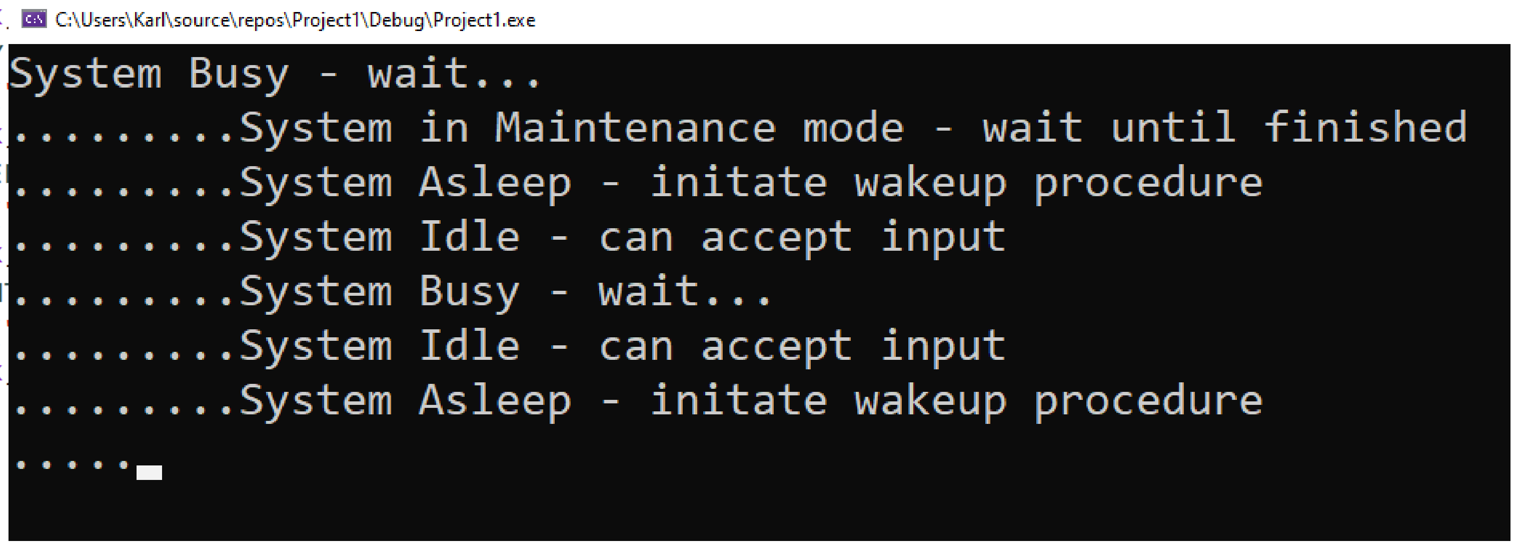
Enumeration C Example Output



```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int getSystemStatus();
void wait(int delay);

enum STATUS { IDLE, BUSY, ASLEEP, MAINTENANCE };

void main(){
    while (1){
        switch (getSystemStatus()){
        case IDLE:
            puts("System Idle - can accept input");
            break
        case BUSY
            puts(
            break
        case ASLE
            puts(
            break
        case MAIN
            puts(
            break
        }
        wait(10);
    }
}
```



```c
int getSystemStatus(){
    return rand() % 4;
}

void wait(int delay)
{
    long t = time(NULL);
    long tplus = t + delay;
    long tplusone = t + 1;

    while (t < tplus){
        if (t == tplusone){
            printf(".");
            tplusone = t + 1;
```



C:\Users\Karl\source\repos\Project1\Debug\Project1.exe

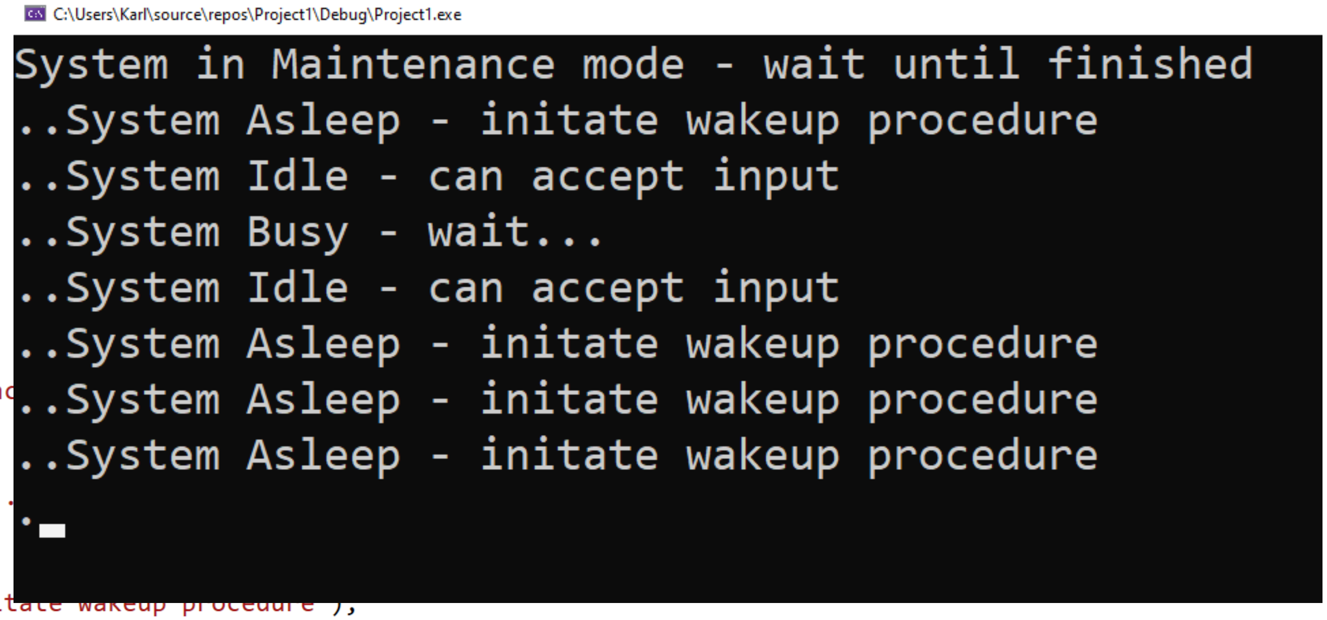
```
System Busy - wait...
.........System in Maintenance mode - wait until finished
.........System Asleep - initate wakeup procedure
.........System Idle - can accept input
.........System Busy - wait...
.........System Idle - can accept input
.........System Asleep - initate wakeup procedure
.....
```

Enumeration Part 1 Using Typedef

```c
typedef enum { IDLE, BUSY, ASLEEP, MAINTENANCE } STATUS;

STATUS getSystemStatus();
void wait(int delay);

void main(){

    STATUS systemStatus;
    systemStatus = getSystemStatus();

    while (1){
        switch (getSystemStatus()){
        case IDLE:
            puts("System Idle - can accept input");
            break;
        case BUSY:
            puts("System Busy - wait...");
            break;
        case ASLEEP:
            puts("System Asleep - initate wakeup procedure");
            break;
        case MAINTENANCE:
            puts("System in Maintenance mode - wait until finished");
            break;
        }
        wait(3);
    }
}

STATUS getSystemStatus(){
    return (STATUS)(rand() % 4);
}
```

Enumeration Using Typedef Output

```c
typedef enum { IDLE, BUSY, ASLEEP, MAINTENANCE } STATUS;

STATUS getSystemStatus();
void wait(int delay);

void main(){

    STATUS systemStatus;
    systemStatus = getSystemStatus();

    while (1){
        switch (getSystemStatus()){
        case IDLE:
            puts("System Idle - can ac
            break;
        case BUSY:
            puts("System Busy - wait..
            break;
        case ASLEEP:
            puts("System Asleep - initate wakeup procedure");
            break;
        case MAINTENANCE:
            puts("System in Maintenance mode - wait until finished");
            break;
        }
        wait(3);
    }
}

STATUS getSystemStatus(){
    return (STATUS)(rand() % 4);
}
```



C:\Users\Karl\source\repos\Project1\Debug\Project1.exe

```
System in Maintenance mode - wait until finished
..System Asleep - initate wakeup procedure
..System Idle - can accept input
..System Busy - wait...
..System Idle - can accept input
..System Asleep - initate wakeup procedure
..System Asleep - initate wakeup procedure
..System Asleep - initate wakeup procedure
.
```

SORTING

Sorting

- **Definition**: Sorting algorithms are algorithms that put items in the correct order.
- Most commonly, the order is based on numeric value.
- This can be from smallest to largest, or vice versa.

Sorting

- Why do we need sorting algorithms?

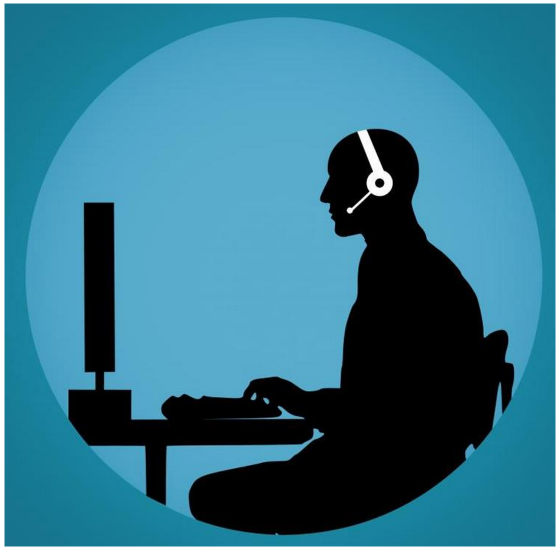
- We need sorting algorithms because data are often not in any order. Putting the data in correct order is needed for many applications.

- There are many situations that would require data to be sorted into the correct order!

Sorting Example

- A customer services department of a business might have 50 customer complaints to deal with on a given day.

- The system stores these complaints alphabetically, however you want to respond to the complaints that have been waiting for a response the longest.



Sorting Example

- A customer services department of a business might have 50 customer complaints to deal with on a given day.
- Complaint wait times (hours): [50,2,3,64, …, 14,3,61]
- How do we sort [50,2,3,64, …, 14,3,61] so that it is in the correct order?
  - [88,85,72,64, …, 3,3,2,2,1]
- We use a sorting algorithm!

Sorting Algorithms

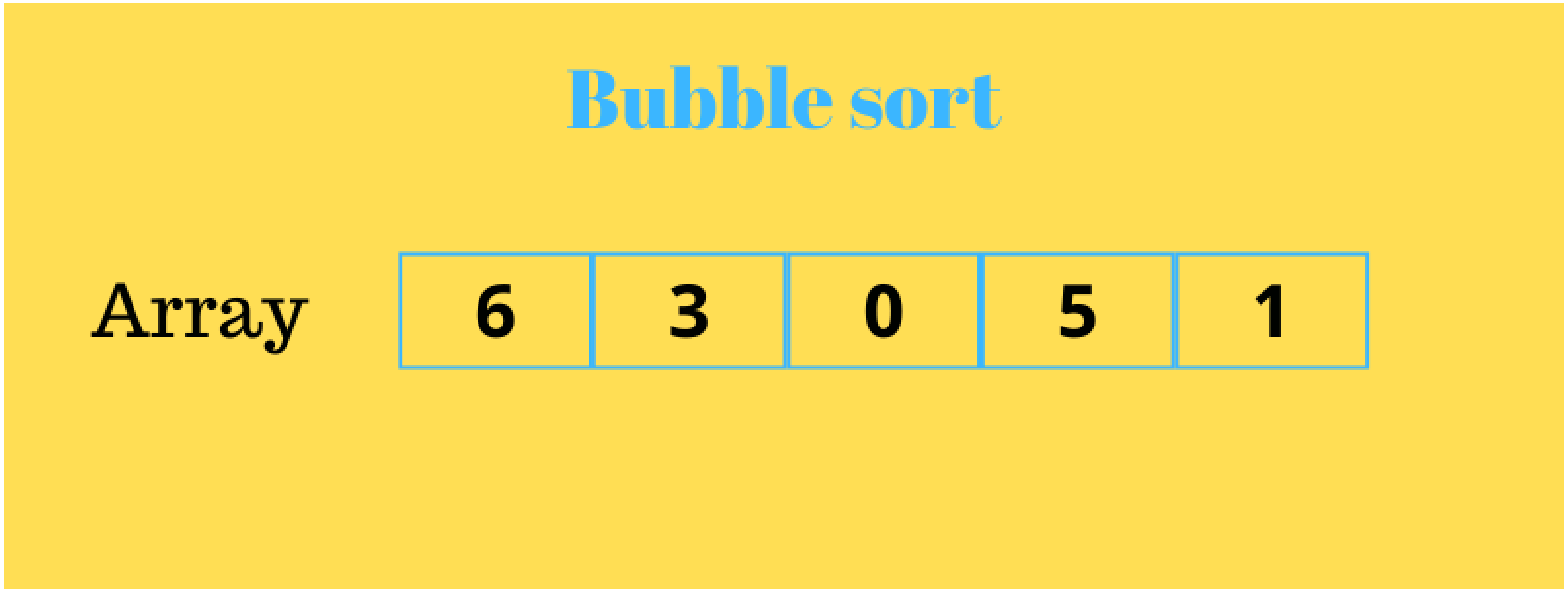
- There are many different sorting algorithms:
  - **Bubble sort**.
  - Merge sort.
  - Insertion sort.
  - Quick sort.
  - Selection sort.

Bubble Sort

- Straightforward concept – comparing elements to make the largest move to the right in an array
- Largest elements in array ‘bubble’ to the top (right)
- Not the most efficient sort algorithm, but OK for small arrays and easy to understand
- Well documented, e.g.
  - https://www.programmingsimplified.com/c/source-code/c-program-bubble-sort
  - https://www.youtube.com/watch?v=nmhjrI-aW5o
  - https://www.w3schools.in/data-structures-tutorial/sorting-techniques/bubble-sort-algorithm/
  - https://www.geeksforgeeks.org/bubble-sort/

Bubble Sort Illustration

- On Wednesday, we will implement bubble sort in C.



Graphic from: programmingsimplified.com

EXAMPLE PROBLEM

Employment Software

- Write program to record hours worked for each day of the week.
- Create an enum to represent each day of the week.
- Ask the user how many hours were worked each day of the week.
- Print the total hours worked to the screen.
- Write a function to check if the employee should be paid for over time, i.e. did they work more than 40 hours? Display a message to the screen if they should be paid for over time.

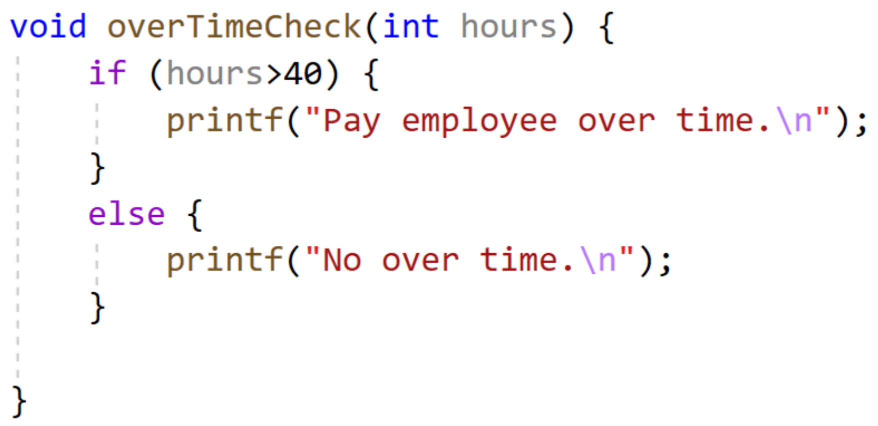
Employment Software

- Go to C program solution.

Employment Software

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

typedef enum { Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday } DAY;
char days[7][10] = { "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday" };

void overTimeCheck(int hours);

void main(){
    DAY today;
    int totalHours = 0, todayHours = 0;

    for (today = Monday; today <= Sunday; today = (DAY)((int)(today)+1)){
        printf("Enter hours for %s: ", days[today]);
        scanf_s("%d", &todayHours);
        totalHours += todayHours;
    }

    printf("\nTotal weekly hours are %d\n", totalHours);
    overTimeCheck(totalHours);
}
```

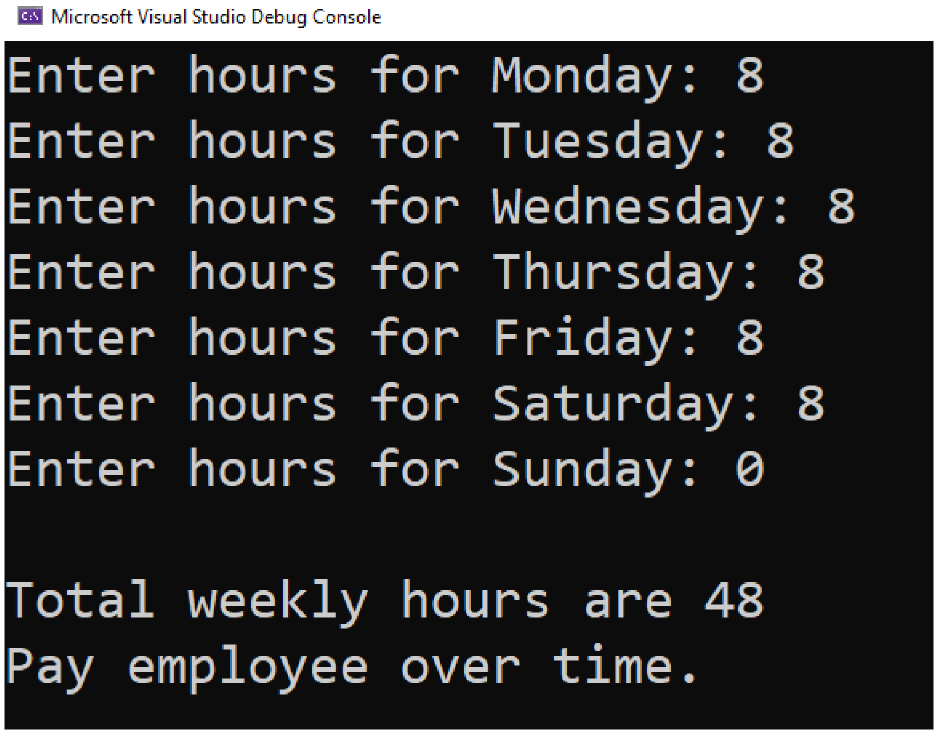
Employment Software



```
void overTimeCheck(int hours) {
    if (hours>40) {
        printf("Pay employee over time.\n");
    }
    else {
        printf("No over time.\n");
    }

}
```

Employment Software

- C Program Output:



```
Enter hours for Monday: 8
Enter hours for Tuesday: 8
Enter hours for Wednesday: 8
Enter hours for Thursday: 8
Enter hours for Friday: 8
Enter hours for Saturday: 8
Enter hours for Sunday: 0

Total weekly hours are 48
Pay employee over time.
```

PROGRAMMING

CT103

Week 12a (11b)

Sign in on Blackboard

- Please sign in on blackboard for CT103.
- I will leave the sign in option open for an hour after the lecture.

Lecture Content

- Last lecture (Week 11a):
  - Enumeration
  - Sorting
  - Example C program

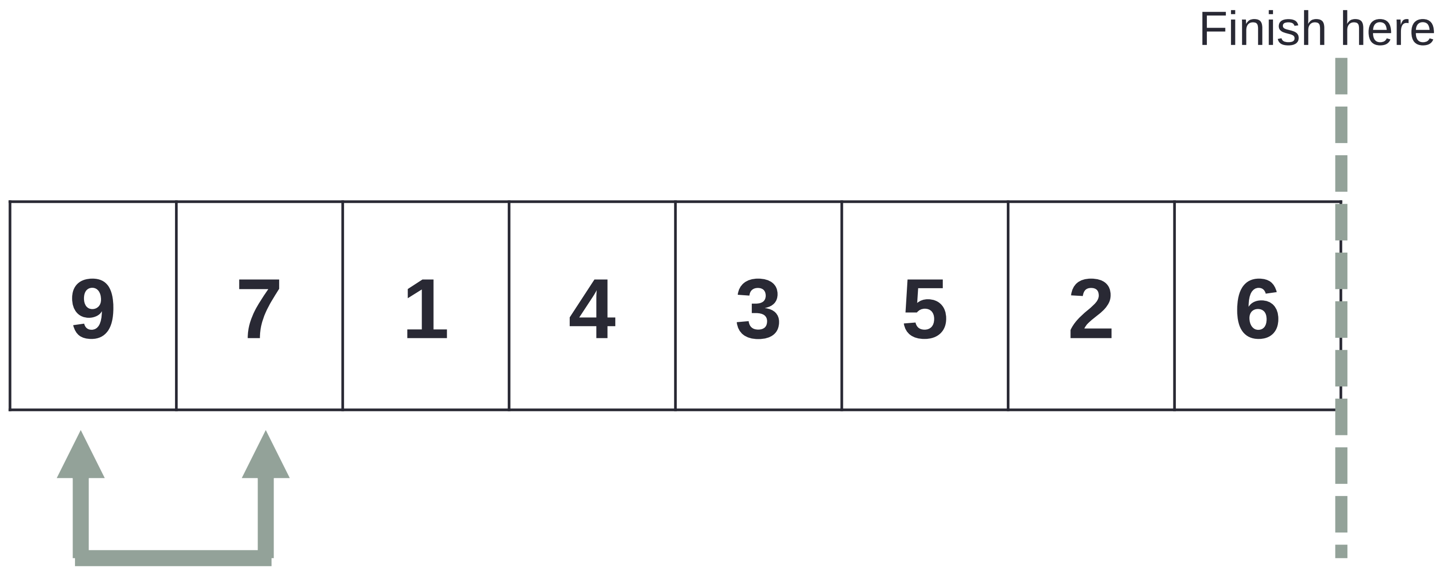
- Today’s lecture (Week 12a):
  - Bubble sort
  - Sorting
  - Example C program

BUBBLE SORT

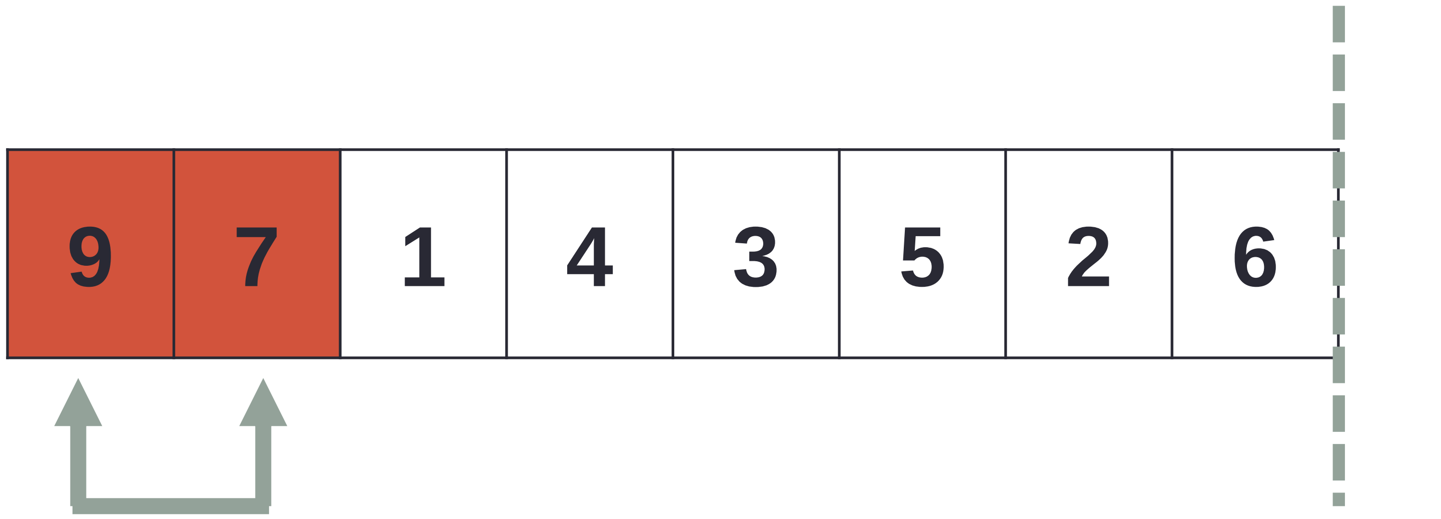
Bubble Sort Recap

- Straightforward concept – comparing elements to make the largest move to the right in an array
- Largest elements in array ‘bubble’ to the top (right)
- Not the most efficient sort algorithm, but OK for small arrays and easy to understand
- Well documented, e.g.
  - https://www.programmingsimplified.com/c/source-code/c-program-bubble-sort
  - https://www.youtube.com/watch?v=nmhjrI-aW5o
  - https://www.w3schools.in/data-structures-tutorial/sorting-techniques/bubble-sort-algorithm/
  - https://www.geeksforgeeks.org/bubble-sort/

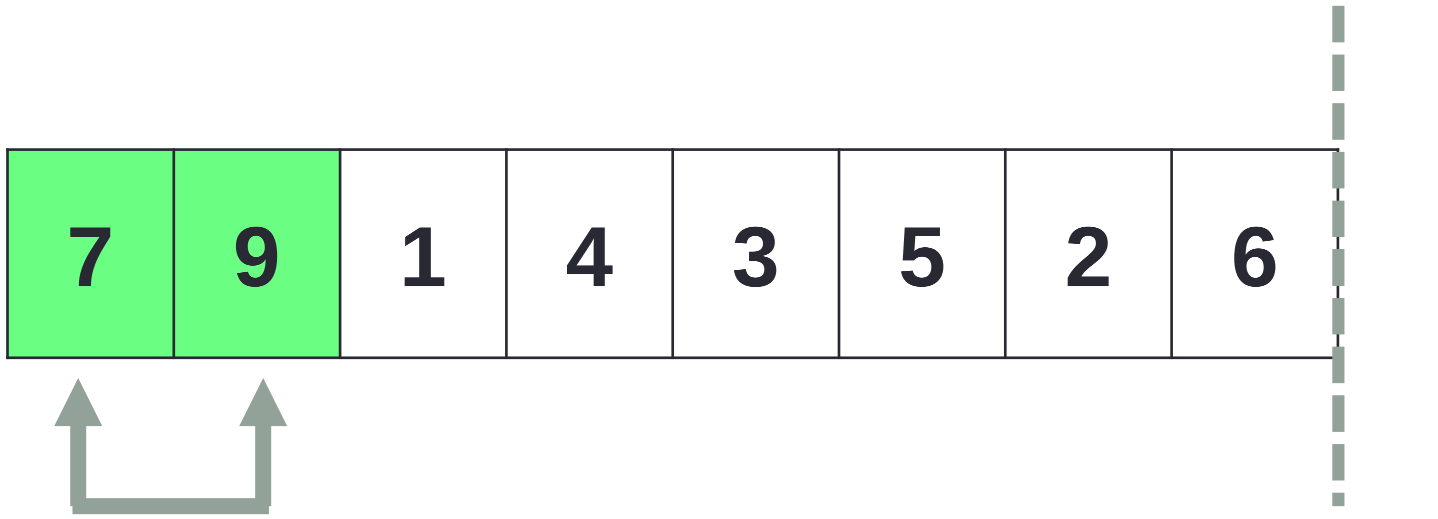
Start by comparing first two elements



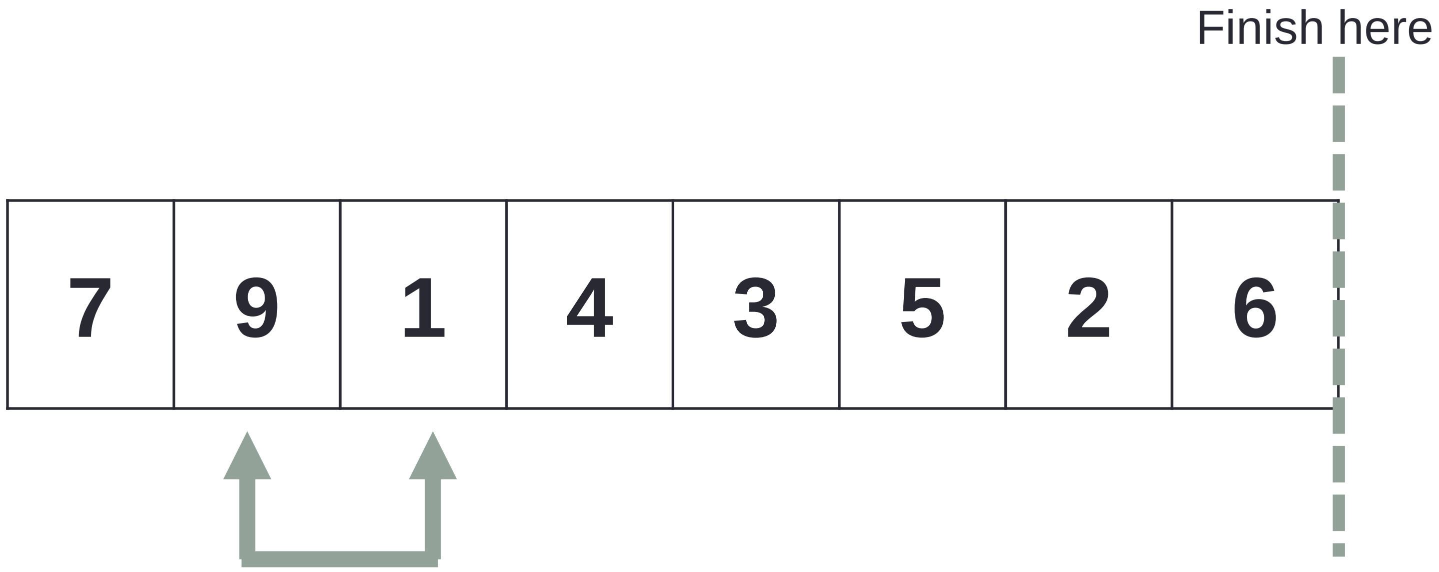
Incorrect order, so need to swap



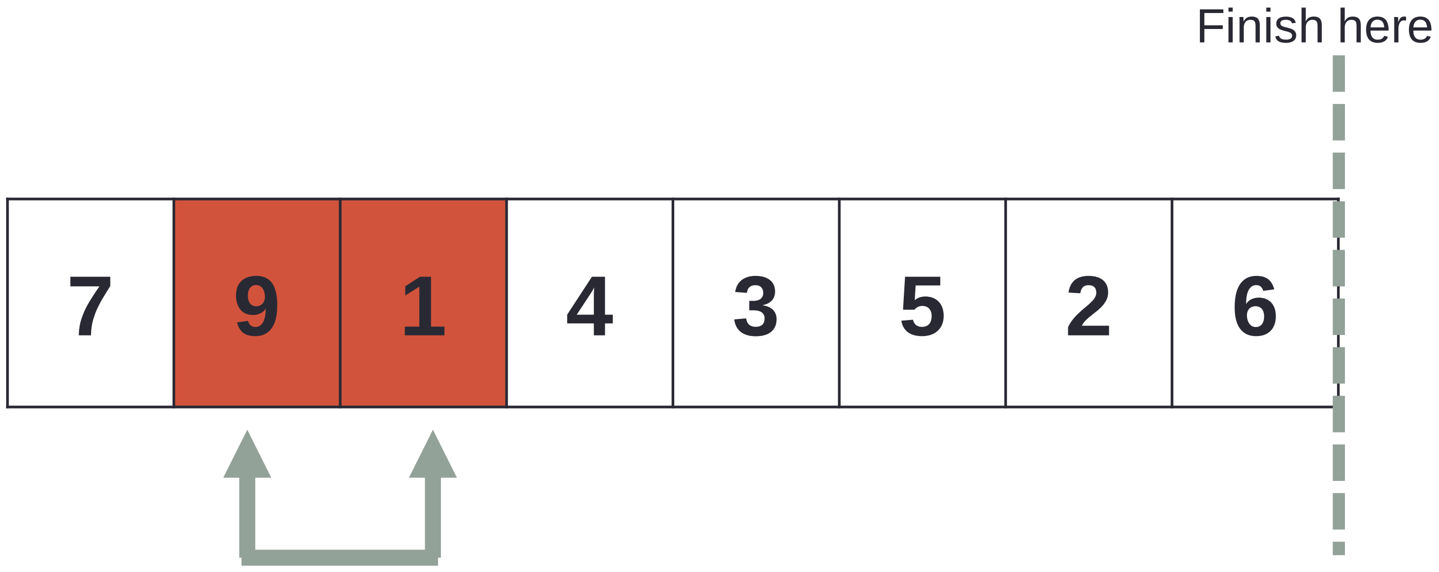
Swap them



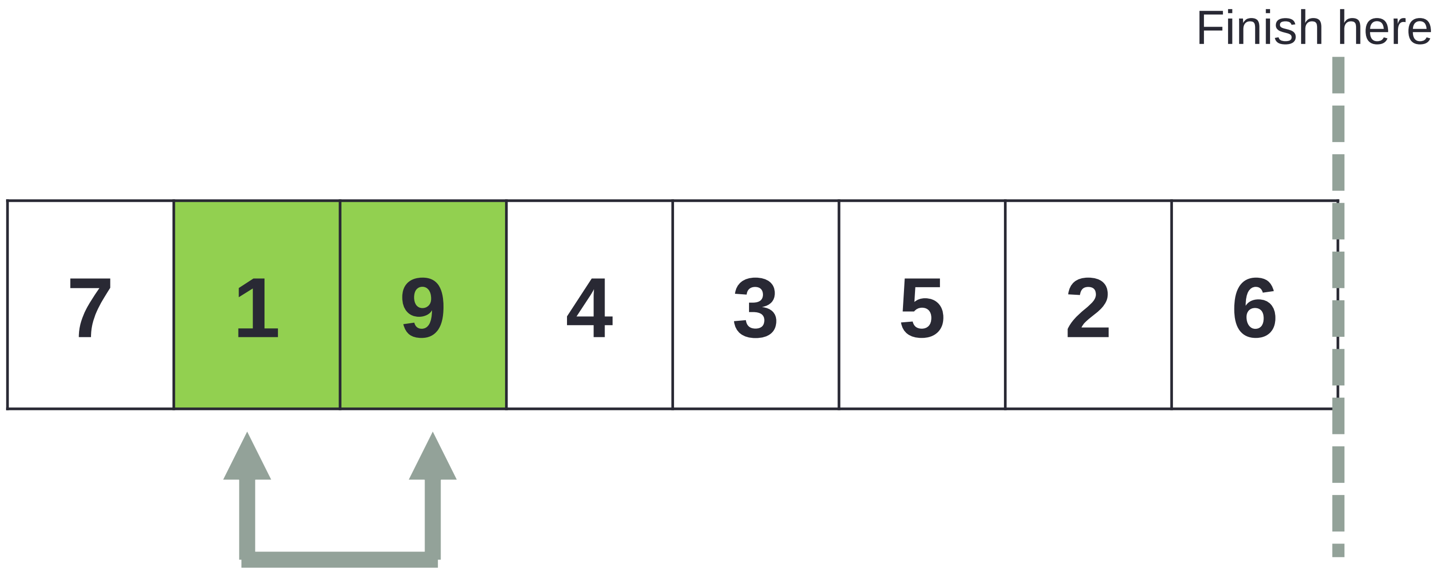
Now compare next two elements



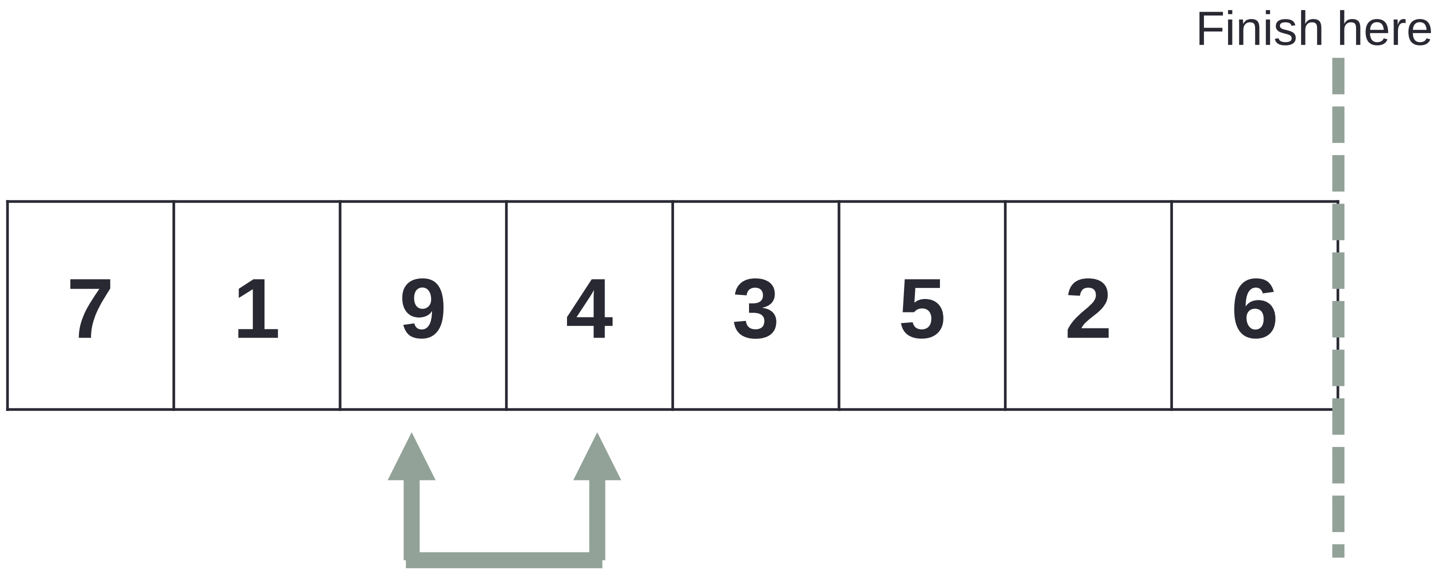
Incorrect order



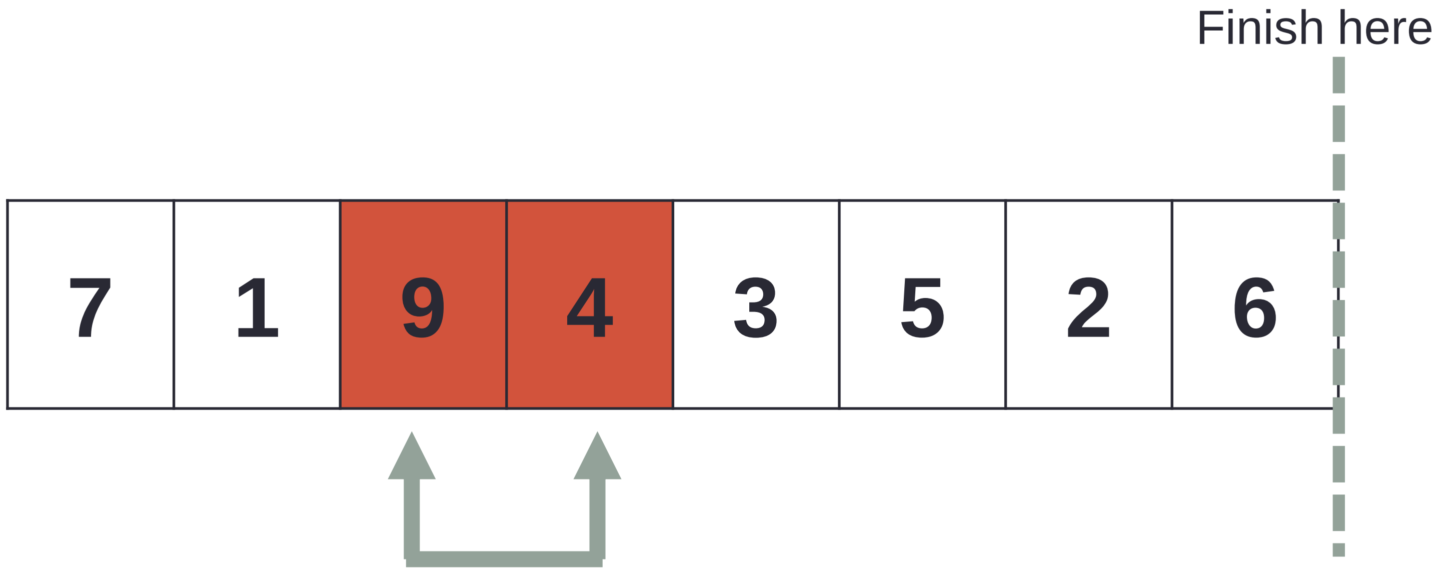
So swap them



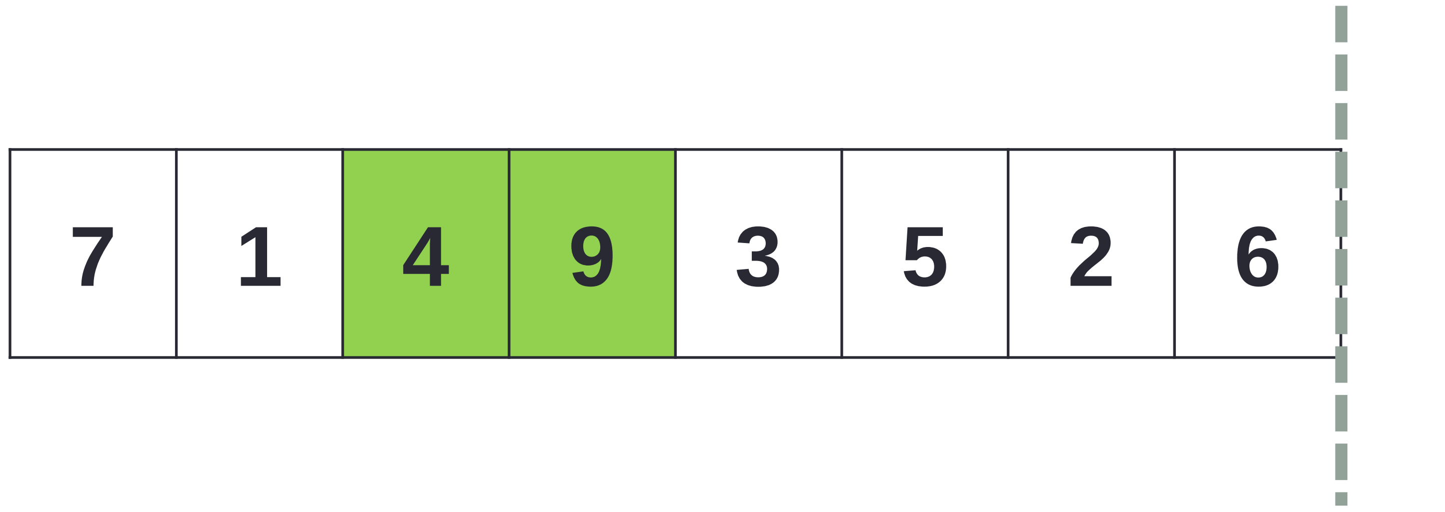
Compare next two elements



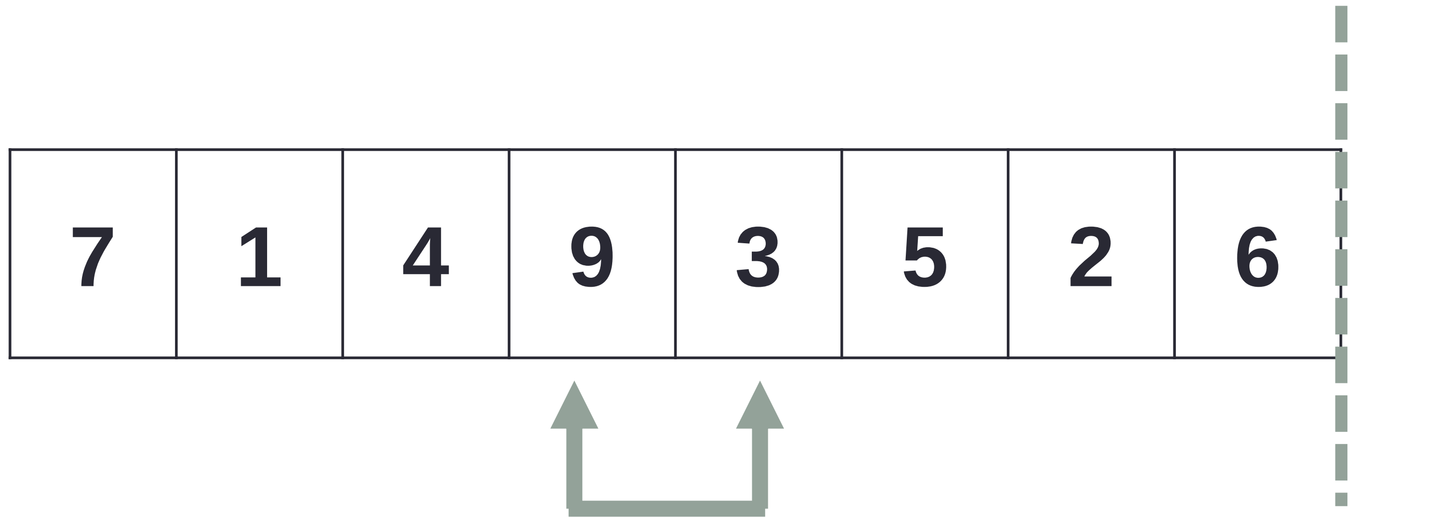
Incorrect order



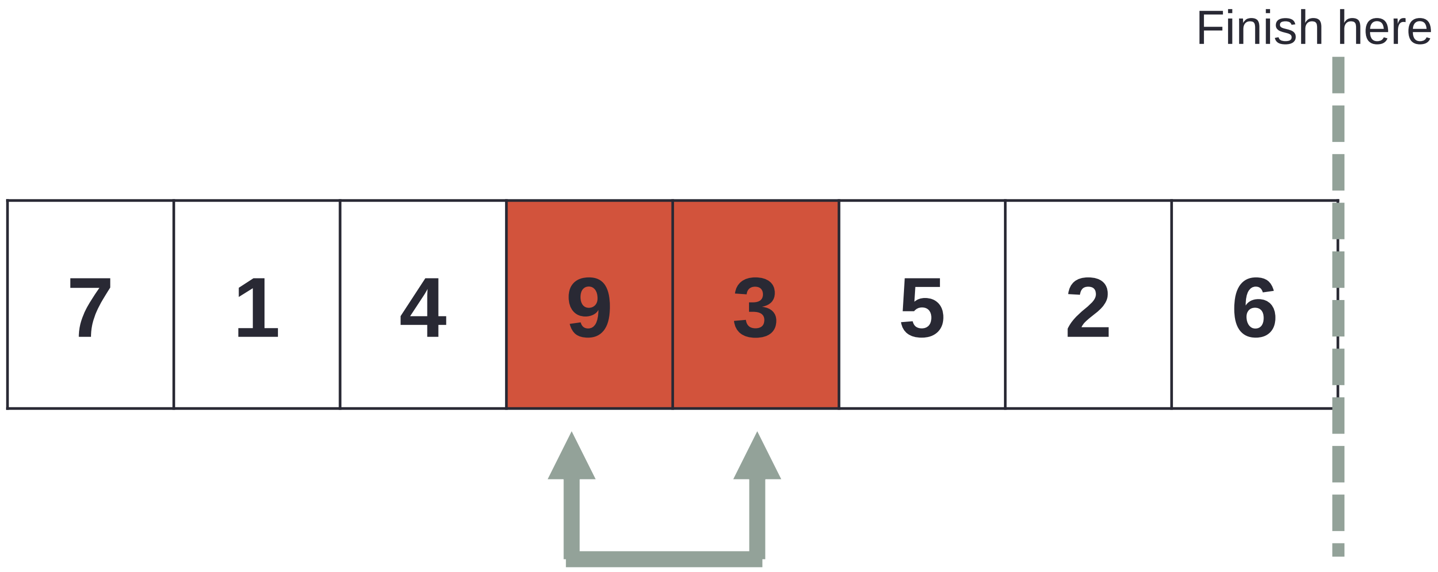
Swap them



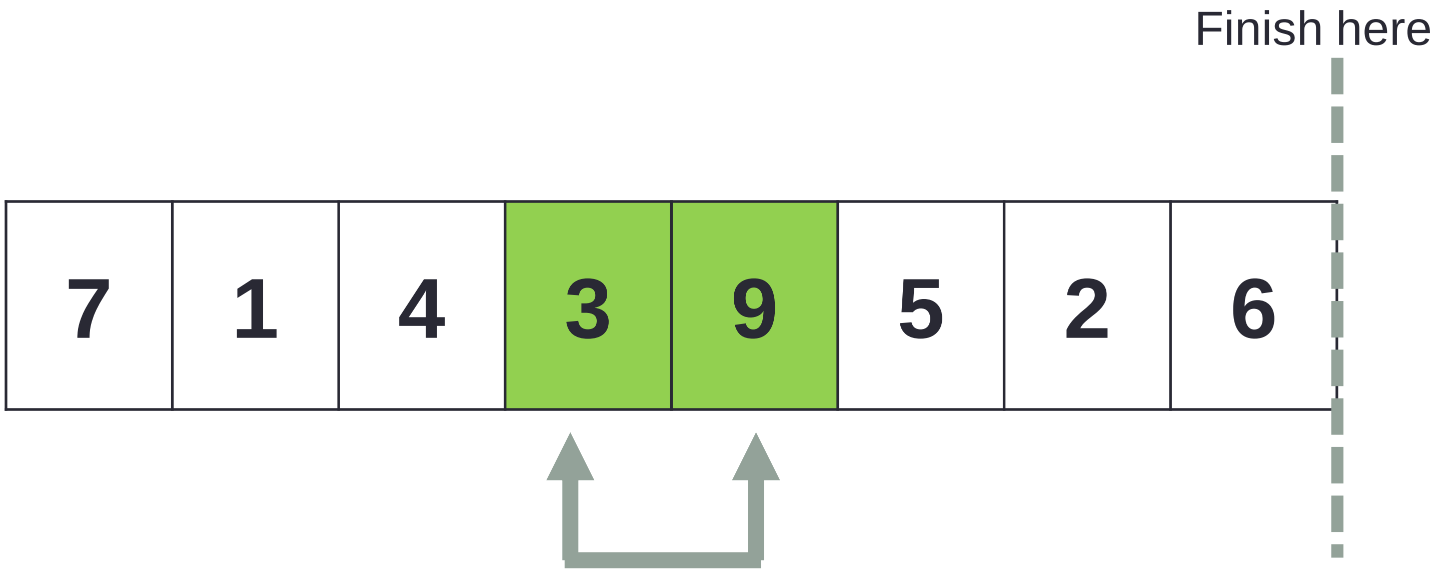
Compare next two elements



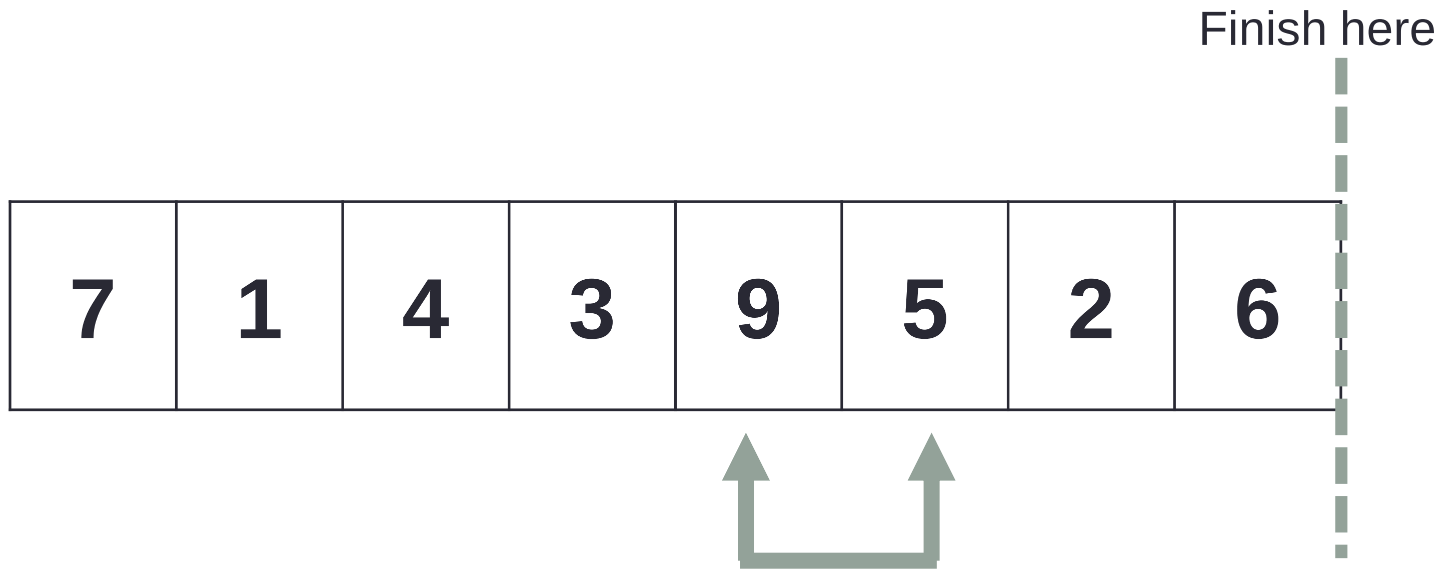
Incorrect order



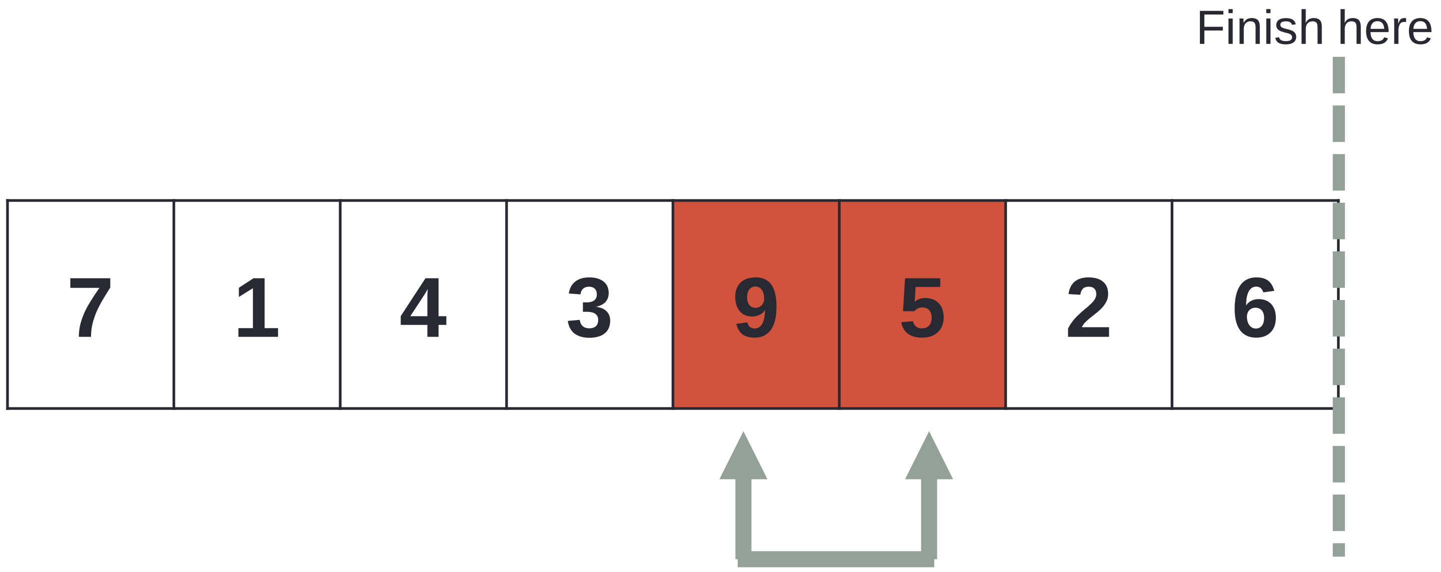
Swap them



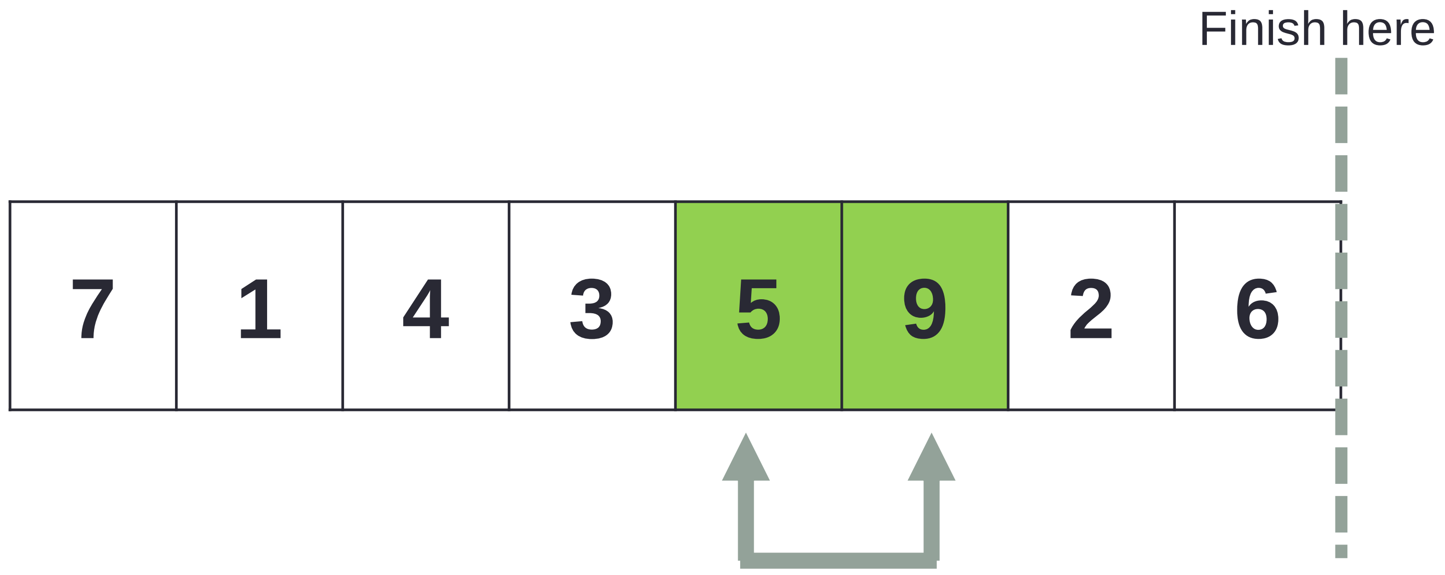
Compare next two elements



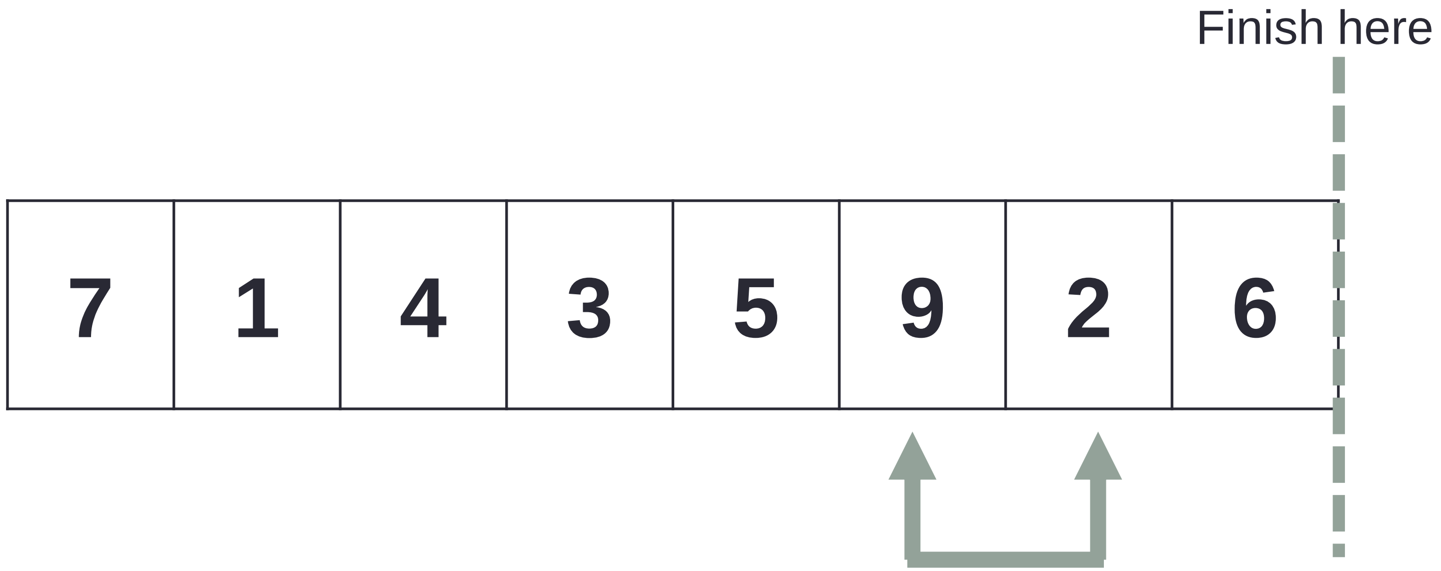
Incorrect order



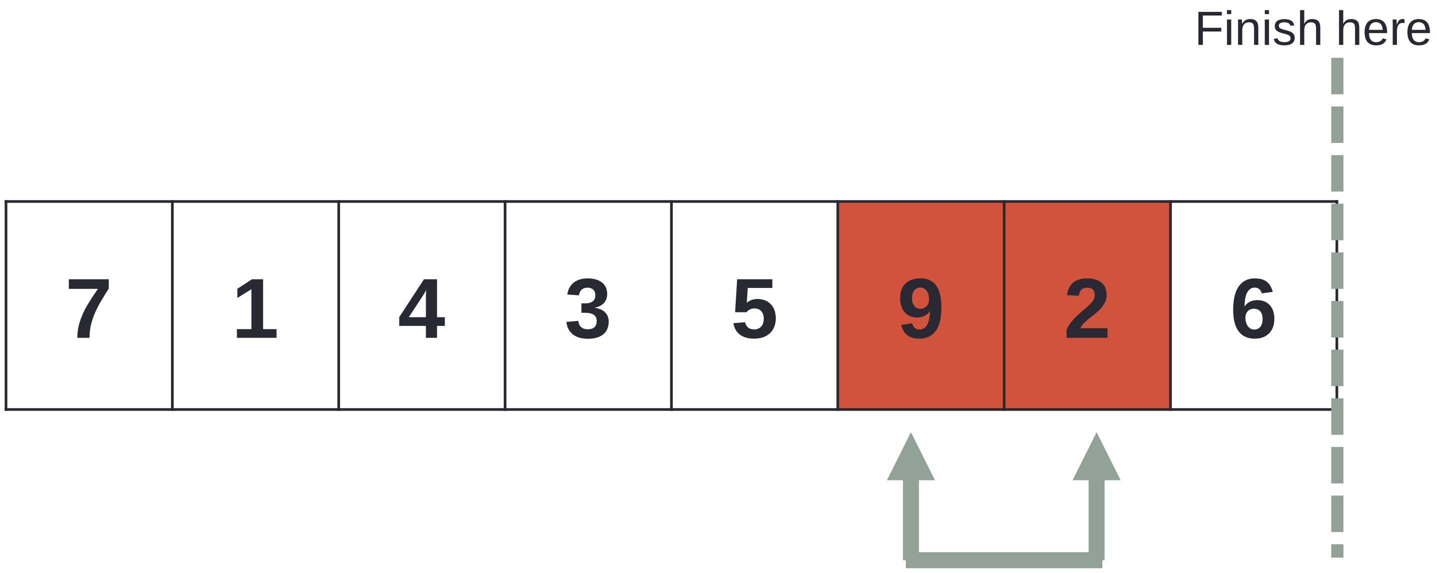
Swap them



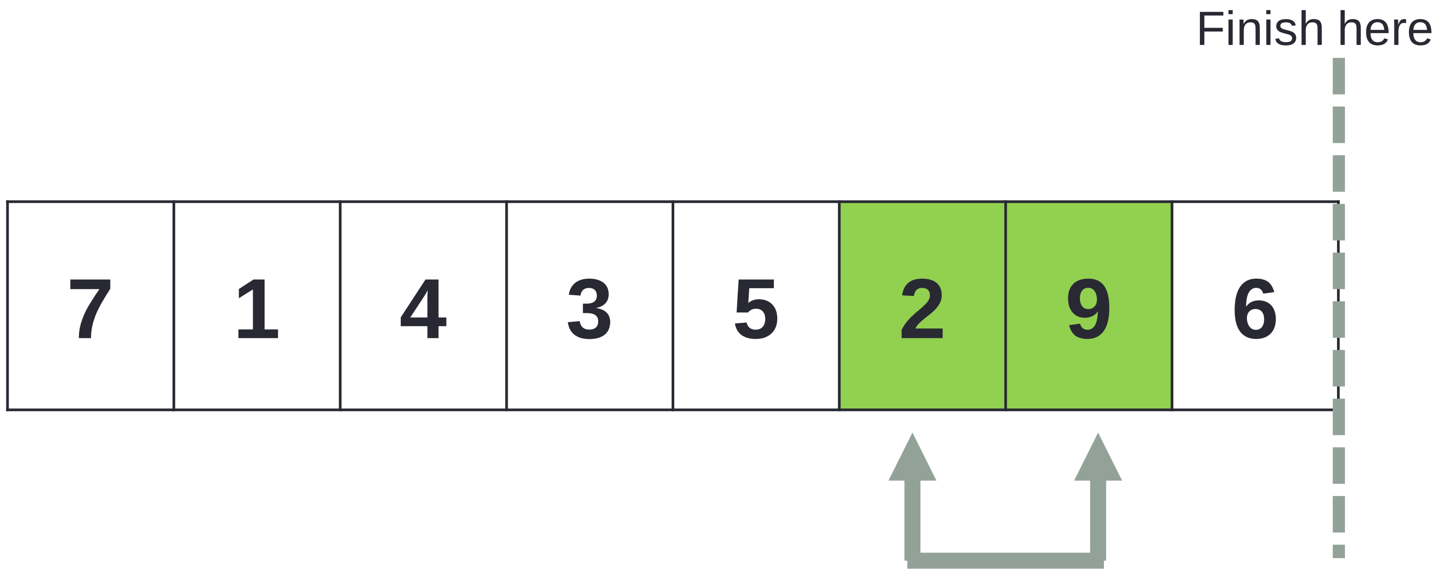
Compare next two elements



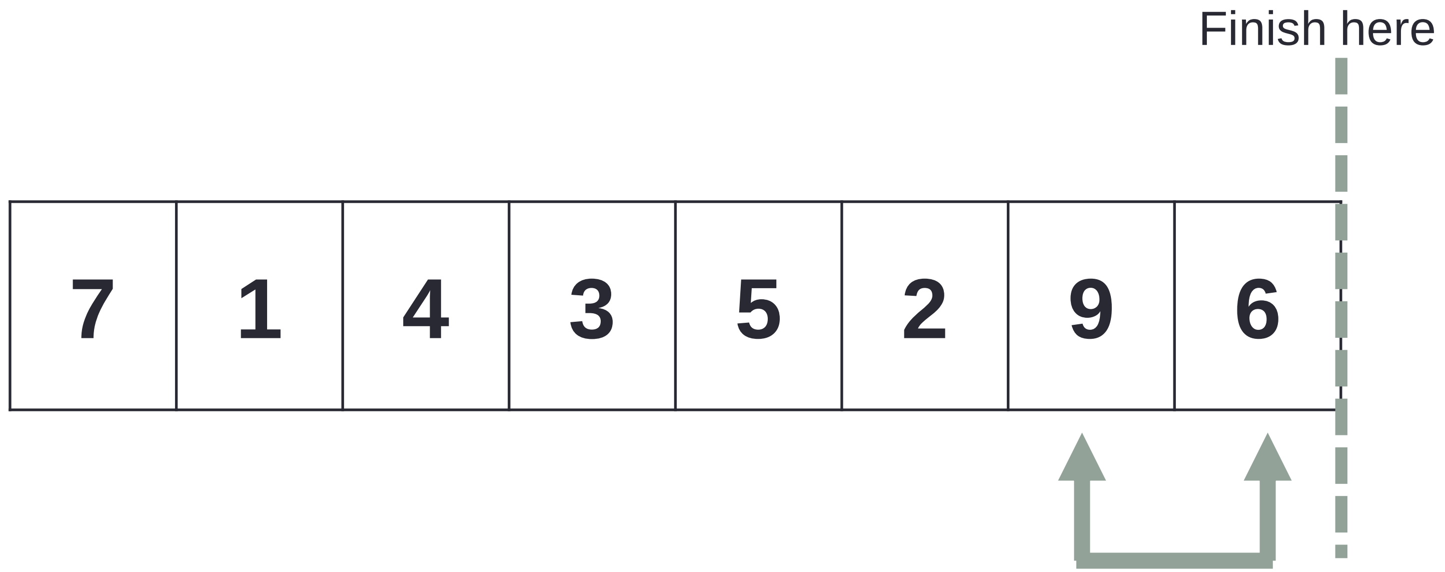
Incorrect order



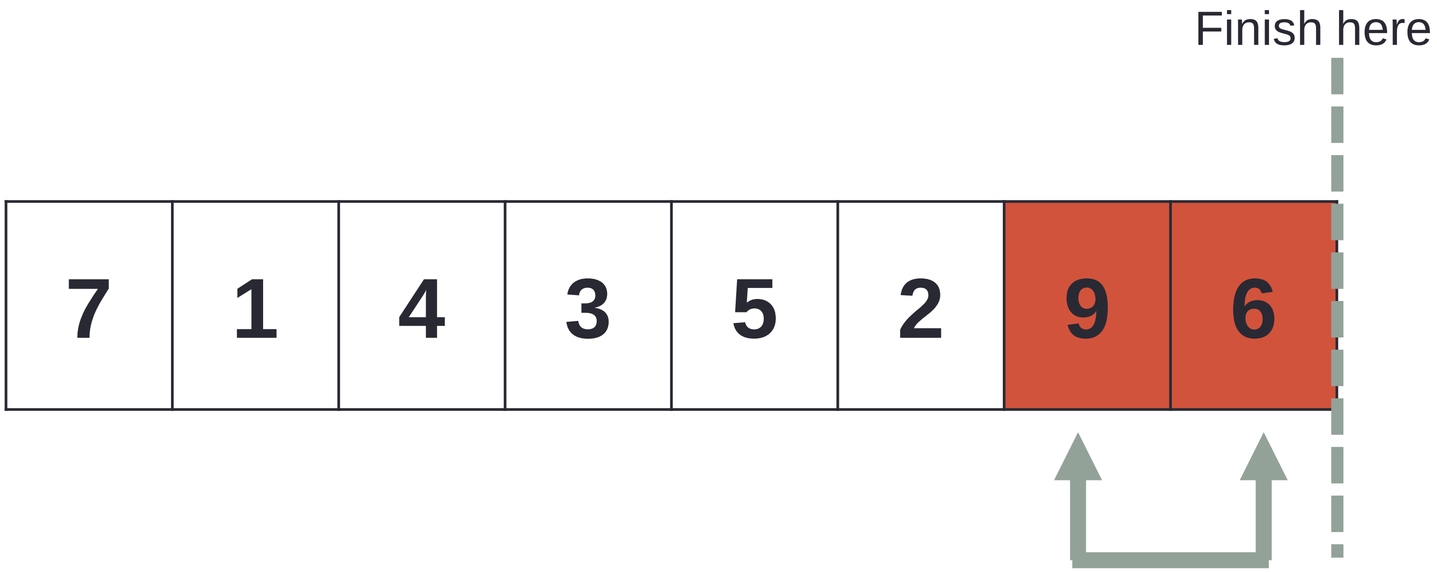
Swap them



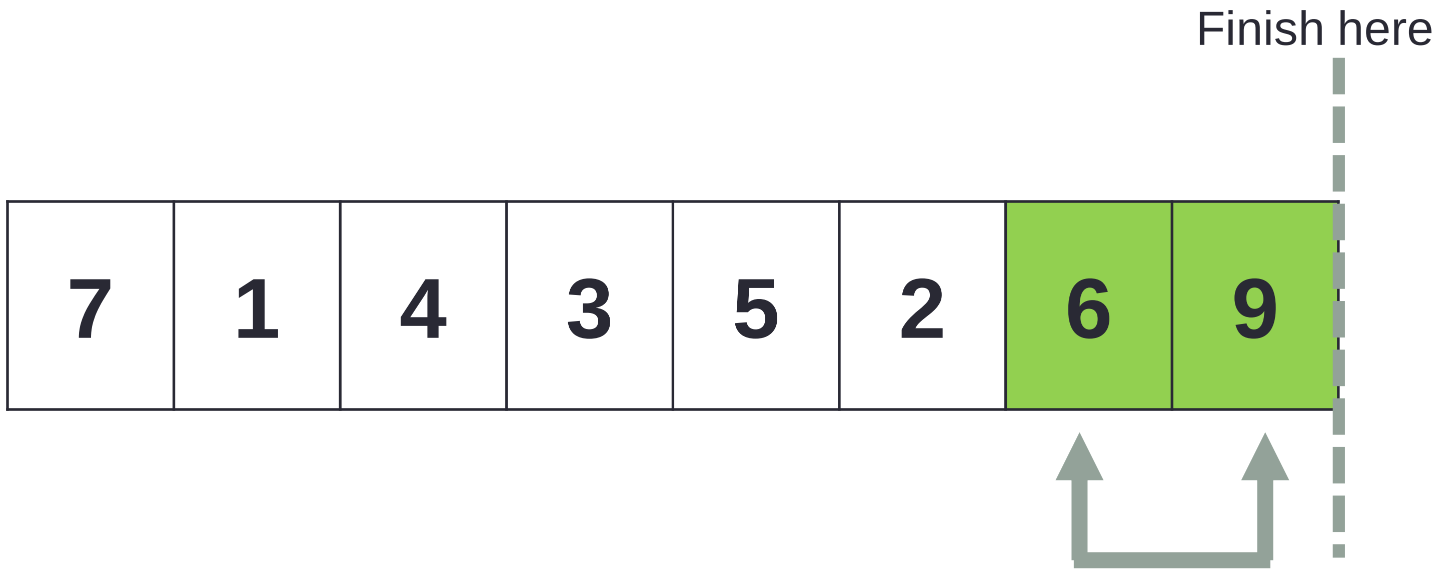
Compare next two elements



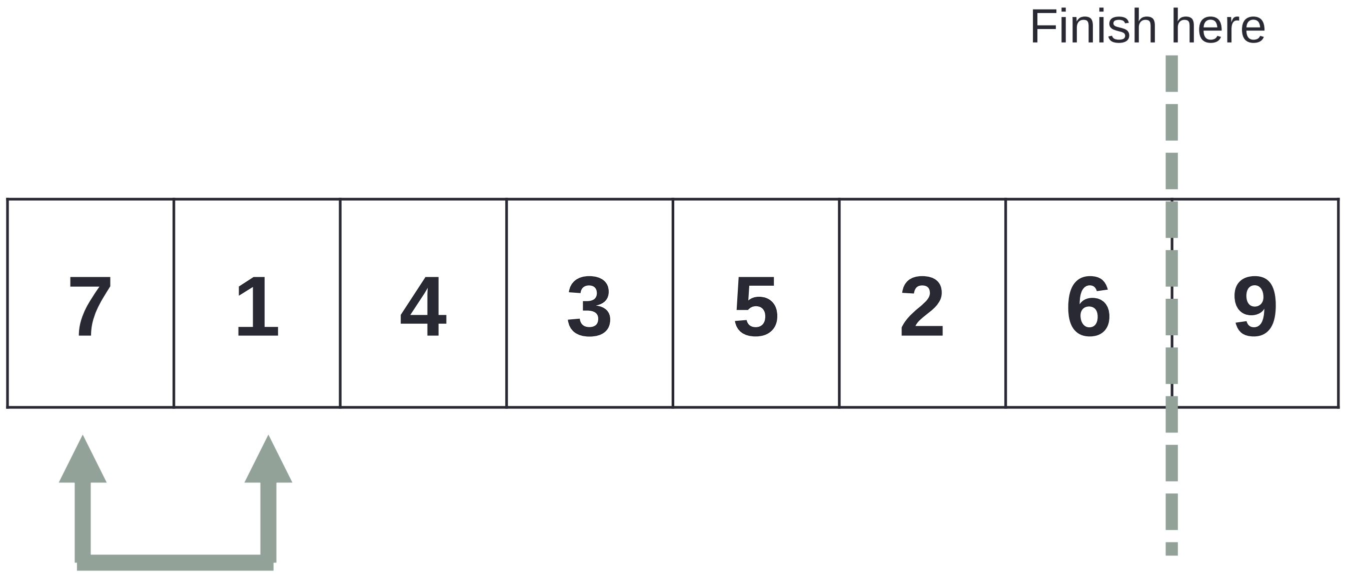
Incorrect order

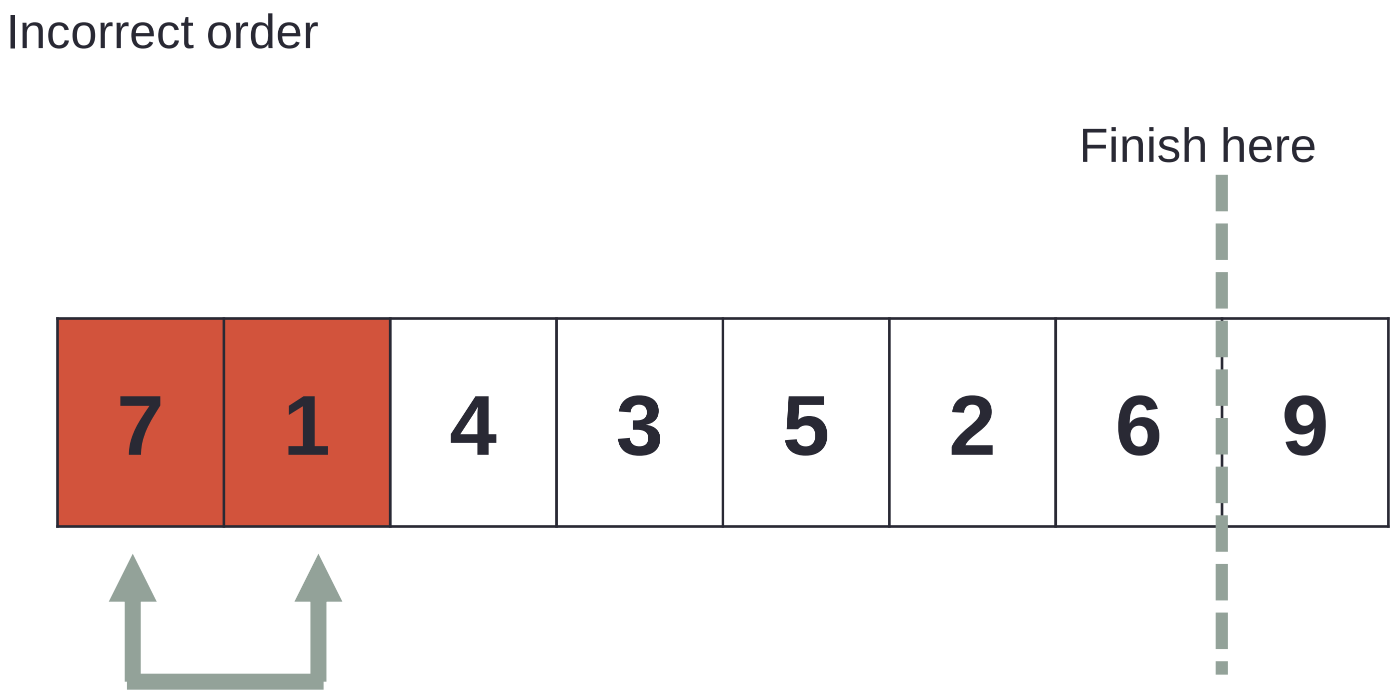


Swap them



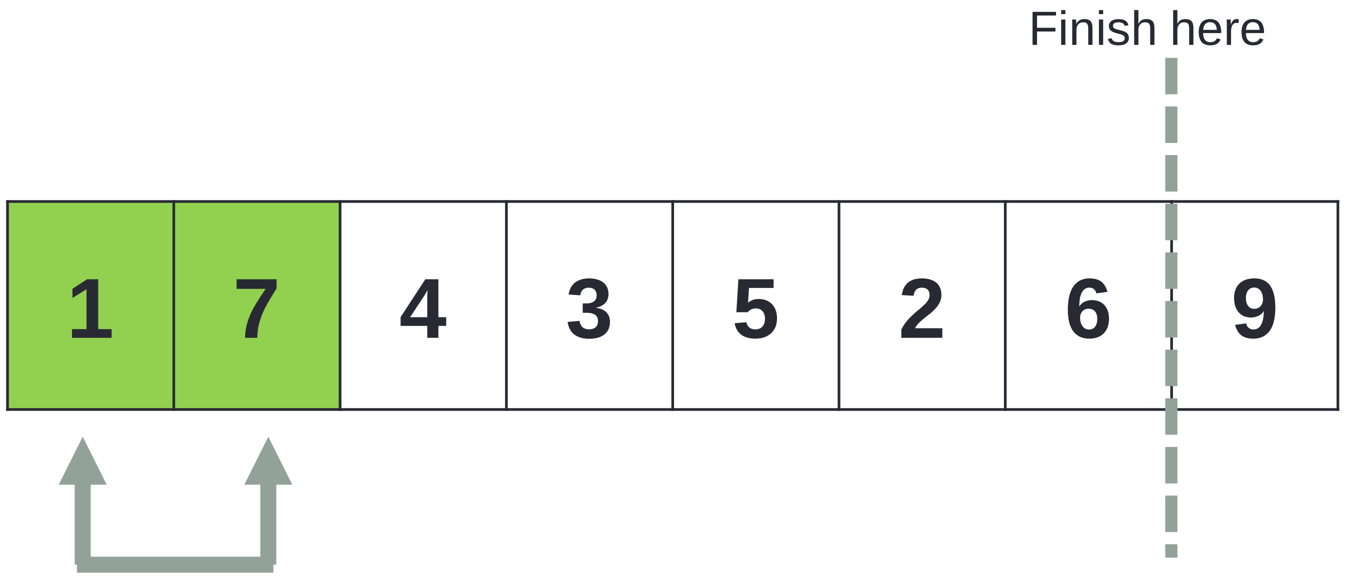
Go back and start at first element – don’t need to include last element



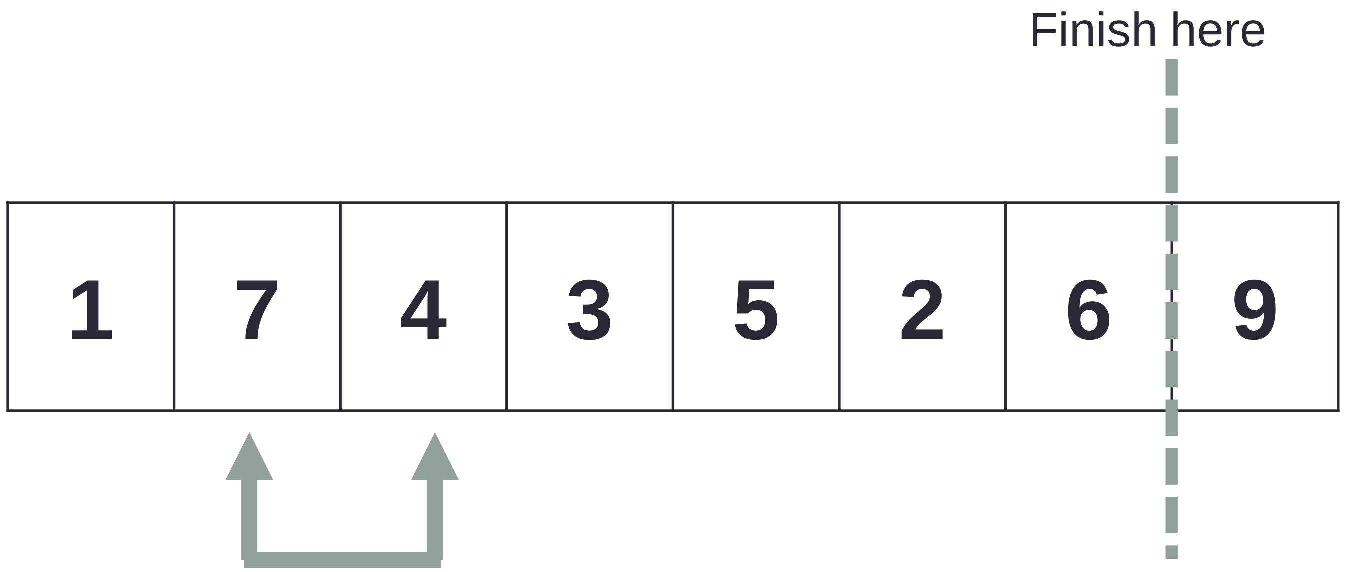


Incorrect order
Finish here
7
1
4
3
5
2
6
9

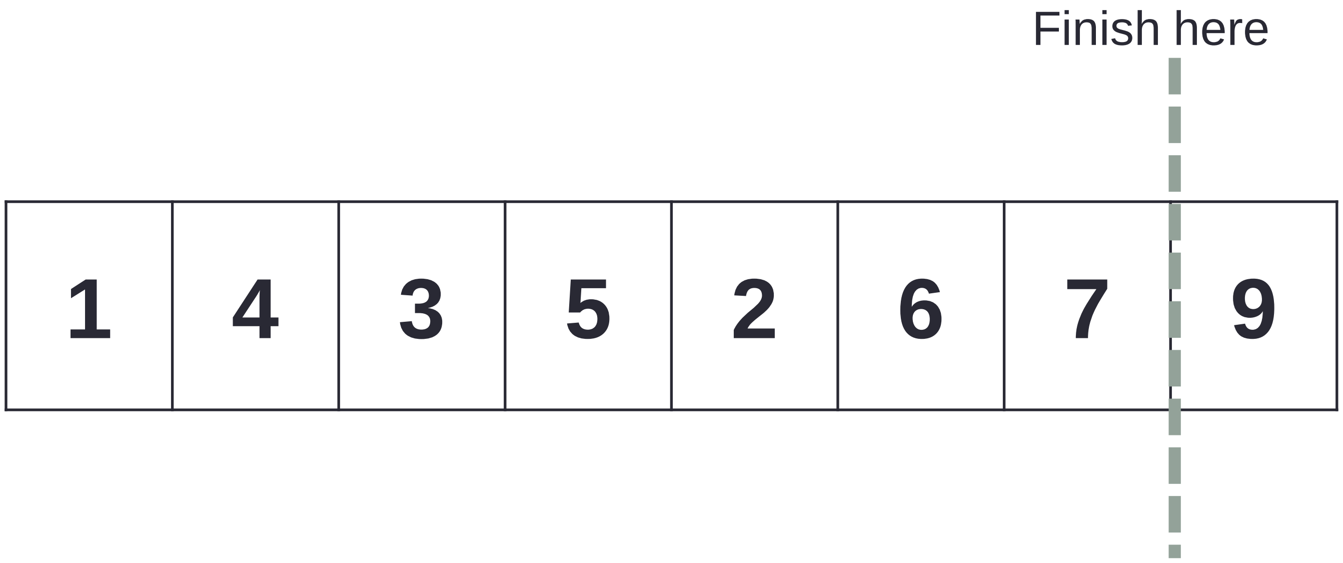
swap



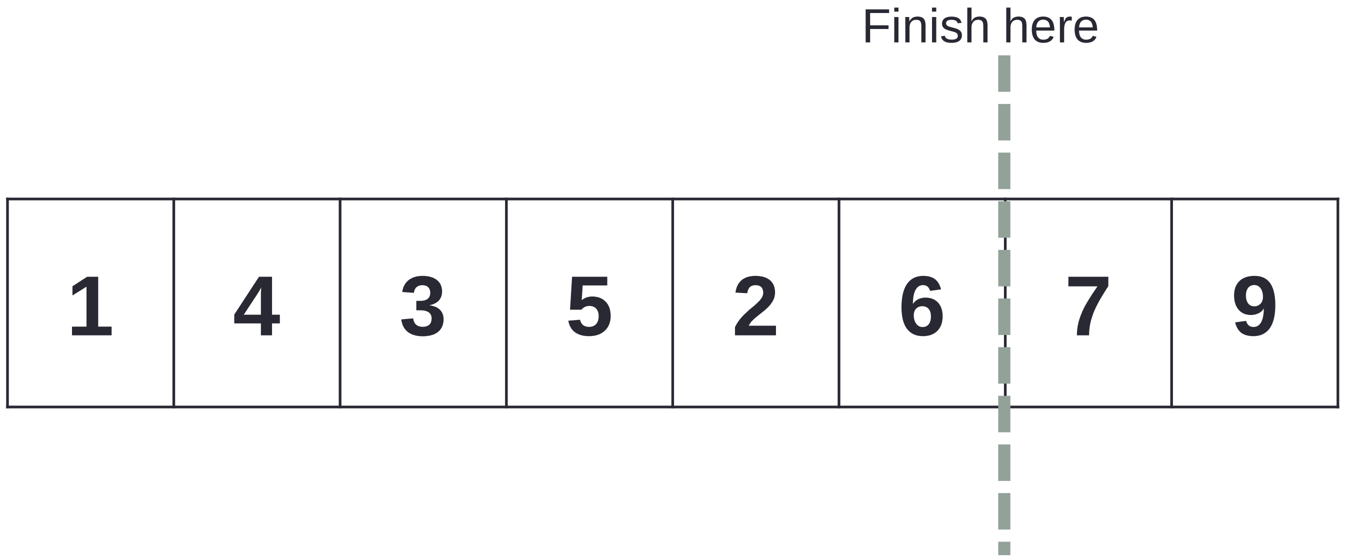
Compare next two elements



End so on until largest bubbles to the end

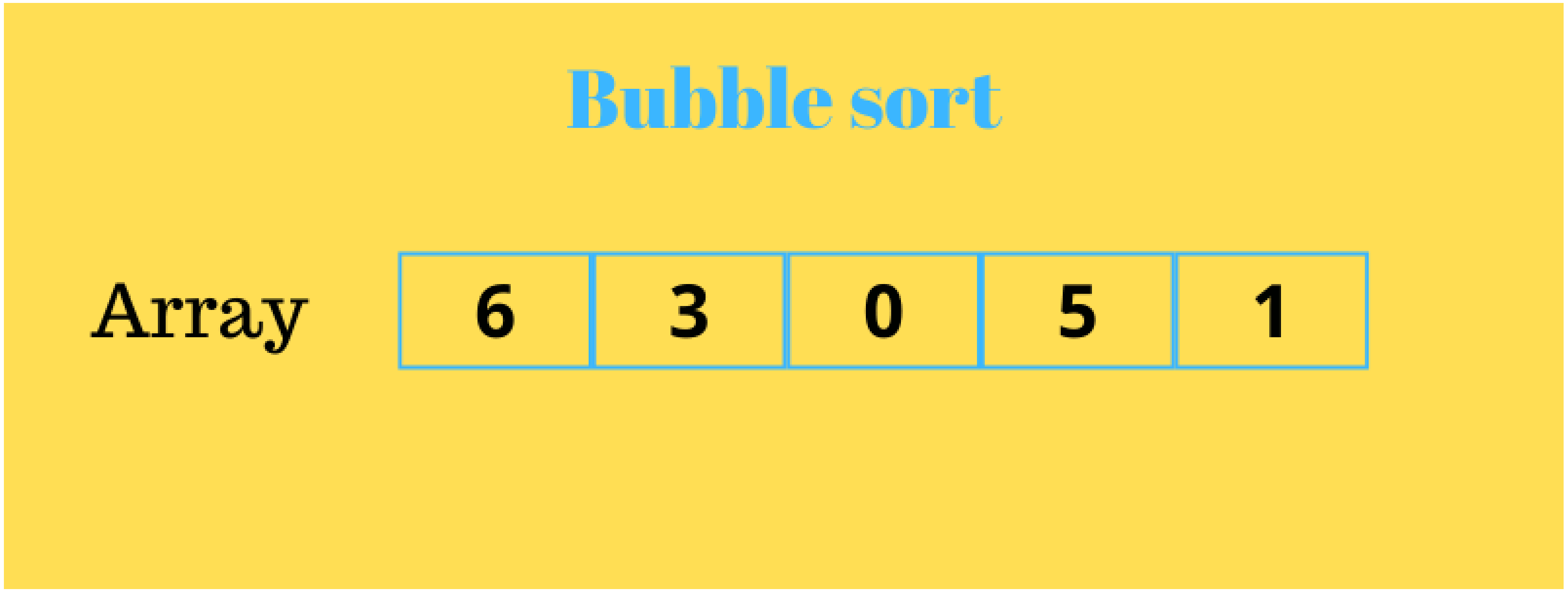


Now go back and start again at first element, finishing one place earlier



Bubble Sort Illustration

- We saw this illustration of bubble sort on Monday.

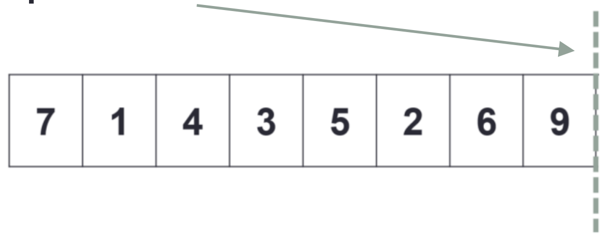


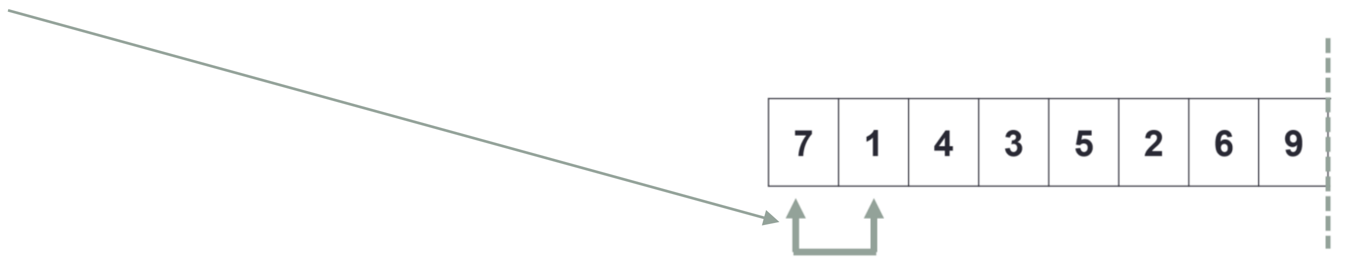
Graphic from: programmingsimplified.com

BUBBLE SORT IN C

Bubble Sort

- What will our program need?
  - Initialize array, indices, etc.
  - Outer loop to set stopping point of each pass.
  - Inner loop to do each pass.
  - If statement to compare values





Bubble Sort Pseudocode

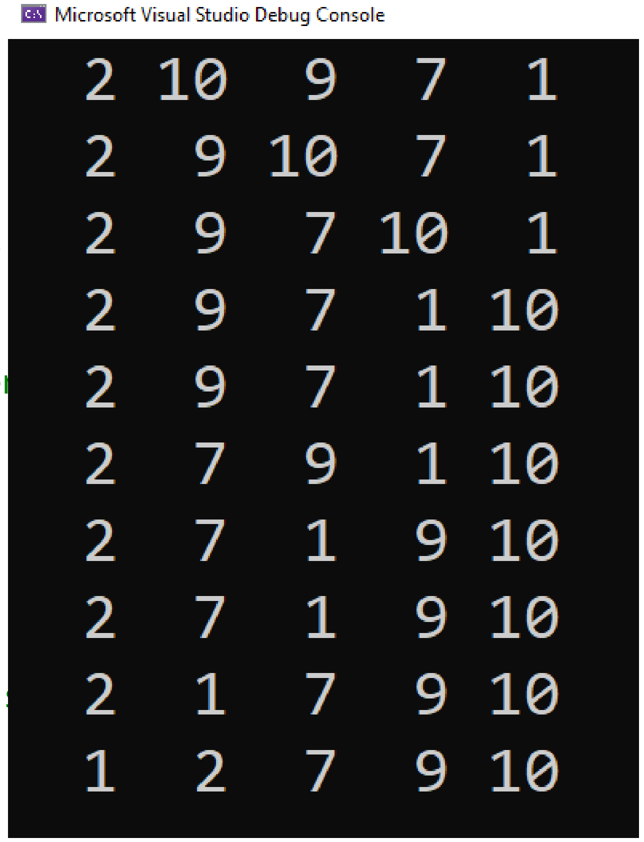
```
1. Initialize array, indices, etc.
2. For p = 0 up to array length.
3.     For i = 0 up to array length – p -1
4.         If (item at position i > item at position i + 1)
5.             Swap items
```

Bubble Sort in C

```c
void main() {
    int iarray[5] = { 10,2,9,7,1 };
    int temp;
    int len = 5, pass, i, j;

    // loop to control number of passes
    for (pass = 0; pass < len; pass++){
        //each pass we do one comparison less, as the highest number bubbles to the
        // right / top
        for (i = 0; i < len - pass - 1; i++){
            // compare adjacent elements and swap them if first element is greater
            // than second element
            if (iarray[i] > iarray[i + 1]){
                temp = iarray[i];
                iarray[i] = iarray[i + 1];
                iarray[i + 1] = temp;
            }
            // print out the array after each comparison
            for (j = 0; j < len; j++) {
                printf("%3d", iarray[j]);
            }
            printf("\n");
        }
    }
}
```

Bubble Sort in C Output

```c
void main() {
    int iarray[5] = { 10,2,9,7,1 };
    int temp;
    int len = 5, pass, i, j;

    // loop to control number of passes
    for (pass = 0; pass < len; pass++){
        //each pass we do one comparison less, as the
        // right / top
        for (i = 0; i < len - pass - 1; i++){
            // compare adjacent elements and swap ther
            // than second element
            if (iarray[i] > iarray[i + 1]){
                temp = iarray[i];
                iarray[i] = iarray[i + 1];
                iarray[i + 1] = temp;
            }
            // print out the array after each compari
            for (j = 0; j < len; j++) {
                printf("%3d", iarray[j]);
            }
            printf("\n");
        }
    }
}
```



Microsoft Visual Studio Debug Console

```
  2 10  9  7  1
  2  9 10  7  1
  2  9  7 10  1
  2  9  7  1 10
  2  9  7  1 10
  2  7  9  1 10
  2  7  1  9 10
  2  7  1  9 10
  2  1  7  9 10
  1  2  7  9 10
```

EXAMPLE PROBLEM

Card Deck Simulator

- Write software to simulate a deck of cards.
- Each card should be represented as a structure with a:
  - Face value character, e.g. A, 2, 3, 4…
  - Suit defined using an enum.
  - Integer card number.
- Write the following functions:
  - FillDeck() – This should fill up a global array of cards with 52 cards.
  - Shuffle() – This should put the cards in a random order.
  - printDeck() – This should display the deck of cards to the screen.
  - sortDeck() – This should sort the deck of cards into the order:
    - H,D,S,C, and within each suit sort A, 2, 3,4…, J, Q, K.
- Test your code by creating a deck of cards, print it, shuffle it, print it, sort it, then print it again.

Card Deck Simulator

- Go to C program solution.

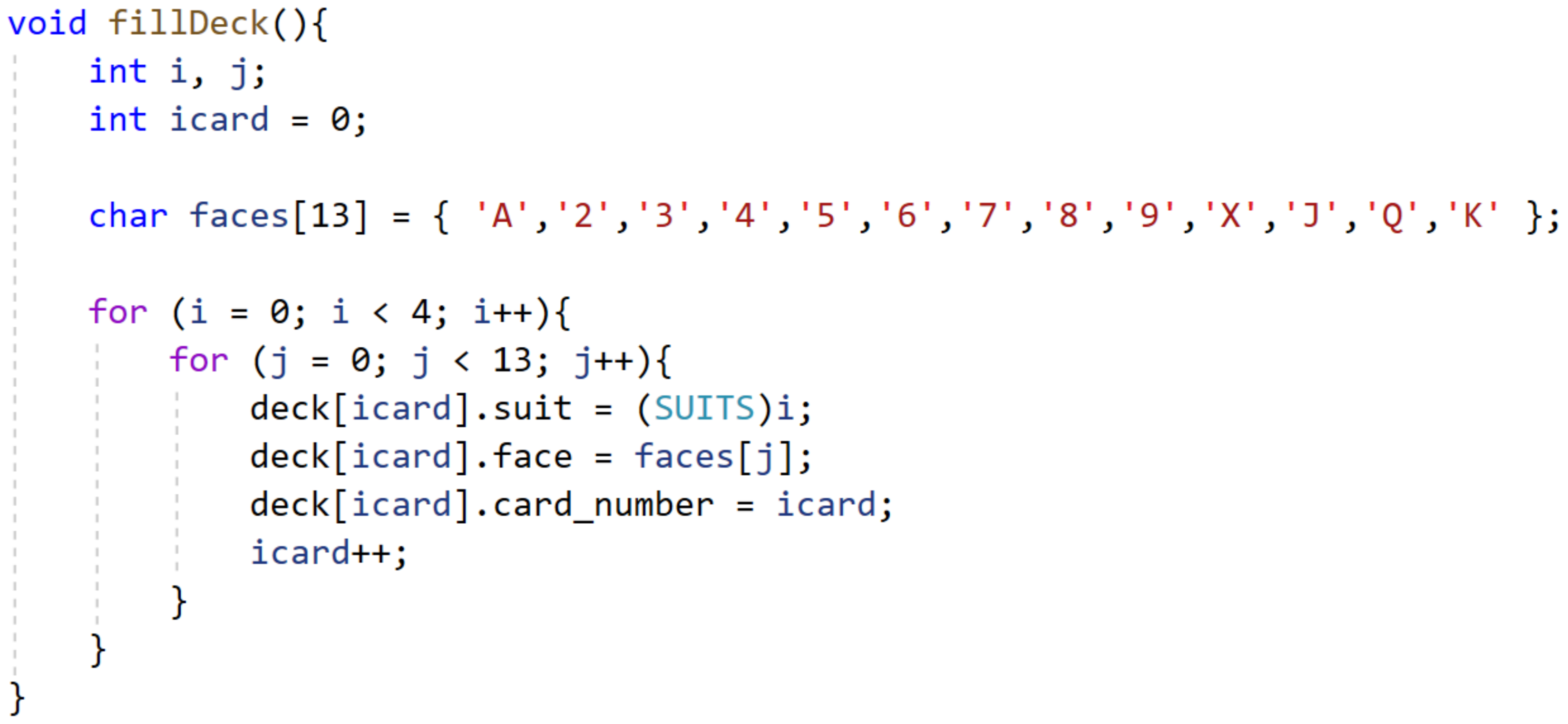
Card Deck Simulator

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>


typedef enum {hearts, diamonds, spades, clubs} SUITS;
char suitString[4][15] = { "hearts", "diamonds", "spades", "clubs"};

typedef struct {
    char face;
    SUITS suit;
    int card_number;
} card;

card deck[52];

void fillDeck();
void shuffle();
void printDeck();
void sortDeck();
```

Card Deck Simulator

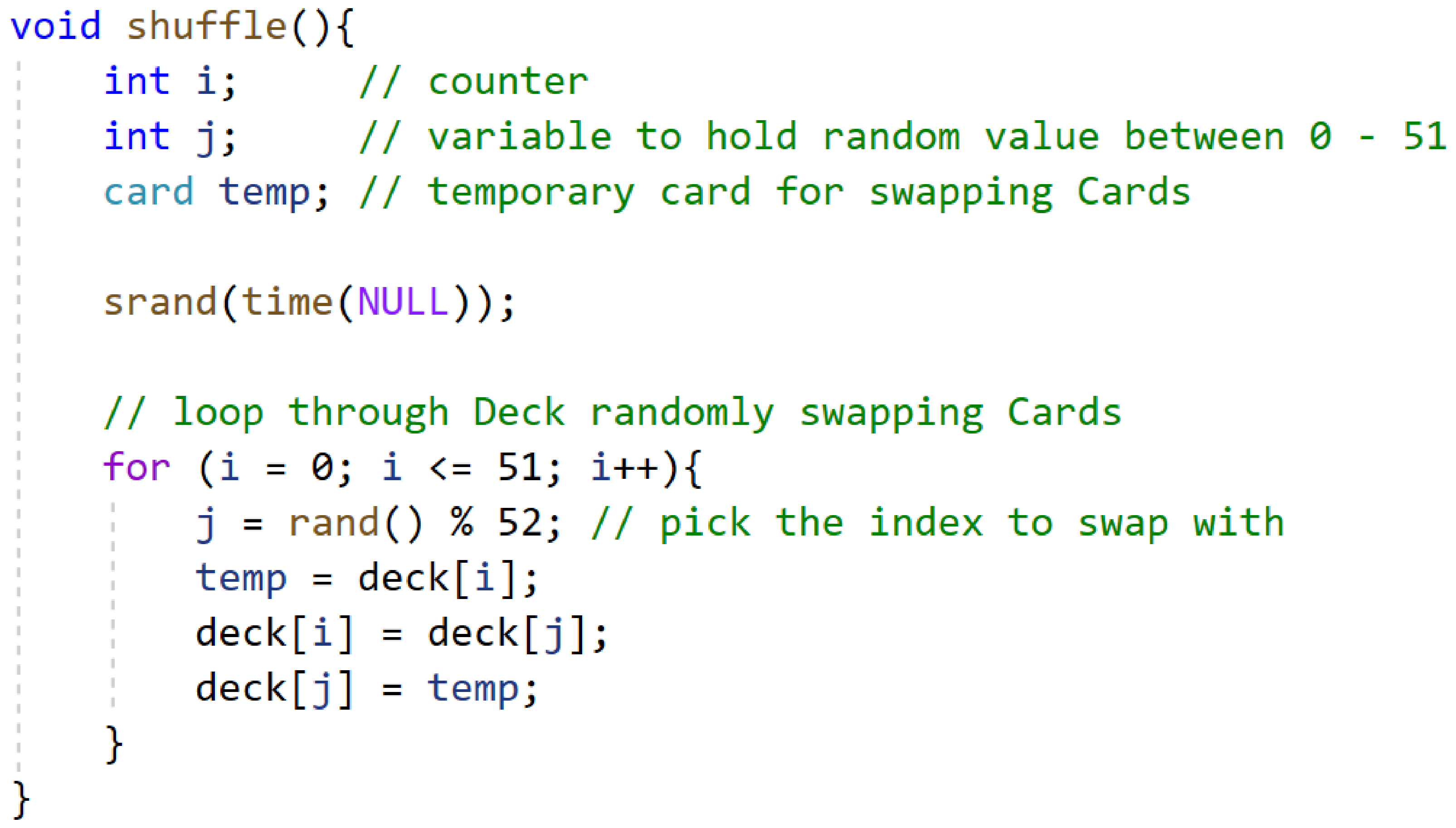
```
void main()
{
    fillDeck(deck);
    puts("Fresh Deck:");
    printDeck(deck);

    shuffle(deck);

    puts("Shuffled Deck:");
    printDeck(deck);

    sortDeck(deck);

    puts("Sorted Deck:");
    printDeck(deck);
}
```

Card Deck Simulator



```
void fillDeck(){
    int i, j;
    int icard = 0;

    char faces[13] = { 'A','2','3','4','5','6','7','8','9','X','J','Q','K' };

    for (i = 0; i < 4; i++){
        for (j = 0; j < 13; j++){
            deck[icard].suit = (SUITS)i;
            deck[icard].face = faces[j];
            deck[icard].card_number = icard;
            icard++;
        }
    }
}
```

Card Deck Simulator



```
void shuffle(){
    int i;      // counter
    int j;      // variable to hold random value between 0 - 51
    card temp; // temporary card for swapping Cards

    srand(time(NULL));

    // loop through Deck randomly swapping Cards
    for (i = 0; i <= 51; i++){
        j = rand() % 52; // pick the index to swap with
        temp = deck[i];
        deck[i] = deck[j];
        deck[j] = temp;
    }
}
```

Card Deck Simulator

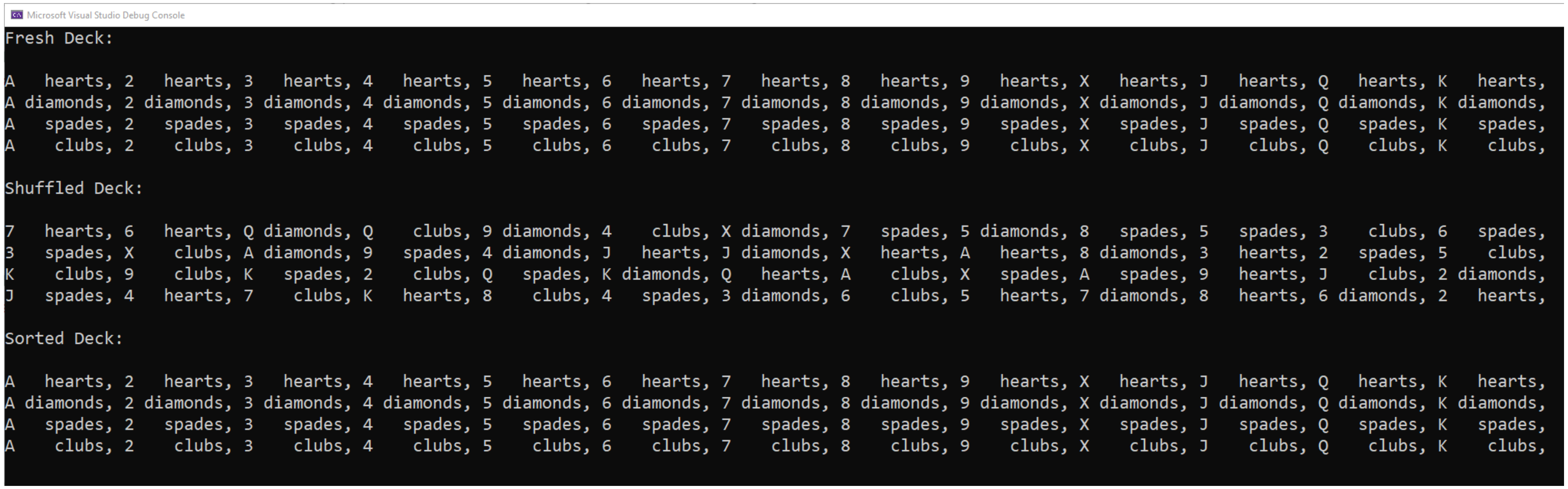
```
void printDeck(){
    int i;
    for (i = 0; i < 52; i++){
        if (i % 13 == 0) {
            printf("\n");
        }
        printf("%c %8s, ", deck[i].face, suitString[deck[i].suit]);

    }
    printf("\n\n");
}
```

Card Deck Simulator

```
void sortDeck(){
    int i, pass;
    card temp;
    for (pass = 0; pass < 52; pass++){
        for (i = 0; i < 51-pass; i++){
            if (deck[i].card_number > deck[i + 1].card_number){
                temp = deck[i];
                deck[i] = deck[i + 1];
                deck[i + 1] = temp;
            }
        }
    }
}
```

Card Deck Simulator

- C Program Output:



Microsoft Visual Studio Debug Console

```
Fresh Deck:

A   hearts, 2   hearts, 3   hearts, 4   hearts, 5   hearts, 6   hearts, 7   hearts, 8   hearts, 9   hearts, X   hearts, J   hearts, Q   hearts, K   hearts,
A diamonds, 2 diamonds, 3 diamonds, 4 diamonds, 5 diamonds, 6 diamonds, 7 diamonds, 8 diamonds, 9 diamonds, X diamonds, J diamonds, Q diamonds, K diamonds,
A   spades, 2   spades, 3   spades, 4   spades, 5   spades, 6   spades, 7   spades, 8   spades, 9   spades, X   spades, J   spades, Q   spades, K   spades,
A    clubs, 2    clubs, 3    clubs, 4    clubs, 5    clubs, 6    clubs, 7    clubs, 8    clubs, 9    clubs, X    clubs, J    clubs, Q    clubs, K    clubs,

Shuffled Deck:

7   hearts, 6   hearts, Q diamonds, Q    clubs, 9 diamonds, 4    clubs, X diamonds, 7   spades, 5 diamonds, 8   spades, 5   spades, 3    clubs, 6   spades,
3   spades, X    clubs, A diamonds, 9   spades, 4 diamonds, J   hearts, J diamonds, X   hearts, A   hearts, 8 diamonds, 3   hearts, 2   spades, 5    clubs,
K    clubs, 9    clubs, K   spades, 2    clubs, Q   spades, K diamonds, Q   hearts, A    clubs, X   spades, A   spades, 9   hearts, J    clubs, 2 diamonds,
J   spades, 4   hearts, 7    clubs, K   hearts, 8    clubs, 4   spades, 3 diamonds, 6    clubs, 5   hearts, 7 diamonds, 8   hearts, 6 diamonds, 2   hearts,

Sorted Deck:

A   hearts, 2   hearts, 3   hearts, 4   hearts, 5   hearts, 6   hearts, 7   hearts, 8   hearts, 9   hearts, X   hearts, J   hearts, Q   hearts, K   hearts,
A diamonds, 2 diamonds, 3 diamonds, 4 diamonds, 5 diamonds, 6 diamonds, 7 diamonds, 8 diamonds, 9 diamonds, X diamonds, J diamonds, Q diamonds, K diamonds,
A   spades, 2   spades, 3   spades, 4   spades, 5   spades, 6   spades, 7   spades, 8   spades, 9   spades, X   spades, J   spades, Q   spades, K   spades,
A    clubs, 2    clubs, 3    clubs, 4    clubs, 5    clubs, 6    clubs, 7    clubs, 8    clubs, 9    clubs, X    clubs, J    clubs, Q    clubs, K    clubs,
```

PROGRAMMING

CT103

Week 12b

Sign in on Blackboard

- Please sign in on blackboard for CT103.
- I will leave the sign in option open for an hour after the lecture.

Lecture Content

- Last lecture (Week 12a):
  - Bubble sort
  - Sorting
  - Example C program

- Today’s lecture (Week 12b):
  - Revision

REVISION

Data Types

- So we can see that we need different variable types, or data types, to hold information
- The basic set of C data types is:
  - **int** - this holds an integer

    e.g. 10 21 456 -6899
  - **float** – holds a floating point number

    e.g. 125.467
  - **double** – holds a very big floating point number

    e.g. up to 1.797e+308
  - **char** – holds a character

    e.g. ‘A’ ‘c’ ‘%’
  - Also **strings** – holds multiple characters

    e.g. ‘hello’

Modifiers

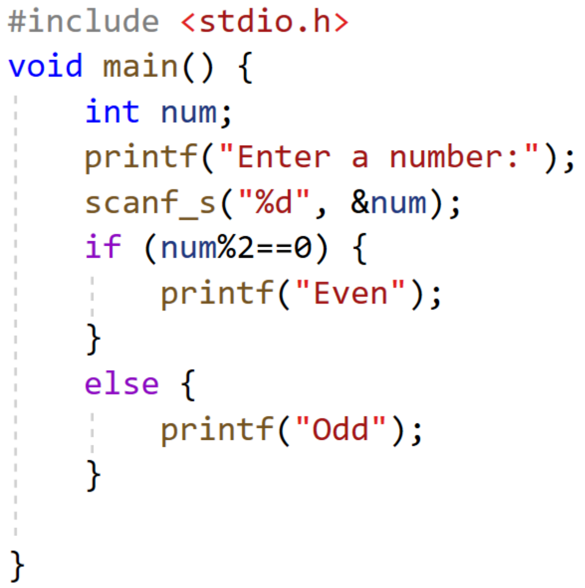
- **Short,** i.e. smaller (less memory)
- **Long**, i.e. larger (more memory)
- **Signed**, i.e. positive or negative
- **Unsigned**, i.e. non negative
- The amount of storage used for each data type (+ modifier) is not set in stone
- ANSI has the following rules:

  short int <= int <= long int

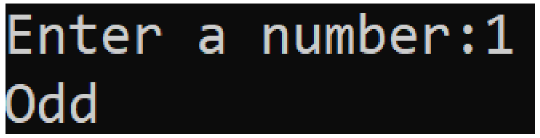
  float <= double <= long double

Modulus

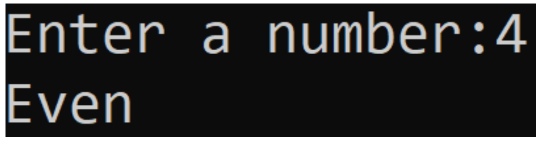
- The modulus operator allows us to get the remainder when doing integer division.
- What is the remainder?
  - When dividing two numbers that don't divide evenly, the remainder is what is left over.
  - E.g. 9/4 = 2 with a remainder of 1.
  - In C: 9%4 = 1.



```c
#include <stdio.h>
void main() {
    int num;
    printf("Enter a number:");
    scanf_s("%d", &num);
    if (num%2==0) {
        printf("Even");
    }
    else {
        printf("Odd");
    }

}
```



```
Enter a number:1
Odd
```

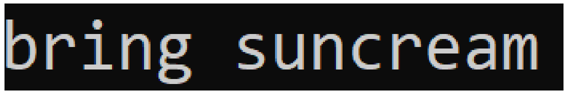


```
Enter a number:4
Even
```

If Else Statements

- We use an **if statement** to check if a statement is true.
- If it is false, do what is in the **else statement**.
- We can also have **else if statements** for multiple checks.

```c
int temp = 35; // deg C
int rain = 0; // 0 = no rain, 1 = rain
if (temp > 18) {
    if (!rain) {
        printf("bring suncream \n");
    }
}
else {
    printf("don't bring suncream \n");
}
```



```
bring suncream
```

Boolean Operators

- The primary Boolean operators are:
  - AND (In C: **&&**)
  - OR (In C: **||**)
  - NOT (In C: **!**)
  - XOR (In C: **!=**)

Truth Tables

- The following truth table shows how each of these operators work.
- In C: 1 = True, 0 = False

NOT

x	x'
0	1
1	0

AND

x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

OR

x	y	$x+y$
0	0	0
0	1	1
1	0	1
1	1	1

XOR

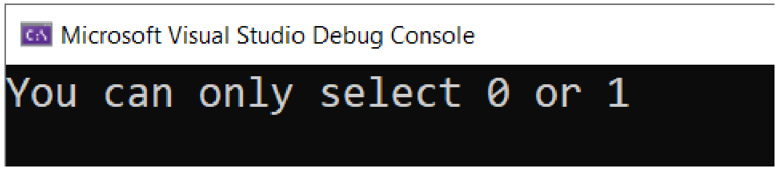
x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Source: https://introcs.cs.princeton.edu/java/71boolean/

Switch Statement

- We also talked about **Switch statements**.

```
// switch statement
int num = 11;
switch (num) {
case 0:
    printf("You have selected 0\n");
    break;
case 1:
    printf("You have selected 1\n");
    break;
default:
    printf("You can only select 0 or 1\n");
    break;
}
```

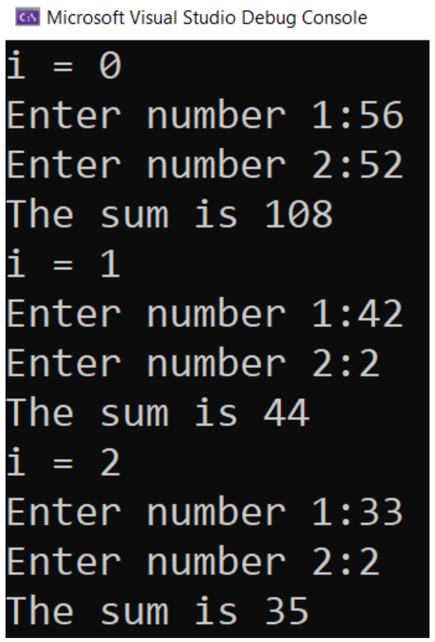


```
You can only select 0 or 1
```

While Loops

- The **while loop** will repeat a block of code over and over while some condition is true.

```c
#include <stdio.h>
void main() {
    int num1;
    int num2;
    int total;
    int i = 0;

    while (i<3){
        printf("i = %d\n",i);
        printf("Enter number 1:");
        scanf_s("%d", &num1);
        printf("Enter number 2:");
        scanf_s("%d", &num2);
        total = num1 + num2;
        printf("The sum is %d\n", total);
        i++;
    }
}
```


Microsoft Visual Studio Debug Console

```
i = 0
Enter number 1:56
Enter number 2:52
The sum is 108
i = 1
Enter number 1:42
Enter number 2:2
The sum is 44
i = 2
Enter number 1:33
Enter number 2:2
The sum is 35
```

Do While Loops

- You can use a do while loop if you want to ensure that you execute a block of code **at least once**.

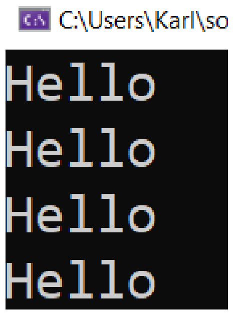
```
int j = 0;
do {
    printf("Hello\n");
    j++;
} while (j<4);
```

“Hello” will be printed at a minimum of once, irrespective of what value j has.

For Loops

- **For loops** are useful if we want to repeat some code a predetermined number of times.

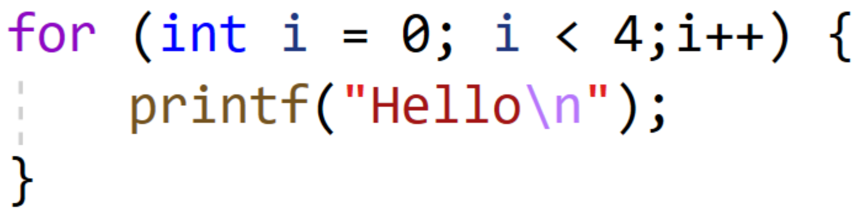
```c
int i;
for (i = 0; i < 4;i++) {
    printf("Hello\n");
}
```


C:\Users\Karl\so

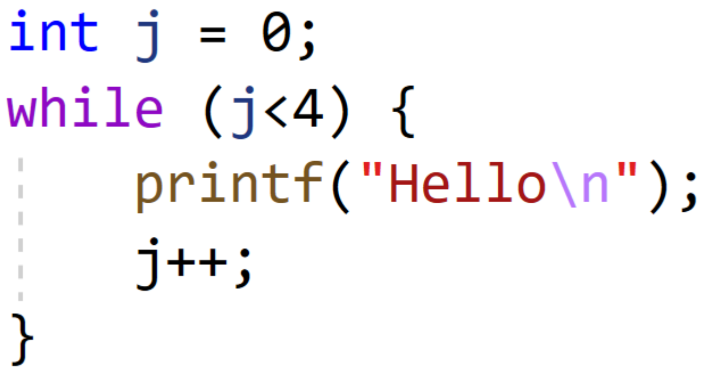
```
Hello
Hello
Hello
Hello
```

For Loop vs While Loop

- Lets compare the structure of for loops and while loops.



```
for (int i = 0; i < 4;i++) {
    printf("Hello\n");
}
```



```
int j = 0;
while (j<4) {
    printf("Hello\n");
    j++;
}
```

For Loop

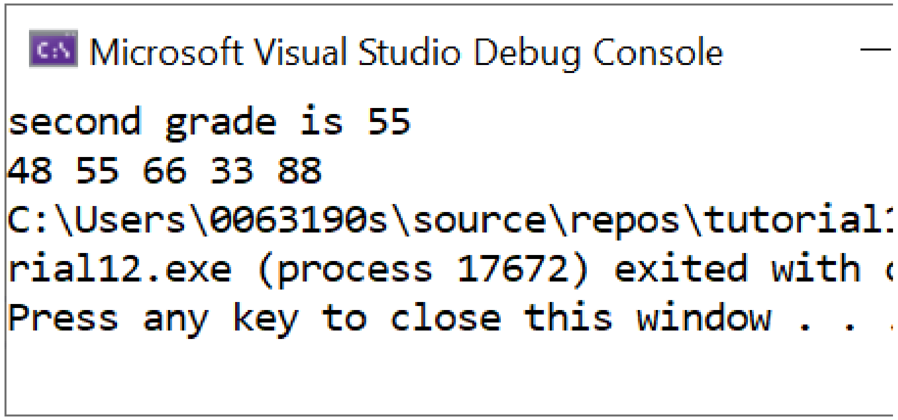
While Loop

Arrays

- An array is used to store a collection of data.

- You can think of an array as a collection of variables of the same type.

```
                 [0]  [1]  [2]  [3]  [4]
int grades[5] = { 44, 55, 66, 33, 88 };
grades[0] = 48; // easy to access/change any member of an array
printf("second grade is %d\n", grades[1]);

for (int i = 0;i < 5;i++){
  printf("%d ",grades[i]);
}
```



```
second grade is 55
48 55 66 33 88
C:\Users\0063190s\source\repos\tutorial1
rial12.exe (process 17672) exited with
Press any key to close this window . .
```

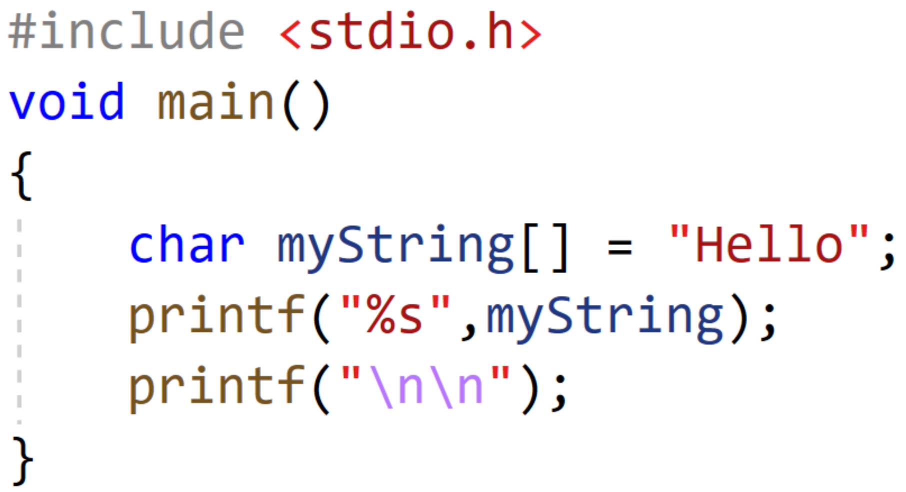
2D Arrays

- What if we have 2 dimensional (2D) data that we need to use in our program? We use 2D arrays!
- How do I loop over elements in a 2D array?
- You need 2 loops:
  - Outer loop for the rows
  - Inner loop for the columns
  - In the first part of this example, we use two loops to set the values in a 4x4 array. We use a separate variable (val) for the values in the array.

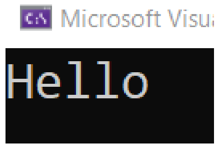
```
int x[4][4];
int r, c, val = 0;

// set array values
for (r = 0; r < 4; r++){
  for (c = 0; c < 4; c++){
    x[r][c] = val;
    val++;
  }
}
```

Strings

- A string is a collection of characters, i.e. text.
- Specifically, in C strings are defined as an array of characters.
- You should simply use %s to print strings.



```c
#include <stdio.h>
void main()
{
    char myString[] = "Hello";
    printf("%s",myString);
    printf("\n\n");
}
```



```
Hello
```

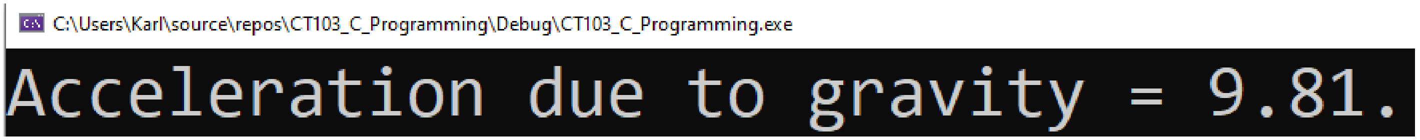
Common String functions

- Strcpy_s() Copy one string to another (**seen already**)
  - Strncpy_s() Copy n characters from one string to another
- Strcat_s() Link together (concatenate) two strings
  - Strncat_s() concatenate n characters from two strings
- strcmp() Compare two strings
  - strncmp() Compare n characters from two strings

Constants

- **Constants** refer to fixed values that the program cannot change during its execution. These are also often called literals.

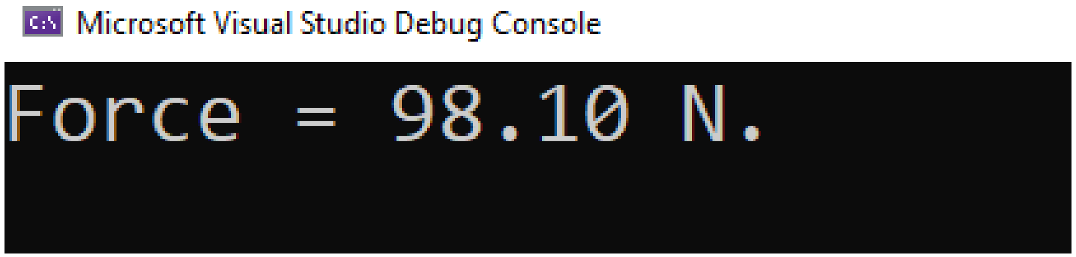
```c
const float gravity = 9.81;
printf("Acceleration due to gravity = %0.2f.\n\n", gravity);
```



```
C:\Users\Karl\source\repos\CT103_C_Programming\Debug\CT103_C_Programming.exe
Acceleration due to gravity = 9.81.
```

#Define in C Example

```c
#include <stdio.h>
#include "string.h"

#define g 9.81
void main() {
    float mass = 10;
    float F;
    F = mass * g;
    printf("Force = %0.2f N.\n",F);
}
```



Microsoft Visual Studio Debug Console

```
Force = 98.10 N.
```

Character Tests

- This is a library with functions that are useful for testing and mapping characters.

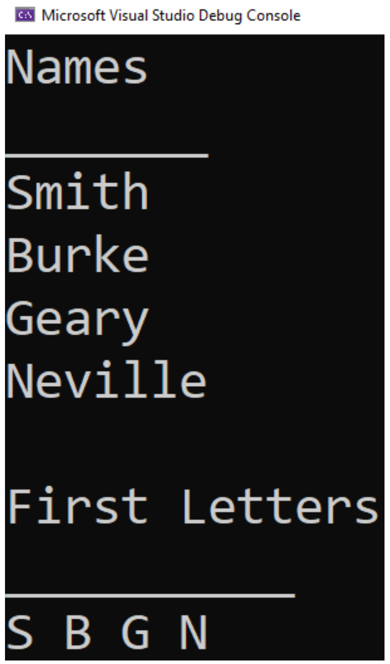
```
#include <ctype.h>
```

- Some useful character testing functions:
  - isalpha
  - isdigit
  - isupper
  - islower
  - isspace

Arrays of Strings

- As with the other 2D arrays we have seen, we can create a 2D array of characters.

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void main() {
    char names[10][20] = { "Smith", "Burke", "Geary", "Neville" };
    int i;

    puts("Names\n_______");
    for (i = 0; i < 4; i++){
        puts(names[i]);
    }

    puts("\nFirst Letters\n__________");
    for (i = 0; i < 4; i++){
        printf("%c ", names[i][0]);
    }
}
```



```
Names
_______
Smith
Burke
Geary
Neville

First Letters
__________
S B G N
```

Function Template

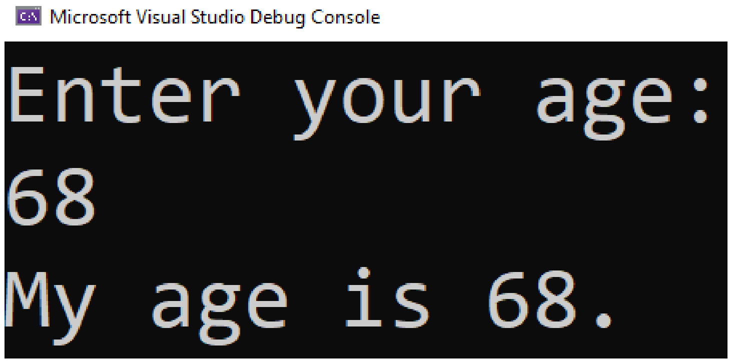
- All functions have the following template:

```
type name (parameters){
        return;
}
```

- Type = data type returned by the function (can be void).
- Name = function name.
- Parameters = data we are giving to the function (can be empty).
- Return = what data is returned by the function (can also return nothing).

C Program with Function

- C program that creates a function to read in an age.
- Notice how this function does not read in any parameters.



Microsoft Visual Studio Debug Console

```
Enter your age:
68
My age is 68.
```

```
#include <string.h>
#include <ctype.h>
#include <stdio.h>

int readAge();

void main() {
    int myAge = readAge();
    printf("My age is %d.\n", myAge);
}

int readAge() {
    int age;
    puts("Enter your age:");
    scanf_s("%d",&age);
    return age;
}
```

Function prototype

Main (we should be familiar with this one)

Function itself

Global Variables

- Global variables are variables that are created outside of a function.
- Lets look at the following example where we declare a global variable:

```
void playGame();
void displayGames();
int gamesPlayed = 0;

void main() {
    displayGames();
    playGame();
    playGame();
    playGame();
    gamesPlayed--;
    displayGames();
}

void playGame() {
    gamesPlayed++;
}

void displayGames() {
    printf("%d games have been played.\n", gamesPlayed);
}
```

Conditional Operator

- The conditional operator is a compact way of representing decision making statements.

```
void main()
{
    double taxThreshold = 30000.00;
    int lowRate = 25, highRate = 45;
    double salary;
    int rate;

    printf("Enter salary: ");
    scanf_s("%lf", &salary);

    rate = (salary >= taxThreshold) ? highRate : lowRate;

    printf("your tax rate is %d%% \n", rate);
}
```

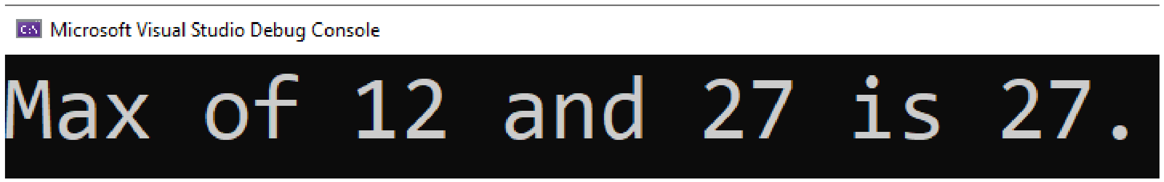
Maths in C

- Math.h has many useful functions:
  - Pow() – raise one number to the power of another.
  - Sqrt() – Square root of a number.
  - Sin() – returns the sine of an angle in radians.
  - Cos() – returns the cosine of an angle in radians.
  - Tan() – returns the tangent of an angle in radians.
  - Exp() – returns the exponent on a value.
  - Log() – returns the natural log of a value.

Function like Macros

- Function like macro:

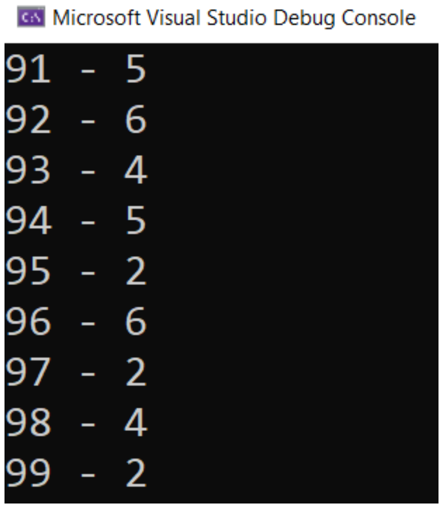
```c
#include <stdio.h>
#include <math.h>

#define MAX(x,y)(x>y?x:y)

void main() {
    int num = MAX(12,27);
    printf("Max of %d and %d is %d.\n", 12, 27,num);
}
```



```
Max of 12 and 27 is 27.
```

Random Numbers

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void main()
{
	int randNum;

	// this converts a time structure (special C type) to
	// a long integer
	// lt will contain the number of seconds since 1 Jan 1970, 00:00:00
	long lt = time(NULL);

	// srand is the seeding function - gives rand() a starting point
	srand(lt);

	for (int i = 0; i < 100; i++){
		randNum = rand() % 6 + 1;
		printf("%d - %d\n", i, randNum);
	}
}
```



```
91 - 5
92 - 6
93 - 4
94 - 5
95 - 2
96 - 6
97 - 2
98 - 4
99 - 2
```

Structures

- We can use **structures** to store information in a more organised manner.

- We declare a structure using the keyword **struct**, as follows:

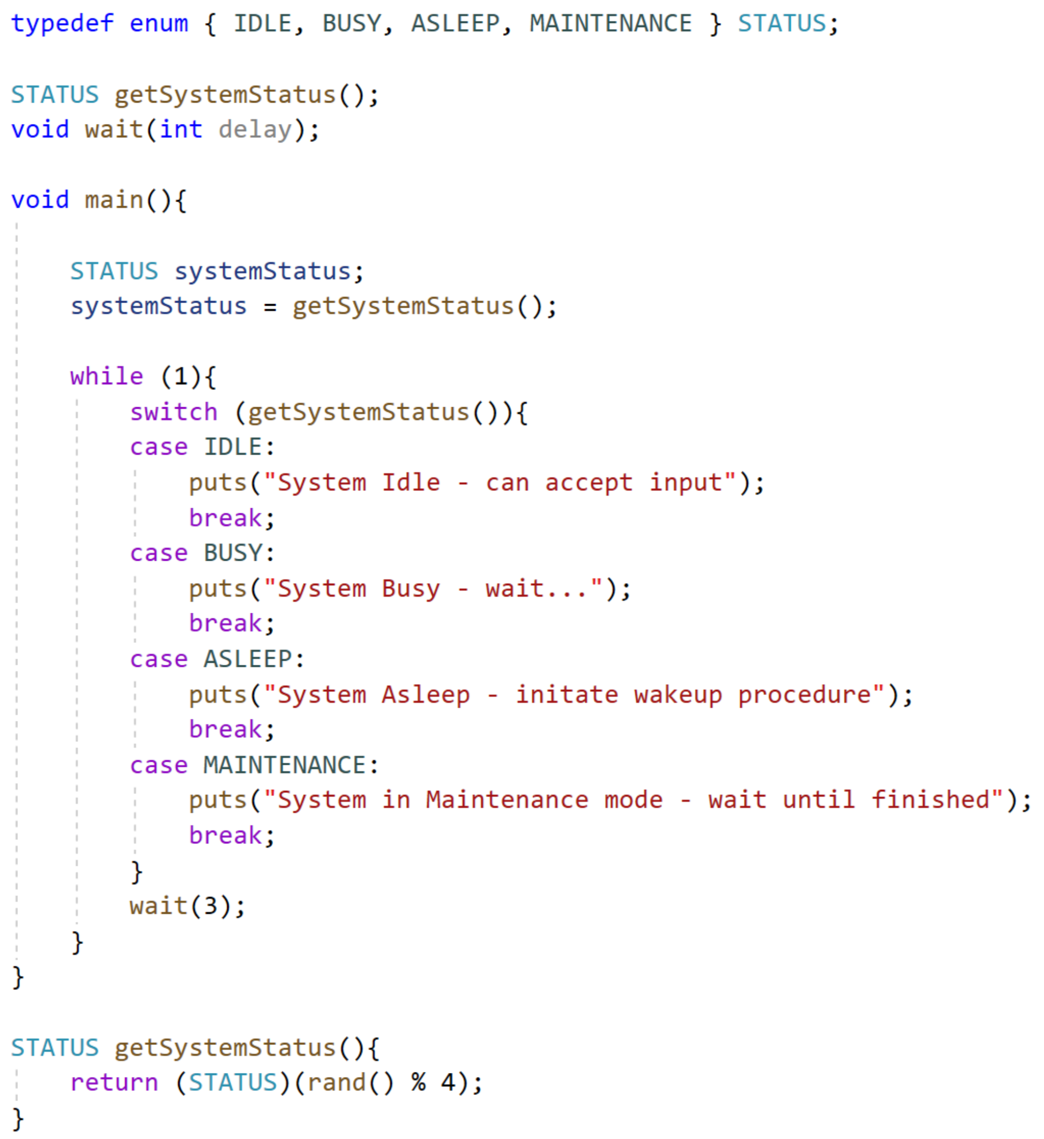
```
typedef struct customer
{
    char name[20];
    int accountNumber;
    float balance;
    char address[20];
} customer;
```

Enumeration

- Enumeration is a user defined datatype in the C programming language.

- It is used to assign names to values.

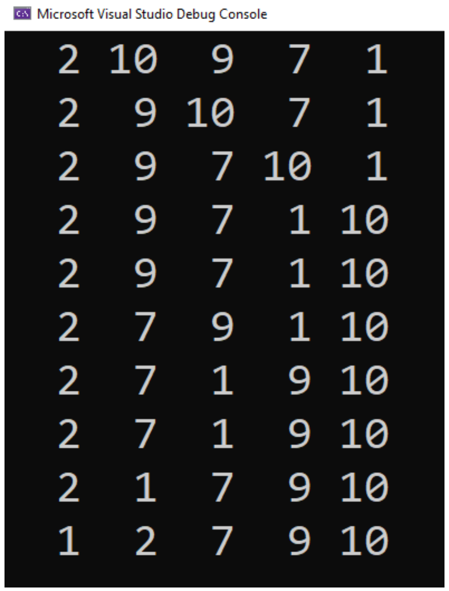
- This makes the program more readable.



```c
typedef enum { IDLE, BUSY, ASLEEP, MAINTENANCE } STATUS;

STATUS getSystemStatus();
void wait(int delay);

void main(){

    STATUS systemStatus;
    systemStatus = getSystemStatus();

    while (1){
        switch (getSystemStatus()){
        case IDLE:
            puts("System Idle - can accept input");
            break;
        case BUSY:
            puts("System Busy - wait...");
            break;
        case ASLEEP:
            puts("System Asleep - initate wakeup procedure");
            break;
        case MAINTENANCE:
            puts("System in Maintenance mode - wait until finished");
            break;
        }
        wait(3);
    }
}

STATUS getSystemStatus(){
    return (STATUS)(rand() % 4);
}
```

Bubble Sort

```c
void main() {
    int iarray[5] = { 10,2,9,7,1 };
    int temp;
    int len = 5, pass, i, j;

    // loop to control number of passes
    for (pass = 0; pass < len; pass++){
        //each pass we do one comparison less, as the highest number bubbles to the
        // right / top
        for (i = 0; i < len - pass - 1; i++){
            // compare adjacent elements and swap them if first element is greater
            // than second element
            if (iarray[i] > iarray[i + 1]){
                temp = iarray[i];
                iarray[i] = iarray[i + 1];
                iarray[i + 1] = temp;
            }
            // print out the array after each comparison
            for (j = 0; j < len; j++) {
                printf("%3d", iarray[j]);
            }
            printf("\n");
        }
    }
}
```

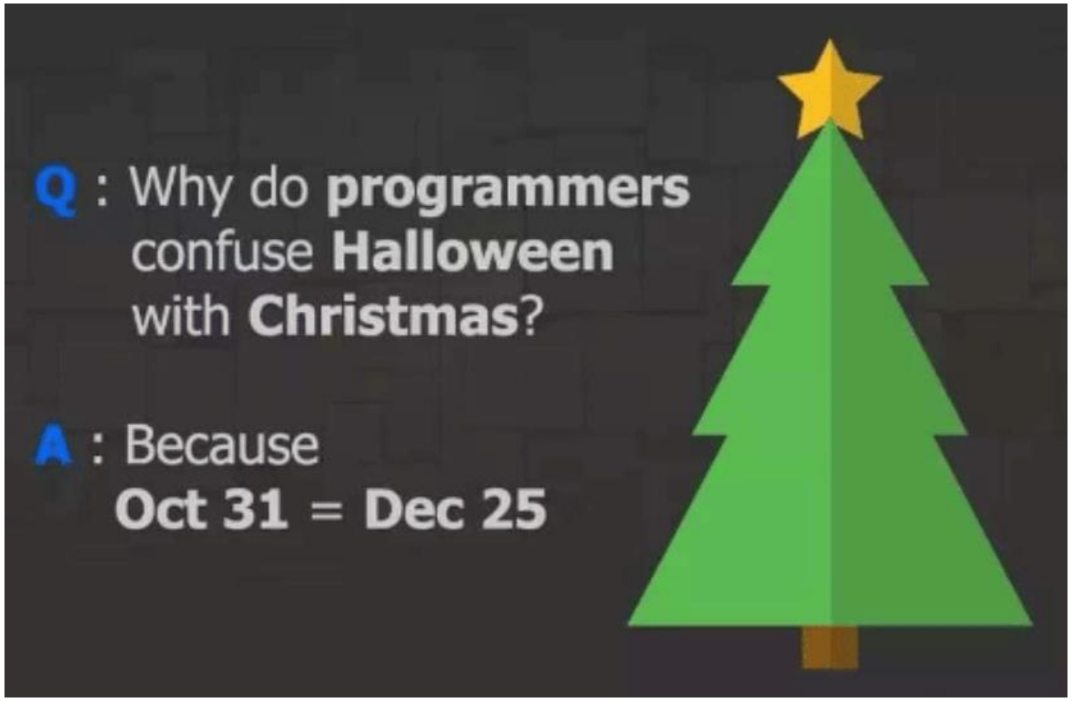


Microsoft Visual Studio Debug Console

```
  2 10  9  7  1
  2  9 10  7  1
  2  9  7 10  1
  2  9  7  1 10
  2  9  7  1 10
  2  7  9  1 10
  2  7  1  9 10
  2  7  1  9 10
  2  1  7  9 10
  1  2  7  9 10
```

Finally

- Have a good Christmas break ☺



Q : Why do **programmers** confuse **Halloween** with **Christmas**?

A : Because **Oct 31 = Dec 25**

PROGRAMMING

CT103

Week 13

Sign in on Blackboard

- Please sign in on blackboard for CT103.
- I will leave the sign in option open for an hour after the lecture.

Course Info Semester 2

- Lectures – 2 hours per week
  - **Thursday 1pm**, IT125.
  - **Thursday 1pm**, MY129 Lecture Theatre 2.
  - Attendance will be take at each lecture

- Labs – 2 hours per week
  - **Tuesday** IT106
  - **Group 1**: 2pm-4pm **Group 2**: 4pm-6pm
  - **Starts 18th January**

- Tutorials
  - **Wednesday 11am**, AC213
  - Attendance not mandatory, only if you need extra help.
  - **Starts 19th January**

Lab Groups

- **Group 1:**
  - 2pm to 4pm surnames **A** to **K**

- **Group 2:**
  - 4pm to 6pm surnames **L** to **Z**

CT103 Semester 2

- I will be lecturing this course until week 6 of semester 2.
- From week 7 until the end of semester 2, Sam will be your lecturer.

Lecture Content

- Today’s lecture (Week 13):
  - Functions Recap
  - Recursion
  - Fibonacci Sequence
  - Example C Problem
  - 3D Animated Donut in C
  - C Code in Visual Studio

FUNCTIONS RECAP

Functions

- We will be using functions a lot when we learn about recursion today.

- Let’s do a quick recap on functions.

Functions

- What is a function?

- **Definition**: A function is a piece of code that can be called whenever we need to execute that code.

Functions

- What is the point of functions?

- Benefits:
  - Functions allow us to reuse code, therefore avoid repetition.
  - More readable programs.
  - Enables us to divide complex problems into simpler ones.
  - Easier to make changes to program.

Function Template

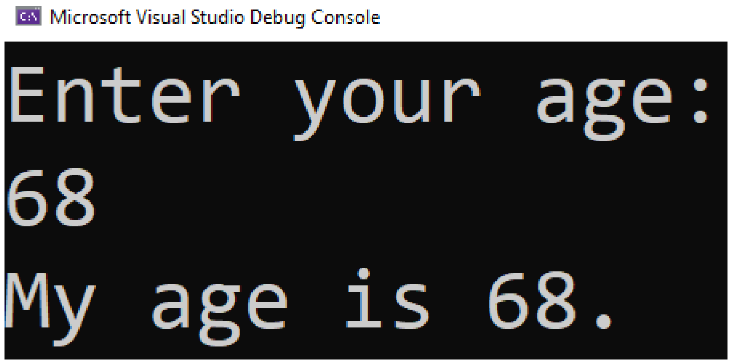
- All functions have the following template:

```
        type name (parameters){
                return;
        }
```

- Type = data type returned by the function (can be void).
- Name = function name.
- Parameters = data we are giving to the function (can be empty).
- Return = what data is returned by the function (can also return nothing).

C Program with Function

- C program that creates a function to read in an age.
- Notice how this function does not read in any parameters.



Microsoft Visual Studio Debug Console

```
Enter your age:
68
My age is 68.
```

```c
#include <string.h>
#include <ctype.h>
#include <stdio.h>

int readAge();

void main() {
    int myAge = readAge();
    printf("My age is %d.\n", myAge);
}

int readAge() {
    int age;
    puts("Enter your age:");
    scanf_s("%d",&age);
    return age;
}
```

Function prototype

Main (we should be familiar with this one)

Function itself

RECURSION

What is Recursion?

- **Recursion** is a method of problem solving where problems are solved by reducing them to smaller problems that resemble the form of the original problem.

- Recursion can make your code more readable.

Recursive Functions

- Recursive functions are functions that use recursion.
- A recursive function will call itself within the body of the function.
- You therefore need to be careful to avoid infinite loops and running out of memory…

Recursive Function Syntax

- What does a recursive function look like?

```
#include <stdio.h>

void myFunction();

void main() {
        myFunction();
}

void myFunction() {
        printf("hi.\n");
        myFunction();
}
```

Function calling itself

Recursive Function Syntax

- Be careful running this code.
- If you run this code it won't stop running and printing out "hi." until you run out of memory.
- You will get a stack overflow error.
- How do we fix this?

```
#include <stdio.h>

void myFunction();

void main() {
        myFunction();
}

void myFunction() {
        printf("hi.\n");
        myFunction();
}
```

Recursive Function V2

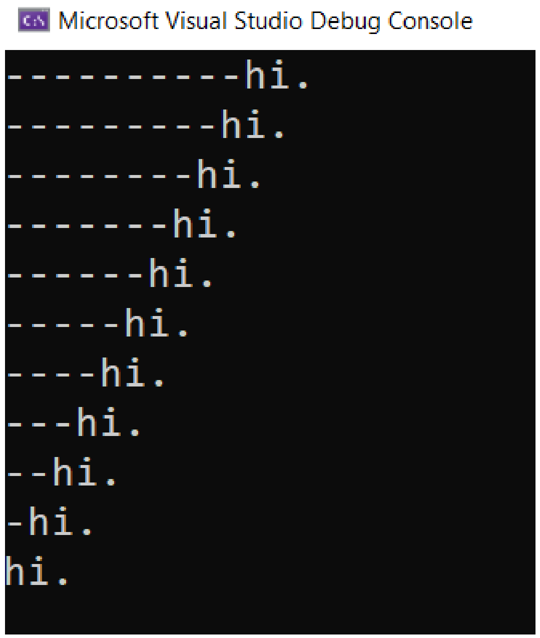
- Will this code work?

```
#include <stdio.h>

void myFunction(int n);

void main() {
   myFunction(10);
}

void myFunction(int n) {
   for (int i = 0; i < n; i++) {
      printf("-");
   }
   printf("hi.\n");
   if (n >= 1) {
      myFunction(n - 1);
   }
}
```

Function still calling itself

Recursive Function V2

- Works perfectly!



```
----------hi.
---------hi.
--------hi.
-------hi.
------hi.
-----hi.
----hi.
---hi.
--hi.
-hi.
hi.
```

```c
#include <stdio.h>

void myFunction(int n);

void main() {
    myFunction(10);
}

void myFunction(int n) {
    for (int i = 0; i < n; i++) {
        printf("-");
    }
    printf("hi.\n");
    if (n >= 1) {
        myFunction(n - 1);
    }
}
```

No Recursion Version

- Recursion is not the only way to write the program we saw on the previous slide.

- We could write a similar program in C without using recursion.

No Recursion Version

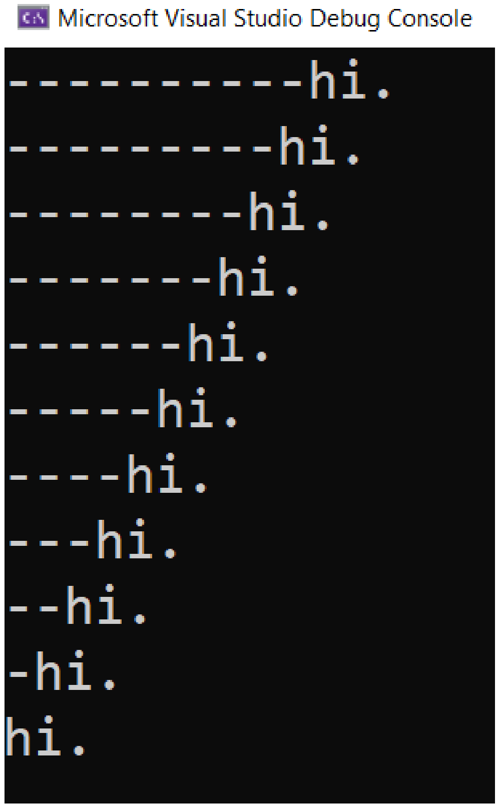
- Will this code produce the same output?

- It no longer recursively calls itself.

```
#include <stdio.h>
void myFunction(int n);

void main() {
   myFunction(10);
}

void myFunction(int n) {
   int m=n;
   for (int k = 0; k <= n;k++) {
      for (int i = 0; i < m; i++) {
         printf("-");
      }
      printf("hi.\n");
      m -=1;
   }
}
```

No Recursion Version

- Gives the same output.


Microsoft Visual Studio Debug Console

```
----------hi.
---------hi.
--------hi.
-------hi.
------hi.
-----hi.
----hi.
---hi.
--hi.
-hi.
hi.
```

```c
#include <stdio.h>
void myFunction(int n);

void main() {
   myFunction(10);
}

void myFunction(int n) {
   int m=n;
   for (int k = 0; k <= n;k++) {
      for (int i = 0; i < m; i++) {
         printf("-");
      }
      printf("hi.\n");
      m -=1;
   }
}
```

Side By Side Comparison

```
void myFunction(int n) {
   for (int i = 0; i < n; i++) {
      printf("-");
   }
   printf("hi.\n");
   if (n >= 1) {
      myFunction(n - 1);
   }
}
```

With Recursion

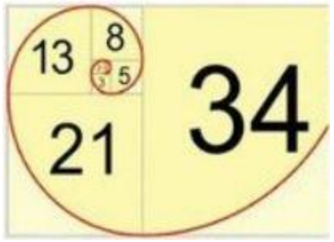
```
void myFunction(int n) {
   int m=n;
   for (int k = 0; k <= n;k++) {
      for (int i = 0; i < m; i++) {
         printf("-");
      }
      printf("hi.\n");
      m -=1;
   }
}
```

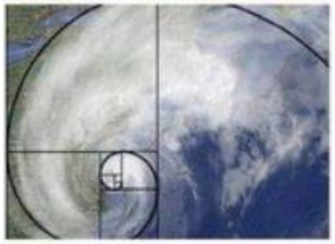
Without Recursion

FIBONACCI SEQUENCE

Fibonacci Sequence

- The **Fibonacci Sequence** is a sequence of numbers in which each number is the sum of the two preceding numbers. The sequence starts from 0 and 1.
- The sequence looks like: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …
- The Fibonacci Sequence often appears in nature.





Fibonacci Sequence in C

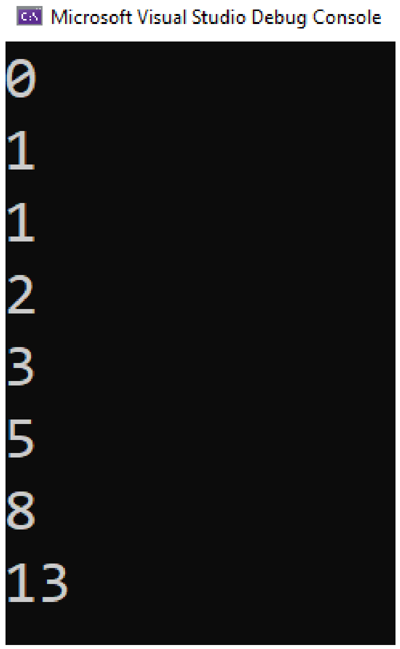
- Next we will look at how you would write a computer program that will print the Fibonacci Sequence both with and without using recursion.

Fibonacci Sequence with Recursion

```
#include <stdio.h>

int fibonacci(int n);

void main() {
  int maxN = 8;
  for (int i = 0; i < maxN;i++) {
    int ans = fibonacci(i);
    printf("%d\n", ans);
  }
}

int fibonacci(int n) {
  if (n<=1) {
    return n;
  }
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

Fibonacci Sequence with Recursion

```c
#include <stdio.h>

int fibonacci(int n);

void main() {
  int maxN = 8;
  for (int i = 0; i < maxN;i++) {
    int ans = fibonacci(i);
    printf("%d\n", ans);
  }
}

int fibonacci(int n) {
  if (n<=1) {
    return n;
  }
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```


Microsoft Visual Studio Debug Console

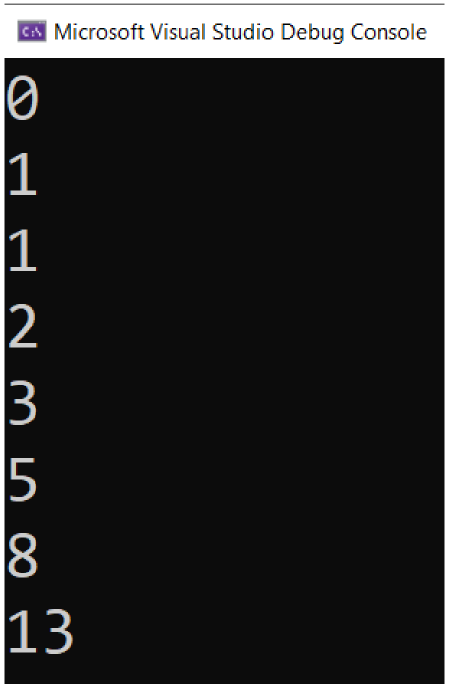
```
0
1
1
2
3
5
8
13
```

Fibonacci Sequence without Recursion

```c
#include <stdio.h>

int fibonacciNoR(int n1, int n2);

void main() {
  int maxN = 8;
  int curNum = 0;
  int prevNum = 0;
  for (int i = 0; i < maxN; i++) {
    if (i<=1) {
      curNum = i;
      prevNum = curNum - 1;
    }
    else {
      int tempNum = fibonacciNoR(curNum, prevNum);
      prevNum = curNum;
      curNum = tempNum;
    }
    printf("%d\n", curNum);
  }
}

int fibonacciNoR(int n1, int n2) {
  return n1+n2;
}
```

Fibonacci Sequence without Recursion

```c
#include <stdio.h>

int fibonacciNoR(int n1, int n2);

void main() {
  int maxN = 8;
  int curNum = 0;
  int prevNum = 0;
  for (int i = 0; i < maxN; i++) {
    if (i<=1) {
      curNum = i;
      prevNum = curNum - 1;
    }
    else {
      int tempNum = fibonacciNoR(curNum, prevNum);
      prevNum = curNum;
      curNum = tempNum;
    }
    printf("%d\n", curNum);
  }
}

int fibonacciNoR(int n1, int n2) {
  return n1+n2;
}
```



```
0
1
1
2
3
5
8
13
```

Side By Side Comparison

```
#include <stdio.h>

int fibonacci(int n);

void main() {
  int maxN = 8;
  for (int i = 0; i < maxN;i++) {
    int ans = fibonacci(i);
    printf("%d\n", ans);
  }
}

int fibonacci(int n) {
  if (n<=1) {
    return n;
  }
    return fibonacci(n - 1) + fibonacci(n - 2);
}
```

With Recursion

```
#include <stdio.h>

int fibonacciNoR(int n1, int n2);

void main() {
  int maxN = 8;
  int curNum = 0;
  int prevNum = 0;
  for (int i = 0; i < maxN; i++) {
    if (i<=1) {
      curNum = i;
      prevNum = curNum - 1;
    }
    else {
      int tempNum = fibonacciNoR(curNum, prevNum);
      prevNum = curNum;
      curNum = tempNum;
    }
    printf("%d\n", curNum);
  }
}

int fibonacciNoR(int n1, int n2) {
  return n1+n2;
}
```

Without Recursion

EXAMPLE C PROBLEM

Example C Problem

- You must write a C program that searches for a particular number.
- The target number is set randomly.
- The only information you have is the maximum possible value that the target number can have.
- Write 2 functions to search for this number. One function must use recursion, the other must not.

C Problem No Recursion Function

```c
int searchNumNoRecur(int n) {
  for (int i = 0; i < n;i++) {
    if (i == targetN) {
      return i;
    }
  }
}
```

C Problem Recursion Function

```
int searchNumRecur(int n) {
  if (n== targetN) {
    return n;
  }
  searchNumRecur(n-1);
}
```

C Problem Solution

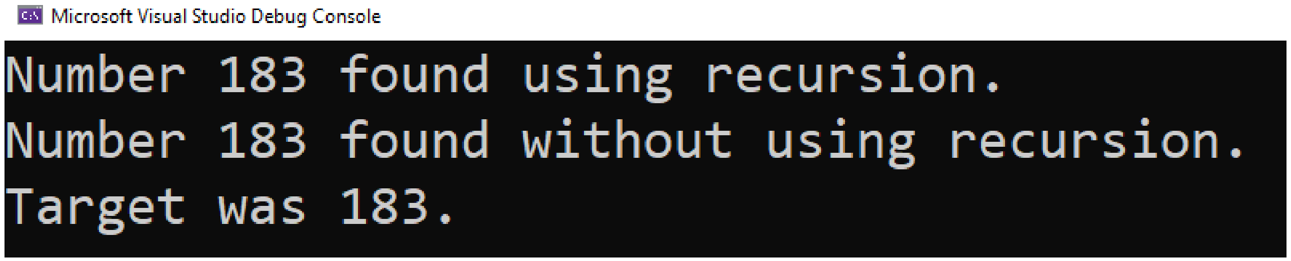
```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>

int targetN;
int searchNumRecur(int n);
int searchNumNoRecur(int n);

void main() {
  srand(time(NULL));
  int maxN = 200;
  targetN = rand() % maxN;

  int ans1 = searchNumRecur(maxN);
  int ans2 = searchNumNoRecur(maxN);
  printf("Number %d found using recursion.\n", ans1);
  printf("Number %d found without using recursion.\n", ans2);
  printf("Target was %d.\n", targetN);
}
```

```c
int searchNumRecur(int n) {
  if (n== targetN) {
    return n;
  }
  searchNumRecur(n-1);
}

int searchNumNoRecur(int n) {
  for (int i = 0; i < n;i++) {
    if (i == targetN) {
      return i;
    }
  }
}
```

C Problem Solution

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>

int targetN;
int searchNumRecur(int n);
int searchNumNoRecur(int n);

void main() {
  srand(time(NULL));
  int maxN = 200;
  targetN = rand() % maxN;

  int ans1 = searchNumRecur(maxN);
  int ans2 = searchNumNoRecur(maxN);
  printf("Number %d found using recursion.\n", ans1);
  printf("Number %d found without using recursion.\n", ans2);
  printf("Target was %d.\n", targetN);
}
```

```c
int searchNumRecur(int n) {
  if (n== targetN) {
    return n;
  }
  searchNumRecur(n-1);
}

int searchNumNoRecur(int n) {
  for (int i = 0; i < n;i++) {
    if (i == targetN) {
      return i;
    }
  }
}
```



```
Number 183 found using recursion.
Number 183 found without using recursion.
Target was 183.
```

Limits of Previous Solution

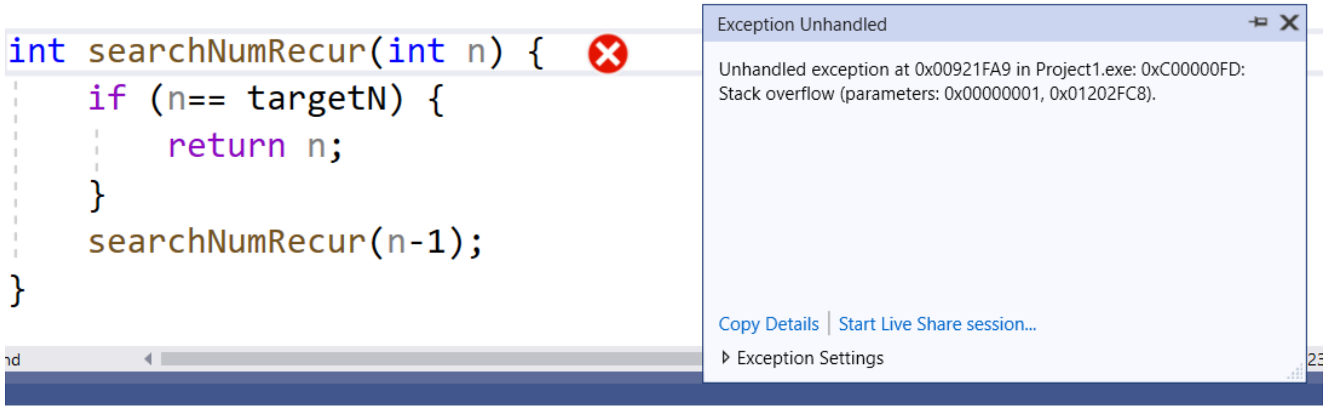
- In the solution on the previous slide, we searched for a number with a max value of 200.
- Is there a max value at which our solution will fail?
- Lets try and increase this max value.

Limits of Previous Solution

- We have now increased the maximum value “maxN” to 20,000.

```
void main() {
    srand(time(NULL));
    int maxN = 20000;
    targetN = rand() % maxN;

    int ans1 = searchNumRecur(maxN);
    int ans2 = searchNumNoRecur(maxN);
    printf("Number %d found using recursion.\n", ans1);
    printf("Number %d found without using recursion.\n", ans2);
    printf("Target was %d.\n", targetN);
}
```

- This results in too many recursive calls and we get a stack overflow error…



```
int searchNumRecur(int n) {
    if (n== targetN) {
        return n;
    }
    searchNumRecur(n-1);
}
```

Exception Unhandled

Unhandled exception at 0x00921FA9 in Project1.exe: 0xC00000FD: Stack overflow (parameters: 0x00000001, 0x01202FC8).

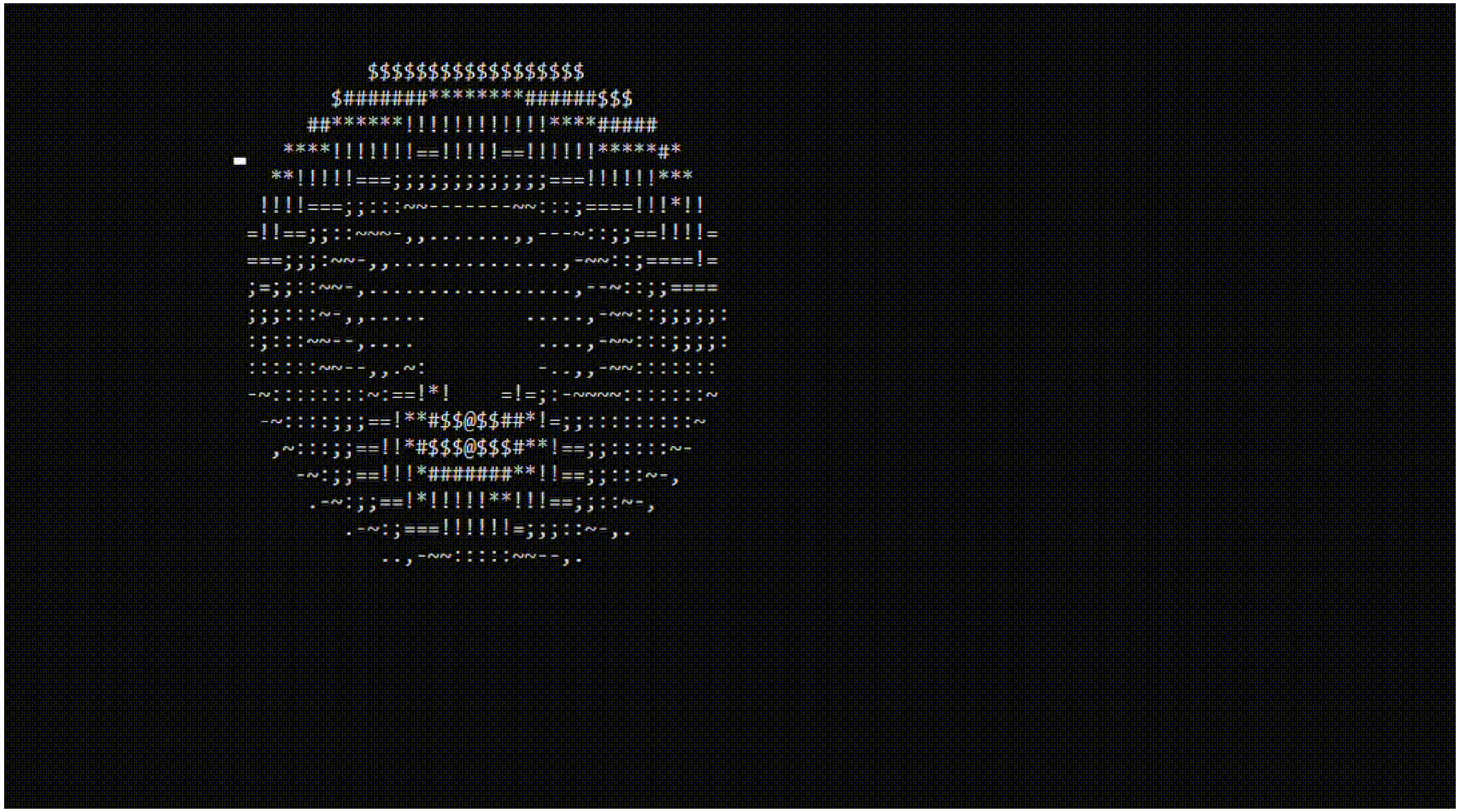
3D SPINNING DONUT USING DONUT SHAPED C CODE

Try Running the Following C Code

```c
                    k; double sin()
             ,cos(); main() {float A =
            0, B = 0, i, j, z[1760]; char b[
            1760]; printf("\x1b[2J"); for (;;
       ) { memset(b, 32, 1760); memset(z, 0, 7040)
    ; for (j = 0; 6.28 > j; j += 0.07)for (i = 0; 6.28
     > i; i += 0.02) {float c = sin(i), d = cos(j), e =
  sin(A), f = sin(j), g = cos(A), h = d + 2, D = 1 / (c *
  h * e + f * g + 5),               l=cos(i),m=cos(B),n=s\
in(B),t=c*h*g - f *                    e;int x=40+30 * D *
(l*h*m -t * n),y =                      12 + 15*D*(l* h * n
+t*m), o = x +80*y,                    N =8*((f*e -c *d* g
 ) * m - c * d *e-f*g-              l *d*n);if (22 > y &&
 y > 0 && x > 0 && 80 > x && D > z[o]) {z[o] = D;;; b[o]=
   ".,-~:;=!*#$@"[N > 0 ? N : 0];}}/*#*************!!-*/
     /***/printf("\x1b[H"); for (k = 0; 1761 > k; k++)
      putchar(k % 80 ? b[k] : 10); A += 0.04; B +=
            0.02;}}/*****##########*******!!=;:~
               ~::==!!!****************!!!==::-
                  .,~~;;;============;;;:~-.
                        ..,--------,*/
```

Don’t forget headers...
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <windows.h>

C Code Output



```
          $$$$$$$$$$$$$$$$$$
       $#######********######$$$
     ##******!!!!!!!!!!!!****#####
   ****!!!!!!!==!!!!!==!!!!!!*****#*
  **!!!!!===;;;;;;;;;;;;;===!!!!!!***
 !!!!===;;:::~~--------~~:::;====!!!*!!
=!!==;;::~~~-,,.........,,---~::;;==!!!!=
===;;;:~~-,,..............,-~~::;====!=
;=;;::~~-,................,--~::;;====
;;;:::~-,,.....        .....,-~~::;;;;;:
:;:::~~--,....          ....,-~~::;;;;:
:::::~~--,,.~:         -..,,-~~:::::::
-~::::::::~:==!*!    =!=;:-~~~~:::::::~
 -~::::;;;==!**#$$@$$##*!=;;:::::::::~
  ,~:::;;==!!*#$$$@$$$#**!==;;:::::~-
    -~:;;==!!!*#######**!!==;;:::~-,
     .-~:;;==!*!!!!!**!!!==;;::~-,
        .-~:;===!!!!!!=;;;::~-,.
           ..,-~~::::~~--,.
```

Similar Code (Not Donut Shaped)

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <windows.h>
```

```c
void main() {
    float A = 0, B = 0;
    float i, j;
    int k;
    float z[1760];
    char b[1760];
    printf("\x1b[2J");

    while(1<2) {
        memset(b, 32, 1760);
        memset(z, 0, 7040);
        for (j = 0; j < 6.28; j += 0.07) {
            for (i = 0; i < 6.28; i += 0.02) {
                float c = sin(i);
                float d = cos(j);
                float e = sin(A);
                float f = sin(j);
                float g = cos(A);
                float h = d + 2;
                float D = 1 / (c * h * e + f * g + 5);
                float l = cos(i);
                float m = cos(B);
                float n = sin(B);
                float t = c * h * g - f * e;
                int x = 40 + 30 * D * (l * h * m - t * n);
                int y = 12 + 15 * D * (l * h * n + t * m);
                int o = x + 80 * y;
                int N = 8 * ((f * e - c * d * g) * m - c * d * e - f * g - l * d * n);
                if (22 > y && y > 0 && x > 0 && 80 > x && D > z[o]) {
                    z[o] = D;
                    b[o] = ".,-~:;=!*#$@"[N > 0 ? N : 0];
                }
            }
        }
        printf("\x1b[H");
        for (k = 0; k < 1761; k++) {
            putchar(k % 80 ? b[k] : 10);
            A += 0.00004;
            B += 0.00002;
        }
        Sleep(10);
    }

}
```

Donut Code Source Material

- Code source - Andy Sloane (https://www.a1k0n.net/).

- If you want to read more about how this code works, see the following links:
  - https://www.a1k0n.net/2011/07/20/donut-math.html
  - https://www.dropbox.com/s/79ga2m7p2bnj1ga/donut_deobfuscated.c?dl=0
  - https://www.youtube.com/watch?v=DEqXNfs_HhY

C CODE

C Code

- Let’s finish today’s lecture by running some C programs in Visual Studio.

PROGRAMMING

CT103

Week 14

Sign in on Blackboard

- Please sign in on blackboard for CT103.
- I will leave the sign in option open for an hour after the lecture.

Lecture Content

- Today’s lecture (Week 14):
  - Reading in data from file
  - Writing to a file
  - Structured data and files
  - Example C programme

FILE INPUT

File Input

- Up until now, all of the information/data in our programs has been either:
  - “hard-coded” into the program as a variable by the programmer.
  - Passed into the program through the console.
- What if we want to use other information/data in our program that is stored somewhere else on our computer?

File Input

- What if we want to use other information/data in our program that is stored somewhere else on our computer?
- We are able to read in data from other files into our C program.
- Today we will focus on text (.txt) files.

File Access

- How do we access the file?

- There are two types of file access:
  - **Sequential access**:
    - You just start at the beginning and read in the data in a continuous stream.
  - **Random access**:
    - You can jump around the file, reading (and writing) data at different locations.

- We will start with sequential access.

File Pointers

- In order to read in our file, we need to use file pointers.
- We mentioned **pointers** a few times in the course so far, we will cover pointers in depth in the coming weeks.
- Briefly, **pointers** are variables that are used to store addresses of other variables.

File Pointers

- What is a file pointer?
- A **file pointer** is a pointer used to manage and keep track of the files being accessed.
- The program needs a physical address to read / write from. For this we use a FILE pointer.
- We *open* the file to set up the pointer.
- When we are finished with the file we must *close* it.

File Pointers

- What does a file pointer look like in C?
- You can create one as follows:

```
FILE* fptr;
```

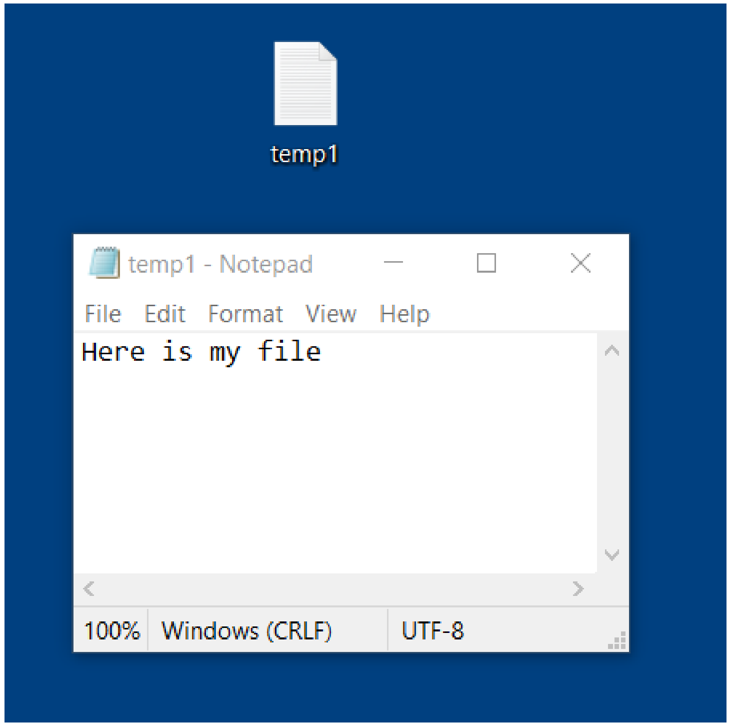
- You will need the following library:

```
#include "stdio.h"
```

OPENING A FILE

Opening a File

- Lets say you have a file on your machine called 'temp1.txt'.



```
Here is my file
```

Opening a File

- The following C program will open the file ‘temp1.txt’.

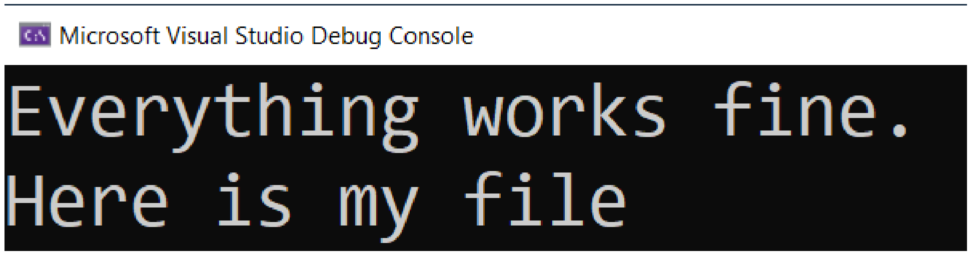
```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
  FILE *fptr;
  fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\temp1.txt", "r");

  if (fptr == NULL){
    puts("Error Opening File \n Exiting ........");
    return;
  }
  else {
    printf("Everything works fine.\n");
    char c = fgetc(fptr);
    while (c != EOF){
      printf("%c", c);
      c = fgetc(fptr);
    }
  }

  fclose(fptr);
}
```

This will be a different file path for you

EOF means ‘End Of File’

Opening a File Output

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
  FILE *fptr;
  fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\temp1.txt", "r");

  if (fptr == NULL){
    puts("Error Opening File \n Exiting ........");
    return;
  }
  else {
    printf("Everything works fine.\n");
    char c = fgetc(fptr);
    while (c != EOF){
      printf("%c", c);
      c = fgetc(fptr);
    }
  }

  fclose(fptr);
}
```



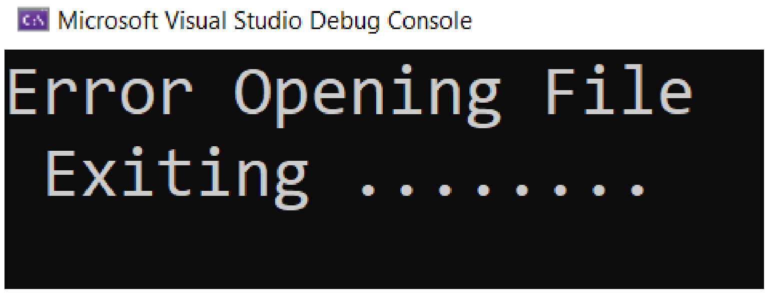
```
Everything works fine.
Here is my file
```

Opening a File

- What if the path name is wrong?

Incorrect spelling

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
  FILE *fptr;
  fopen_s(&fptr, "C:\\Users\\Karl\\Desktopp\\temp1.txt", "r");

  if (fptr == NULL){
    puts("Error Opening File \n Exiting ........");
    return;
  }
  else {
    printf("Everything works fine.\n");
    char c = fgetc(fptr);
    while (c != EOF){
      printf("%c", c);
      c = fgetc(fptr);
    }
  }

  fclose(fptr);
}
```


Microsoft Visual Studio Debug Console

```
Error Opening File
 Exiting ........
```

WRITING STRINGS TO A FILE

Writing to a File

- In the previous example, we read data in from a file.
- This is useful, however what if we want to store data generated my our program in a file?
- We will talk about writing to a file next.

Writing to a File

- We used the following line of code to open our file:

```
fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\temp1.txt", "r");
```

File pointer address   File location   Read indicator

- The “r” indicates we are opening the file for reading only.
- If we want to write to the file, we need to change this mode.

File Open Modes

- “r”: Opens a file for reading.
    - The file must exist.
- “w”: Creates an empty file for writing.
    - If a file with the same name already exists, its content is erased and the file is considered as a new empty file.
- “a”: Appends to a file.
    - Writing operations, append data at the end of the file. The file is created if it does not exist.
- “r+”: Opens a file to update both reading and writing.
    - The file must exist.
- “w+”: Creates an empty file for both reading and writing.
- “a+”: Opens a file for reading and appending.

Opening a File to Write

- So in order to write to the file, we would need to do something like the following:

```
fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\temp2.txt", "w");
```

- This will open a file that we can write to.
- However, after we open the file, how do we actually write to it?

Writing to a File

- How do we actually write to the file?
- There are “file” versions of most input/output functions that you would use to input/output with the keyboard/screen.
- The difference is that you point the input/output to a particular physical location via the FILE* pointer.

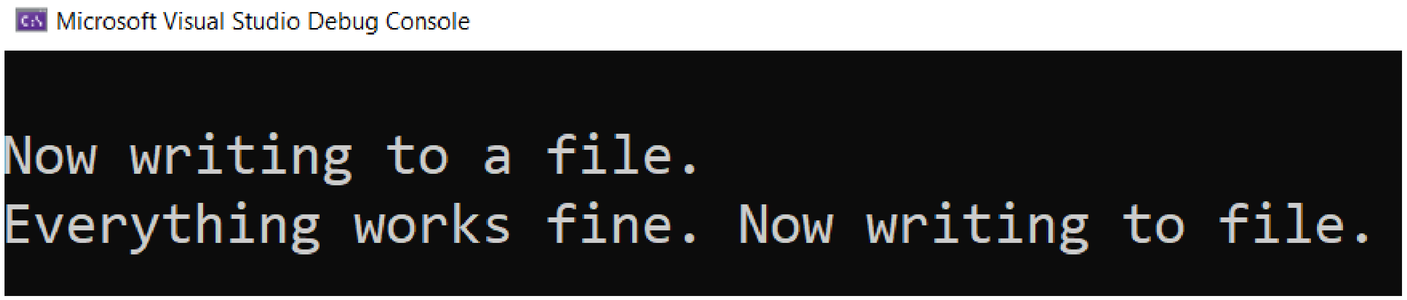
Writing to a File in C

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
  FILE *fptr;
  printf("\nNow writing to a file.\n");

  fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\temp2.txt", "w");

  if (fptr == NULL) {
    puts("Error Opening File \n Exiting ........");
    return;
  }
  else {
    printf("Everything works fine. Now writing to file.\n");
      for (int i = 0; i < 5;i++) {
        fprintf(fptr,"Line %d of text.\n",(i+1));
      }
  }
  fclose(fptr);
}
```

Now writing to a file

‘fprintf’ to print to file

Writing to a File in C Output

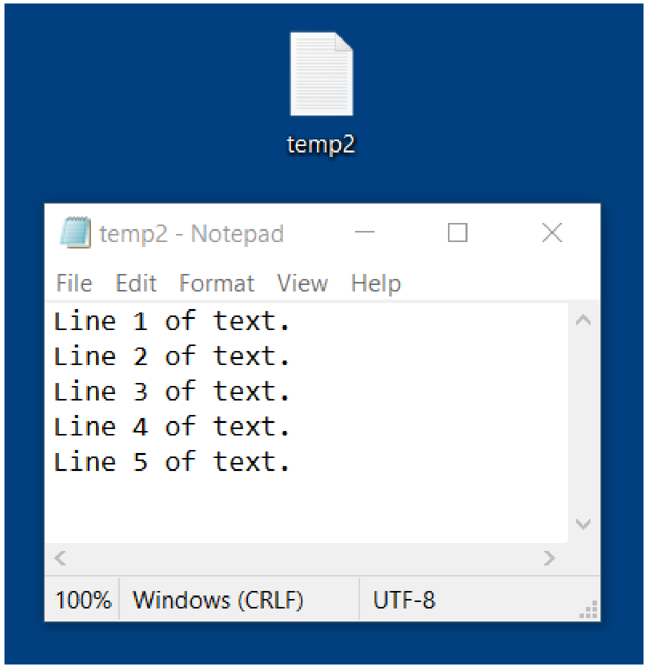

Microsoft Visual Studio Debug Console

```
Now writing to a file.
Everything works fine. Now writing to file.
```

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
  FILE *fptr;
  printf("\nNow writing to a file.\n");

  fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\temp2.txt", "w");

  if (fptr == NULL) {
    puts("Error Opening File \n Exiting ........");
    return;
  }
  else {
    printf("Everything works fine. Now writing to file.\n");
      for (int i = 0; i < 5;i++) {
        fprintf(fptr,"Line %d of text.\n",(i+1));
      }
  }
  fclose(fptr);
}
```

Writing to a File

- We can also see on the desktop, the new file ‘temp2.txt’:
- Note: Of course you can create files anywhere you like.



```
Line 1 of text.
Line 2 of text.
Line 3 of text.
Line 4 of text.
Line 5 of text.
```

Read the File Temp2

- We can read the file we just created, back into memory if we like.
- Previously we read in each character using ‘fgetc()’

```
c = fgetc(fptr);
```

- We can also read in full lines at a time.

Read the File Temp2

- The easiest way to read the data from the file is to use *fgets()* to read it line by line.

- With *fgets()* you have to specify the maximum length array you're reading into.

- If there are less characters on the line than the length of the array, that's no problem – *fgets()* will stop reading when it gets to the newline character.

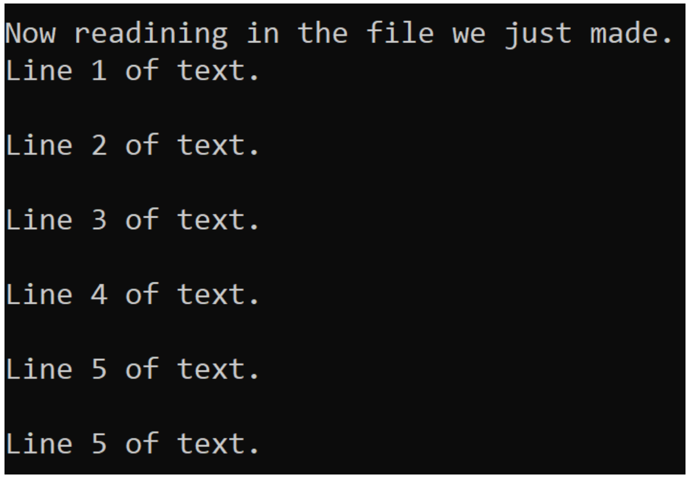
- It reads each line as a string.

Read the File Temp2

```
printf("\nNow readining in the file we just made.\n");

char line[101];
fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\temp2.txt", "r");

if (fptr == NULL){
    puts("Error Opening File \n Exiting ........");
    return;
}

while (!feof(fptr)){
    fgets(line, 101, fptr);
    puts(line);
}
fclose(fptr);
```

Read the File Temp2

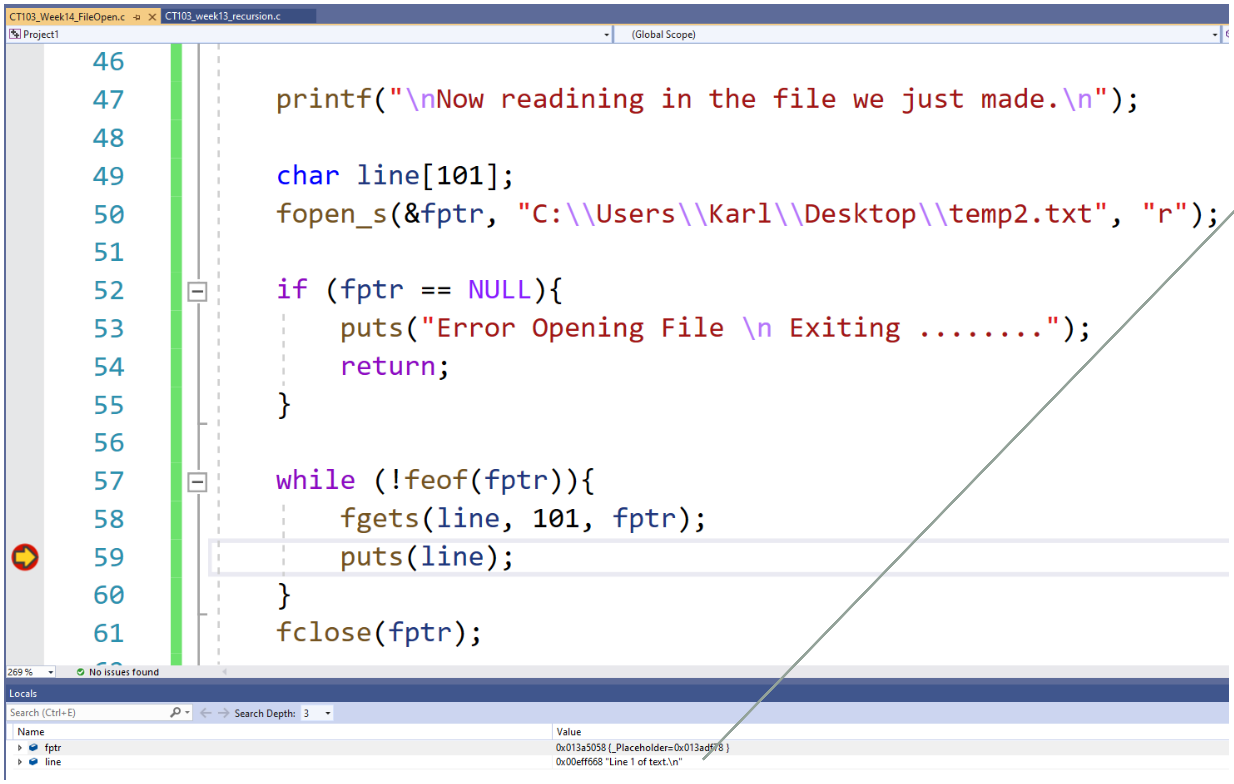
```c
printf("\nNow readining in the file we just made.\n");

char line[101];
fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\temp2.txt", "r");

if (fptr == NULL){
    puts("Error Opening File \n Exiting ........");
    return;
}

while (!feof(fptr)){
    fgets(line, 101, fptr);
    puts(line);
}
fclose(fptr);
```



```
Now readining in the file we just made.
Line 1 of text.

Line 2 of text.

Line 3 of text.

Line 4 of text.

Line 5 of text.

Line 5 of text.
```

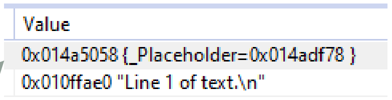
Debugger

- We can use the debugger that we saw in semester 1 to see the data read from the file:



```c

    printf("\nNow readining in the file we just made.\n");

    char line[101];
    fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\temp2.txt", "r");

    if (fptr == NULL){
        puts("Error Opening File \n Exiting ........");
        return;
    }

    while (!feof(fptr)){
        fgets(line, 101, fptr);
        puts(line);
    }
    fclose(fptr);
```

Name	Value
fptr	0x013a5058 {_Placeholder=0x013adf78 }
line	0x00eff668 "Line 1 of text.\n"



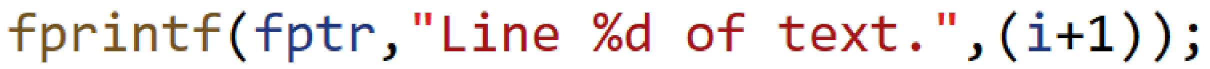
Value
0x014a5058 {_Placeholder=0x014adf78 }
0x010ffae0 "Line 1 of text.\n"

"Line 1 of text.\n"

New Lines

“Line 1 of text.\n”

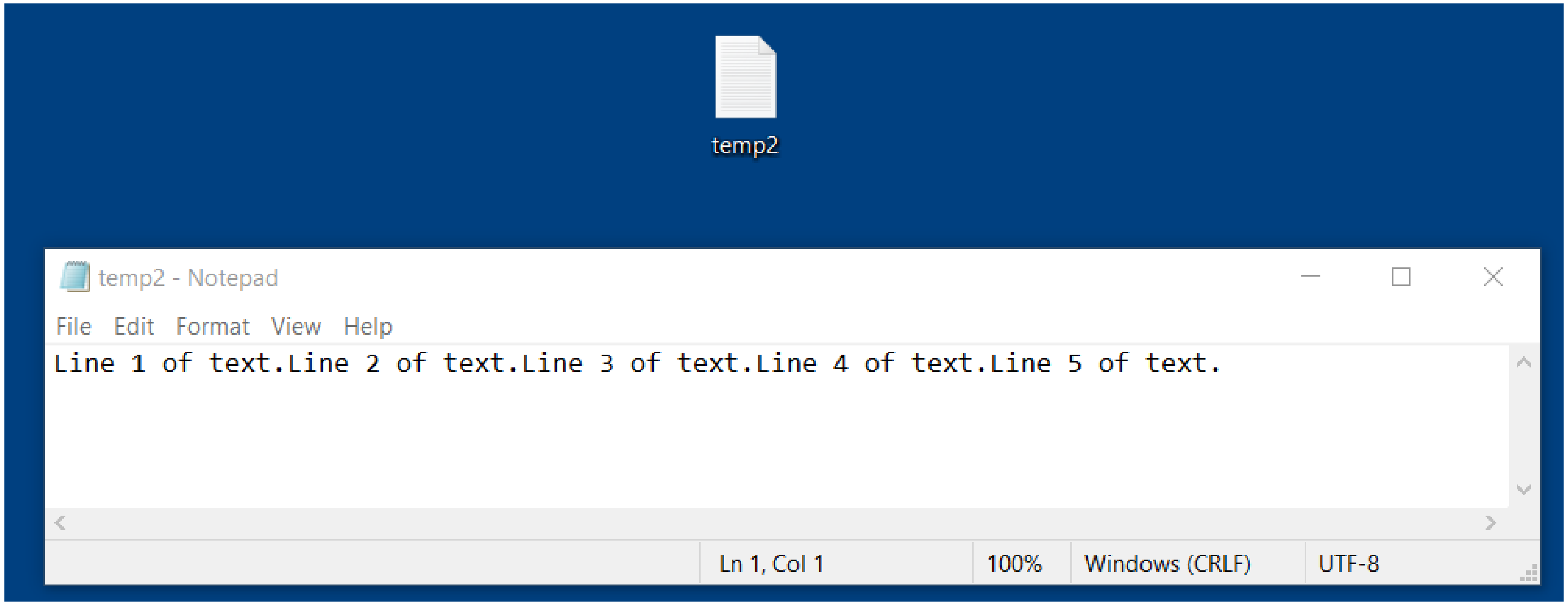
- What if we leave the \n out when writing to the file?



```
fprintf(fptr,"Line %d of text.",(i+1));
```

New Lines

- What if we leave the \n out when writing to the file?
- We will end up with this:



```
Line 1 of text.Line 2 of text.Line 3 of text.Line 4 of text.Line 5 of text.
```

fputs

- We used fprintf() to write to a file before.
- We can also use fputs():

```
fputs("Line of text.\n", fptr);
```

End of File

- We already saw feof(fptr).
- This is a C library function that checks the file stream (via the FILE* pointer) for an ‘end-of-file’ indicator.
- This indicator would be set when a function reading from the file reaches the end of the file (data stream).
- `feof()` returns false as long as the end-of-file indicator has not been set.

WRITING DATA TO FILE

Write Using Data

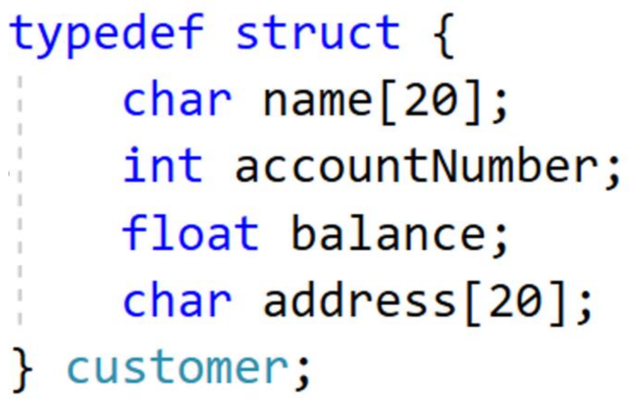
- Up until now, we have been writing strings to a file.

- Let’s now look at how we can write data organised in structures to a file.

Structures Recap

- We can use **structures** to store information in a more organised manner.

- We declare a structure using the keyword **struct**, as follows:



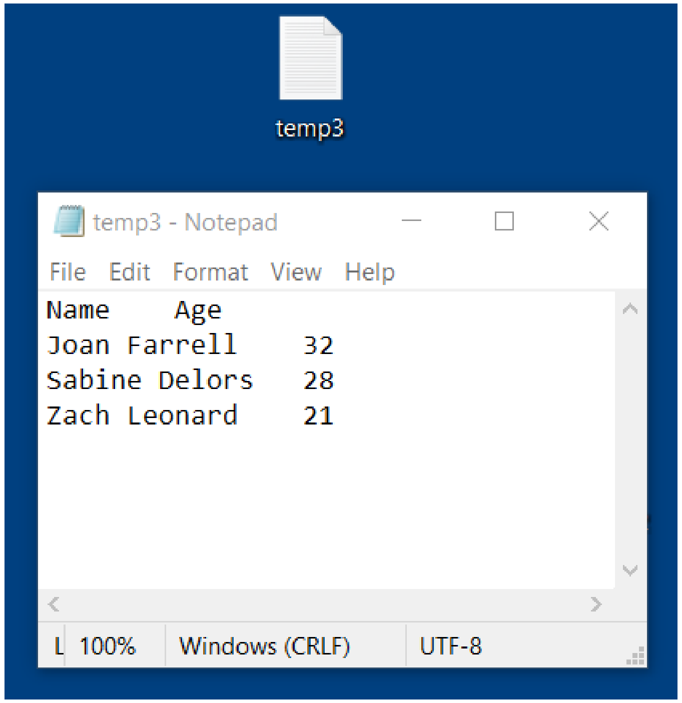
```
typedef struct {
    char name[20];
    int accountNumber;
    float balance;
    char address[20];
} customer;
```

Write Structured Data to File

```c
typedef struct{
    char
    name[100];
    int age;
} person;
```

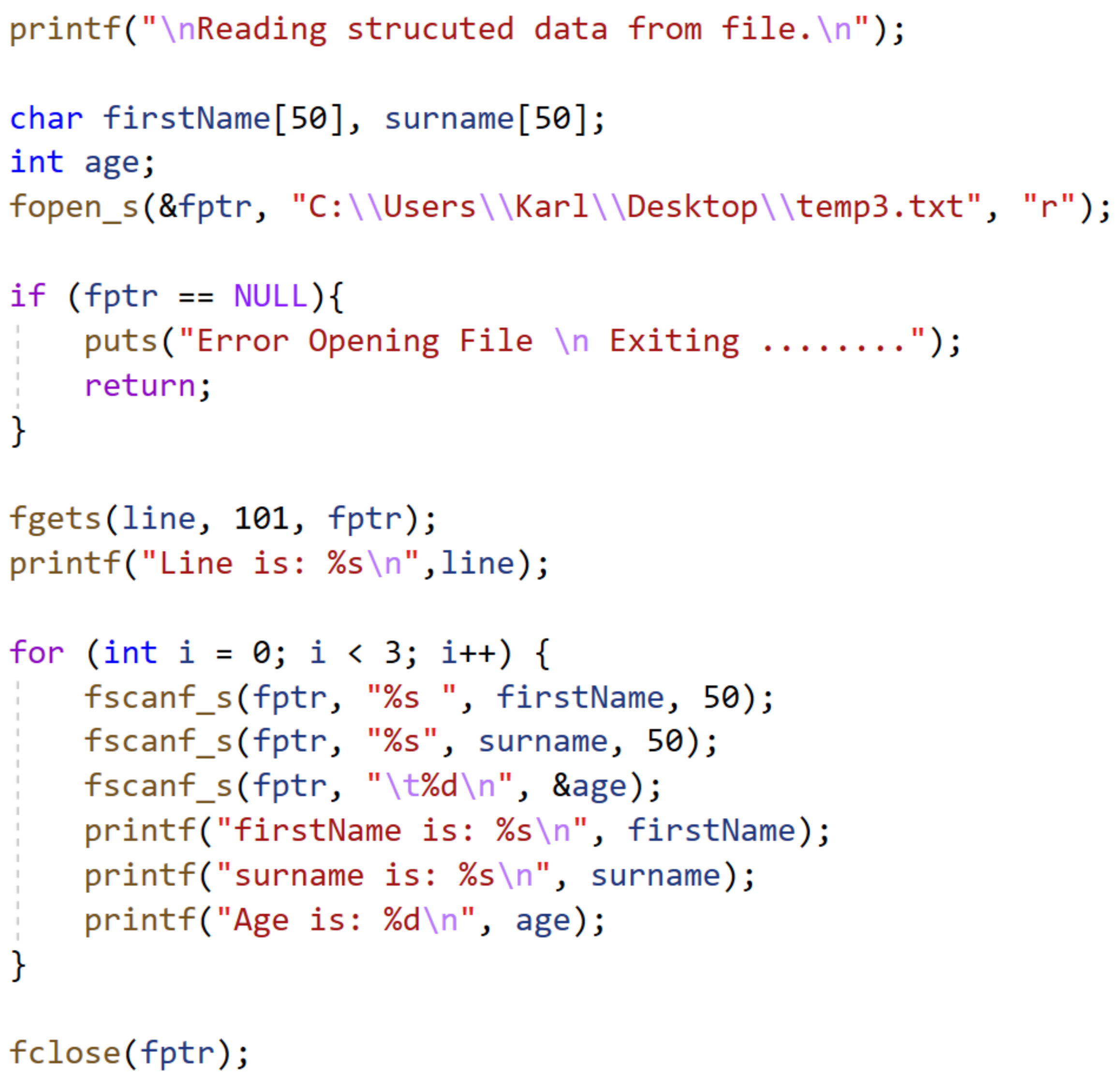
```c
printf("\nExample using structs.\n");

person p1 = { "Joan Farrell", 32 };
person p2 = { "Sabine Delors", 28 };
person p3 = { "Zach Leonard", 21 };

fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\temp3.txt", "w");

if (fptr == NULL){
    puts("Error Opening File \n Exiting ........");
    return;
}

fprintf(fptr, "Name\tAge\n");
fprintf(fptr, "%s\t%d\n", p1.name, p1.age);
fprintf(fptr, "%s\t%d\n", p2.name, p2.age);
fprintf(fptr, "%s\t%d\n", p3.name, p3.age);

fclose(fptr);
```

Structured Data File



```
Name     Age
Joan Farrell    32
Sabine Delors   28
Zach Leonard    21
```

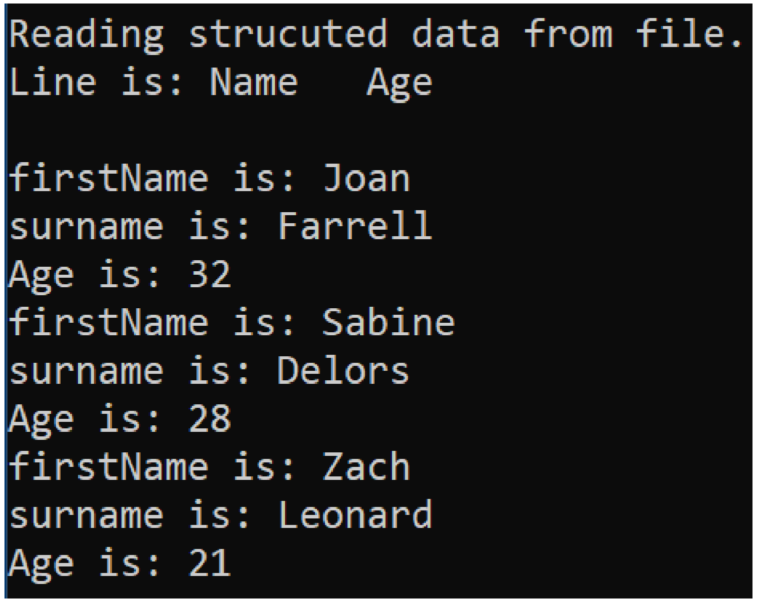
Read Data Back From File



```
printf("\nReading strucuted data from file.\n");

char firstName[50], surname[50];
int age;
fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\temp3.txt", "r");

if (fptr == NULL){
    puts("Error Opening File \n Exiting ........");
    return;
}

fgets(line, 101, fptr);
printf("Line is: %s\n",line);

for (int i = 0; i < 3; i++) {
    fscanf_s(fptr, "%s ", firstName, 50);
    fscanf_s(fptr, "%s", surname, 50);
    fscanf_s(fptr, "\t%d\n", &age);
    printf("firstName is: %s\n", firstName);
    printf("surname is: %s\n", surname);
    printf("Age is: %d\n", age);
}

fclose(fptr);
```

Read Data Back From File

```c
printf("\nReading strucuted data from file.\n");

char firstName[50], surname[50];
int age;
fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\temp3.txt", "r");

if (fptr == NULL){
    puts("Error Opening File \n Exiting ........");
    return;
}

fgets(line, 101, fptr);
printf("Line is: %s\n",line);

for (int i = 0; i < 3; i++) {
    fscanf_s(fptr, "%s ", firstName, 50);
    fscanf_s(fptr, "%s", surname, 50);
    fscanf_s(fptr, "\t%d\n", &age);
    printf("firstName is: %s\n", firstName);
    printf("surname is: %s\n", surname);
    printf("Age is: %d\n", age);
}

fclose(fptr);
```



```
Reading strucuted data from file.
Line is: Name    Age

firstName is: Joan
surname is: Farrell
Age is: 32
firstName is: Sabine
surname is: Delors
Age is: 28
firstName is: Zach
surname is: Leonard
Age is: 21
```

Reading Multiple Data Types

- Awkward to read multiple data types.
- Have to use fixes like we saw in the previous example:

```
fscanf_s(fptr, "%s ", firstName, 50);
fscanf_s(fptr, "%s", surname, 50);
fscanf_s(fptr, "\t%d\n", &age);
```

- We will see later on how we can tackle this using the `strtok` function to parse strings.

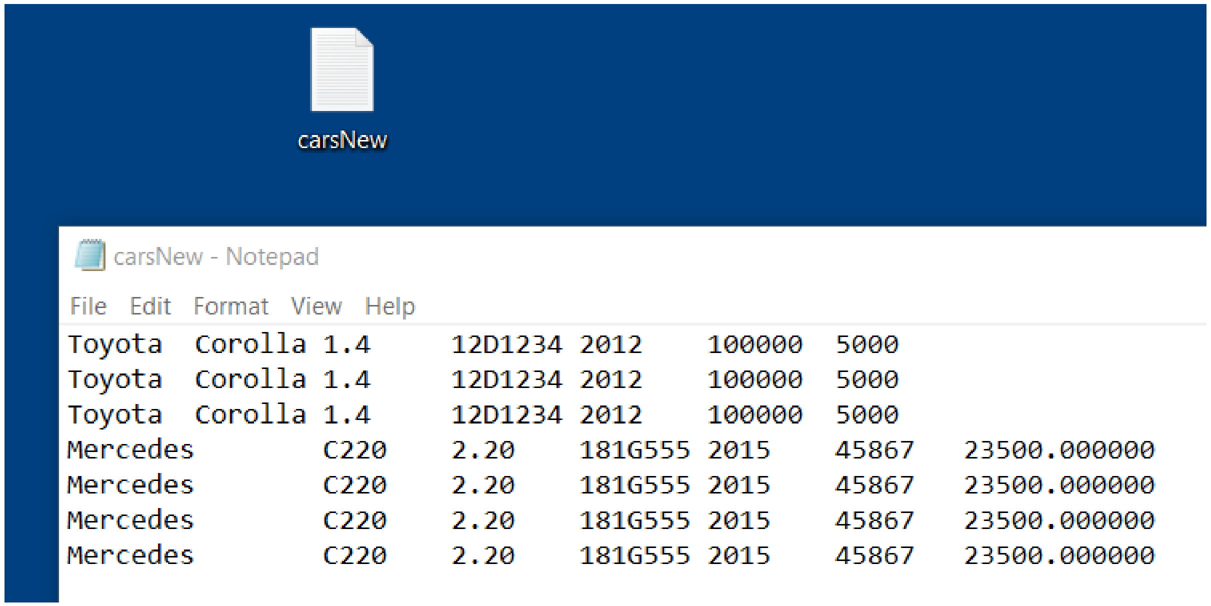
EXAMPLE C PROBLEM

Example C Problem

- You are writing software to keep track of vehicles for a mechanic.
- Write a struct to organise the information about each car in your program: make, model, price, etc.
- A file called ‘carsNew.txt’ exists and is used to store all vehicle information.
- Write a function to read in the data stored in this text file.
- Write another function to add a new vehicle to this text file.
- Test both functions.

Text File

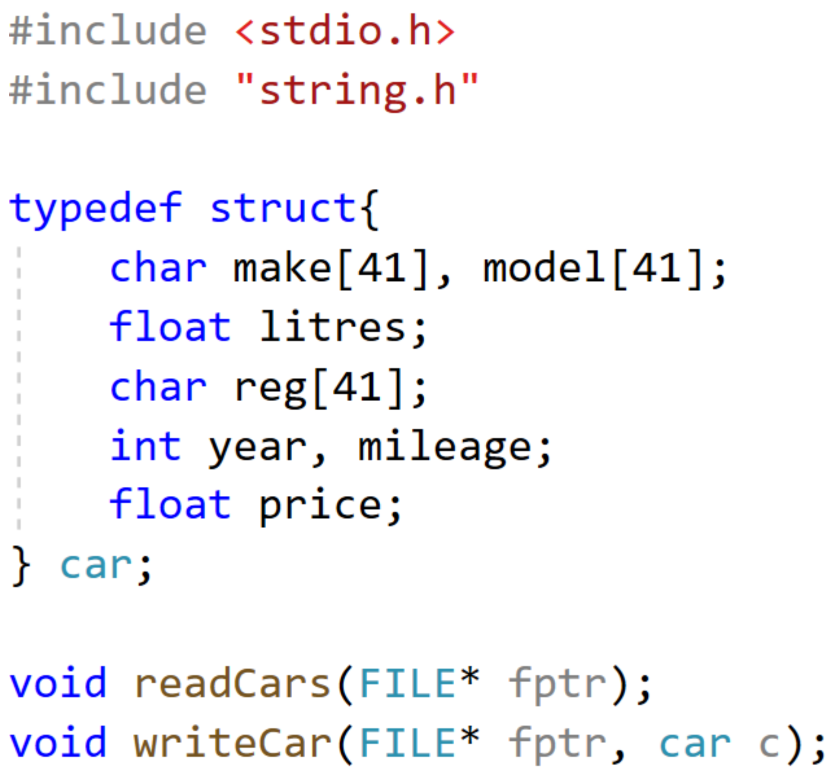
- Here is our text file stored on the desktop:



```
Toyota  Corolla 1.4     12D1234 2012    100000  5000
Toyota  Corolla 1.4     12D1234 2012    100000  5000
Toyota  Corolla 1.4     12D1234 2012    100000  5000
Mercedes        C220    2.20    181G555 2015    45867   23500.000000
Mercedes        C220    2.20    181G555 2015    45867   23500.000000
Mercedes        C220    2.20    181G555 2015    45867   23500.000000
Mercedes        C220    2.20    181G555 2015    45867   23500.000000
```

Structure

- Here is our structure and function prototypes:



```c
#include <stdio.h>
#include "string.h"

typedef struct{
    char make[41], model[41];
    float litres;
    char reg[41];
    int year, mileage;
    float price;
} car;

void readCars(FILE* fptr);
void writeCar(FILE* fptr, car c);
```

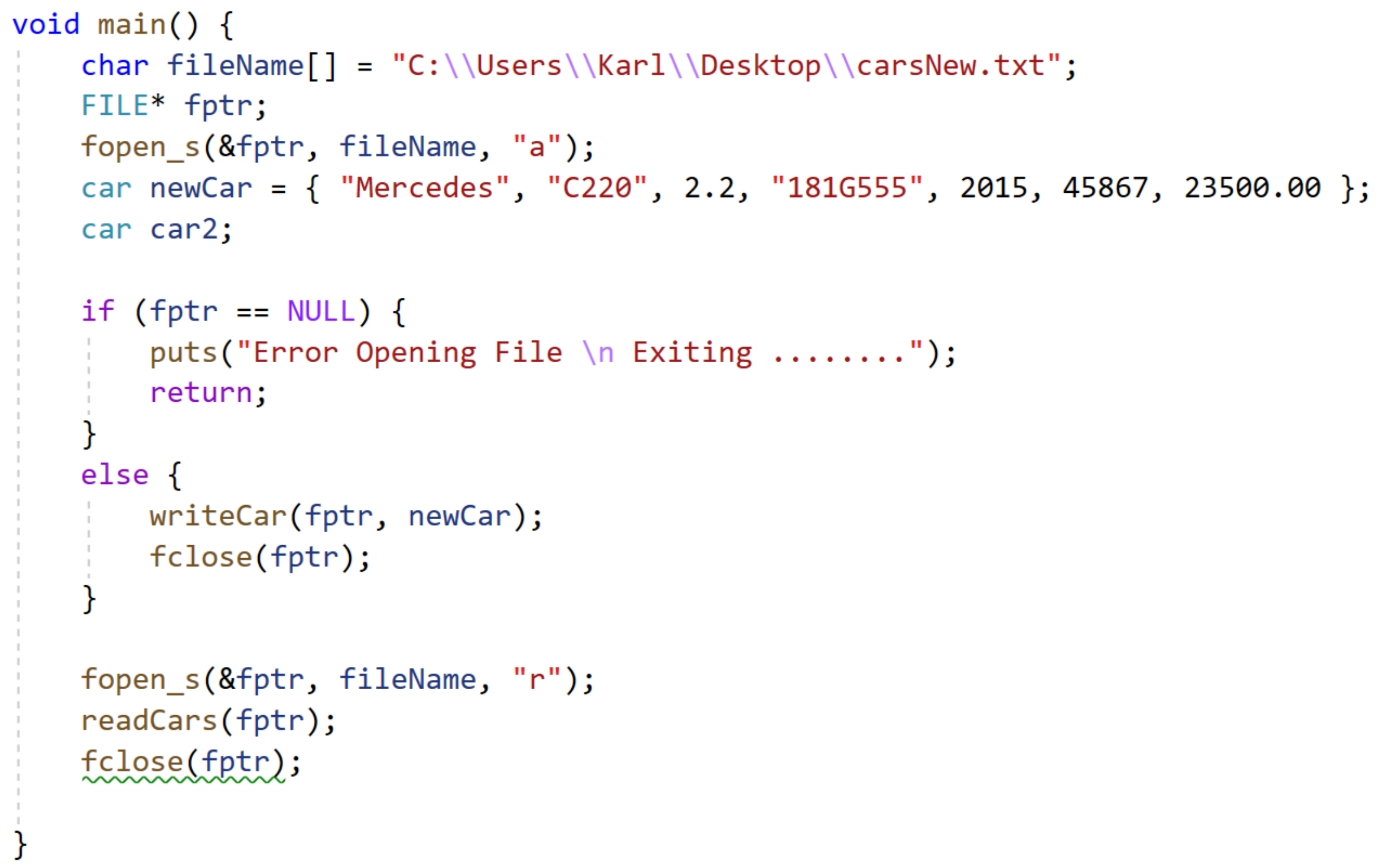
Functions

- Here are our functions:

```
void readCars(FILE* fptr) {
    car c;
    while (!feof(fptr)){
        fscanf_s(fptr, "%s\t", c.make, 41);
        fscanf_s(fptr, "%s\t", c.model, 41);
        fscanf_s(fptr, "%f\t", &c.litres);
        fscanf_s(fptr, "%s\t", c.reg, 41);
        fscanf_s(fptr, "%d\t", &c.year);
        fscanf_s(fptr, "%d\t", &c.mileage);
        fscanf_s(fptr, "%f\t", &c.price);
        printf("%s\t%s\t%lf\t%s\t%d\t%d\t%lf\n", c.make, c.model, c.litres, c.reg, c.year, c.mileage, c.price);
    }
}

void writeCar(FILE* fptr, car c){
    fprintf(fptr, "%s\t%s\t%.2lf\t%s\t%d\t%d\t%lf\n", c.make, c.model, c.litres, c.reg, c.year, c.mileage, c.price);
}
```

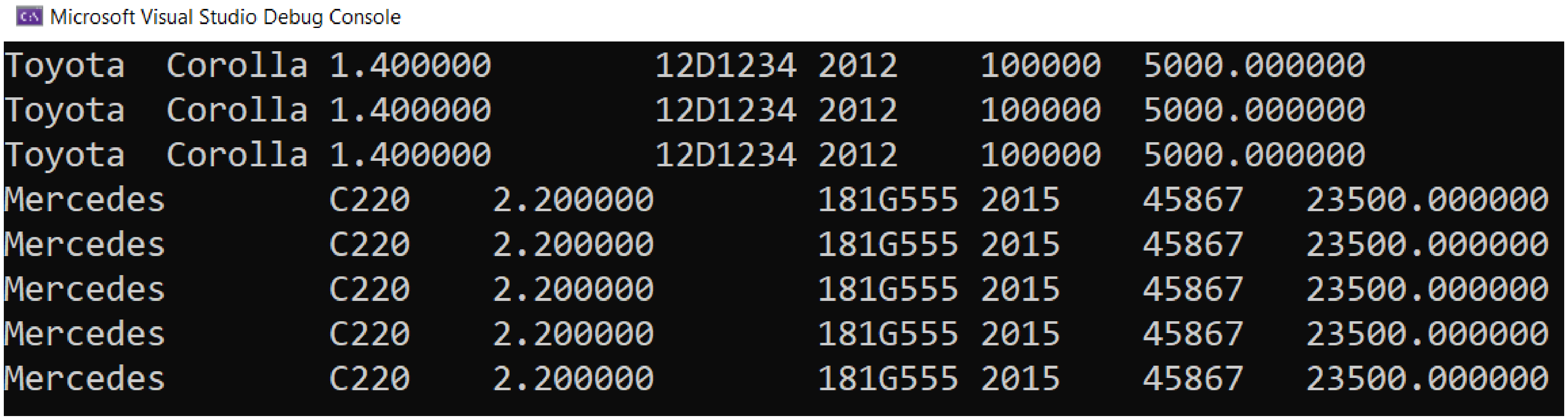
Main

- Here is main:



```
void main() {
    char fileName[] = "C:\\Users\\Karl\\Desktop\\carsNew.txt";
    FILE* fptr;
    fopen_s(&fptr, fileName, "a");
    car newCar = { "Mercedes", "C220", 2.2, "181G555", 2015, 45867, 23500.00 };
    car car2;

    if (fptr == NULL) {
        puts("Error Opening File \n Exiting ........");
        return;
    }
    else {
        writeCar(fptr, newCar);
        fclose(fptr);
    }

    fopen_s(&fptr, fileName, "r");
    readCars(fptr);
    fclose(fptr);

}
```

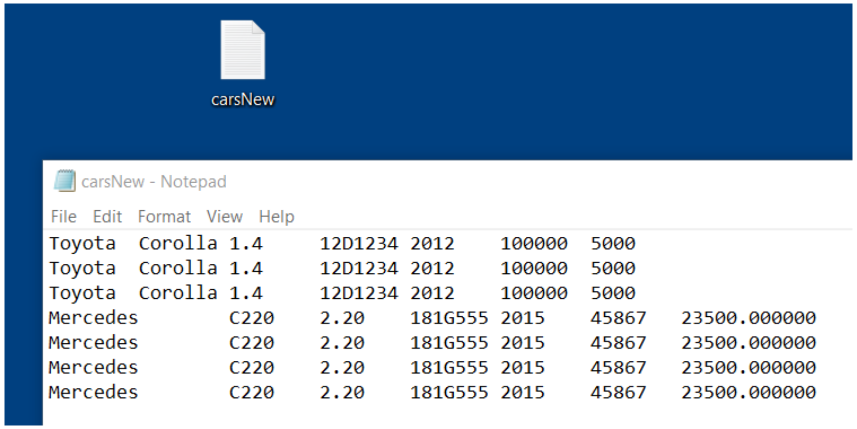
C Program Output



```
Toyota   Corolla 1.400000        12D1234 2012    100000  5000.000000
Toyota   Corolla 1.400000        12D1234 2012    100000  5000.000000
Toyota   Corolla 1.400000        12D1234 2012    100000  5000.000000
Mercedes         C220    2.200000        181G555 2015    45867   23500.000000
Mercedes         C220    2.200000        181G555 2015    45867   23500.000000
Mercedes         C220    2.200000        181G555 2015    45867   23500.000000
Mercedes         C220    2.200000        181G555 2015    45867   23500.000000
Mercedes         C220    2.200000        181G555 2015    45867   23500.000000
```

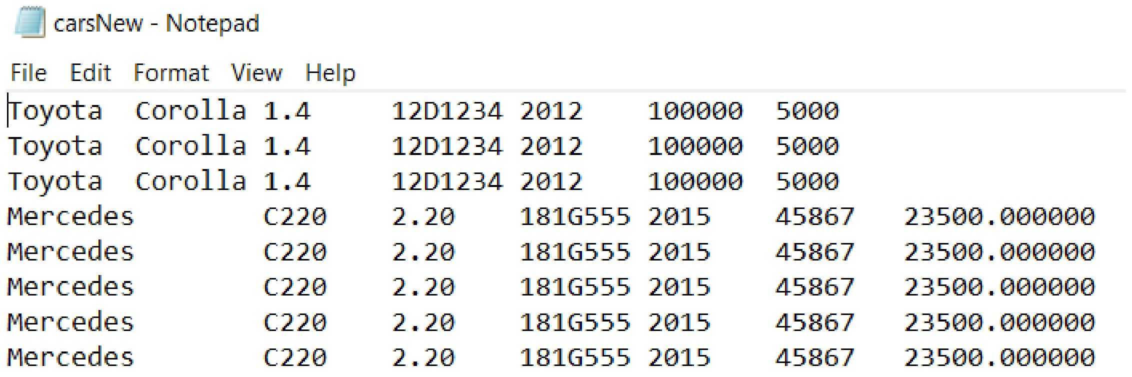
Updated Text File

- Here is our updated text file:



```
Toyota  Corolla 1.4     12D1234 2012    100000  5000
Toyota  Corolla 1.4     12D1234 2012    100000  5000
Toyota  Corolla 1.4     12D1234 2012    100000  5000
Mercedes        C220    2.20    181G555 2015    45867   23500.000000
Mercedes        C220    2.20    181G555 2015    45867   23500.000000
Mercedes        C220    2.20    181G555 2015    45867   23500.000000
Mercedes        C220    2.20    181G555 2015    45867   23500.000000
```

Extra Car



```
Toyota  Corolla 1.4     12D1234 2012    100000  5000
Toyota  Corolla 1.4     12D1234 2012    100000  5000
Toyota  Corolla 1.4     12D1234 2012    100000  5000
Mercedes        C220    2.20    181G555 2015    45867   23500.000000
Mercedes        C220    2.20    181G555 2015    45867   23500.000000
Mercedes        C220    2.20    181G555 2015    45867   23500.000000
Mercedes        C220    2.20    181G555 2015    45867   23500.000000
Mercedes        C220    2.20    181G555 2015    45867   23500.000000
```

C CODE

C Code

- Let’s finish today’s lecture by running some C programs in Visual Studio.

PROGRAMMING

CT103

Week 15

Sign in on Blackboard

- Please sign in on blackboard for CT103.
- I will leave the sign in option open for an hour after the lecture.

Lecture Content

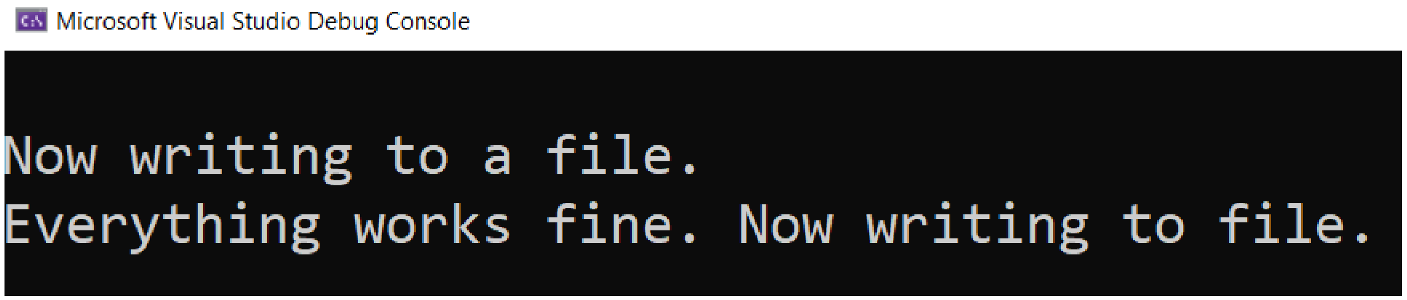
- Today's lecture (Week 15):
  - More file reading/writing
  - Updating files
  - Problems reading in files
  - Example C programme

FILE READING/WRITING

File Open Modes

- “r”: Opens a file for reading.
  - The file must exist.
- “w”: Creates an empty file for writing.
  - If a file with the same name already exists, its content is erased and the file is considered as a new empty file.
- “a”: Appends to a file.
  - Writing operations, append data at the end of the file. The file is created if it does not exist.
- “r+”: Opens a file to update both reading and writing.
  - The file must exist.
- “w+”: Creates an empty file for both reading and writing.
- “a+”: Opens a file for reading and appending.

Writing to a File in C Output

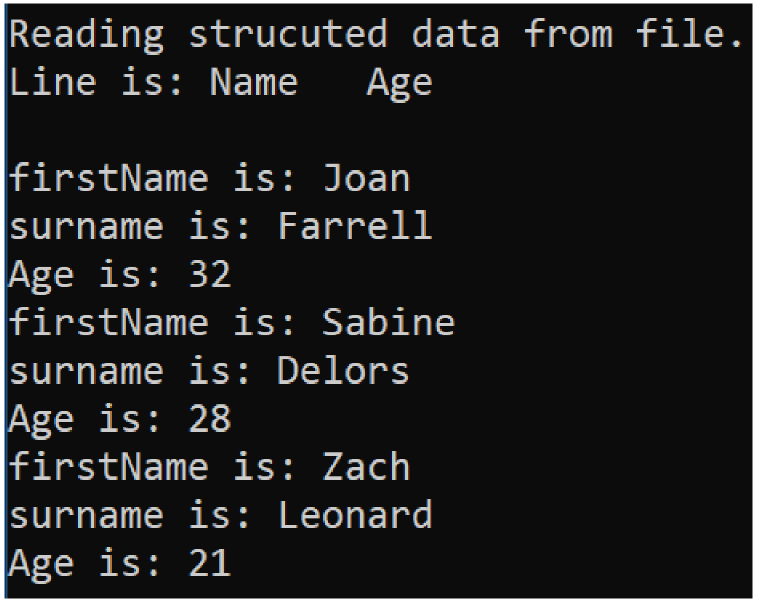

Microsoft Visual Studio Debug Console

```
Now writing to a file.
Everything works fine. Now writing to file.
```

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
  FILE *fptr;
  printf("\nNow writing to a file.\n");

  fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\temp2.txt", "w");

  if (fptr == NULL) {
    puts("Error Opening File \n Exiting ........");
    return;
  }
  else {
    printf("Everything works fine. Now writing to file.\n");
      for (int i = 0; i < 5;i++) {
        fprintf(fptr,"Line %d of text.\n",(i+1));
      }
  }
  fclose(fptr);
}
```

Read Data From File

```c
printf("\nReading strucuted data from file.\n");
FILE *fptr;
char firstName[50], surname[50];
int age;
fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\temp3.txt", "r");

if (fptr == NULL){
    puts("Error Opening File \n Exiting ........");
    return;
}

fgets(line, 101, fptr);
printf("Line is: %s\n",line);

for (int i = 0; i < 3; i++) {
    fscanf_s(fptr, "%s ", firstName, 50);
    fscanf_s(fptr, "%s", surname, 50);
    fscanf_s(fptr, "\t%d\n", &age);
    printf("firstName is: %s\n", firstName);
    printf("surname is: %s\n", surname);
    printf("Age is: %d\n", age);
}

fclose(fptr);
```



```
Reading strucuted data from file.
Line is: Name    Age

firstName is: Joan
surname is: Farrell
Age is: 32
firstName is: Sabine
surname is: Delors
Age is: 28
firstName is: Zach
surname is: Leonard
Age is: 21
```

UPDATING A FILE

Updating a File

- Sequential access is fairly straightforward.

- You can read or write from the beginning or end of the file

- Sequential Access is not usually used to update records *in place*.
  - Why?

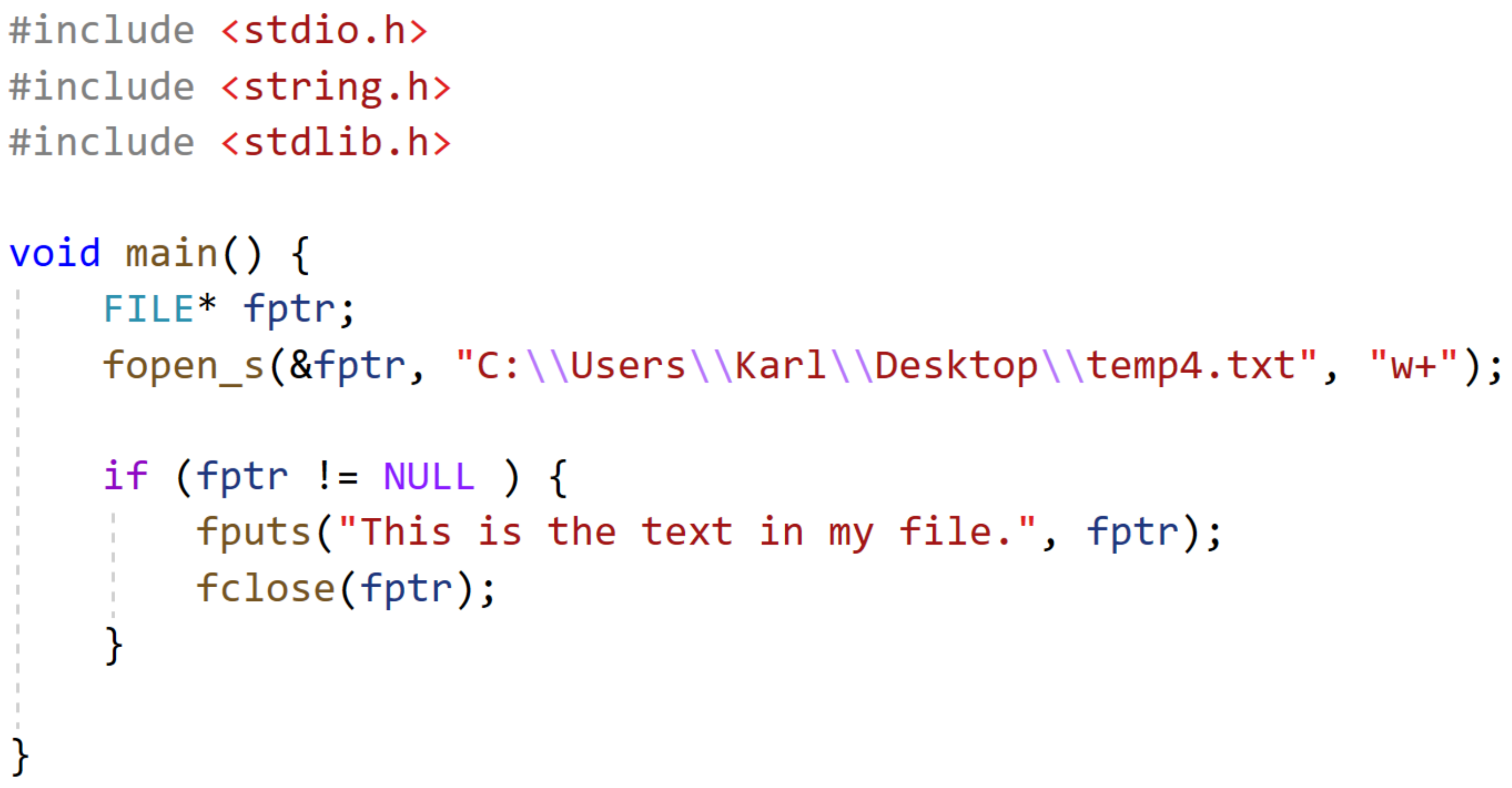
- Note: Updating records ‘**in place**’ refers to reading the records, processing them, and writing them back to their **original position** without destroying other records.

Updating a File

- Why is Sequential Access not usually used to update records *in place*?
- Imagine the following line in a text file:
  - 300 White 0.00
- How would we replace ‘White’ with ‘Washington’ so it looks like the following?
  - 300 Washington 0.00
- We would find it very difficult to avoid overwriting the 0.00 using the formatted input / output because the fields in each record will vary in size.
- Normally if we want to change records in a sequential access file we write out all records again to a new file.

Updating a File Example

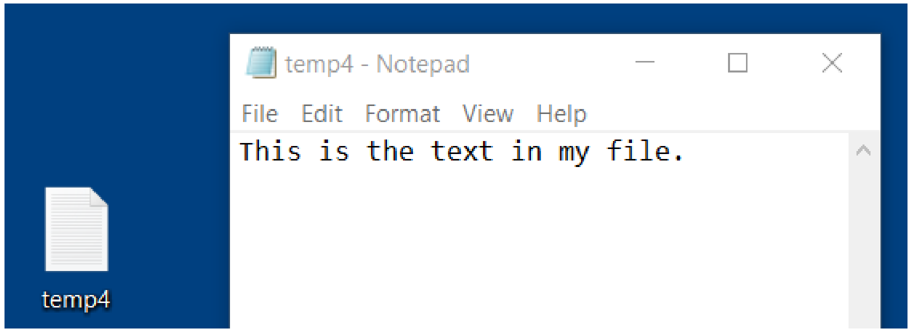
- Lets say you had the following code:



```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
    FILE* fptr;
    fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\temp4.txt", "w+");

    if (fptr != NULL ) {
        fputs("This is the text in my file.", fptr);
        fclose(fptr);
    }

}
```

Updating a File Example

- This code will run and produce a text file called ‘temp4.txt’.

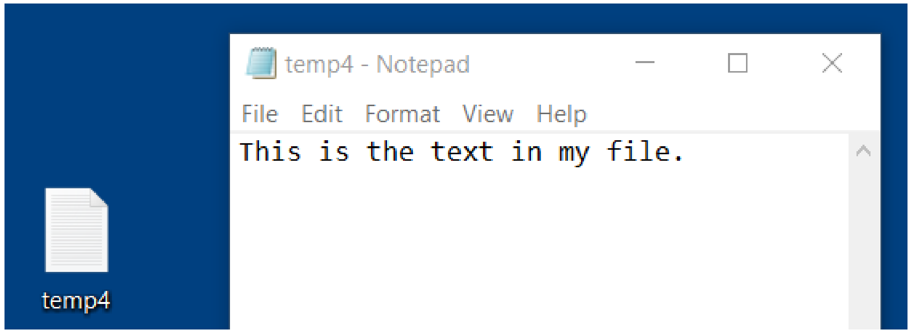


```
This is the text in my file.
```

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
    FILE* fptr;
    fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\temp4.txt", "w+");

    if (fptr != NULL ) {
        fputs("This is the text in my file.", fptr);
        fclose(fptr);
    }

}
```

Updating a File Example

- What if we wanted to change the text at some point in the middle of the text file?



```
This is the text in my file.
```

- We could use a function called fseek().

Updating a File Example

- Lets see how we use fseek() to update the file:

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
    FILE* fptr;
    fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\temp4.txt", "r+");

    if (fptr != NULL ) {
        fseek(fptr, 7, SEEK_SET);
        fputs(" new text in my file.", fptr);
        fclose(fptr);
    }
}
```

Now r+

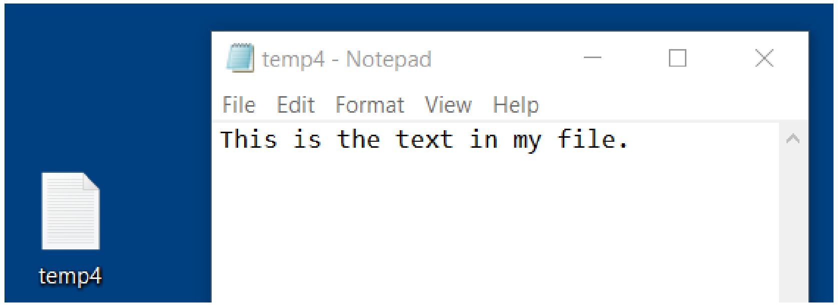
- Note: We will explain fseek() in the coming slides.

Updating a File Example

- This code will run and alter the text file‘temp4.txt’.

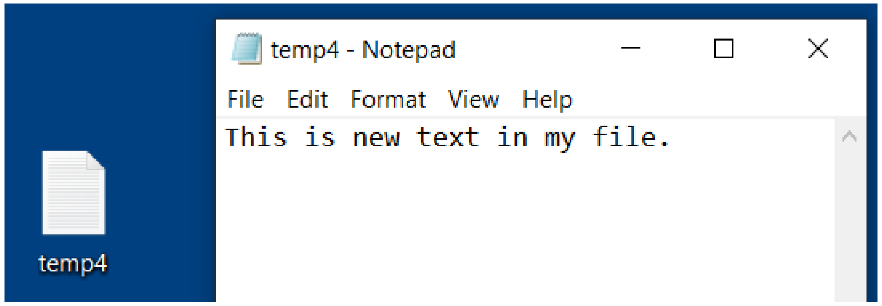
```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
    FILE* fptr;
    fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\temp4.txt", "r+");

    if (fptr != NULL ) {
        fseek(fptr, 7, SEEK_SET);
        fputs(" new text in my file.", fptr);
        fclose(fptr);
    }
}
```

Temp4.txt **before** running above code



```
This is the text in my file.
```

Temp4.txt **after** running above code

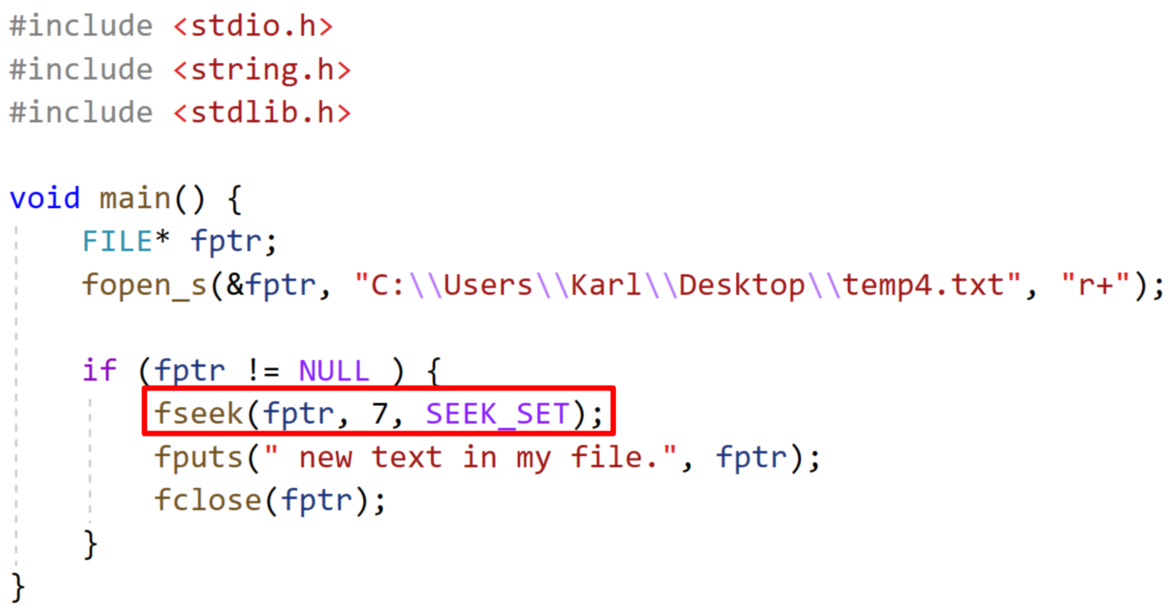


```
This is new text in my file.
```

FSEEK

fseek()

- We saw in the previous example how fseek() can be used to adjust a portion of the data in a file.



```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
    FILE* fptr;
    fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\temp4.txt", "r+");

    if (fptr != NULL ) {
        fseek(fptr, 7, SEEK_SET);
        fputs(" new text in my file.", fptr);
        fclose(fptr);
    }
}
```

- How does it work?

fseek()

- What is fseek()?
- We use *fseek()* to move around in a file
- *fseek()* moves the file pointer so that you can read and write at different places.

fseek()

- fseek() has the following function signature:

- fseek(**filePtr**, **offset**, **origin**)

  - **filePtr** is the file pointer.

  - **offset** (a long int) is the **number of bytes** to skip forwards or backwards in the file.
    - It can be positive or negative.

  - **origin** tells *fseek()* from where to start 'seeking'.
    - We need to select a value for origin.

Possible Origin Values

- You can use any of the three values for origin when calling fseek().

origin	Description
SEEK_SET	Beginning of file
SEEK_CUR	Current position in file
SEEK_END	End of file

fseek()

- Once you position the file pointer with fseek() you can use the file input and output functions to read from and write to the file.
- Using SEEK_END will act as append if you are writing to the file.
- If you position the file pointer over existing data, and then write new data, it will replace the existing data.

fseek() example 2

- What if we start change the value of the offset from 7?

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
    FILE* fptr;
    fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\temp4.txt", "r+");

    if (fptr != NULL ) {
        fseek(fptr, 3, SEEK_SET);
        fputs("MORE TEXT", fptr);
        fclose(fptr);
    }
}
```

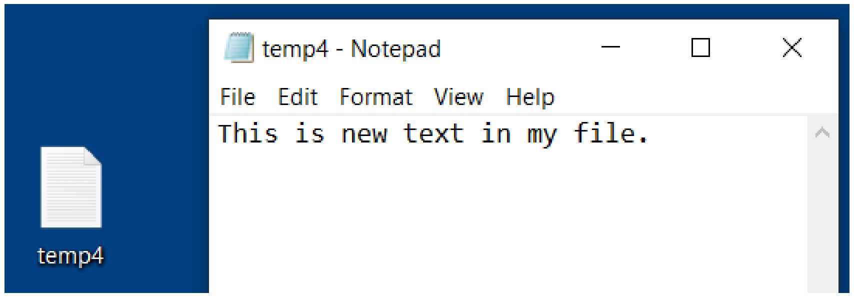
Now 3

fseek() example 2

- What if we start change the value of the offset from 7?

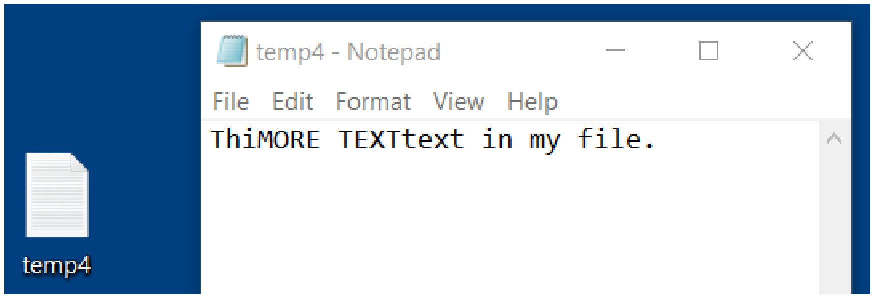
```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
    FILE* fptr;
    fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\temp4.txt", "r+");

    if (fptr != NULL ) {
        fseek(fptr, 3, SEEK_SET);
        fputs("MORE TEXT", fptr);
        fclose(fptr);
    }
}
```

Temp4.txt **before** running above code



```
This is new text in my file.
```

Temp4.txt **after** running above code



```
ThiMORE TEXTtext in my file.
```

fseek() example 3

- What if we want the origin to be at the end of the file?

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
    FILE* fptr;
    fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\temp4.txt", "r+");

    if (fptr != NULL ) {
        fseek(fptr, -5, SEEK_END);
        fputs("fantastic text file!", fptr);
        fclose(fptr);
    }
}
```

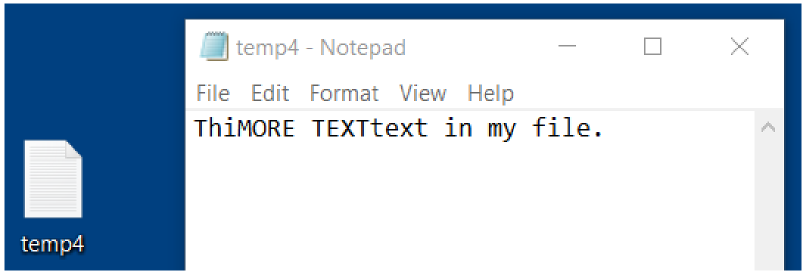
New offset and origin

fseek() example 3

- What if we want the origin to be at the end of the file?

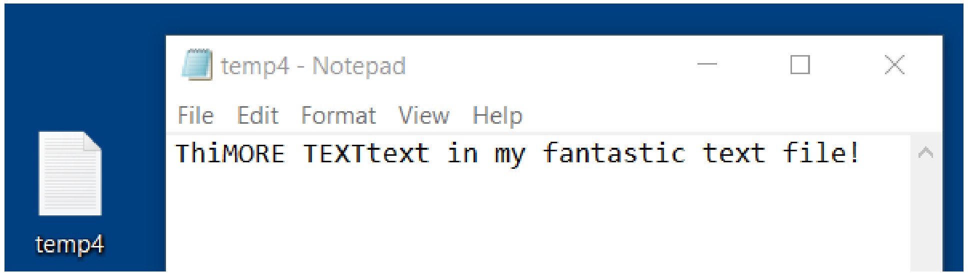
```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
    FILE* fptr;
    fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\temp4.txt", "r+");

    if (fptr != NULL ) {
        fseek(fptr, -5, SEEK_END);
        fputs("fantastic text file!", fptr);
        fclose(fptr);
    }
}
```

Temp4.txt **before** running above code



```
ThiMORE TEXTtext in my file.
```

Temp4.txt **after** running above code

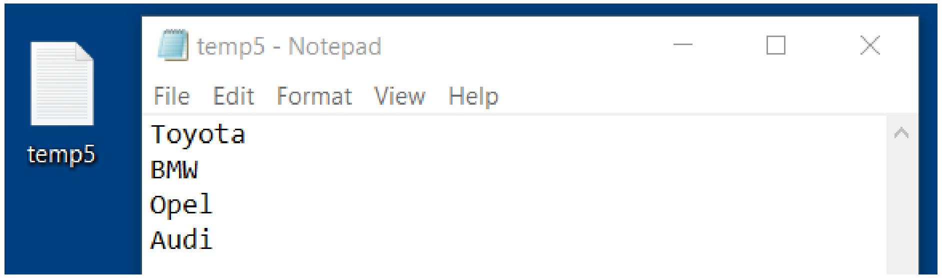


```
ThiMORE TEXTtext in my fantastic text file!
```

fseek() example 4

- Does fseek() work if my text file has multiple lines?

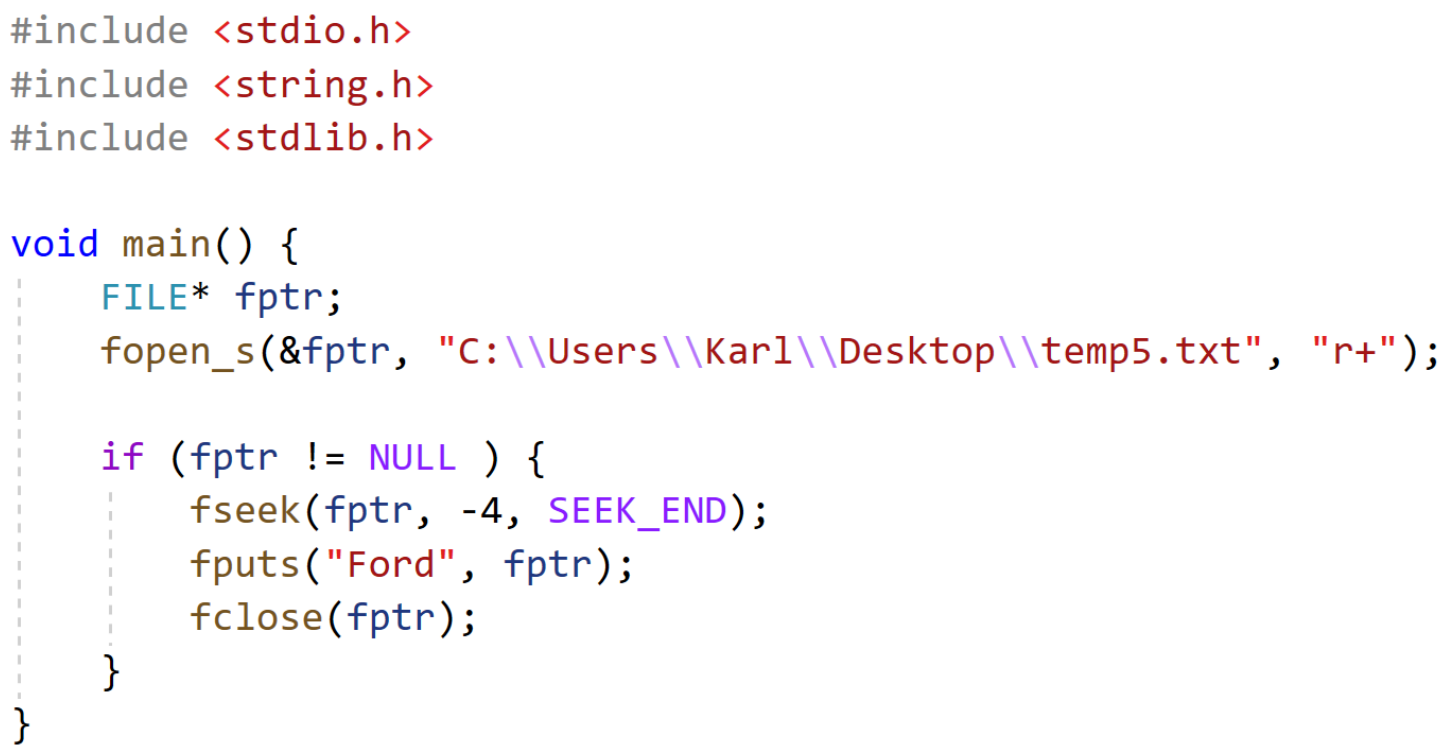
- Lets use the following file ‘temp5.txt’:



```
Toyota
BMW
Opel
Audi
```

fseek() example 4

- Does fseek() work if my text file has multiple lines?



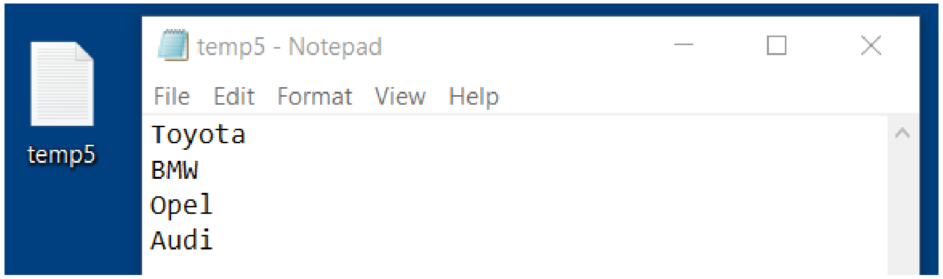
```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
    FILE* fptr;
    fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\temp5.txt", "r+");

    if (fptr != NULL ) {
        fseek(fptr, -4, SEEK_END);
        fputs("Ford", fptr);
        fclose(fptr);
    }
}
```

fseek() example 4

- Does fseek() work if my text file has multiple lines?

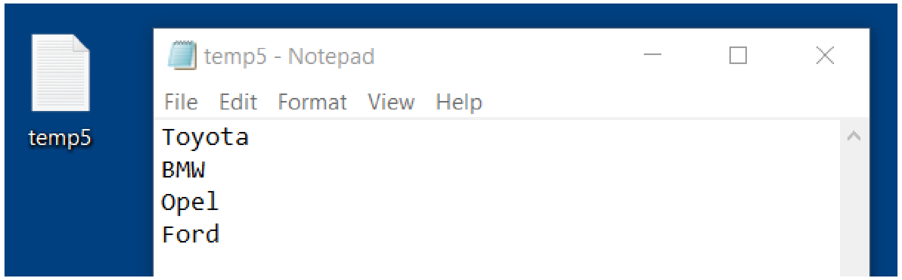
```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
    FILE* fptr;
    fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\temp5.txt", "r+");

    if (fptr != NULL ) {
        fseek(fptr, -4, SEEK_END);
        fputs("Ford", fptr);
        fclose(fptr);
    }
}
```

Temp5.txt **before** running above code



```
Toyota
BMW
Opel
Audi
```

Temp5.txt **after** running above code



```
Toyota
BMW
Opel
Ford
```

FTELL

ftell()

- ftell() is another useful function when reading/writing to file.
- ftell() allows us to get the current file position of the stream.
- This can be useful for getting the file size!

ftell()

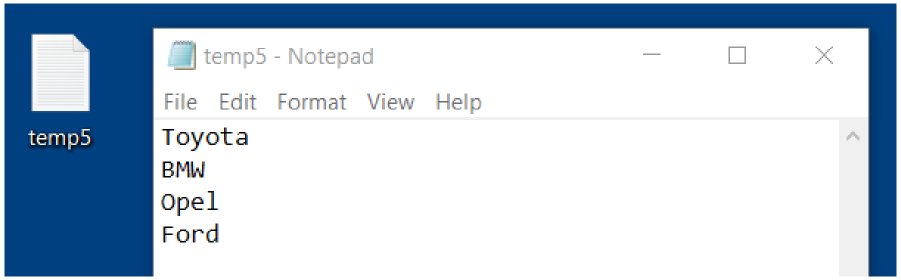
- ftell() function signature:

  long int ftell(FILE* f)

- The function returns a **long int**.
- The functions name is **ftell()**.
- The function reads in a **pointer** to the file.

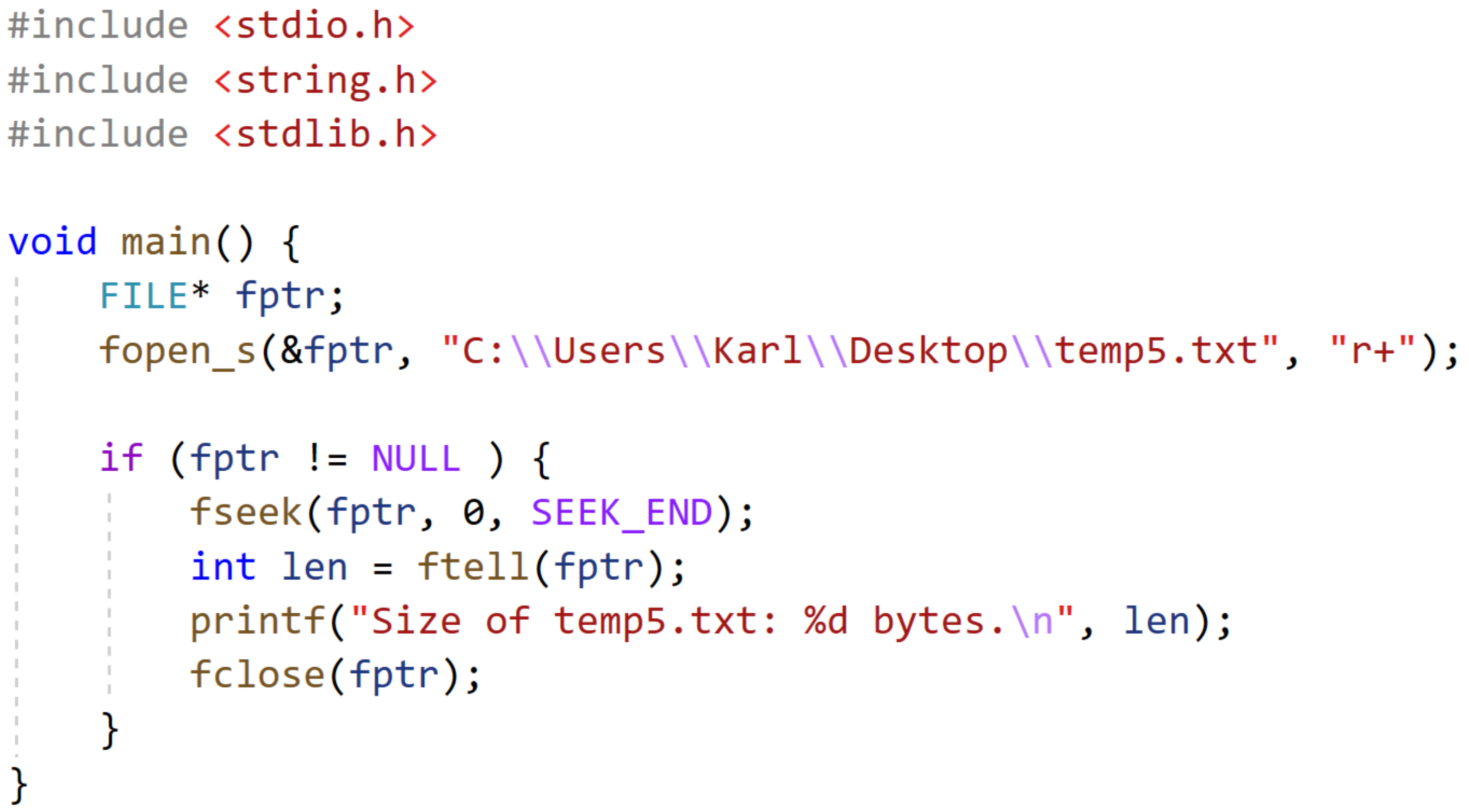
ftell() example

- Lets have a look at an example using ftell().
- Lets use the file temp5.txt from before:



```
Toyota
BMW
Opel
Ford
```

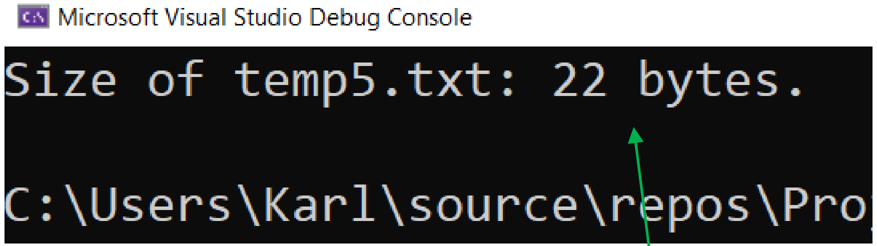
ftell() example



```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
    FILE* fptr;
    fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\temp5.txt", "r+");

    if (fptr != NULL ) {
        fseek(fptr, 0, SEEK_END);
        int len = ftell(fptr);
        printf("Size of temp5.txt: %d bytes.\n", len);
        fclose(fptr);
    }
}
```

ftell() example

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
    FILE* fptr;
    fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\temp5.txt", "r+");

    if (fptr != NULL ) {
        fseek(fptr, 0, SEEK_END);
        int len = ftell(fptr);
        printf("Size of temp5.txt: %d bytes.\n", len);
        fclose(fptr);
    }
}
```

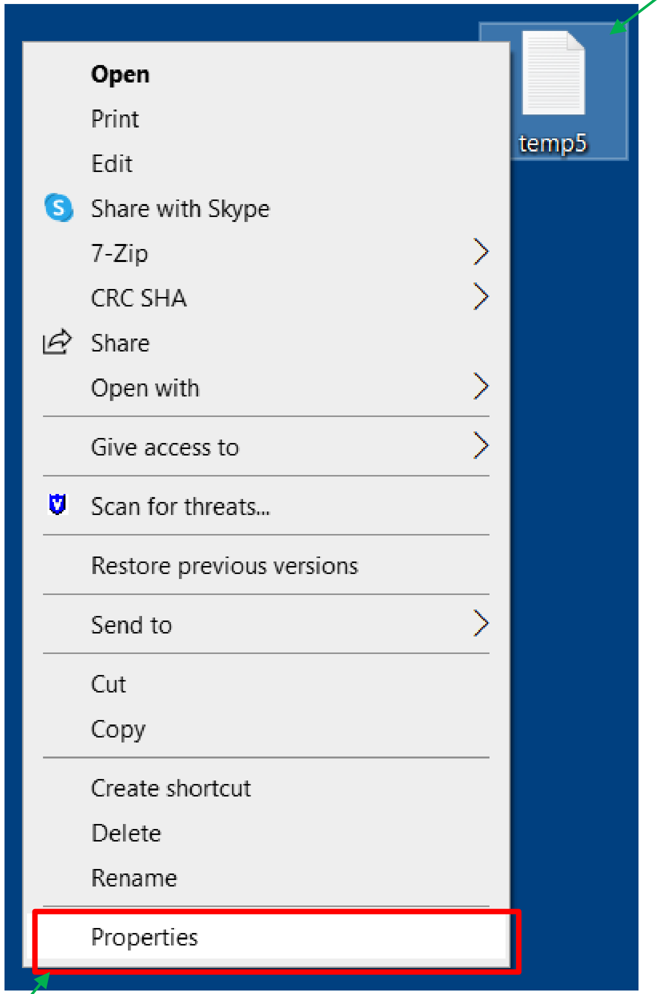


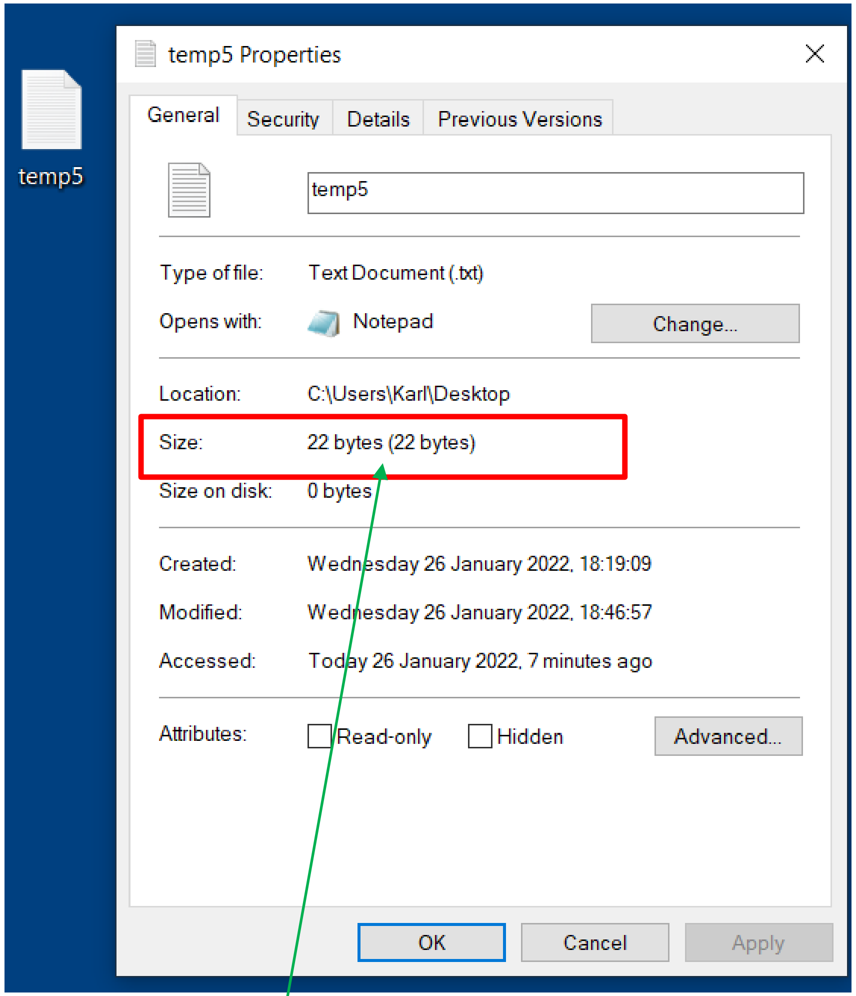
```
Size of temp5.txt: 22 bytes.

C:\Users\Karl\source\repos\Pro
```

Is this correct?

Temp5.txt Size

Right click





temp5

Type of file: Text Document (.txt)

Opens with: Notepad

Location: C:\Users\Karl\Desktop

Size: 22 bytes (22 bytes)

Size on disk: 0 bytes

Created: Wednesday 26 January 2022, 18:19:09

Modified: Wednesday 26 January 2022, 18:46:57

Accessed: Today 26 January 2022, 7 minutes ago

Attributes: ☐ Read-only ☐ Hidden

Go to properties

Size = 22 bytes (same as our code output)

END OF FIRST LECTURE

START OF SECOND LECTURE

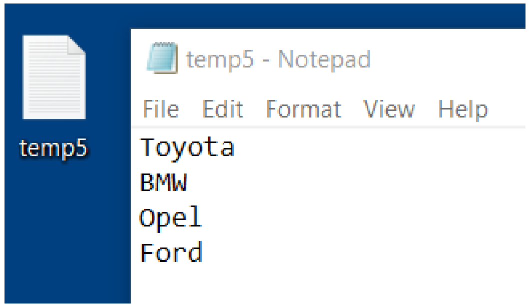
FILE READ ISSUES

Comments on fseek()

- Using fseek is pretty useful if you have a very simple file structure and you know exactly where in your file you want to go.
- Unfortunately, life is not that easy – for example, we are still left with the problem that in most cases the size of individual fields will vary in records (such as names, addresses, etc.)
- In those cases we are better off just reading in the file line by line (sequential access) until we find the data we want.
- We can then make any changes we want to the data and write out the entire file again.

Example problem with fseek()

- Recall temp5.txt.



```
Toyota
BMW
Opel
Ford
```

- What if we want to change line two from “BMW” to “Mitsubishi”?

Example problem with fseek()

- What if we want to change line two from “**BMW**” to “**Mitsubishi**”?

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
    FILE* fptr;
    fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\temp5.txt", "r+");

    if (fptr != NULL ) {
        fseek(fptr, 7, SEEK_SET);
        fputs("Mitsubishi", fptr);
        fclose(fptr);
    }
}
```

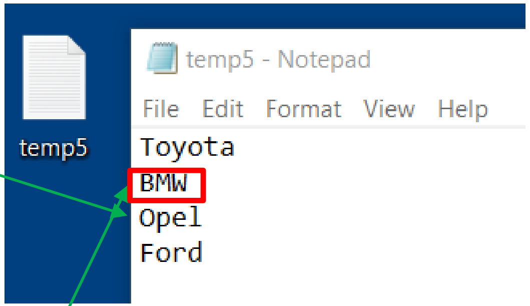
- Would this work?

Example problem with fseek()

- What if we want to change line two from "**BMW**" to "**Mitsubishi**"?

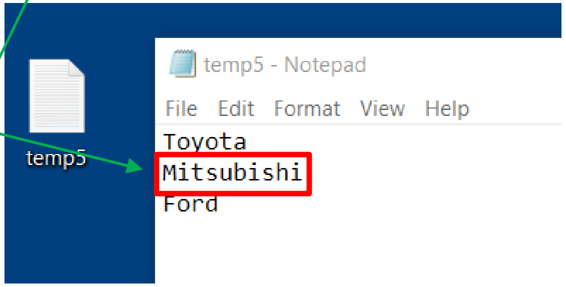
```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
    FILE* fptr;
    fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\temp5.txt", "r+");

    if (fptr != NULL ) {
        fseek(fptr, 7, SEEK_SET);
        fputs("Mitsubishi", fptr);
        fclose(fptr);
    }
}
```

We lost Opel



```
Toyota
BMW
Opel
Ford
```

We replaced BMW with Mitsubishi

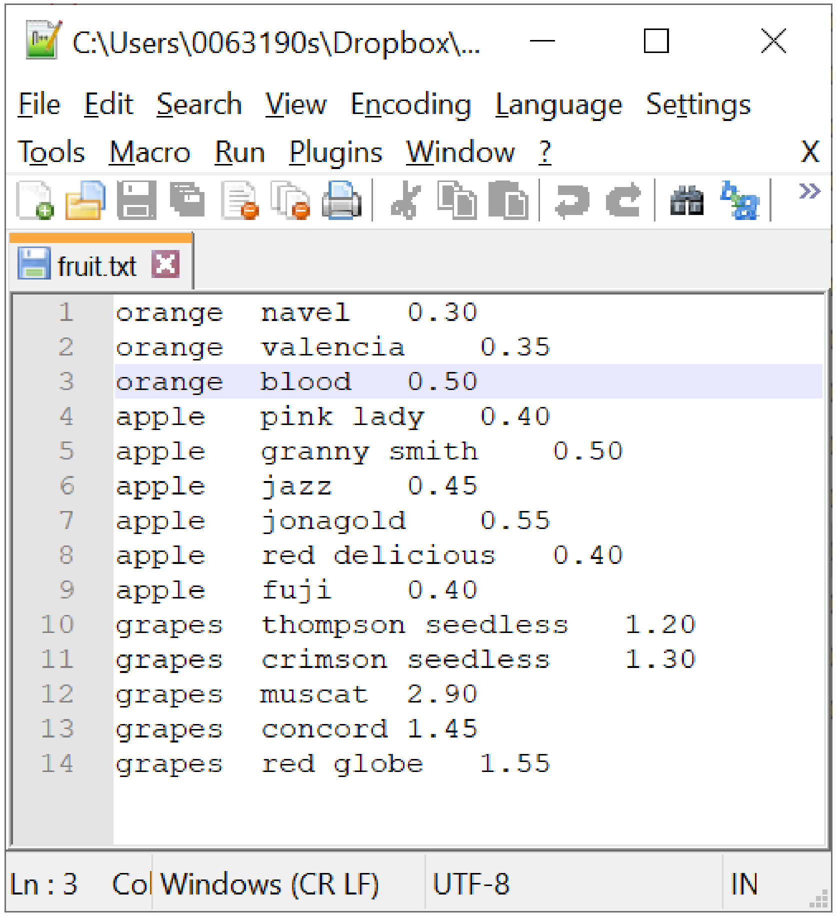


```
Toyota
Mitsubishi
Ford
```

- Did it work? Not how we want it to…

Problems Reading Data Files

- 1. Fields of unpredictable length / content.
- Multi-word fields make %s pretty useless.

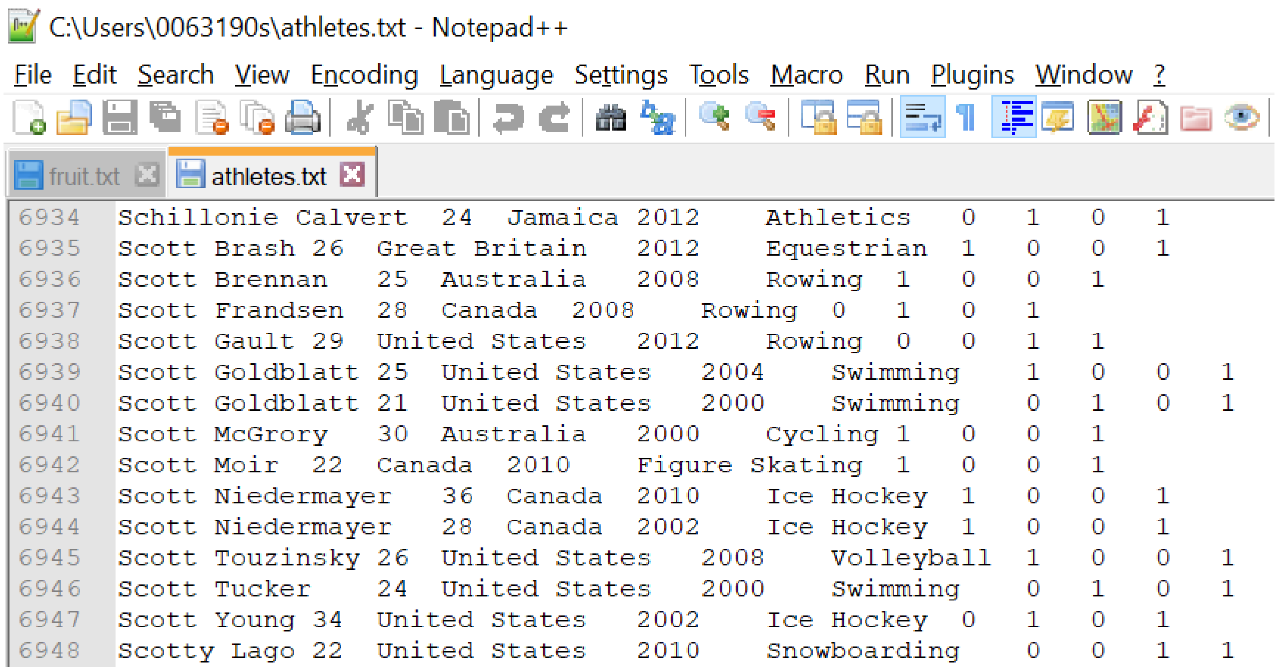

C:\Users\0063190s\Dropbox\...
File Edit Search View Encoding Language Settings
Tools Macro Run Plugins Window ?
fruit.txt

```
orange   navel    0.30
orange   valencia     0.35
orange   blood    0.50
apple    pink lady    0.40
apple    granny smith     0.50
apple    jazz     0.45
apple    jonagold     0.55
apple    red delicious    0.40
apple    fuji     0.40
grapes   thompson seedless    1.20
grapes   crimson seedless     1.30
grapes   muscat   2.90
grapes   concord 1.45
grapes   red globe    1.55
```

Ln : 3 Col Windows (CR LF) UTF-8 IN

Problems Reading Data Files

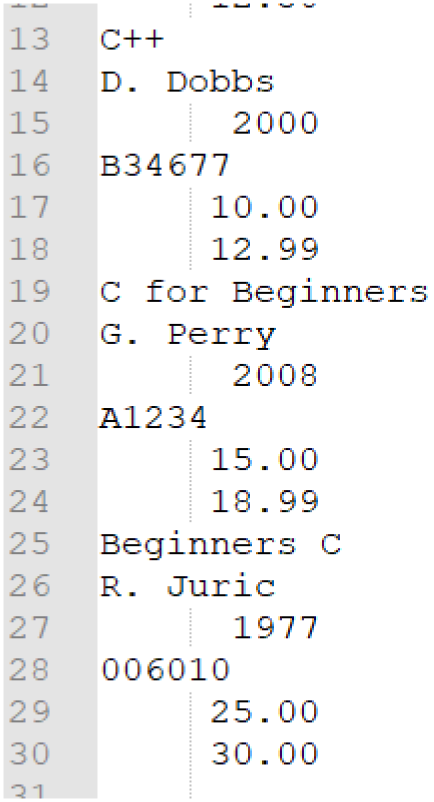
- 2. Choosing an appropriate data structure.
- 3. People with same name or different people?


C:\Users\0063190s\athletes.txt - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
fruit.txt
athletes.txt

```
6934  Schillonie Calvert  24  Jamaica 2012    Athletics   0   1   0   1
6935  Scott Brash 26  Great Britain   2012    Equestrian  1   0   0   1
6936  Scott Brennan   25  Australia   2008    Rowing  1   0   0   1
6937  Scott Frandsen  28  Canada  2008    Rowing  0   1   0   1
6938  Scott Gault 29  United States   2012    Rowing  0   0   1   1
6939  Scott Goldblatt 25  United States   2004    Swimming    1   0   0   1
6940  Scott Goldblatt 21  United States   2000    Swimming    0   1   0   1
6941  Scott McGrory   30  Australia   2000    Cycling 1   0   0   1
6942  Scott Moir  22  Canada  2010    Figure Skating  1   0   0   1
6943  Scott Niedermayer   36  Canada  2010    Ice Hockey  1   0   0   1
6944  Scott Niedermayer   28  Canada  2002    Ice Hockey  1   0   0   1
6945  Scott Touzinsky 26  United States   2008    Volleyball  1   0   0   1
6946  Scott Tucker    24  United States   2000    Swimming    0   1   0   1
6947  Scott Young 34  United States   2002    Ice Hockey  0   1   0   1
6948  Scotty Lago 22  United States   2010    Snowboarding    0   0   1   1
```

Problems Reading Data Files

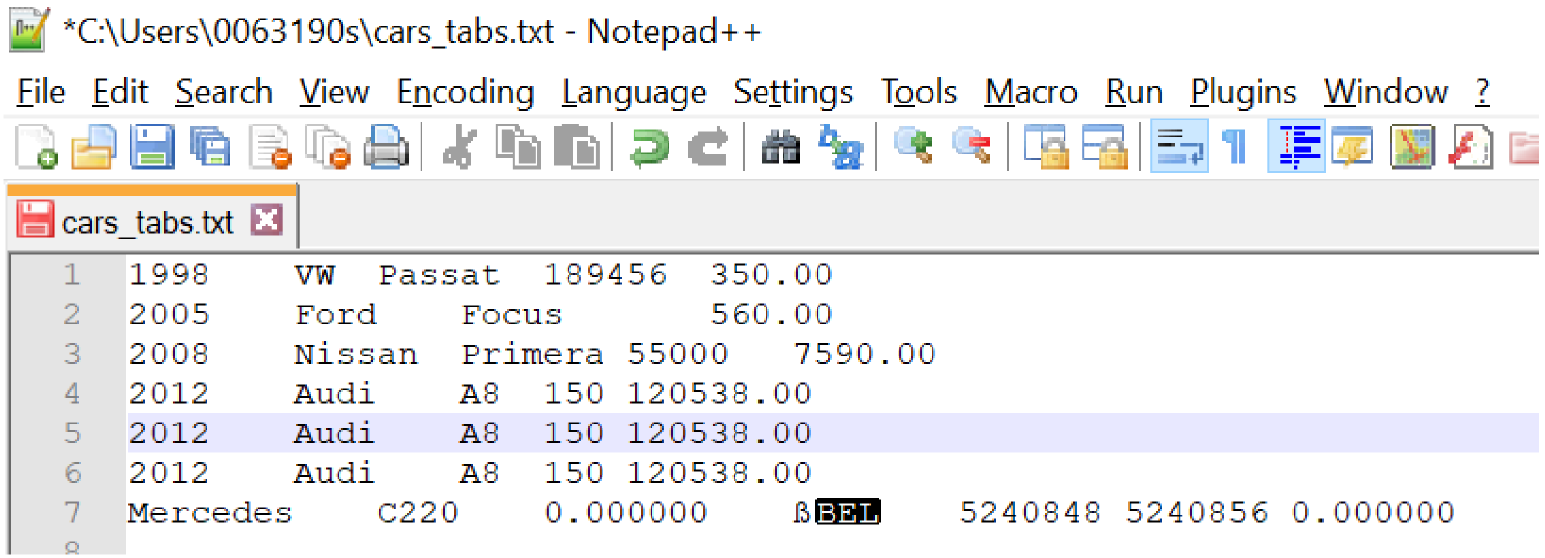
- 4. Unusual / unique / non-standard formats.



```
C++
D. Dobbs
          2000
B34677
         10.00
         12.99
C for Beginners
G. Perry
          2008
A1234
         15.00
         18.99
Beginners C
R. Juric
          1977
006010
         25.00
         30.00
```

Problems Reading Data Files

- 5. Missing / corrupt data or illegal characters

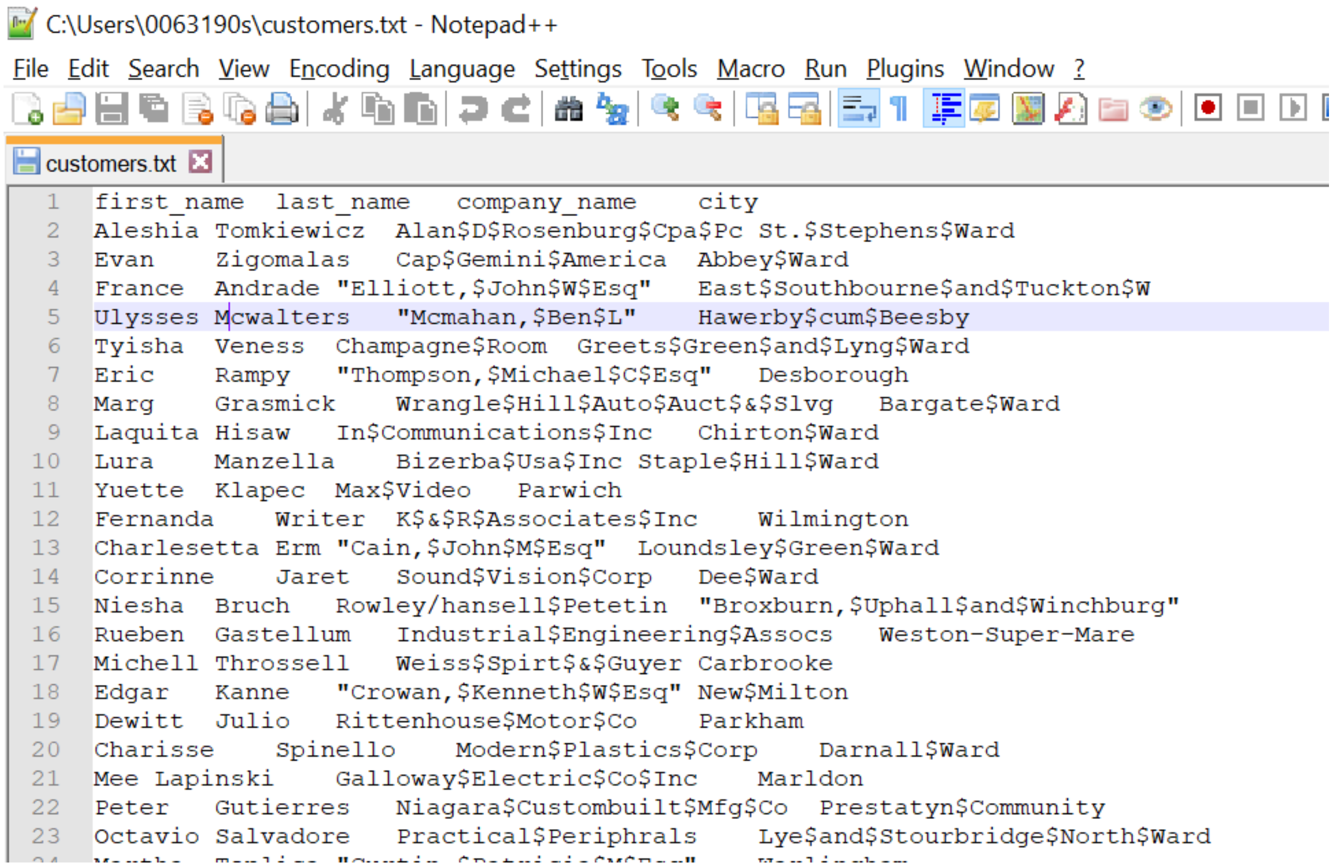

*C:\Users\0063190s\cars_tabs.txt - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
cars_tabs.txt

```
1998    VW  Passat  189456  350.00
2005    Ford    Focus       560.00
2008    Nissan  Primera 55000   7590.00
2012    Audi    A8  150 120538.00
2012    Audi    A8  150 120538.00
2012    Audi    A8  150 120538.00
Mercedes    C220    0.000000    ßBEL    5240848 5240856 0.000000
```

Problems Reading Data Files

- 6. What delimiters were used?

You have to figure out what delimiter(s) were used and if some characters (e.g. $) are being to replace another one (space)


C:\Users\0063190s\customers.txt - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
customers.txt

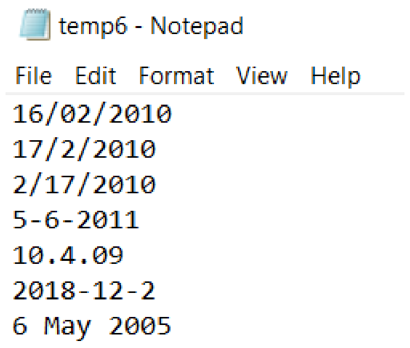
```
first_name  last_name   company_name    city
Aleshia Tomkiewicz  Alan$D$Rosenburg$Cpa$Pc St.$Stephens$Ward
Evan    Zigomalas   Cap$Gemini$America  Abbey$Ward
France  Andrade "Elliott,$John$W$Esq"   East$Southbourne$and$Tuckton$W
Ulysses Mcwalters   "Mcmahan,$Ben$L"    Hawerby$cum$Beesby
Tyisha  Veness  Champagne$Room  Greets$Green$and$Lyng$Ward
Eric    Rampy   "Thompson,$Michael$C$Esq"   Desborough
Marg    Grasmick    Wrangle$Hill$Auto$Auct$&$Slvg   Bargate$Ward
Laquita Hisaw   In$Communications$Inc   Chirton$Ward
Lura    Manzella    Bizerba$Usa$Inc Staple$Hill$Ward
Yuette  Klapec  Max$Video   Parwich
Fernanda    Writer  K$&$R$Associates$Inc    Wilmington
Charlesetta Erm "Cain,$John$M$Esq"  Loundsley$Green$Ward
Corrinne    Jaret   Sound$Vision$Corp   Dee$Ward
Niesha  Bruch   Rowley/hansell$Petetin  "Broxburn,$Uphall$and$Winchburg"
Rueben  Gastellum   Industrial$Engineering$Assocs   Weston-Super-Mare
Michell Throssell   Weiss$Spirt$&$Guyer Carbrooke
Edgar   Kanne   "Crowan,$Kenneth$W$Esq" New$Milton
Dewitt  Julio   Rittenhouse$Motor$Co    Parkham
Charisse    Spinello    Modern$Plastics$Corp    Darnall$Ward
Mee Lapinski    Galloway$Electric$Co$Inc    Marldon
Peter   Gutierres   Niagara$Custombuilt$Mfg$Co  Prestatyn$Community
Octavio Salvadore   Practical$Periphrals    Lye$and$Stourbridge$North$Ward
```

Delimiters

- Quick note on delimiters.
- What are delimiters?
- **Delimiters** refer to: one or more characters that outline the boundary between data.
- These can be: tab, whitespace, new line, comma, etc.

Problems Reading Data Files

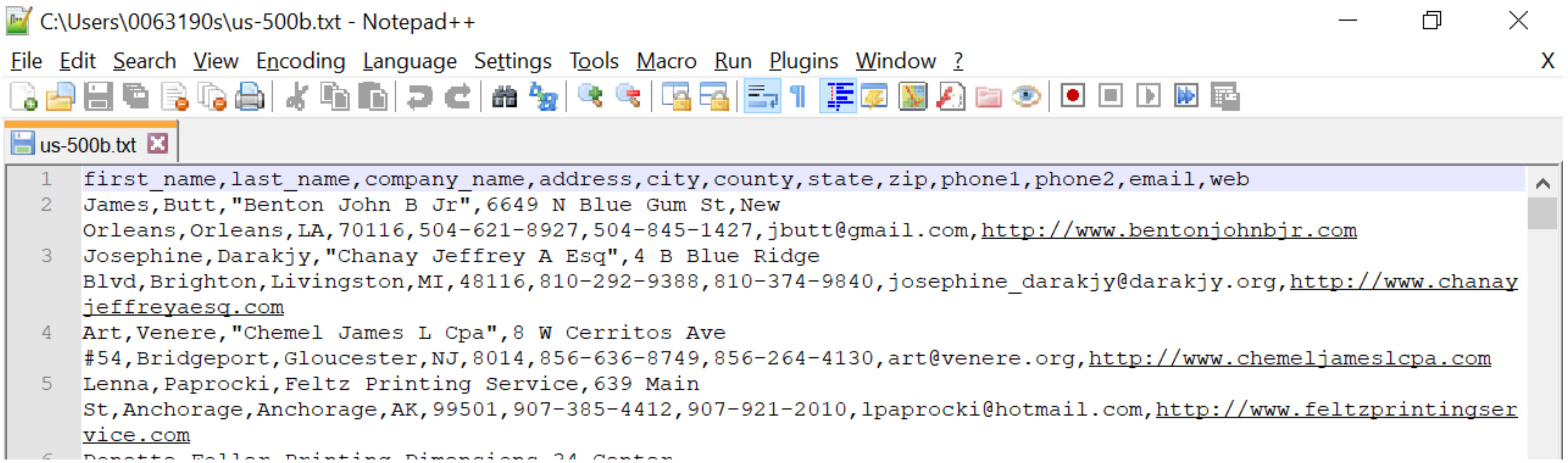
- 7. dates – what format was used?



```
16/02/2010
17/2/2010
2/17/2010
5-6-2011
10.4.09
2018-12-2
6 May 2005
```

Problems Reading Data Files

- 8. What are the formatting rules? (zip codes, phone numbers, urls, emails,…)

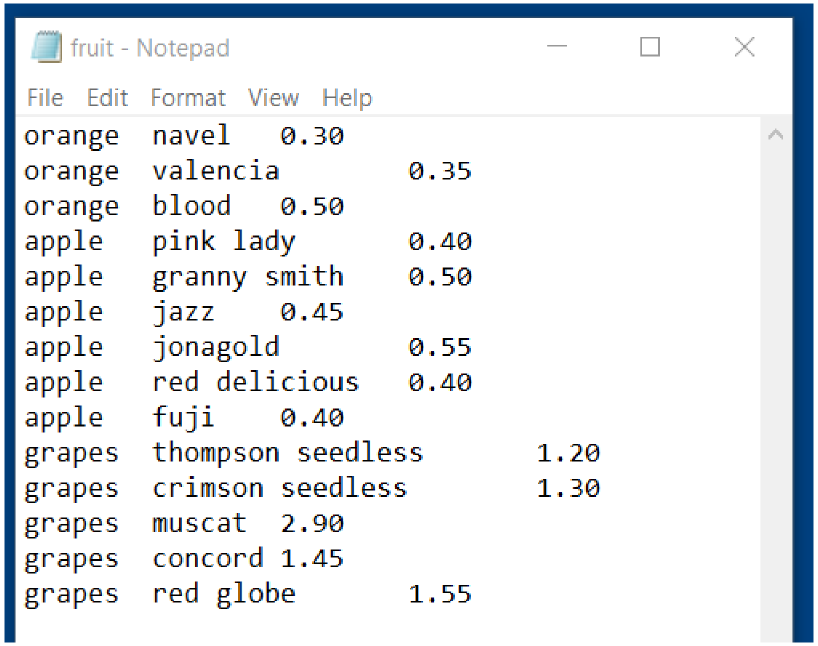

C:\Users\0063190s\us-500b.txt - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
us-500b.txt

```
first_name,last_name,company_name,address,city,county,state,zip,phone1,phone2,email,web
James,Butt,"Benton John B Jr",6649 N Blue Gum St,New
Orleans,Orleans,LA,70116,504-621-8927,504-845-1427,jbutt@gmail.com,http://www.bentonjohnbjr.com
Josephine,Darakjy,"Chanay Jeffrey A Esq",4 B Blue Ridge
Blvd,Brighton,Livingston,MI,48116,810-292-9388,810-374-9840,josephine_darakjy@darakjy.org,http://www.chanay
jeffreyaesq.com
Art,Venere,"Chemel James L Cpa",8 W Cerritos Ave
#54,Bridgeport,Gloucester,NJ,8014,856-636-8749,856-264-4130,art@venere.org,http://www.chemeljameslcpa.com
Lenna,Paprocki,Feltz Printing Service,639 Main
St,Anchorage,Anchorage,AK,99501,907-385-4412,907-921-2010,lpaprocki@hotmail.com,http://www.feltzprintingser
vice.com
```

FRUIT EXAMPLE DATASET

Fruit Example

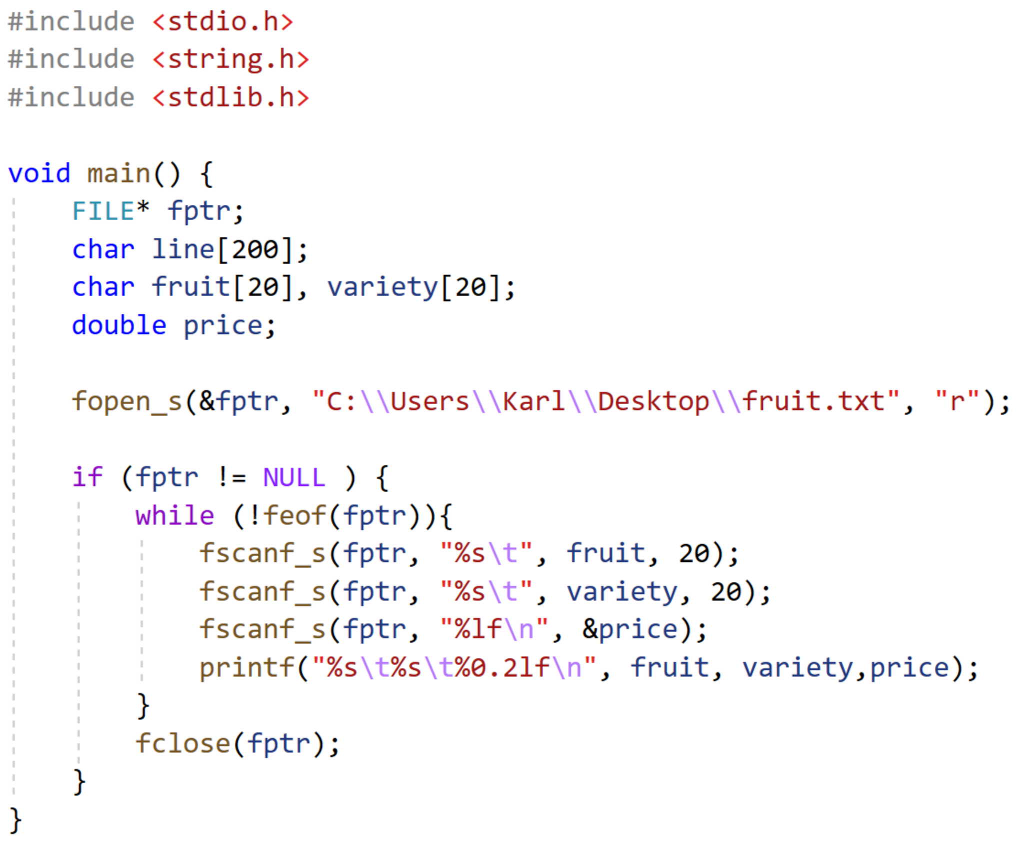
- Lets look at the ‘fruit.txt’ file.
- How can we scan in data when some fields have multiple words and some don’t?



```
orange	navel	0.30
orange	valencia	0.35
orange	blood	0.50
apple	pink lady	0.40
apple	granny smith	0.50
apple	jazz	0.45
apple	jonagold	0.55
apple	red delicious	0.40
apple	fuji	0.40
grapes	thompson seedless	1.20
grapes	crimson seedless	1.30
grapes	muscat	2.90
grapes	concord	1.45
grapes	red globe	1.55
```

Fruit Example

- Will this code work?

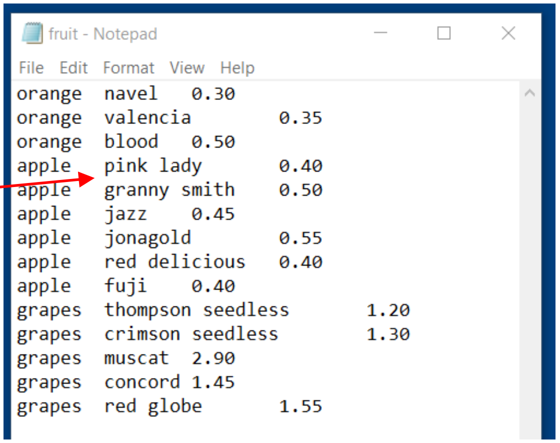


```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
    FILE* fptr;
    char line[200];
    char fruit[20], variety[20];
    double price;

    fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\fruit.txt", "r");

    if (fptr != NULL ) {
        while (!feof(fptr)){
            fscanf_s(fptr, "%s\t", fruit, 20);
            fscanf_s(fptr, "%s\t", variety, 20);
            fscanf_s(fptr, "%lf\n", &price);
            printf("%s\t%s\t%0.2lf\n", fruit, variety,price);
        }
        fclose(fptr);
    }
}
```

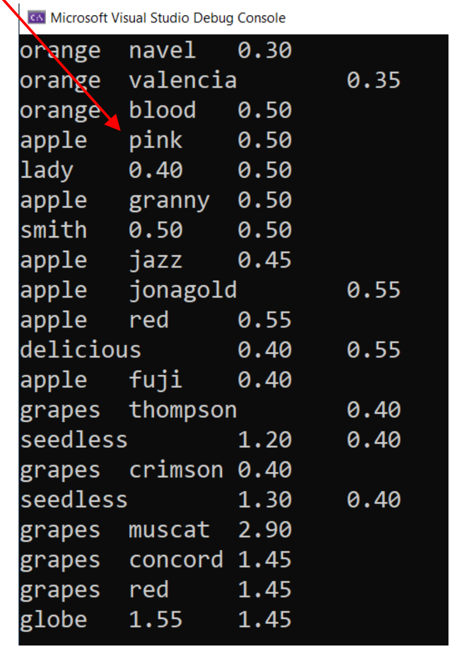
Fruit Example

- Will this code work? Sort of…

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
    FILE* fptr;
    char line[200];
    char fruit[20], variety[20];
    double price;

    fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\fruit.txt", "r");

    if (fptr != NULL ) {
        while (!feof(fptr)){
            fscanf_s(fptr, "%s\t", fruit, 20);
            fscanf_s(fptr, "%s\t", variety, 20);
            fscanf_s(fptr, "%lf\n", &price);
            printf("%s\t%s\t%0.2lf\n", fruit, variety,price);
        }
        fclose(fptr);
    }
}
```



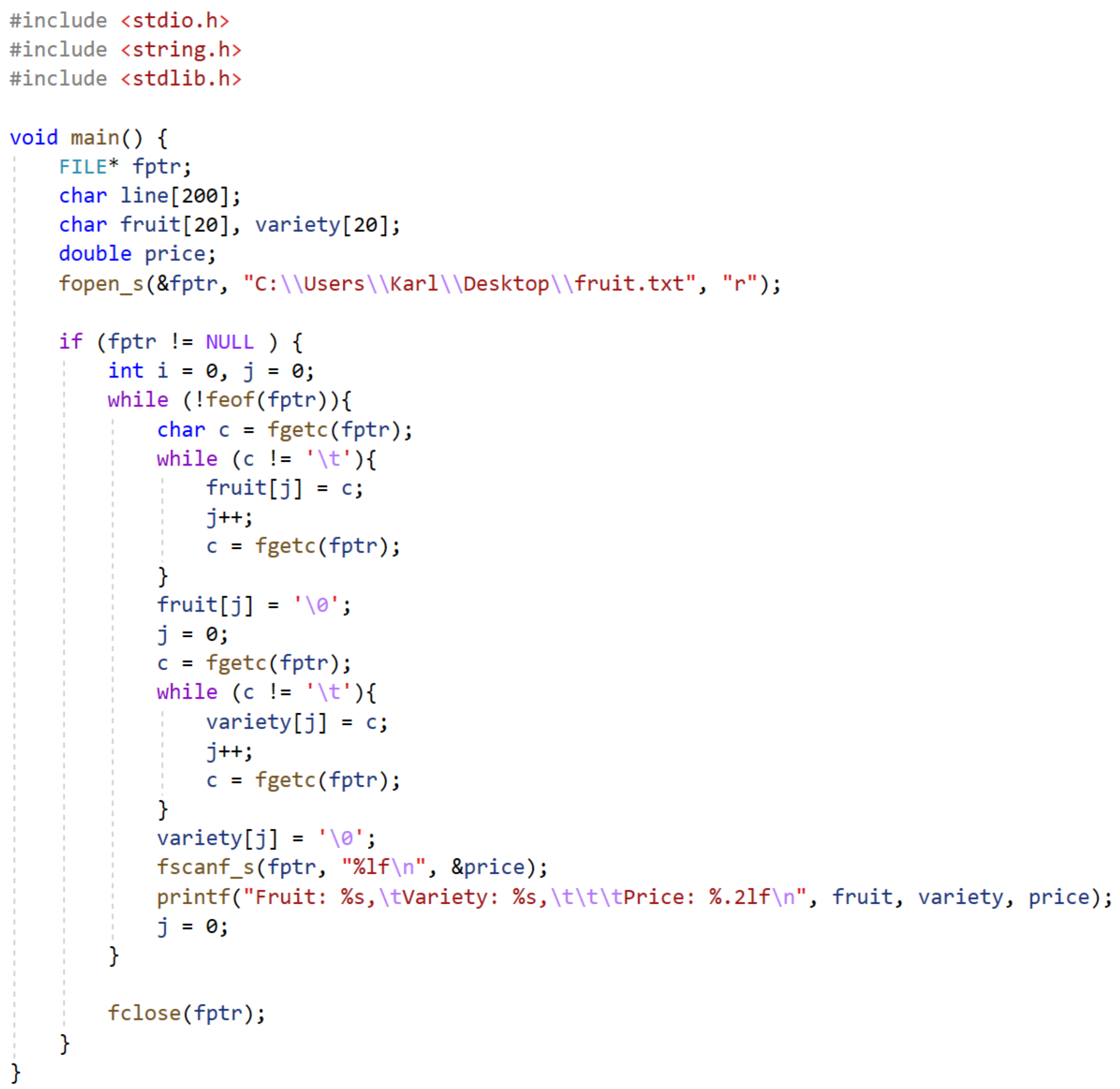
```
orange  navel   0.30
orange  valencia        0.35
orange  blood   0.50
apple   pink lady       0.40
apple   granny smith    0.50
apple   jazz    0.45
apple   jonagold        0.55
apple   red delicious   0.40
apple   fuji    0.40
grapes  thompson seedless       1.20
grapes  crimson seedless        1.30
grapes  muscat  2.90
grapes  concord 1.45
grapes  red globe       1.55
```



```
orange  navel   0.30
orange  valencia        0.35
orange  blood   0.50
apple   pink    0.50
lady    0.40    0.50
apple   granny  0.50
smith   0.50    0.50
apple   jazz    0.45
apple   jonagold        0.55
apple   red     0.55
delicious       0.40    0.55
apple   fuji    0.40
grapes  thompson        0.40
seedless        1.20    0.40
grapes  crimson 0.40
seedless        1.30    0.40
grapes  muscat  2.90
grapes  concord 1.45
grapes  red     1.45
globe   1.55    1.45
```

Fruit Example 2

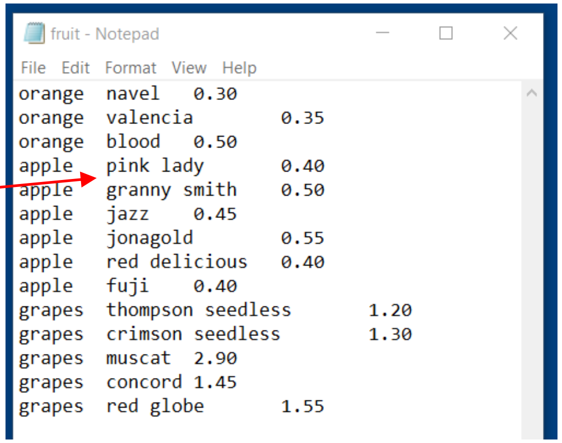
- How about this?



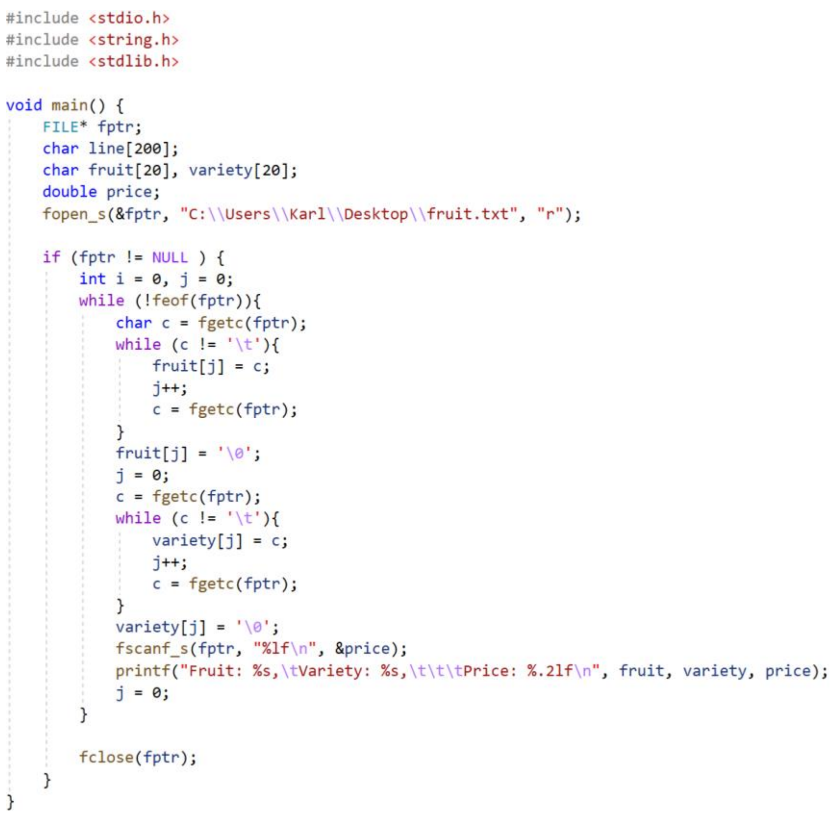
```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
    FILE* fptr;
    char line[200];
    char fruit[20], variety[20];
    double price;
    fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\fruit.txt", "r");

    if (fptr != NULL ) {
        int i = 0, j = 0;
        while (!feof(fptr)){
            char c = fgetc(fptr);
            while (c != '\t'){
                fruit[j] = c;
                j++;
                c = fgetc(fptr);
            }
            fruit[j] = '\0';
            j = 0;
            c = fgetc(fptr);
            while (c != '\t'){
                variety[j] = c;
                j++;
                c = fgetc(fptr);
            }
            variety[j] = '\0';
            fscanf_s(fptr, "%lf\n", &price);
            printf("Fruit: %s,\tVariety: %s,\t\t\tPrice: %.2lf\n", fruit, variety, price);
            j = 0;
        }

        fclose(fptr);
    }
}
```

Fruit Example 2

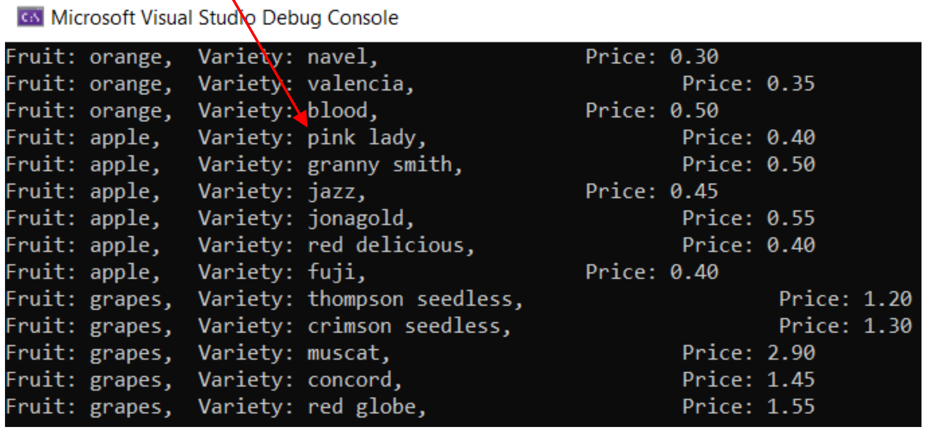
- How about this? Much better



```
orange	navel	0.30
orange	valencia	0.35
orange	blood	0.50
apple	pink lady	0.40
apple	granny smith	0.50
apple	jazz	0.45
apple	jonagold	0.55
apple	red delicious	0.40
apple	fuji	0.40
grapes	thompson seedless	1.20
grapes	crimson seedless	1.30
grapes	muscat	2.90
grapes	concord 1.45
grapes	red globe	1.55
```



```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
    FILE* fptr;
    char line[200];
    char fruit[20], variety[20];
    double price;
    fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\fruit.txt", "r");

    if (fptr != NULL ) {
        int i = 0, j = 0;
        while (!feof(fptr)){
            char c = fgetc(fptr);
            while (c != '\t'){
                fruit[j] = c;
                j++;
                c = fgetc(fptr);
            }
            fruit[j] = '\0';
            j = 0;
            c = fgetc(fptr);
            while (c != '\t'){
                variety[j] = c;
                j++;
                c = fgetc(fptr);
            }
            variety[j] = '\0';
            fscanf_s(fptr, "%lf\n", &price);
            printf("Fruit: %s,\tVariety: %s,\t\t\tPrice: %.2lf\n", fruit, variety, price);
            j = 0;
        }

        fclose(fptr);
    }
}
```



```
Fruit: orange,  Variety: navel,                 Price: 0.30
Fruit: orange,  Variety: valencia,                       Price: 0.35
Fruit: orange,  Variety: blood,                 Price: 0.50
Fruit: apple,   Variety: pink lady,                      Price: 0.40
Fruit: apple,   Variety: granny smith,                   Price: 0.50
Fruit: apple,   Variety: jazz,                  Price: 0.45
Fruit: apple,   Variety: jonagold,                       Price: 0.55
Fruit: apple,   Variety: red delicious,                  Price: 0.40
Fruit: apple,   Variety: fuji,                  Price: 0.40
Fruit: grapes,  Variety: thompson seedless,                       Price: 1.20
Fruit: grapes,  Variety: crimson seedless,                        Price: 1.30
Fruit: grapes,  Variety: muscat,                         Price: 2.90
Fruit: grapes,  Variety: concord,                        Price: 1.45
Fruit: grapes,  Variety: red globe,                      Price: 1.55
```

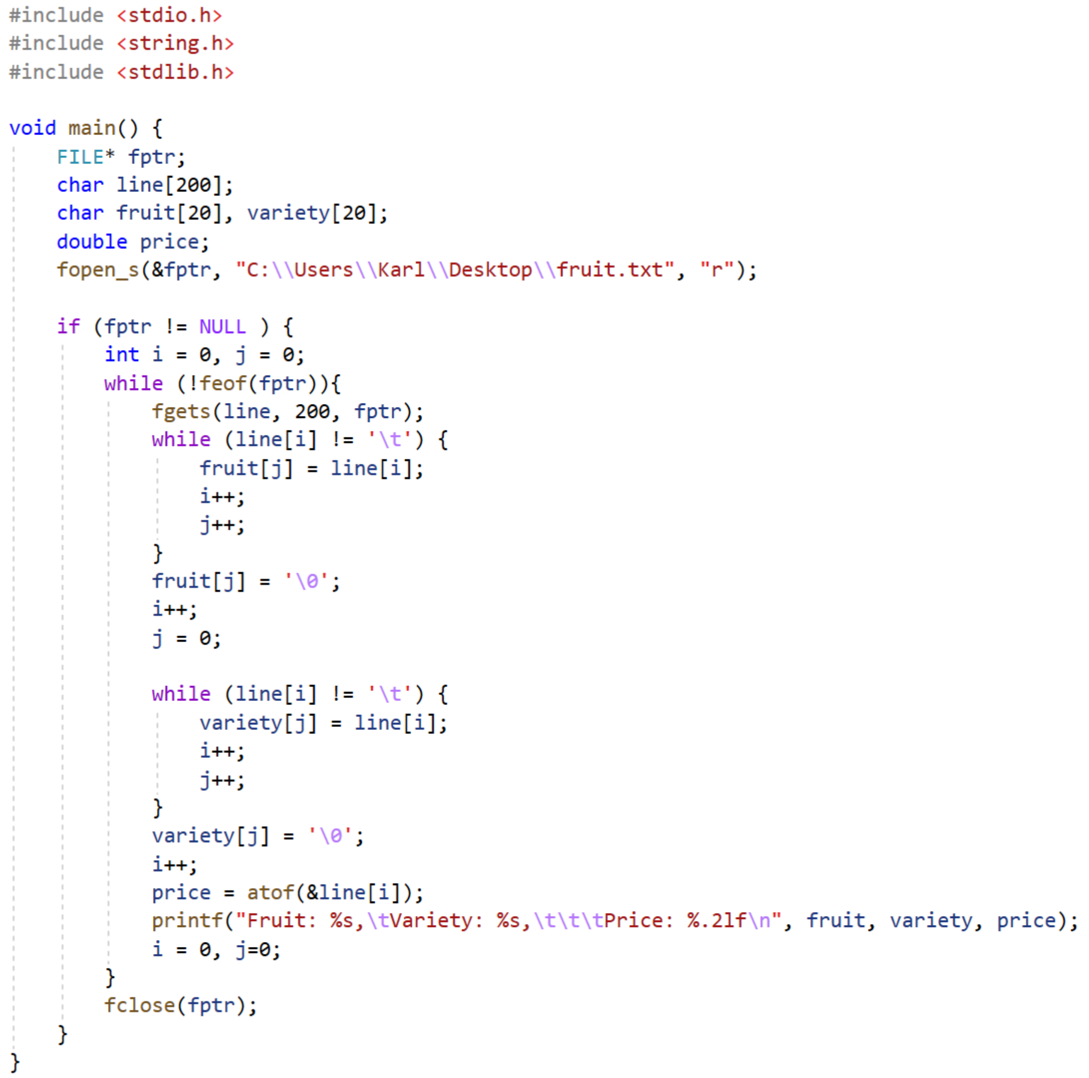
Parsing

- In this solution, we used **parsing**.
- **Parsing** refers to analysing a string or text into logical syntactic components.

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
    FILE* fptr;
    char line[200];
    char fruit[20], variety[20];
    double price;
    fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\fruit.txt", "r");

    if (fptr != NULL ) {
        int i = 0, j = 0;
        while (!feof(fptr)){
            char c = fgetc(fptr);
            while (c != '\t'){
                fruit[j] = c;
                j++;
                c = fgetc(fptr);
            }
            fruit[j] = '\0';
            j = 0;
            c = fgetc(fptr);
            while (c != '\t'){
                variety[j] = c;
                j++;
                c = fgetc(fptr);
            }
            variety[j] = '\0';
            fscanf_s(fptr, "%lf\n", &price);
            printf("Fruit: %s,\tVariety: %s,\t\t\tPrice: %.2lf\n", fruit, variety, price);
            j = 0;
        }

        fclose(fptr);
    }
}
```

Fruit Example 3

- Another way of doing it.

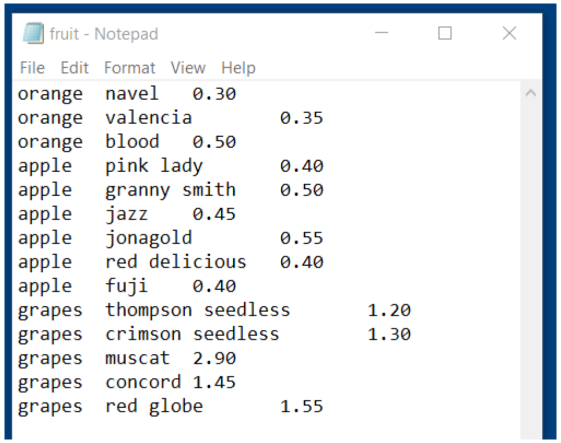


```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
    FILE* fptr;
    char line[200];
    char fruit[20], variety[20];
    double price;
    fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\fruit.txt", "r");

    if (fptr != NULL ) {
        int i = 0, j = 0;
        while (!feof(fptr)){
            fgets(line, 200, fptr);
            while (line[i] != '\t') {
                fruit[j] = line[i];
                i++;
                j++;
            }
            fruit[j] = '\0';
            i++;
            j = 0;

            while (line[i] != '\t') {
                variety[j] = line[i];
                i++;
                j++;
            }
            variety[j] = '\0';
            i++;
            price = atof(&line[i]);
            printf("Fruit: %s,\tVariety: %s,\t\t\tPrice: %.2lf\n", fruit, variety, price);
            i = 0, j=0;
        }
        fclose(fptr);
    }
}
```

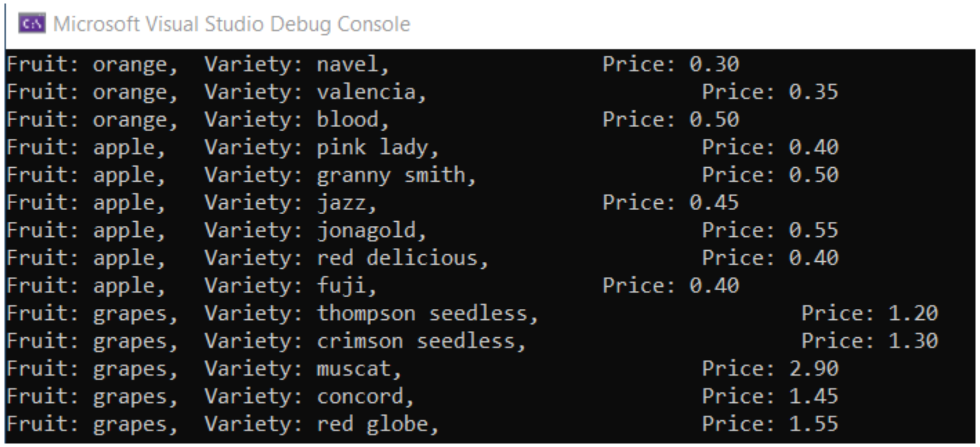
Fruit Example 3

- Works perfectly.

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
    FILE* fptr;
    char line[200];
    char fruit[20], variety[20];
    double price;
    fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\fruit.txt", "r");

    if (fptr != NULL ) {
        int i = 0, j = 0;
        while (!feof(fptr)){
            fgets(line, 200, fptr);
            while (line[i] != '\t') {
                fruit[j] = line[i];
                i++;
                j++;
            }
            fruit[j] = '\0';
            i++;
            j = 0;

            while (line[i] != '\t') {
                variety[j] = line[i];
                i++;
                j++;
            }
            variety[j] = '\0';
            i++;
            price = atof(&line[i]);
            printf("Fruit: %s,\tVariety: %s,\t\t\tPrice: %.2lf\n", fruit, variety, price);
            i = 0, j=0;
        }
        fclose(fptr);
    }
}
```



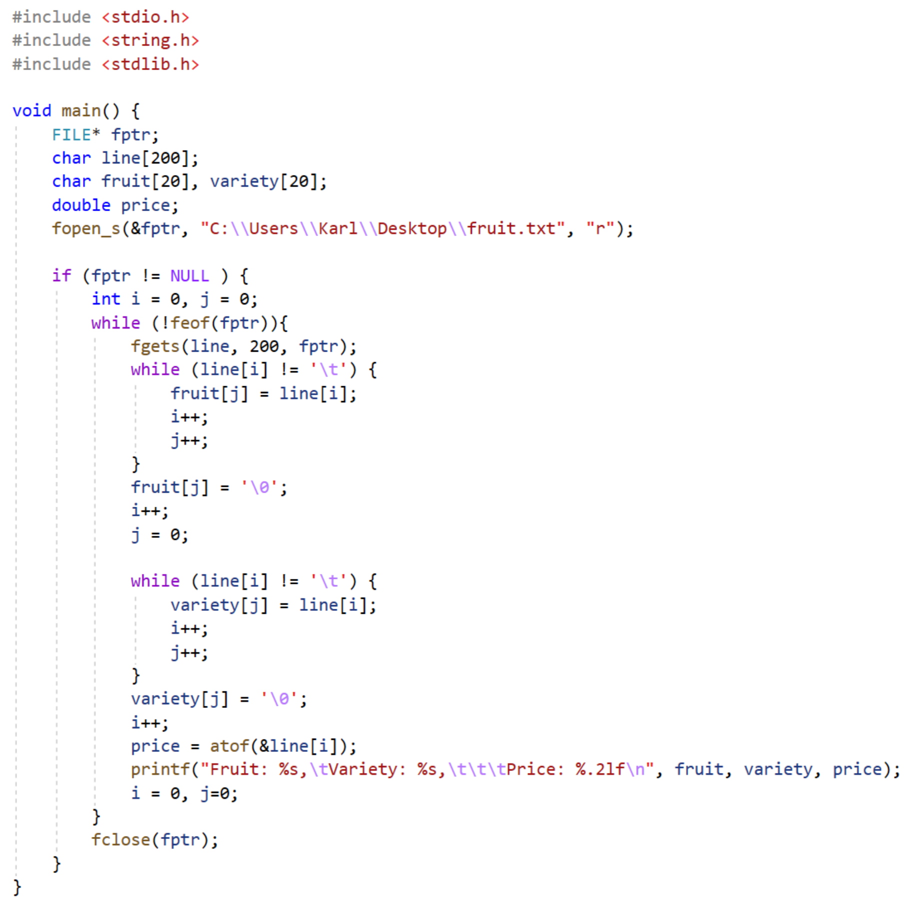
```
orange  navel   0.30
orange  valencia        0.35
orange  blood   0.50
apple   pink lady       0.40
apple   granny smith    0.50
apple   jazz    0.45
apple   jonagold        0.55
apple   red delicious   0.40
apple   fuji    0.40
grapes  thompson seedless       1.20
grapes  crimson seedless        1.30
grapes  muscat  2.90
grapes  concord 1.45
grapes  red globe       1.55
```



```
Fruit: orange,  Variety: navel,                 Price: 0.30
Fruit: orange,  Variety: valencia,                      Price: 0.35
Fruit: orange,  Variety: blood,                 Price: 0.50
Fruit: apple,   Variety: pink lady,                     Price: 0.40
Fruit: apple,   Variety: granny smith,                  Price: 0.50
Fruit: apple,   Variety: jazz,                  Price: 0.45
Fruit: apple,   Variety: jonagold,                      Price: 0.55
Fruit: apple,   Variety: red delicious,                 Price: 0.40
Fruit: apple,   Variety: fuji,                  Price: 0.40
Fruit: grapes,  Variety: thompson seedless,                     Price: 1.20
Fruit: grapes,  Variety: crimson seedless,                      Price: 1.30
Fruit: grapes,  Variety: muscat,                        Price: 2.90
Fruit: grapes,  Variety: concord,                       Price: 1.45
Fruit: grapes,  Variety: red globe,                     Price: 1.55
```

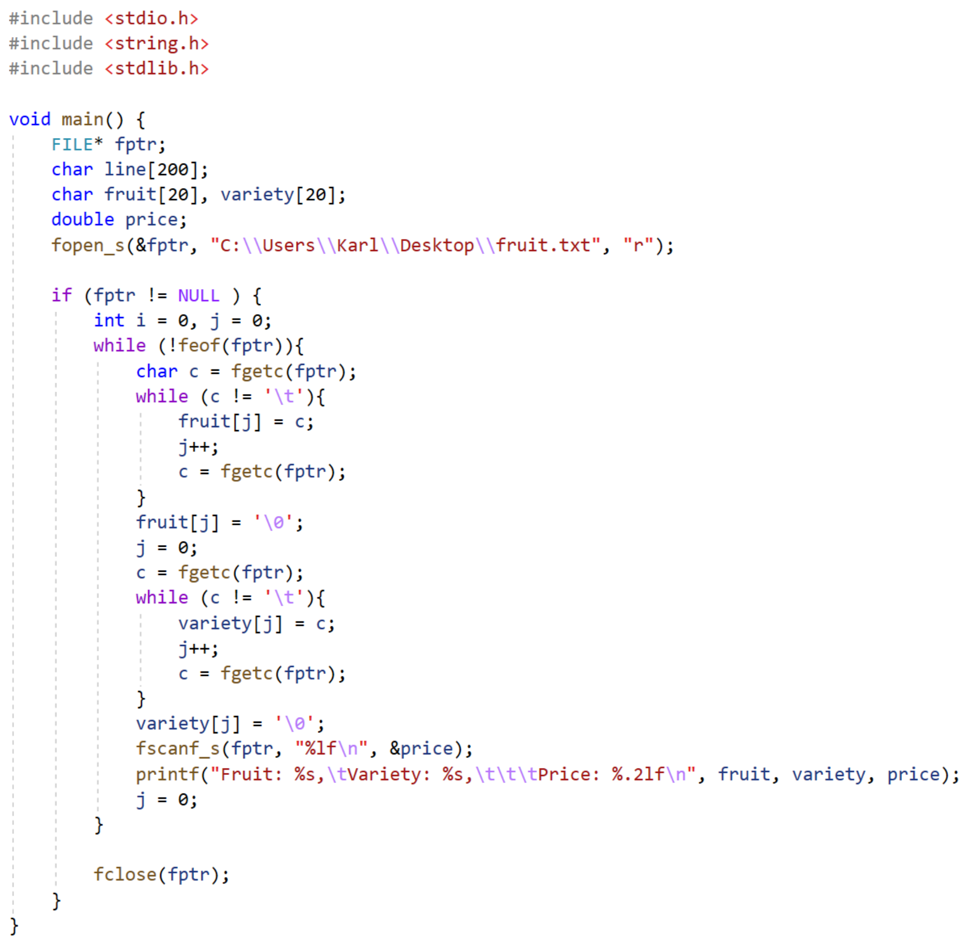
atof()

- We used **atof()** in this solution.
- **atof** is a useful function to convert a string to a **float**.
- In solution 3 we also used fgets() to read in a line at a time.

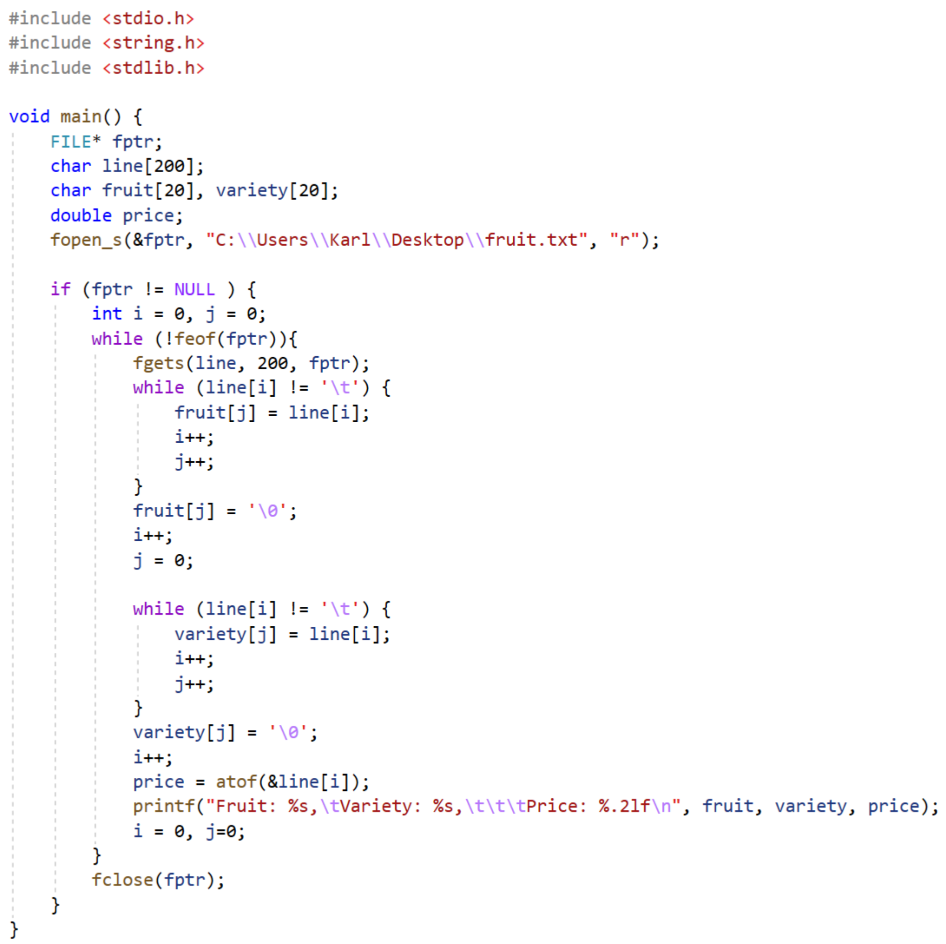


```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
    FILE* fptr;
    char line[200];
    char fruit[20], variety[20];
    double price;
    fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\fruit.txt", "r");

    if (fptr != NULL ) {
        int i = 0, j = 0;
        while (!feof(fptr)){
            fgets(line, 200, fptr);
            while (line[i] != '\t') {
                fruit[j] = line[i];
                i++;
                j++;
            }
            fruit[j] = '\0';
            i++;
            j = 0;

            while (line[i] != '\t') {
                variety[j] = line[i];
                i++;
                j++;
            }
            variety[j] = '\0';
            i++;
            price = atof(&line[i]);
            printf("Fruit: %s,\tVariety: %s,\t\t\tPrice: %.2lf\n", fruit, variety, price);
            i = 0, j=0;
        }
        fclose(fptr);
    }
}
```

Side by side comparison



```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
    FILE* fptr;
    char line[200];
    char fruit[20], variety[20];
    double price;
    fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\fruit.txt", "r");

    if (fptr != NULL ) {
        int i = 0, j = 0;
        while (!feof(fptr)){
            char c = fgetc(fptr);
            while (c != '\t'){
                fruit[j] = c;
                j++;
                c = fgetc(fptr);
            }
            fruit[j] = '\0';
            j = 0;
            c = fgetc(fptr);
            while (c != '\t'){
                variety[j] = c;
                j++;
                c = fgetc(fptr);
            }
            variety[j] = '\0';
            fscanf_s(fptr, "%lf\n", &price);
            printf("Fruit: %s,\tVariety: %s,\t\t\tPrice: %.2lf\n", fruit, variety, price);
            j = 0;
        }

        fclose(fptr);
    }
}
```



```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
    FILE* fptr;
    char line[200];
    char fruit[20], variety[20];
    double price;
    fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\fruit.txt", "r");

    if (fptr != NULL ) {
        int i = 0, j = 0;
        while (!feof(fptr)){
            fgets(line, 200, fptr);
            while (line[i] != '\t') {
                fruit[j] = line[i];
                i++;
                j++;
            }
            fruit[j] = '\0';
            i++;
            j = 0;

            while (line[i] != '\t') {
                variety[j] = line[i];
                i++;
                j++;
            }
            variety[j] = '\0';
            i++;
            price = atof(&line[i]);
            printf("Fruit: %s,\tVariety: %s,\t\t\tPrice: %.2lf\n", fruit, variety, price);
            i = 0, j=0;
        }
        fclose(fptr);
    }
}
```

C CODE

C Code

- Let’s finish today’s lecture by running some C programs in Visual Studio.

PROGRAMMING

CT103

Week 16

Sign in on Blackboard

- Please sign in on blackboard for CT103.
- I will leave the sign in option open for an hour after the lecture.

Lecture Content

- Today’s lecture (Week 16):
  - Memory requirements
  - Pointers
  - Pointers and arrays
  - Pointers and strings
  - Pointers and functions
  - Example C programme

MEMORY REQUIREMENTS

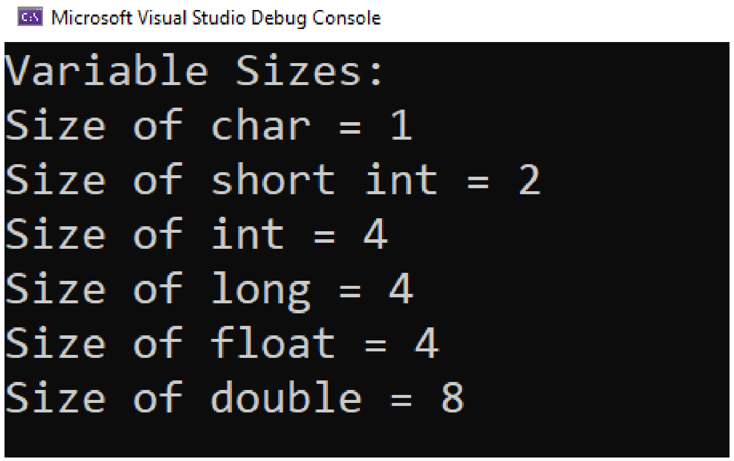
Memory Requirements

- **Different space requirements for different variables:**

```
char c;
short int i;
int j;
long x;
float f;
double d;

puts ("Variable Sizes: ");
printf("Size of char = %d \n", sizeof(c));
printf("Size of short int = %d \n", sizeof(i));
printf("Size of int = %d \n", sizeof(j));
printf("Size of long = %d \n", sizeof(x));
printf("Size of float = %d \n", sizeof(f));
printf("Size of double = %d \n", sizeof(d));
```

Sizeof()

- sizeof() returns the number of bytes of whatever variable (including structures and arrays) you give it.

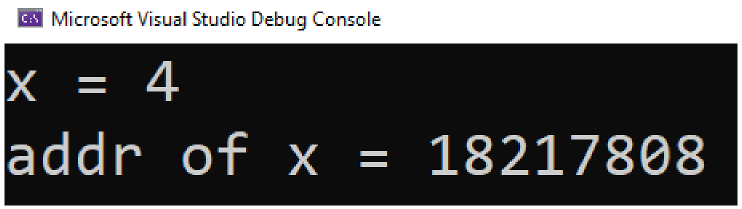


```
Variable Sizes:
Size of char = 1
Size of short int = 2
Size of int = 4
Size of long = 4
Size of float = 4
Size of double = 8
```

Memory Addresses

- We store variables in memory.
- We access the address of a variable by putting & in front of it.
- For example:

```
int x = 4;

printf ("x = %d \n", x);
printf ("addr of x = %d \n",&x);
```

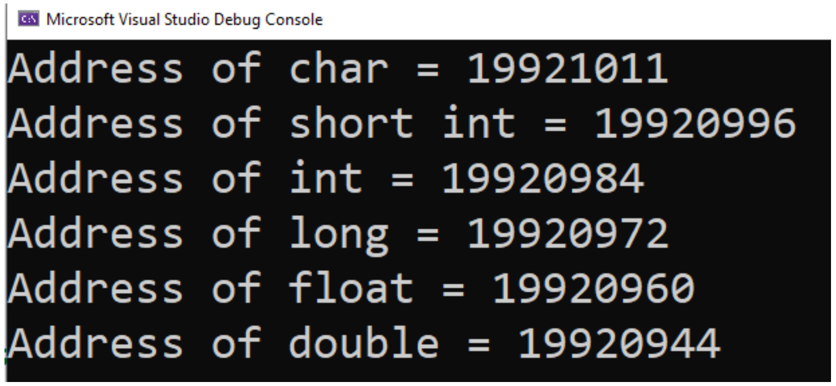

Microsoft Visual Studio Debug Console

```
x = 4
addr of x = 18217808
```

Decimal Addresses

- %d to see address as decimal

```
printf("Address of char = %d \n", &c);
printf("Address of short int = %d \n", &i);
printf("Address of int = %d \n", &j);
printf("Address of long = %d \n", &x);
printf("Address of float = %d \n", &f);
printf("Address of double = %d \n", &d);
```

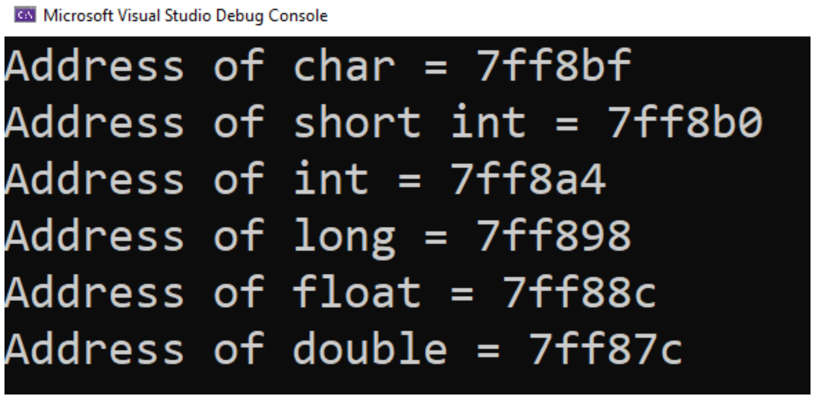


```
Address of char = 19921011
Address of short int = 19920996
Address of int = 19920984
Address of long = 19920972
Address of float = 19920960
Address of double = 19920944
```

Hexadecimal Addresses

- %x to see hexadecimal address (lowercase letters)

```
printf("Address of char = %x \n", &c);
printf("Address of short int = %x \n", &i);
printf("Address of int = %x \n", &j);
printf("Address of long = %x \n", &x);
printf("Address of float = %x \n", &f);
printf("Address of double = %x \n", &d);
```

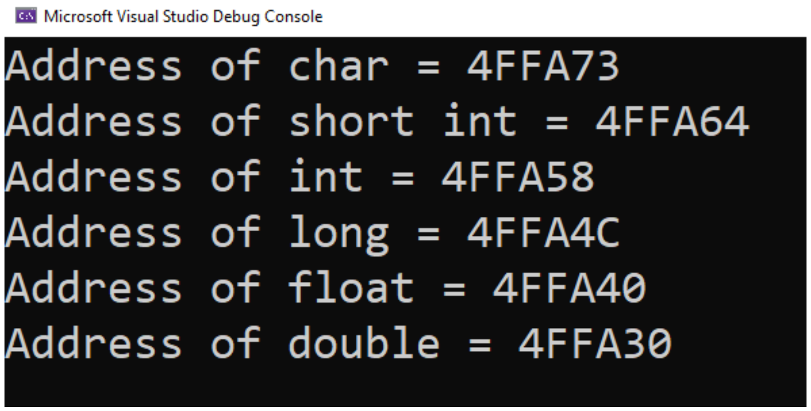


```
Address of char = 7ff8bf
Address of short int = 7ff8b0
Address of int = 7ff8a4
Address of long = 7ff898
Address of float = 7ff88c
Address of double = 7ff87c
```

Hexadecimal Addresses

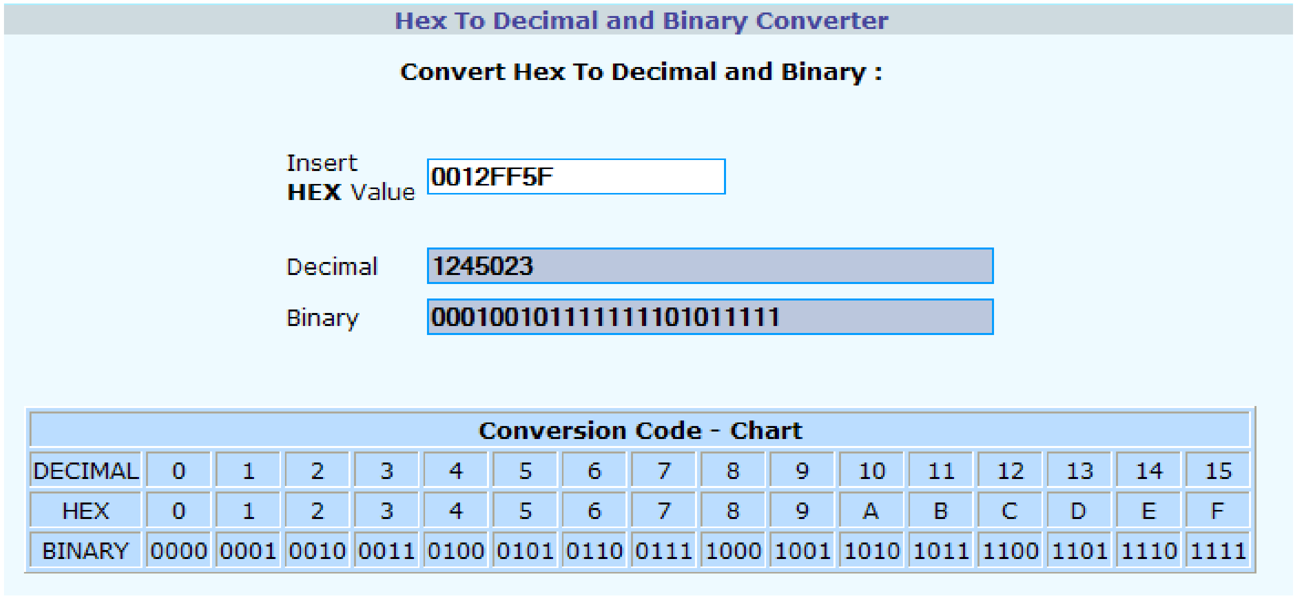
- %X to see hexadecimal address (uppercase letters)

```
printf("Address of char = %X \n", &c);
printf("Address of short int = %X \n", &i);
printf("Address of int = %X \n", &j);
printf("Address of long = %X \n", &x);
printf("Address of float = %X \n", &f);
printf("Address of double = %X \n\n", &d);
```


Microsoft Visual Studio Debug Console

```
Address of char = 4FFA73
Address of short int = 4FFA64
Address of int = 4FFA58
Address of long = 4FFA4C
Address of float = 4FFA40
Address of double = 4FFA30
```

Hexadecimal – Numbers Base 16



Hex To Decimal and Binary Converter

Convert Hex To Decimal and Binary :

Insert **HEX** Value	0012FF5F
Decimal	1245023
Binary	000100101111111101011111

Conversion Code - Chart

DECIMAL	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
BINARY	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

POINTERS

What are pointers?

- We have seen pointers a few times, e.g.
  - When we create file pointers to open a file.

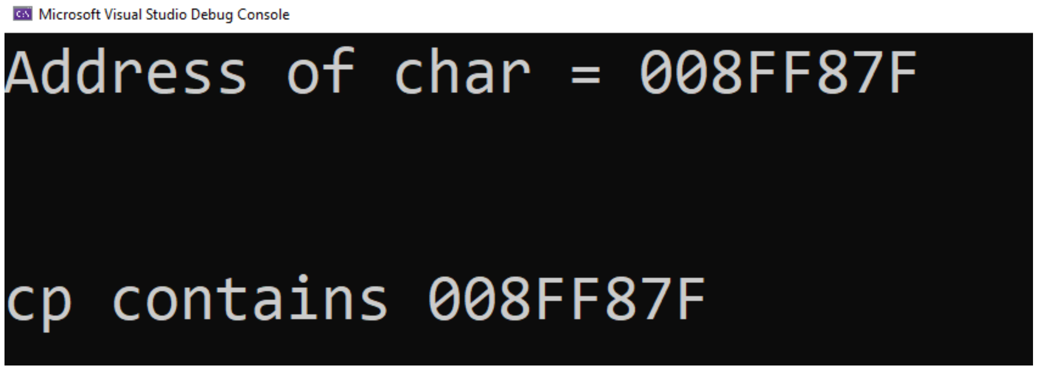
```
FILE* fptr;
fopen_s(&fptr, str, "r");
```

- **Definition**: A pointer is a variable whose value is the address of another variable.

Pointers

- %p will print leading zeros

```
char c;

char *cp = &c;

printf("Address of char = %p \n\n\n", &c);


printf("cp contains %p \n",cp);
```



Microsoft Visual Studio Debug Console

```
Address of char = 008FF87F


cp contains 008FF87F
```

Pointers

- The operator * has special purpose also.
- We usually apply it to a memory address.
  - It returns the variable which that address points to!

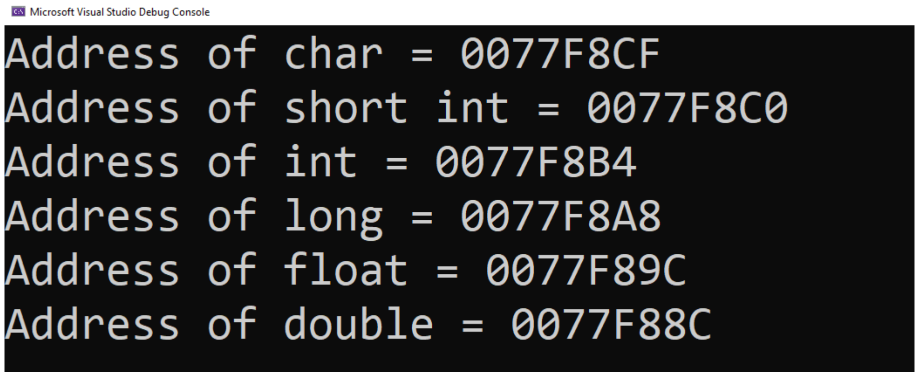
- We have to use pointer variables to hold memory addresses for the appropriate type of variable
  - E.g

```
int * x;      // defines pointer to an integer
float *y;     // defines pointer to a float
```

Print Pointers

- %p will print leading zeros

```
printf("Address of char = %p \n", &c);
printf("Address of short int = %p \n", &i);
printf("Address of int = %p \n", &j);
printf("Address of long = %p \n", &x);
printf("Address of float = %p \n", &f);
printf("Address of double = %p \n", &d);
```

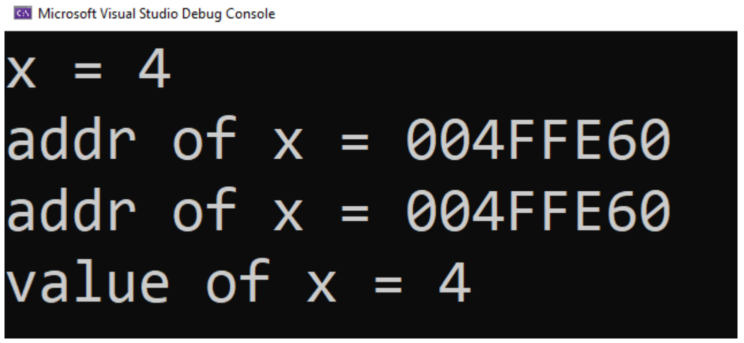


Microsoft Visual Studio Debug Console

```
Address of char = 0077F8CF
Address of short int = 0077F8C0
Address of int = 0077F8B4
Address of long = 0077F8A8
Address of float = 0077F89C
Address of double = 0077F88C
```

POINTERS EXAMPLE

Pointers Example 1



```
x = 4
addr of x = 004FFE60
addr of x = 004FFE60
value of x = 4
```

```
void main(){
        int x = 4;
        int* addr;

        addr = &x;

        printf("x = %d \n", x);
        printf("addr of x = %p \n", &x);
        printf("addr of x = %p \n", addr);
        printf("value of x = %d \n", *addr);
}
```

Remember: A pointer is a variable whose value is the address of another variable

Remember

- Pointer variables hold the addresses of other variables – that’s their purpose.

- We don’t know in advance where the program will store the variables – and normally don’t care.

- *Dereferencing*
  - just means using the pointer to get to the variable!

Dereferencing

- Here we are dereferencing the pointer:

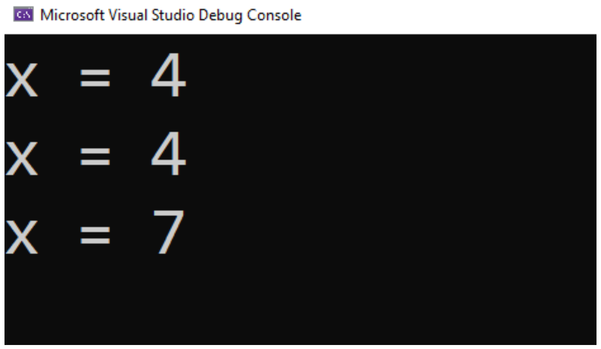
```
void main(){
       int x = 4;
       int* addr;

       addr = &x;

       printf("x = %d \n", x);
       printf("addr of x = %p \n", &x);
       printf("addr of x = %p \n", addr);
       printf("value of x = %d \n", *addr);
}
```

Dereferencing

Pointers Example 2

- Here we are dereferencing the pointer:

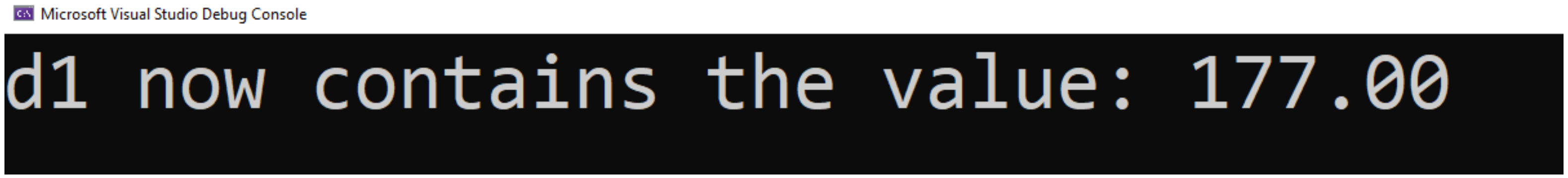
```
void main(){
            int x = 4;  // store 4 in memory location given to x
            int* a1; // create a variable that can hold the address of an integer

            a1 = &x; // store the address of x in our new pointer variable

            printf("x = %d \n", x); // print out the value of x (4)

            printf("x = %d \n", *a1); // dereference the address stored in a1 and print value found there(4)

            *a1 = 7;  // store the value 7 in the variable stored at address a1 (which is the address of x)

            printf("x = %d \n", x); // print out value of variable of x (should be 7!)
}
```


Microsoft Visual Studio Debug Console

```
x = 4
x = 4
x = 7
```

Pointers Example 3

```
void main(){
    double d1 = 88.5;
    double* p1 = &d1;

    *p1 = *p1 * 2.0;

    printf("d1 now contains the value: %.2f \n", d1);
}
```


Microsoft Visual Studio Debug Console

```
d1 now contains the value: 177.00
```

Do’s and Don’t’s

- Get used to pointers.
- Use the & to get the address of a variable.
- Use * to define a pointer variable and to dereference a pointer variable.
- Only use pointer variables with the correct variable data type.
    - E.g. an integer pointer must point to an integer

POINTERS AND ARRAYS

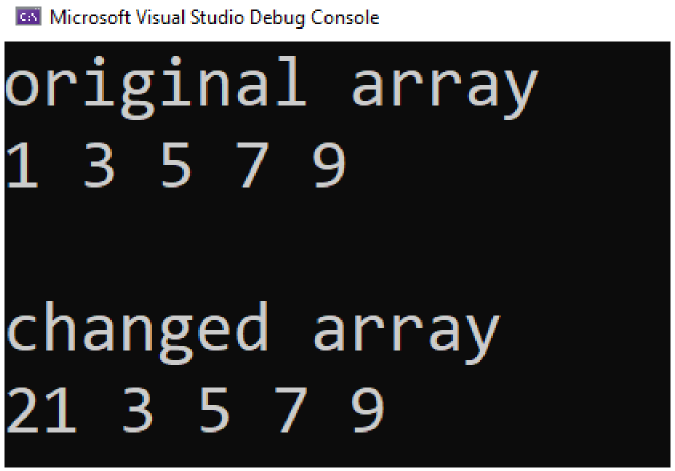
Pointers and Arrays

- So far today, we have looked at pointers for variables like integers, floats, chars, etc.

- Next we will look at using pointers for arrays.

Pointers and Arrays

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void printArray(int arr[], int len);

void main(){
    int array1[] = { 1,3,5,7,9 };
    puts("original array");
    printArray(array1, 5);

    int* ip = array1;
    *ip = 21;
    puts("changed array");
    printArray(array1, 5);
}

void printArray(int arr[], int len){
    for (int i = 0; i < len; i++){
        printf("%d ", arr[i]);
    }
    printf("\n\n");
}
```

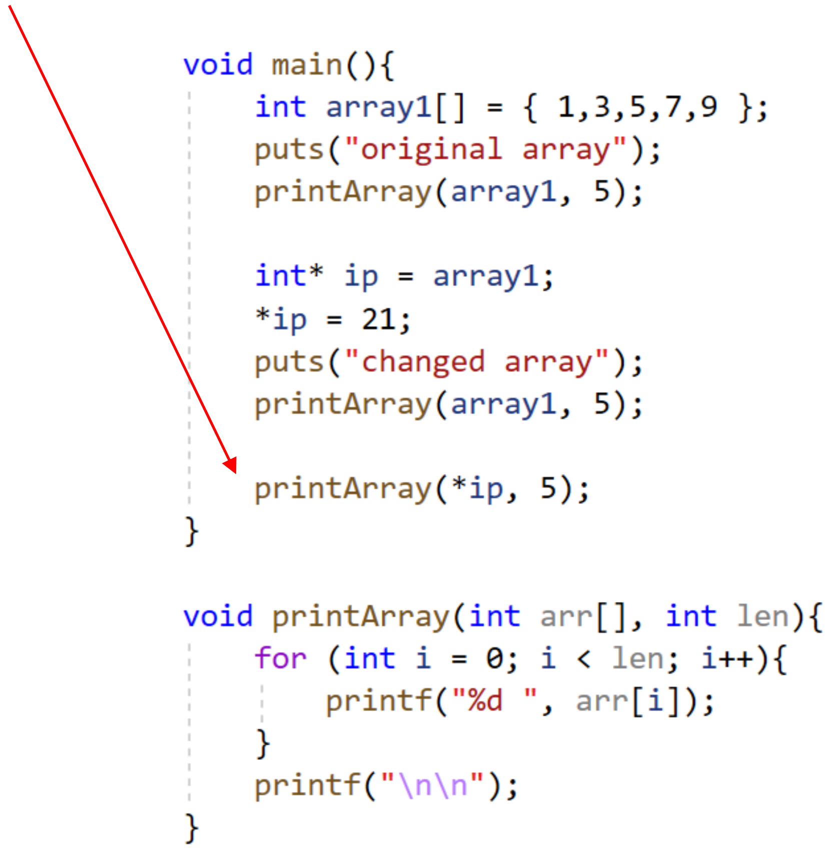


```
original array
1 3 5 7 9

changed array
21 3 5 7 9
```

The pointer 'ip' will point to the 0th element of array1

Pointers and Arrays

- Could I dereference ‘ip’ and pass that into printArray?
  - i.e. would this work?

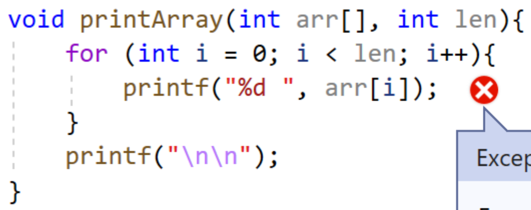


```c
void main(){
    int array1[] = { 1,3,5,7,9 };
    puts("original array");
    printArray(array1, 5);

    int* ip = array1;
    *ip = 21;
    puts("changed array");
    printArray(array1, 5);

    printArray(*ip, 5);
}

void printArray(int arr[], int len){
    for (int i = 0; i < len; i++){
        printf("%d ", arr[i]);
    }
    printf("\n\n");
}
```

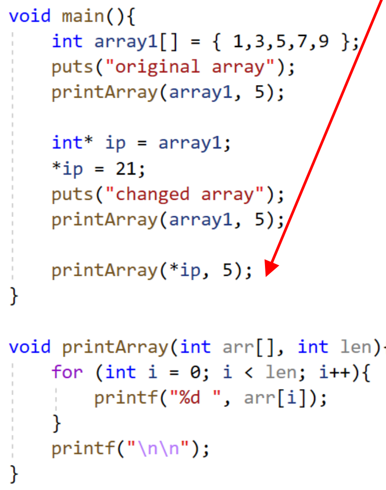
Pointers and Arrays

- Could I dereference ‘ip’ and pass that into printArray?
  - i.e. would this work? **No.**

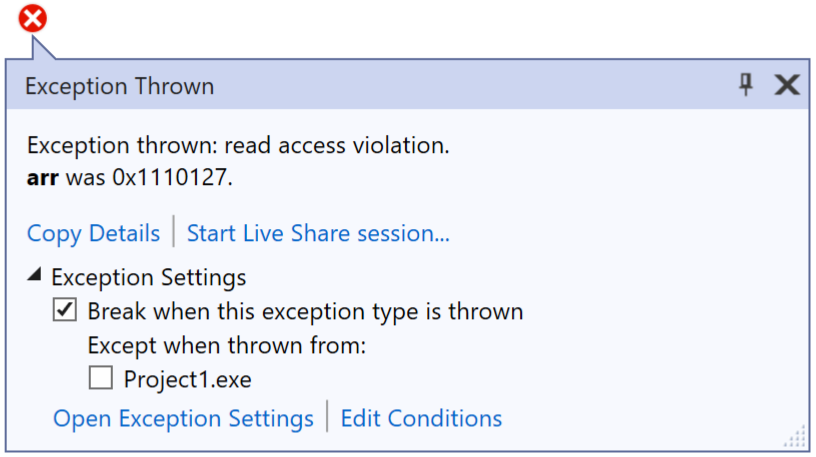
The pointer ‘ip’ will point to the 0th element of array1.
We need to point to the full array.



```
void printArray(int arr[], int len){
    for (int i = 0; i < len; i++){
        printf("%d ", arr[i]);
    }
    printf("\n\n");
}
```



```
void main(){
    int array1[] = { 1,3,5,7,9 };
    puts("original array");
    printArray(array1, 5);

    int* ip = array1;
    *ip = 21;
    puts("changed array");
    printArray(array1, 5);

    printArray(*ip, 5);
}

void printArray(int arr[], int len){
    for (int i = 0; i < len; i++){
        printf("%d ", arr[i]);
    }
    printf("\n\n");
}
```



Exception Thrown

Exception thrown: read access violation.
arr was 0x1110127.

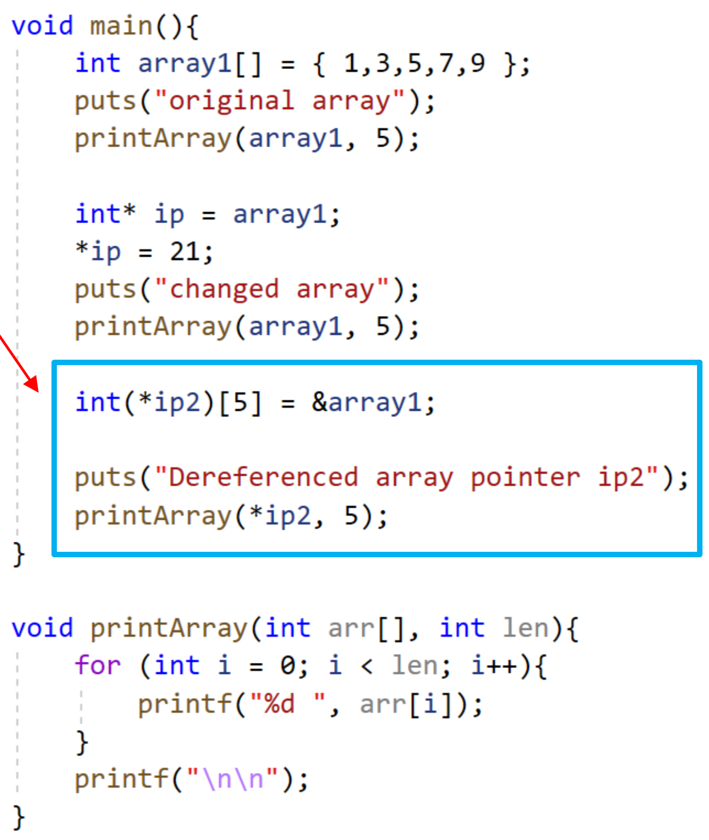
Copy Details | Start Live Share session...

Exception Settings
[✓] Break when this exception type is thrown
Except when thrown from:
[ ] Project1.exe

Open Exception Settings | Edit Conditions

Pointers and Arrays

- We would need to point to the full array.

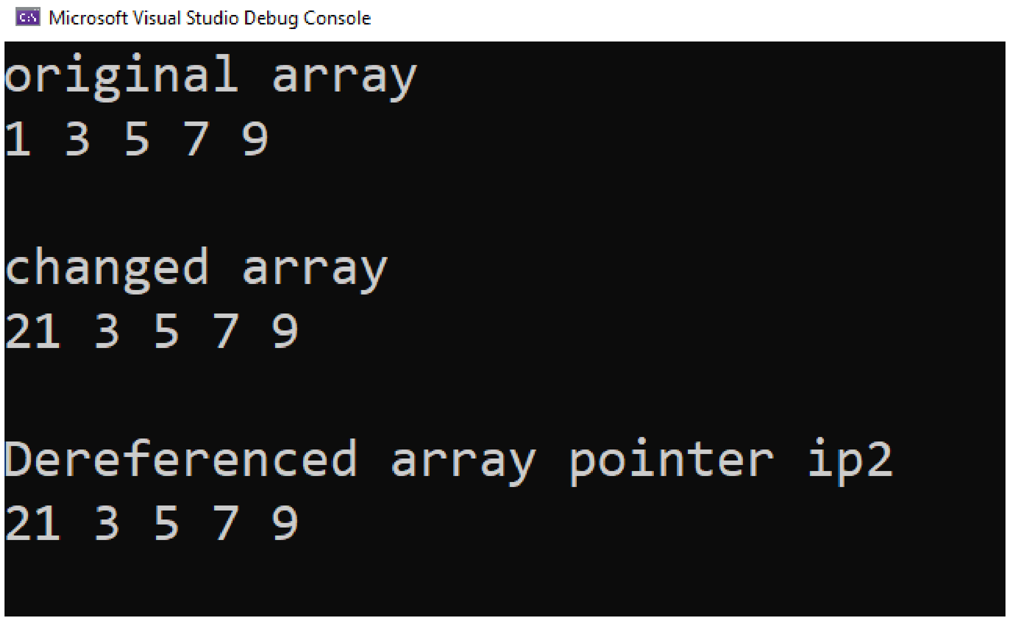


```
void main(){
    int array1[] = { 1,3,5,7,9 };
    puts("original array");
    printArray(array1, 5);

    int* ip = array1;
    *ip = 21;
    puts("changed array");
    printArray(array1, 5);

    int(*ip2)[5] = &array1;

    puts("Dereferenced array pointer ip2");
    printArray(*ip2, 5);
}

void printArray(int arr[], int len){
    for (int i = 0; i < len; i++){
        printf("%d ", arr[i]);
    }
    printf("\n\n");
}
```

Pointers and Arrays

- We would need to point to the full array.

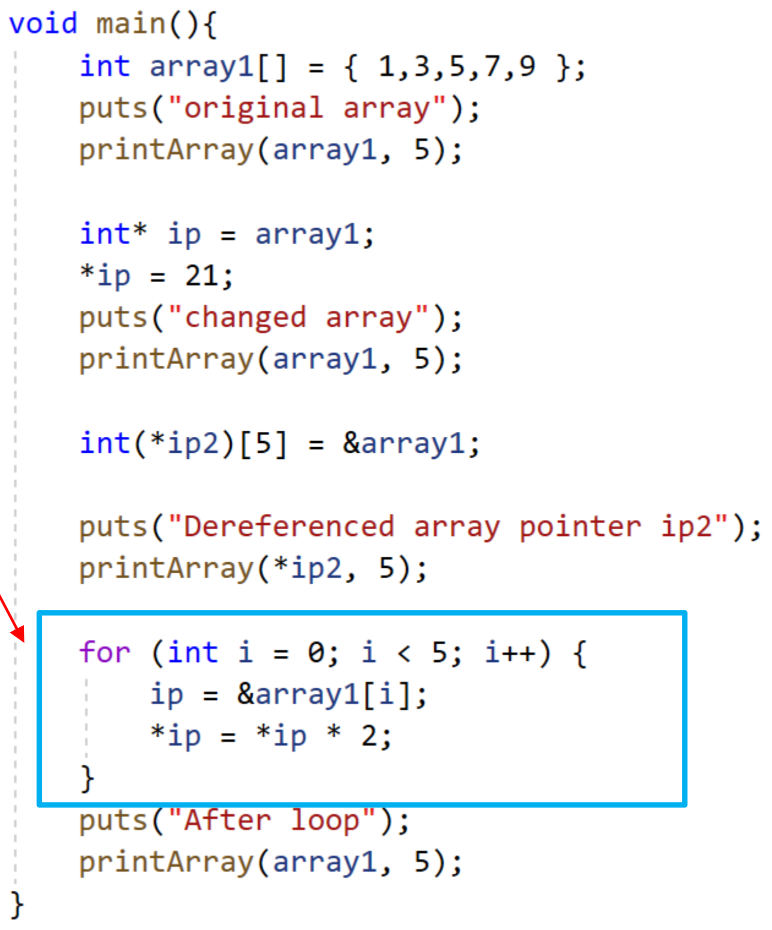
```
void main(){
    int array1[] = { 1,3,5,7,9 };
    puts("original array");
    printArray(array1, 5);

    int* ip = array1;
    *ip = 21;
    puts("changed array");
    printArray(array1, 5);

    int(*ip2)[5] = &array1;

    puts("Dereferenced array pointer ip2");
    printArray(*ip2, 5);
}

void printArray(int arr[], int len){
    for (int i = 0; i < len; i++){
        printf("%d ", arr[i]);
    }
    printf("\n\n");
}
```



```
original array
1 3 5 7 9

changed array
21 3 5 7 9

Dereferenced array pointer ip2
21 3 5 7 9
```

Pointers and Arrays

- Can we reset array pointers?
  - i.e. would this work?

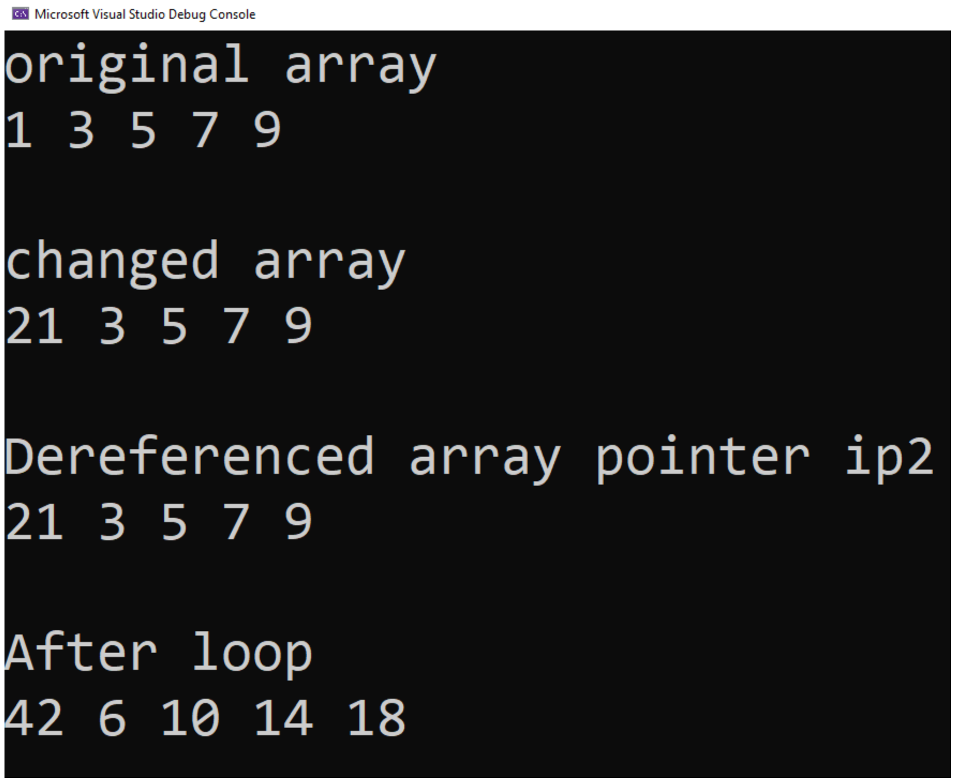


```c
void main(){
    int array1[] = { 1,3,5,7,9 };
    puts("original array");
    printArray(array1, 5);

    int* ip = array1;
    *ip = 21;
    puts("changed array");
    printArray(array1, 5);

    int(*ip2)[5] = &array1;

    puts("Dereferenced array pointer ip2");
    printArray(*ip2, 5);

    for (int i = 0; i < 5; i++) {
        ip = &array1[i];
        *ip = *ip * 2;
    }
    puts("After loop");
    printArray(array1, 5);
}
```

Pointers and Arrays

- Can we reset array pointers?
  - i.e. would this work? **Yes.**

```
void main(){
    int array1[] = { 1,3,5,7,9 };
    puts("original array");
    printArray(array1, 5);

    int* ip = array1;
    *ip = 21;
    puts("changed array");
    printArray(array1, 5);

    int(*ip2)[5] = &array1;

    puts("Dereferenced array pointer ip2");
    printArray(*ip2, 5);

    for (int i = 0; i < 5; i++) {
        ip = &array1[i];
        *ip = *ip * 2;
    }
    puts("After loop");
    printArray(array1, 5);
}
```



```
original array
1 3 5 7 9

changed array
21 3 5 7 9

Dereferenced array pointer ip2
21 3 5 7 9

After loop
42 6 10 14 18
```

END OF FIRST LECTURE

START OF SECOND LECTURE

POINTERS AND STRINGS

Pointers and Strings

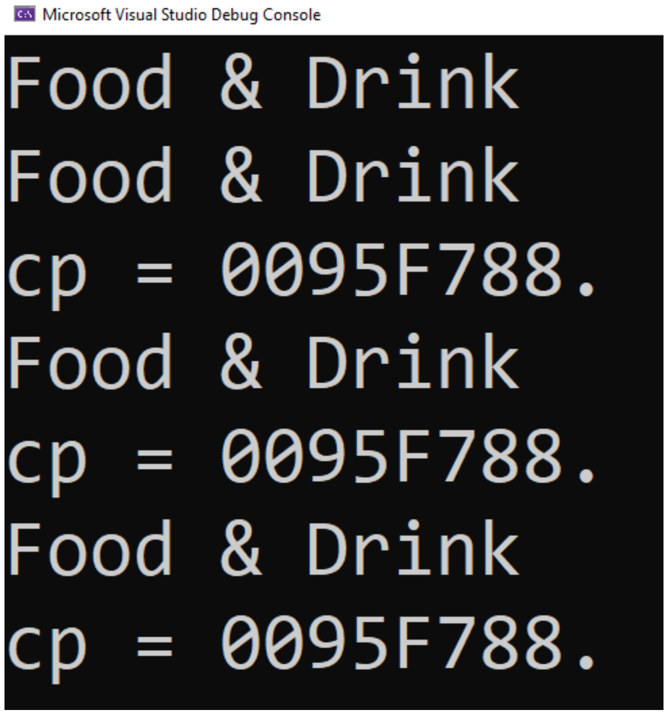
- Let’s now have a look at pointers and strings.
- Which of the following are correct?

```
void main(){
    char string1[] = "Food & Drink";
    puts(string1);

    char* cp;
    cp = string1;
    puts(cp);
    printf("cp = %p.\n",cp);

    cp = &string1;
    puts(cp);
    printf("cp = %p.\n", cp);

    cp = &string1[0];
    puts(cp);
    printf("cp = %p.\n", cp);
}
```

Pointers and Strings

- Which of the following are correct? **All of them.**
- Why?

```
void main(){
     char string1[] = "Food & Drink";
     puts(string1);

     char* cp;
     cp = string1;
     puts(cp);
     printf("cp = %p.\n",cp);

     cp = &string1;
     puts(cp);
     printf("cp = %p.\n", cp);

     cp = &string1[0];
     puts(cp);
     printf("cp = %p.\n", cp);
}
```



```
Food & Drink
Food & Drink
cp = 0095F788.
Food & Drink
cp = 0095F788.
Food & Drink
cp = 0095F788.
```

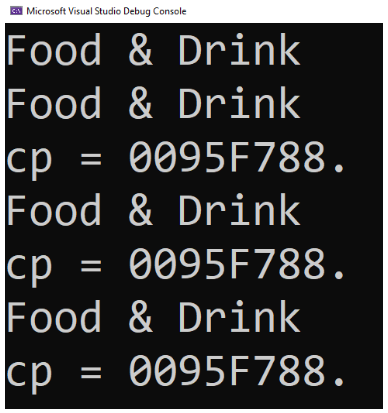
Pointers and Strings

- Which of the following are correct? **All of them.**
- Why?

- An array name often evaluates to a pointer.
  - This is referred to as an array 'decaying' to a pointer.

- Next cp points to 'F' in string
  - Remember 0th element.

- A pointer to an array is the same as a pointer to its first element



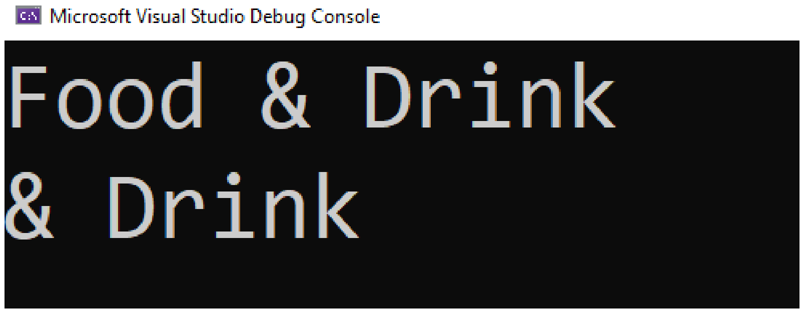
```
Food & Drink
Food & Drink
cp = 0095F788.
Food & Drink
cp = 0095F788.
Food & Drink
cp = 0095F788.
```

```
void main(){
    char string1[] = "Food & Drink";
    puts(string1);

    char* cp;
    cp = string1;
    puts(cp);
    printf("cp = %p.\n",cp);

    cp = &string1;
    puts(cp);
    printf("cp = %p.\n", cp);

    cp = &string1[0];
    puts(cp);
    printf("cp = %p.\n", cp);
}
```

Pointers and Strings

- Puts() will start printing the characters from the address you give it:

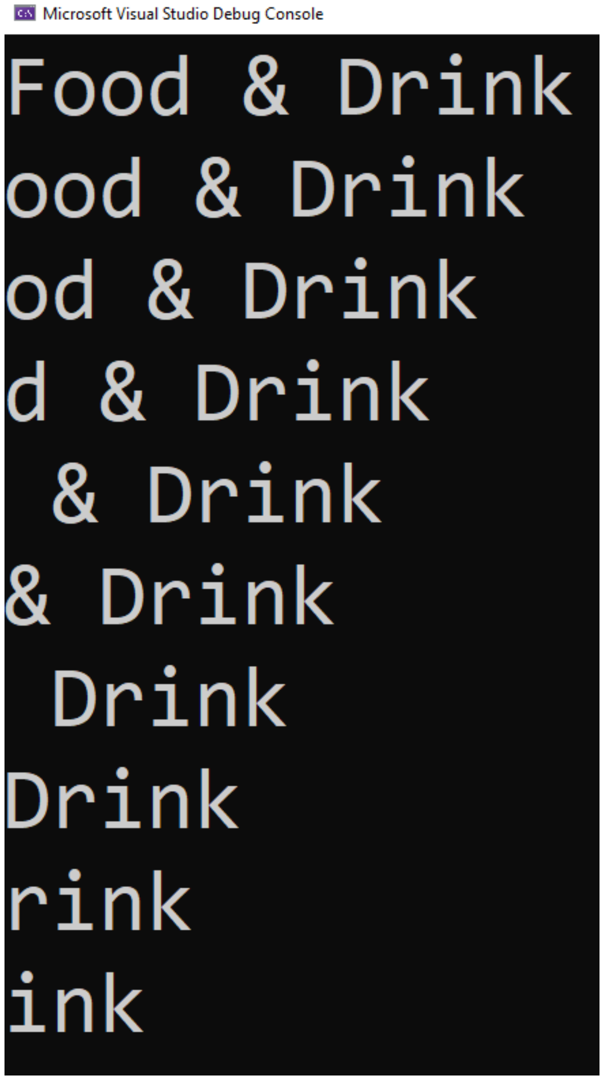
```
void main(){
   char string1[] = "Food & Drink";
   puts(string1);

   char* cp = &string1[5];

   puts(cp);
}
```



```
Food & Drink
& Drink
```

Incrementing Pointers

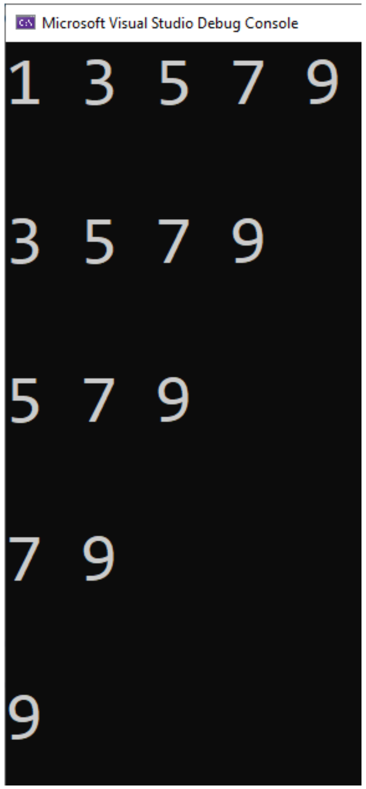
```c
void main(){
    char string1[] = "Food & Drink";

    char* cp = string1;
    for (int i = 0; i < 10;i++) {
        puts(cp);
        cp++;
    }
}
```



```
Food & Drink
ood & Drink
od & Drink
d & Drink
 & Drink
& Drink
 Drink
Drink
rink
ink
```

Incrementing Pointers with Ints

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void printArray(int arr[], int len);

void main(){
    int array1[] = { 1,3,5,7,9 };
    int* ip = array1;
    printArray(ip, 5);
    ip++;
    printArray(ip, 4);
    ip++;
    printArray(ip, 3);
    ip++;
    printArray(ip, 2);
    ip++;
    printArray(ip, 1);
}

void printArray(int arr[], int len) {
    for (int i = 0; i < len; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n\n");
}
```

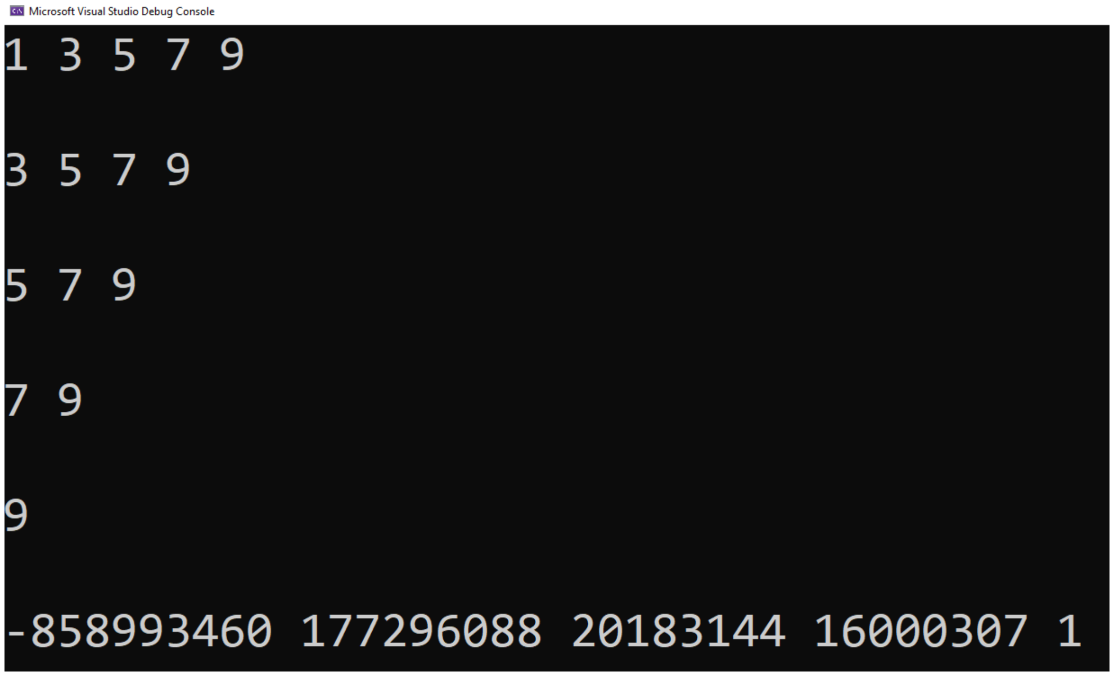


```
1 3 5 7 9

3 5 7 9

5 7 9

7 9

9
```

Be Careful Incrementing Pointers

```
void main(){
    int array1[] = { 1,3,5,7,9 };
    int* ip = array1;
    printArray(ip, 5);
    ip++;
    printArray(ip, 4);
    ip++;
    printArray(ip, 3);
    ip++;
    printArray(ip, 2);
    ip++;
    printArray(ip, 1);
    ip++;
    printArray(ip, 5);
}
```

No warnings or errors and no crash, so these bugs can be hard to find and cause serious problems if giving the wrong data

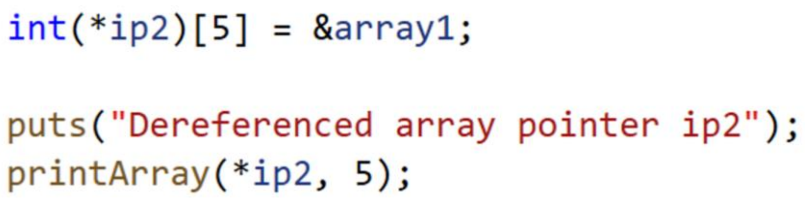


```
1 3 5 7 9

3 5 7 9

5 7 9

7 9

9

-858993460 177296088 20183144 16000307 1
```

POINTERS AND FUNCTIONS

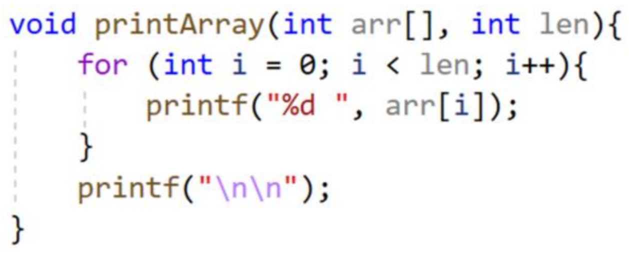
Pointers and Functions

- We already saw earlier how we can pass pointers to full arrays into functions.



```
int(*ip2)[5] = &array1;

puts("Dereferenced array pointer ip2");
printArray(*ip2, 5);
```

- Here printArray() expects an array:

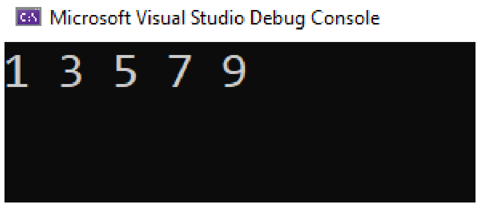


```
void printArray(int arr[], int len){
    for (int i = 0; i < len; i++){
        printf("%d ", arr[i]);
    }
    printf("\n\n");
}
```

Pointers and Functions

- We can see here how printArray() expects a pointer:

```
void printArray(int* ptr, int len);

void main(){
    int array1[] = { 1,3,5,7,9 };

    int* ip = array1; // or &array1[0]

    printArray(ip, 5);
}

void printArray(int* ptr, int len){
    for (int i = 0; i < len; i++){
        printf("%d ", *(ptr + i));
    }
    printf("\n\n");
}
```


Microsoft Visual Studio Debug Console

```
1 3 5 7 9
```

Pointers and Functions

- Will this work?

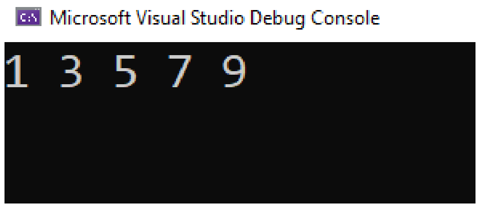
- Since printArray() expects a pointer..

```c
void printArray(int* ptr, int len);

void main(){
    int array1[] = { 1,3,5,7,9 };

    int* ip = array1; // or &array1[0]

    printArray(array1, 5);
}

void printArray(int* ptr, int len){
    for (int i = 0; i < len; i++){
        printf("%d ", *(ptr + i));
    }
    printf("\n\n");
}
```

Pointers and Functions

- Will this work?
  - **Yes!**
- Since printArray() expects a pointer..
- Why?
  - Arrays decay to pointers.

```c
void printArray(int* ptr, int len);

void main(){
    int array1[] = { 1,3,5,7,9 };

    int* ip = array1; // or &array1[0]

    printArray(array1, 5);
}

void printArray(int* ptr, int len){
    for (int i = 0; i < len; i++){
        printf("%d ", *(ptr + i));
    }
    printf("\n\n");
}
```


Microsoft Visual Studio Debug Console

```
1 3 5 7 9
```

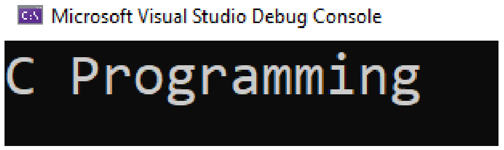
Incrementing Pointers

- When we increment a pointer, we are not simply adding one to the address.

- We are **adding the size in bytes** of whatever data type the pointer points to.

- However, chars are of size 1 byte. So in this case we are actually adding 1 to the address…

```c
void printString(char* cptr);

void main(){
    char string1[] = "C Programming";
    printString(string1);
}

void printString(char* cptr){
    int i = 0;
    while (*(cptr + i) != '\0'){
        printf("%c", *(cptr + i));
        i++;
    }
    printf("\n");
}
```



```
C Programming
```

C CODE

C Code

- Let’s finish today’s lecture by running some C programs in Visual Studio.

PROGRAMMING

CT103

Week 17

Sign in on Blackboard

- Please sign in on blackboard for CT103.
- I will leave the sign in option open for an hour after the lecture.

Question from last week:

- Someone at the end of last week's lecture asked:
  - *'Does the line of code [below] dereference the pointer and **then** add 4?'*

```
printf("\nmyNewPTR +4 dereferenced = %d\n", *myNewPTR+4);
```

- You were correct! The line should have been:

```
printf("\nmyNewPTR +4 dereferenced = %d\n", *(myNewPTR+4));
```

- This will increment the pointer by 4 blocks of memory and **then** dereference it.

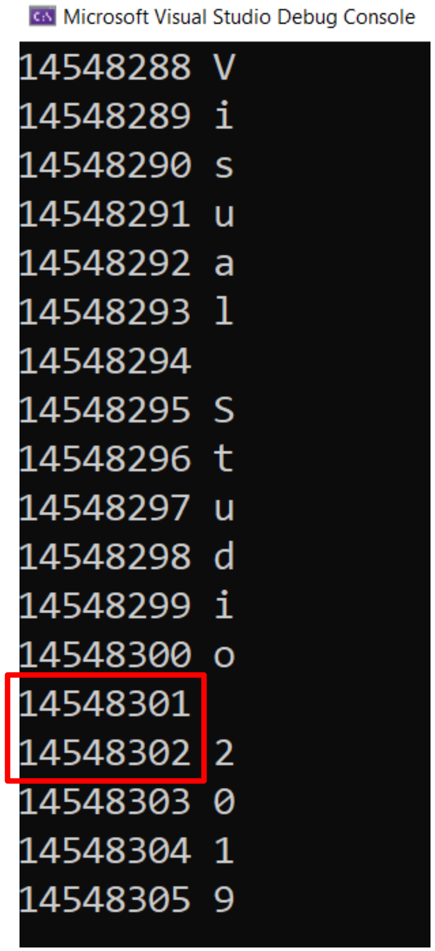
Lecture Content

- Today’s lecture (Week 17):
  - Memory requirements
  - Strcpy_s function
  - Structure pointers
  - Linking structures
  - Example C programme

MEMORY REQUIREMENTS

Characters

```
void main() {
     char string1[] = "Visual Studio 2019";
     int i = 0;

     while (string1[i] != '\0'){
          printf("%d %c\n", &string1[i], string1[i]);
          i++;
     }
}
```

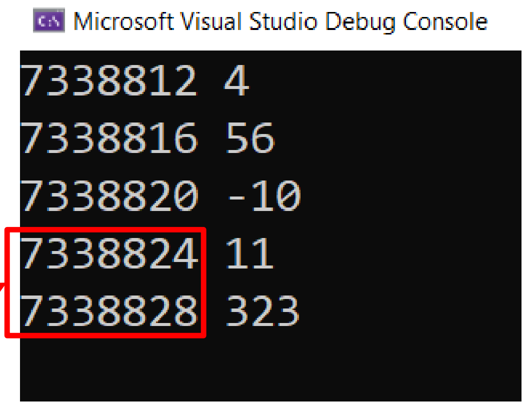


```
14548288 V
14548289 i
14548290 s
14548291 u
14548292 a
14548293 l
14548294
14548295 S
14548296 t
14548297 u
14548298 d
14548299 i
14548300 o
14548301
14548302 2
14548303 0
14548304 1
14548305 9
```

Difference is sizeof(char) = 1

Integers

```c
void main() {
    int array1[] = { 4,56,-10,11,323 };
    int i = 0;

    while (i<5)
    {
        printf("%d %d\n", &array1[i], array1[i]);
        i++;
    }
}
```

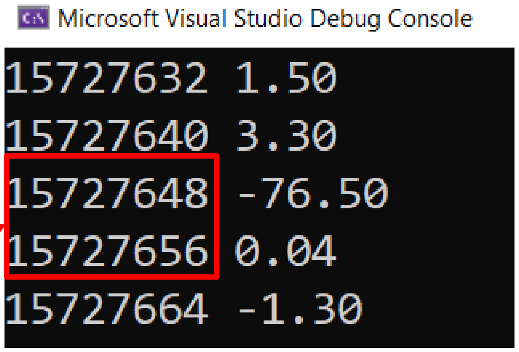


```
7338812 4
7338816 56
7338820 -10
7338824 11
7338828 323
```

Difference is sizeof(int) = 4

Doubles

```
void main() {
	double array1[] = { 1.5, 3.3, -76.5, 0.04, -1.3 };
	int i = 0;

	while (i < 5){
		printf("%d %.2lf\n", &array1[i], array1[i]);
		i++;
	}
}
```

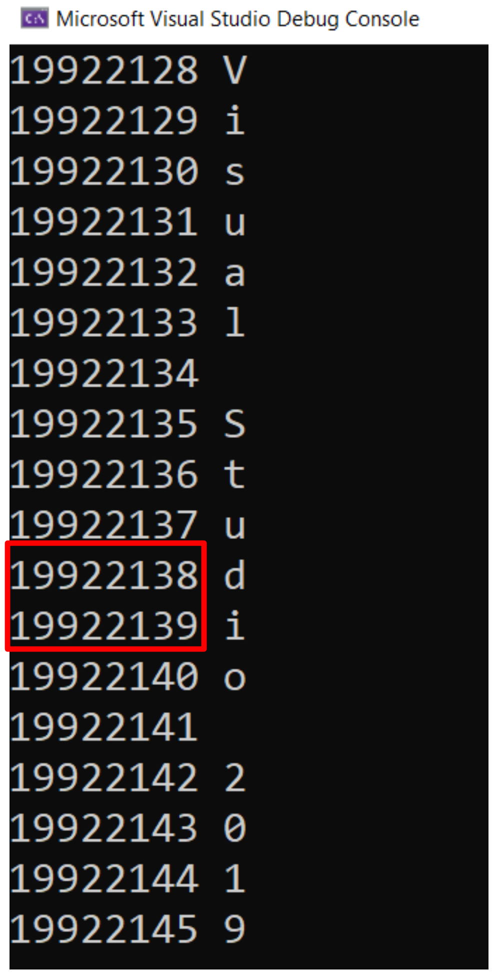


```
15727632 1.50
15727640 3.30
15727648 -76.50
15727656 0.04
15727664 -1.30
```

Difference is sizeof(double) = 8

Characters (Using Pointers)

```c
void main() {
	char string1[] = "Visual Studio 2019";
	char* cp = string1;
	int i = 0;

	while (*cp != '\0'){
		printf("%d %c\n", cp, *cp);
		cp++;
	}
}
```

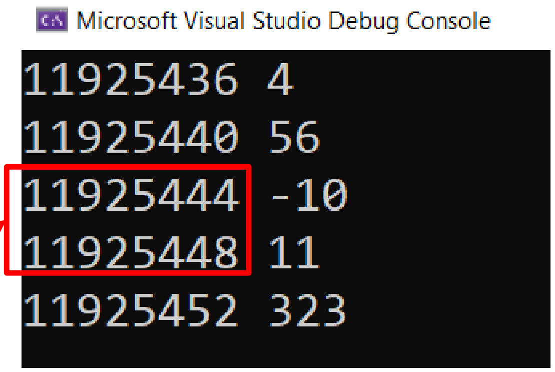

Microsoft Visual Studio Debug Console

```
19922128 V
19922129 i
19922130 s
19922131 u
19922132 a
19922133 l
19922134
19922135 S
19922136 t
19922137 u
19922138 d
19922139 i
19922140 o
19922141
19922142 2
19922143 0
19922144 1
19922145 9
```

Difference is sizeof(char) = 1

Integers (Using Pointers)

```c
void main() {
	int array1[] = { 4,56,-10,11,323 };
	int *ip = array1;
	int i = 0;

	while (i < 5){
		printf("%d %d\n", ip, *ip);
		ip++;
		i++;
	}
}
```

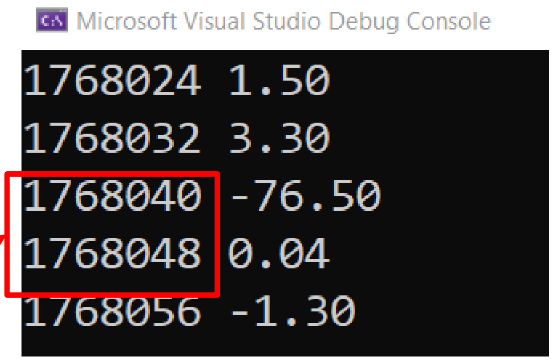

Microsoft Visual Studio Debug Console

```
11925436 4
11925440 56
11925444 -10
11925448 11
11925452 323
```

Difference is sizeof(int) = 4

Doubles (Using Pointers)

```c
void main() {
	double array1[] = { 1.5, 3.3, -76.5, 0.04, -1.3 };
	double* dp = array1;
	int i = 0;

	while (i < 5){
		printf("%d %.2lf\n", dp, *dp);
		dp++;
		i++;
	}
}
```



```
1768024 1.50
1768032 3.30
1768040 -76.50
1768048 0.04
1768056 -1.30
```

Difference is sizeof(double) = 8

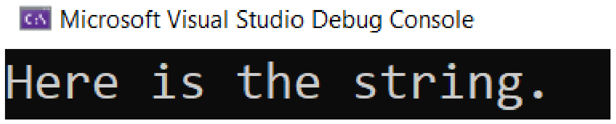
DIY STRCPY_S FUNCTION

Strcpy_s Function

- Remember strcpy_s() from when we learned about strings in semester 1.
- This function overwrites one string with another.
- Let’s see how we can write a function ourselves using pointers to do the same thing.

myStringCopy()

```c
void myStringCopy(char* target, char* source);

void main(){
     char t[100]; // target string
     char s[] = "Here is the string."; // to copy from
     myStringCopy(t, s);
     printf("%s",t);
}


void myStringCopy(char* target, char* source){
     while (*source != '\0'){
          *target = *source;
          target++;
          source++;
     }
     *target = '\0';
}
```


Microsoft Visual Studio Debug Console

```
Here is the string.
```

myStringCopy()

- How would we modify the new function we wrote to also return the array pointer?

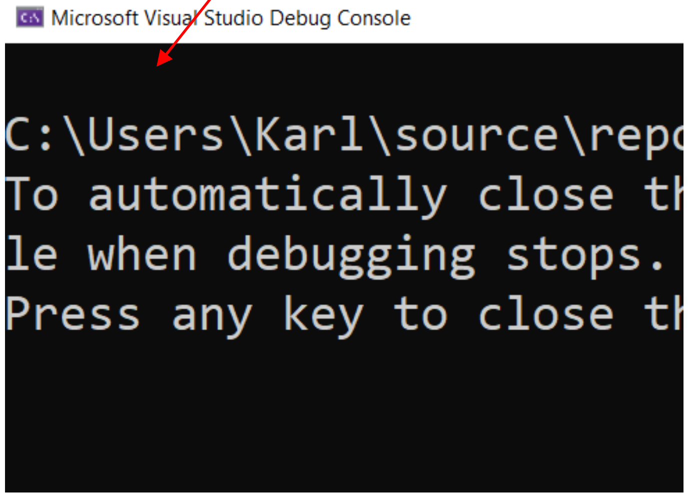
Will this work?

```
char* myStringCopy(char* target, char* source);

void main(){
     char t[100]; // target string
     char s[] = "Here is the string."; // to copy from
     char * tPtr = myStringCopy(t, s);
     printf("%s",tPtr);
}


char* myStringCopy(char* target, char* source){
     while (*source != '\0'){
          *target = *source;
          target++;
          source++;
     }
     *target = '\0';
     return target;
}
```

Will this work?

```
char* myStringCopy(char* target, char* source);

void main(){
     char t[100]; // target string
     char s[] = "Here is the string."; // to copy from
     char * tPtr = myStringCopy(t, s);
     printf("%s",tPtr);
}


char* myStringCopy(char* target, char* source){
     while (*source != '\0'){
          *target = *source;
          target++;
          source++;
     }
     *target = '\0';
     return target;
}
```

Unfortunately not.
Why?



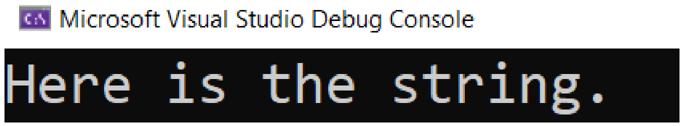
```

C:\Users\Karl\source\repo
To automatically close th
le when debugging stops.
Press any key to close th
```

myStringCopy()

- We need to return the pointer to the original address of the start of the target string
- But in myStringCopy we have kept incrementing this value as we copied characters
- So we need to fix this

It works

```
char* myStringCopy(char* target, char* source);

void main(){
     char t[100]; // target string
     char s[] = "Here is the string."; // to copy from
     char * tPtr = myStringCopy(t, s);
     printf("%s",tPtr);
}


char* myStringCopy(char* target, char* source){
     char* origTarget = target;
     while (*source != '\0'){
          *target = *source;
          target++;
          source++;
     }
     *target = '\0';
     return origTarget;
}
```



```
Here is the string.
```

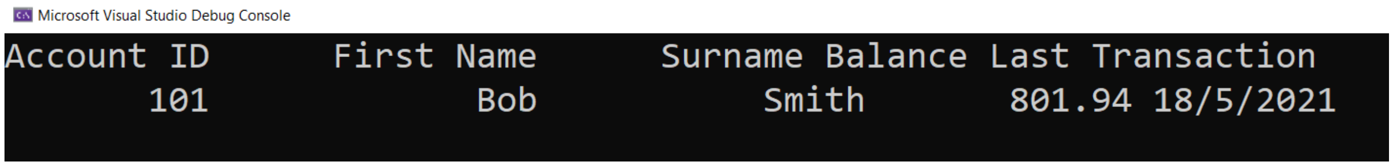
We kept the original pointer

We returned this instead

STRUCTURE POINTERS

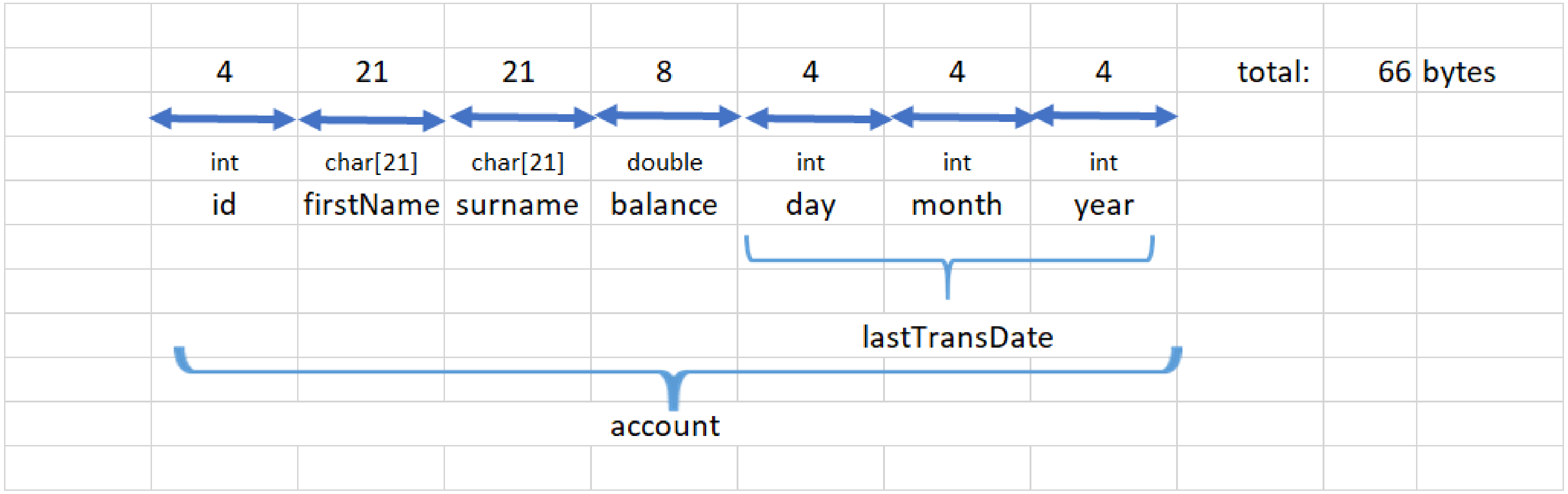
Structures

```
typedef struct{
     int day, month, year;
} date;

typedef struct{
      int id;
      char firstName[21];
      char surname[21];
      double balance;
      date lastTransDate;
} account;

void displayAccount(account acc1);

void main(){
     account myAccount = { 101, "Bob", "Smith", 801.94, {18,5,2021} };
     displayAccount(myAccount);
}

void displayAccount(account acc1){
     printf("Account ID\tFirst Name\tSurname\tBalance\tLast
     Transaction\n%10d\t%10s\t%10s\t%7.2lf\t%d/%d/%d  \n\n",
     acc1.id, acc1.firstName, acc1.surname, acc1.balance, acc1.lastTransDate.day,
     acc1.lastTransDate.month, acc1.lastTransDate.year);
}
```

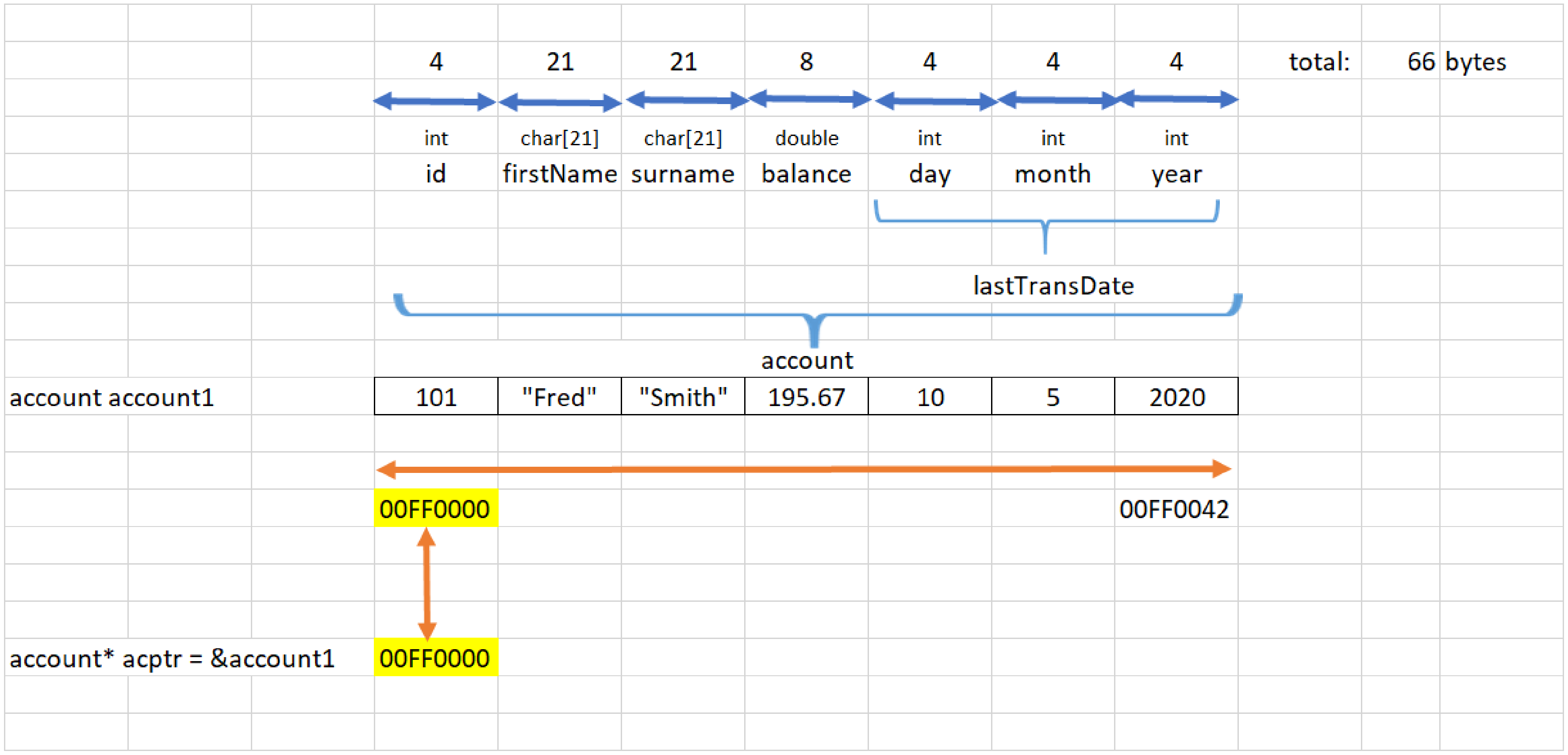


```
Account ID        First Name        Surname Balance Last Transaction
       101               Bob          Smith      801.94 18/5/2021
```

Account Structure Memory

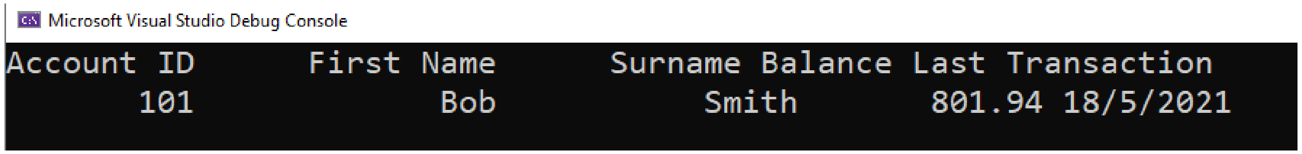


Structure Pointers



Structures – With Pointers

```c
typedef struct{
     int day, month, year;
} date;

typedef struct{
      int id;
      char firstName[21];
      char surname[21];
      double balance;
      date lastTransDate;
} account;

void displayAccount(account* accptr);

void main(){
     account myAccount = { 101, "Bob", "Smith", 801.94, {18,5,2021} };
     displayAccount(&myAccount);
}

void displayAccount(account* accptr){
     printf("Account ID\tFirst Name\tSurname\tBalance\tLast
     Transaction\n%10d\t%10s\t%10s\t%7.2lf\t%d/%d/%d  \n\n",
     (*accptr).id, (*accptr).firstName, (*accptr).surname, (*accptr).balance,
     (*accptr).lastTransDate.day, (*accptr).lastTransDate.month,
     (*accptr).lastTransDate.year);
}
```


Microsoft Visual Studio Debug Console

```
Account ID      First Name      Surname Balance Last Transaction
       101             Bob        Smith          801.94 18/5/2021
```

Differences Structure Example

1. We send the address of the structure to the function `displayAccount`
2. The function receives the value of the address which it uses to initialise the (local) variable `accptr`
3. To access the contents of a structure via a pointer, we dereference the pointer first.
4. `lastTransDate` is not a pointer, so we use the . to access it’s members.

Dereferencing Structure Pointers

- We could have used the ‘->’ symbol with our structure as follows:

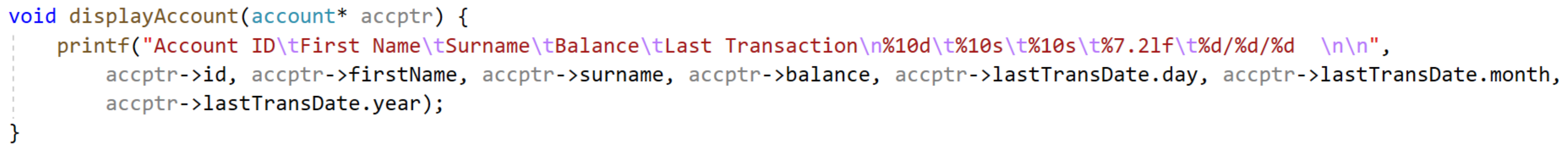
```c
void displayAccount(account* accptr) {
    printf("Account ID\tFirst Name\tSurname\tBalance\tLast Transaction\n%10d\t%10s\t%10s\t%7.2lf\t%d/%d/%d  \n\n",
        accptr->id, accptr->firstName, accptr->surname, accptr->balance, accptr->lastTransDate.day, accptr->lastTransDate.month,
        accptr->lastTransDate.year);
}
```

- This works the same as what we had originally:

```c
void displayAccount(account* accptr){
    printf("Account ID\tFirst Name\tSurname\tBalance\tLast Transaction\n%10d\t%10s\t%10s\t%7.2lf\t%d/%d/%d  \n\n",
        (*accptr).id, (*accptr).firstName, (*accptr).surname, (*accptr).balance, (*accptr).lastTransDate.day, (*accptr).lastTransDate.month,
        (*accptr).lastTransDate.year);
}
```

->

- We could have used the '->' symbol with our structure as follows:



```
void displayAccount(account* accptr) {
    printf("Account ID\tFirst Name\tSurname\tBalance\tLast Transaction\n%10d\t%10s\t%10s\t%7.2lf\t%d/%d/%d  \n\n",
        accptr->id, accptr->firstName, accptr->surname, accptr->balance, accptr->lastTransDate.day, accptr->lastTransDate.month,
        accptr->lastTransDate.year);
}
```

- -> will dereference the structure pointer and access the data member within the structure.

- E.g. **accptr-> id** is the same as **(*accptr).id**

C Code

- We will finish the first half of the lecture by running some C code in Visual Studio.

END OF FIRST LECTURE

START OF SECOND LECTURE

LINKING STRUCTURES

Linking Structures

- Pointing to structures is a powerful tool in C.
- When used with dynamic allocation, we can build chains of linked data structures of unlimited size on the fly.
- You will learn more about dynamic memory allocation later on in the course.
  - Briefly, it is the process of allocating memory during run time.

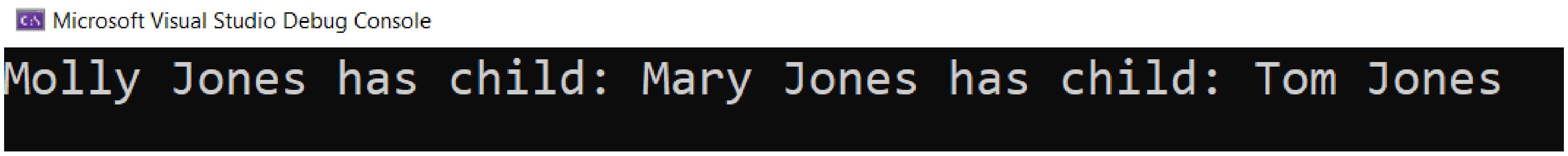
Structures – With Pointers

```
typedef struct{
    char name[100];
    struct person* child;
}person;

void display_person(person* personPointer);

void main(){
    person* personPtr;
    person p1 = { "Molly Jones", NULL }; // create a person
    person p2 = { "Mary Jones", NULL }; // create second person

    p1.child = &p2; // now first person ‘points’ to it’s child
    person p3 = { "Tom Jones", NULL }; // create third person
    p2.child = &p3; // now second person ‘points’ to it’s child
    display_person(&p1); // call function to print them out
}
```

Structures – With Pointers

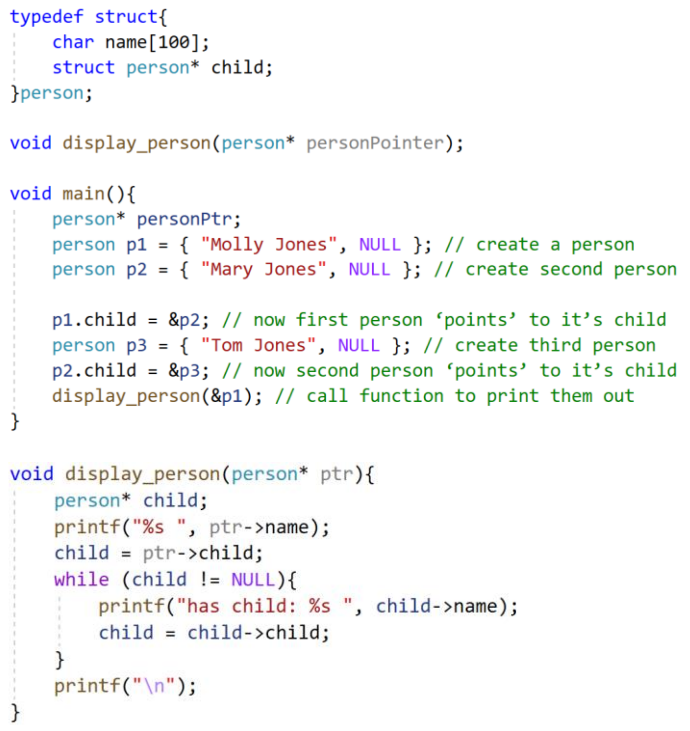
```
void display_person(person* ptr){
    person* child;
    printf("%s ", ptr->name);
    child = ptr->child;
    while (child != NULL){
        printf("has child: %s ", child->name);
        child = child->child;
    }
    printf("\n");
}
```

Structures – With Pointers



Microsoft Visual Studio Debug Console

```
Molly Jones has child: Mary Jones has child: Tom Jones
```



```c
typedef struct{
    char name[100];
    struct person* child;
}person;

void display_person(person* personPointer);

void main(){
    person* personPtr;
    person p1 = { "Molly Jones", NULL }; // create a person
    person p2 = { "Mary Jones", NULL }; // create second person

    p1.child = &p2; // now first person ‘points’ to it’s child
    person p3 = { "Tom Jones", NULL }; // create third person
    p2.child = &p3; // now second person ‘points’ to it’s child
    display_person(&p1); // call function to print them out
}

void display_person(person* ptr){
    person* child;
    printf("%s ", ptr->name);
    child = ptr->child;
    while (child != NULL){
        printf("has child: %s ", child->name);
        child = child->child;
    }
    printf("\n");
}
```

EXAMPLE PROBLEM

Example C Problem

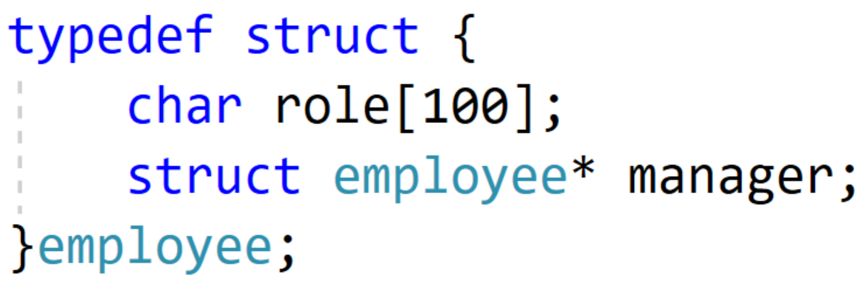
- You are writing software for a software company to organise the various employment positions at the company, i.e. the management structure.
- Create a structure for an **employee** type that holds the title of the position and a pointer to their manager.
- The company has 5 different levels of seniority:
  1. Junior developer.
  2. Senior developer.
  3. Lead developer.
  4. Director of software engineering.
  5. CEO.
- Write a function that will display the management structure of the company.

C Code

- Go to Visual Studio.

Problem solution

- Employee struct



```
typedef struct {
    char role[100];
    struct employee* manager;
}employee;
```

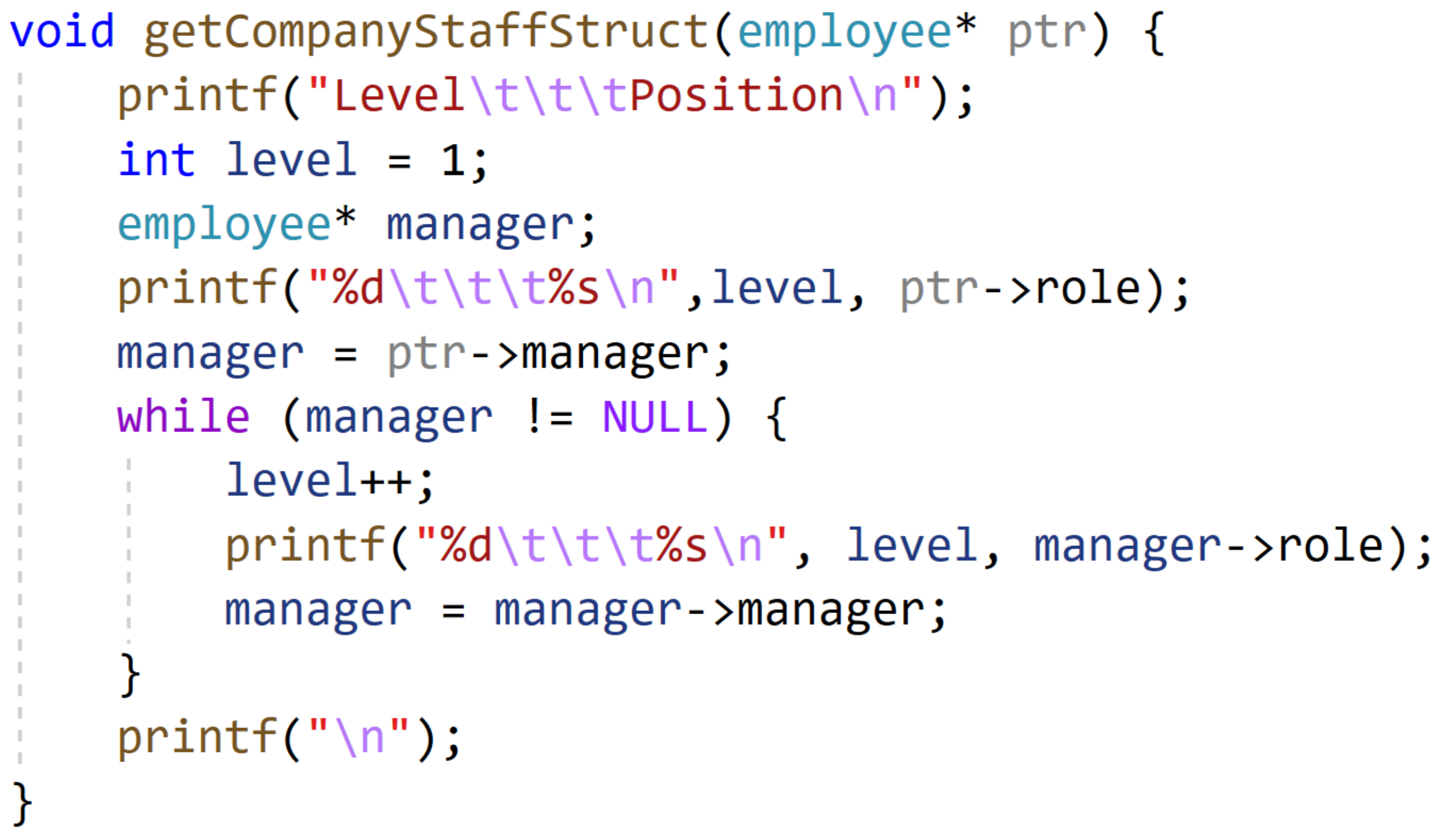
Problem solution

- Creating employees:

```
void main() {

    employee CEO = { "CEO", NULL }; // CEO
    employee direct = { "Director of Software Eng", &CEO }; // Director
    employee lead = { "Lead Developer", &direct }; // Lead Developer
    employee senior = { "Senior Developer", &lead }; // Senior Developer
    employee junior = { "Junior Developer", &senior }; // Junior Developer

    getCompanyStaffStruct(&junior); // call function to print them out
}
```

Problem solution

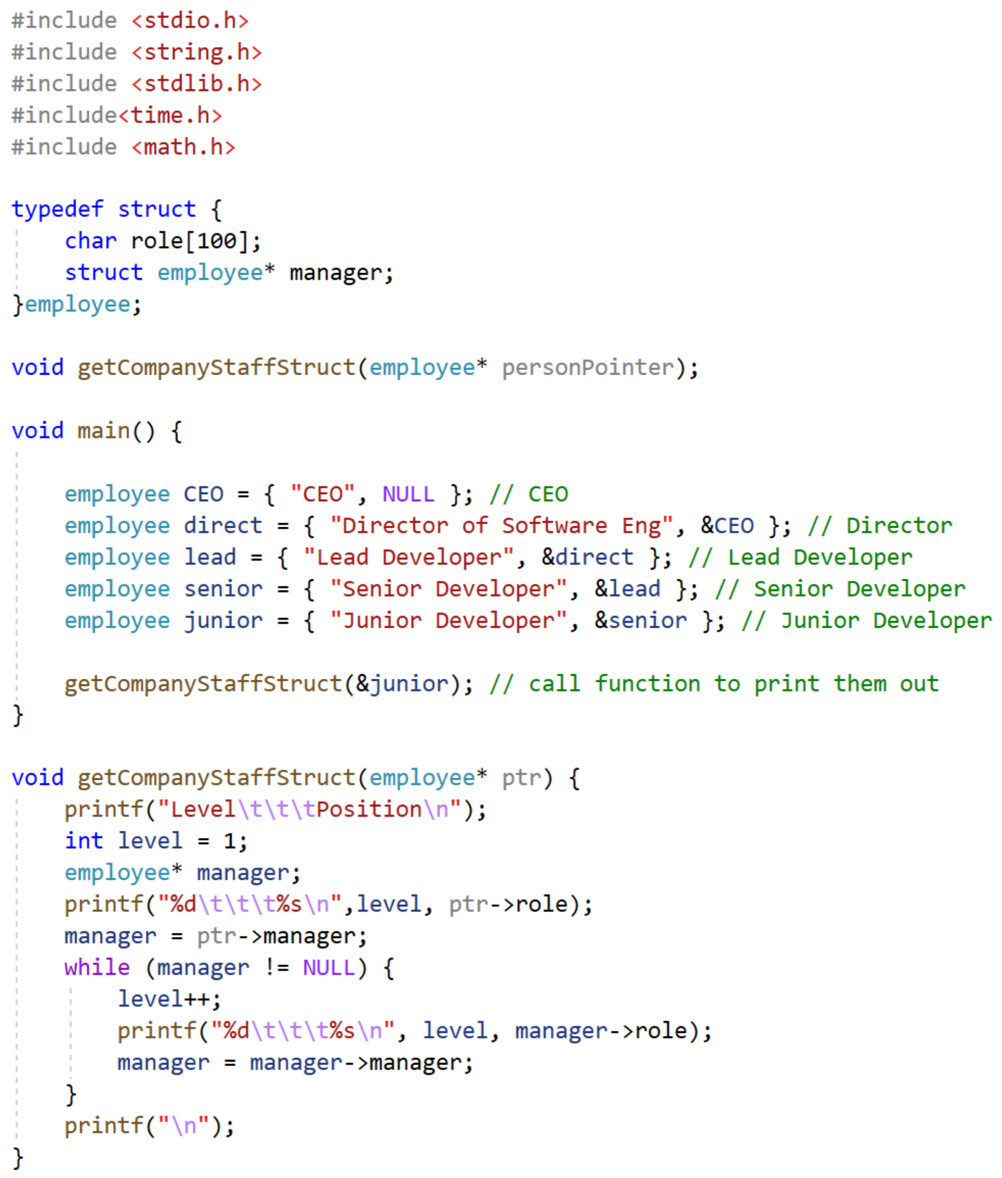
- Function to display structure:



```
void getCompanyStaffStruct(employee* ptr) {
    printf("Level\t\t\tPosition\n");
    int level = 1;
    employee* manager;
    printf("%d\t\t\t%s\n",level, ptr->role);
    manager = ptr->manager;
    while (manager != NULL) {
        level++;
        printf("%d\t\t\t%s\n", level, manager->role);
        manager = manager->manager;
    }
    printf("\n");
}
```

Problem solution

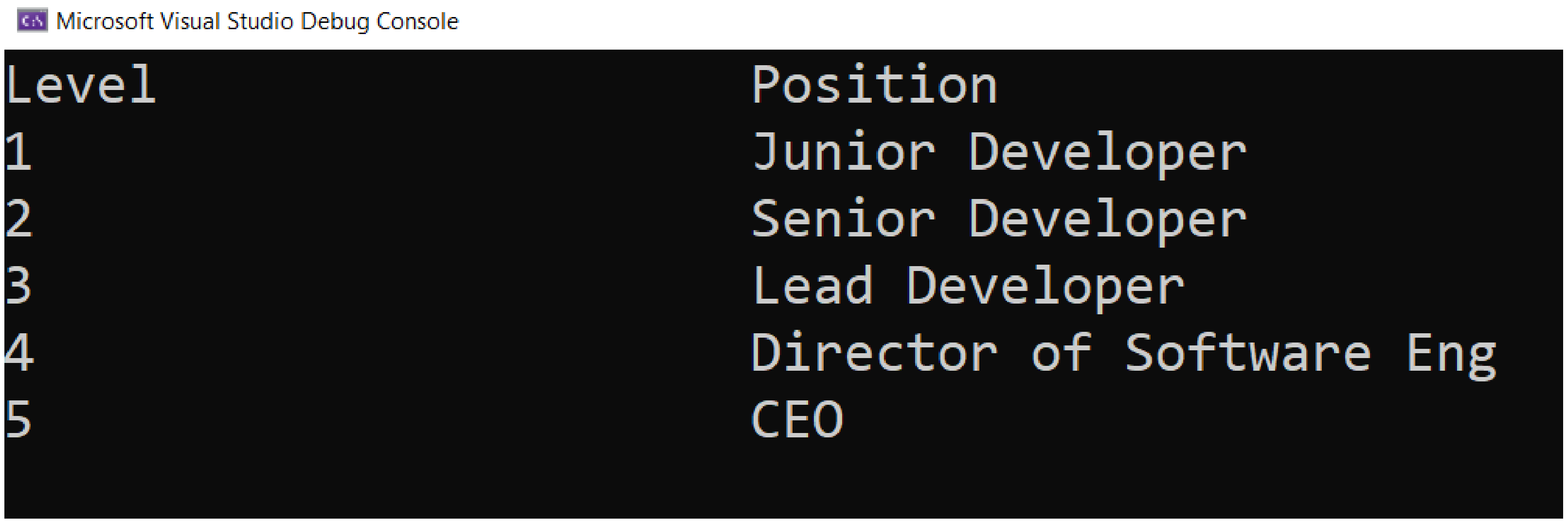
- Full solution



```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include<time.h>
#include <math.h>

typedef struct {
    char role[100];
    struct employee* manager;
}employee;

void getCompanyStaffStruct(employee* personPointer);

void main() {

    employee CEO = { "CEO", NULL }; // CEO
    employee direct = { "Director of Software Eng", &CEO }; // Director
    employee lead = { "Lead Developer", &direct }; // Lead Developer
    employee senior = { "Senior Developer", &lead }; // Senior Developer
    employee junior = { "Junior Developer", &senior }; // Junior Developer

    getCompanyStaffStruct(&junior); // call function to print them out
}

void getCompanyStaffStruct(employee* ptr) {
    printf("Level\t\t\tPosition\n");
    int level = 1;
    employee* manager;
    printf("%d\t\t\t%s\n",level, ptr->role);
    manager = ptr->manager;
    while (manager != NULL) {
        level++;
        printf("%d\t\t\t%s\n", level, manager->role);
        manager = manager->manager;
    }
    printf("\n");
}
```

Problem solution

- Code output:


Microsoft Visual Studio Debug Console

```
Level                    Position
1                        Junior Developer
2                        Senior Developer
3                        Lead Developer
4                        Director of Software Eng
5                        CEO
```

PROGRAMMING

CT103

Week 18

Sign in on Blackboard

- Please sign in on blackboard for CT103.
- I will leave the sign in option open for an hour after the lecture.

Lecture Content

- Today’s lecture (Week 18):
  - Strtok_s
  - Reading CSV Files
  - Strtok_s and CSV Files

STRTOK_S

Remember the problem of parsing a string or file record, where a field can contain more than 1 word?

- In this situation fscanf_s using %s is no use to us.
- For example you have a file like this:

Name	Age	Occupation	Birth Date
David Vose	46	Risk Analyst	13/10/1974
Mary Smith Burke	57	Software Engineer	01/09/1963

So the data line looks like:

- It depends on the delimiter used.
- If it is tab delimited, a line will look like this:

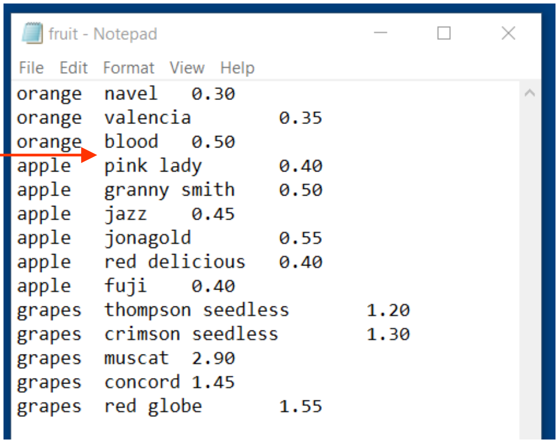
```
Mary Smith Burke\t57\tSoftware Engineer\t01/09/1963\n
```

So how to parse this?

- We have seen how you can copy from the line into temporary strings, stopping at the delimiter (\t in this case):
  - Repeating this then for each field.
  - Converting to doubles, ints, etc. as needed using functions like atoi().
- This definitely works, no problem.

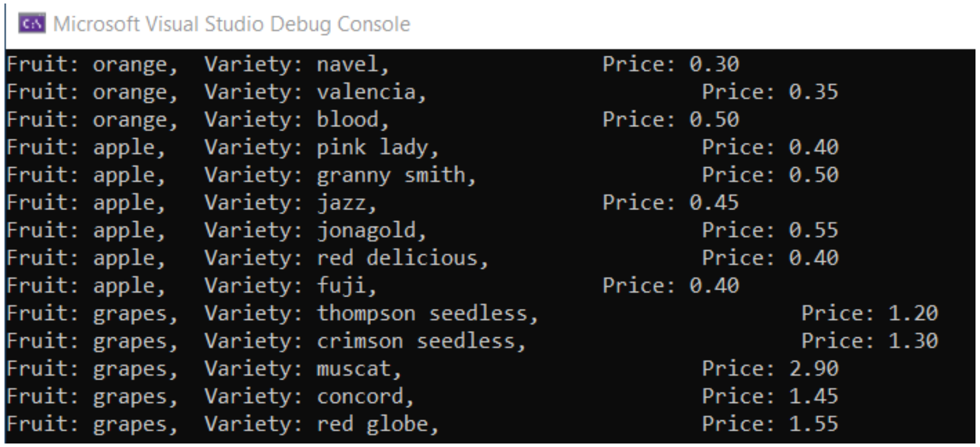
Example From Week 15

- Scanning strings with spaces.



```
orange	navel	0.30
orange	valencia	0.35
orange	blood	0.50
apple	pink lady	0.40
apple	granny smith	0.50
apple	jazz	0.45
apple	jonagold	0.55
apple	red delicious	0.40
apple	fuji	0.40
grapes	thompson seedless	1.20
grapes	crimson seedless	1.30
grapes	muscat	2.90
grapes	concord	1.45
grapes	red globe	1.55
```

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void main() {
    FILE* fptr;
    char line[200];
    char fruit[20], variety[20];
    double price;
    fopen_s(&fptr, "C:\\Users\\Karl\\Desktop\\fruit.txt", "r");

    if (fptr != NULL ) {
        int i = 0, j = 0;
        while (!feof(fptr)){
            fgets(line, 200, fptr);
            while (line[i] != '\t') {
                fruit[j] = line[i];
                i++;
                j++;
            }
            fruit[j] = '\0';
            i++;
            j = 0;

            while (line[i] != '\t') {
                variety[j] = line[i];
                i++;
                j++;
            }
            variety[j] = '\0';
            i++;
            price = atof(&line[i]);
            printf("Fruit: %s,\tVariety: %s,\t\t\tPrice: %.2lf\n", fruit, variety, price);
            i = 0, j=0;
        }
        fclose(fptr);
    }
}
```



```
Fruit: orange,  Variety: navel,                 Price: 0.30
Fruit: orange,  Variety: valencia,                      Price: 0.35
Fruit: orange,  Variety: blood,                 Price: 0.50
Fruit: apple,   Variety: pink lady,                     Price: 0.40
Fruit: apple,   Variety: granny smith,                  Price: 0.50
Fruit: apple,   Variety: jazz,                  Price: 0.45
Fruit: apple,   Variety: jonagold,                      Price: 0.55
Fruit: apple,   Variety: red delicious,                 Price: 0.40
Fruit: apple,   Variety: fuji,                  Price: 0.40
Fruit: grapes,  Variety: thompson seedless,                     Price: 1.20
Fruit: grapes,  Variety: crimson seedless,                      Price: 1.30
Fruit: grapes,  Variety: muscat,                        Price: 2.90
Fruit: grapes,  Variety: concord,                       Price: 1.45
Fruit: grapes,  Variety: red globe,                     Price: 1.55
```

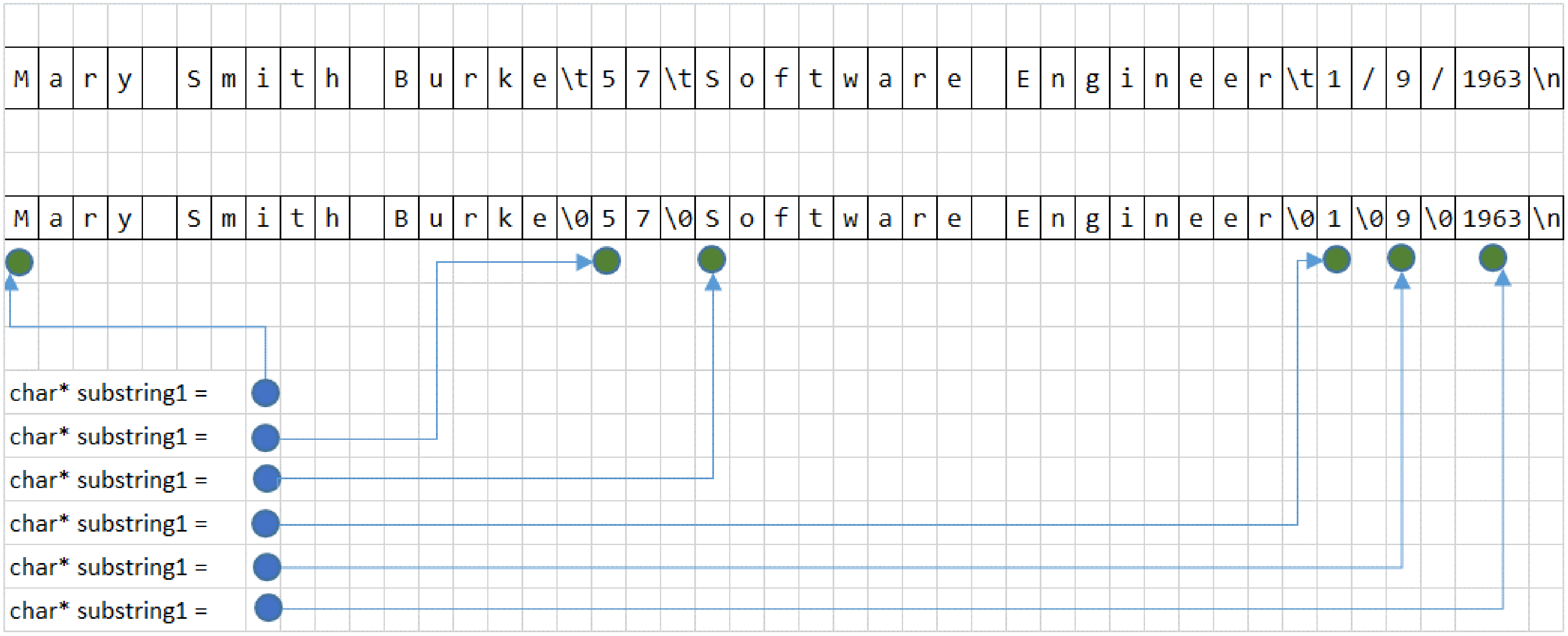
Another approach

- We could also replace the delimiters with ‘\0’ s
- Effectively splitting the source (line) string into sub-strings.

- All you need then is the starting point for each sub-string, which you can store in a char* pointer.

- We can use strtok() for this!

Replace delimiters with ‘\0’ and save pointers to start of substrings

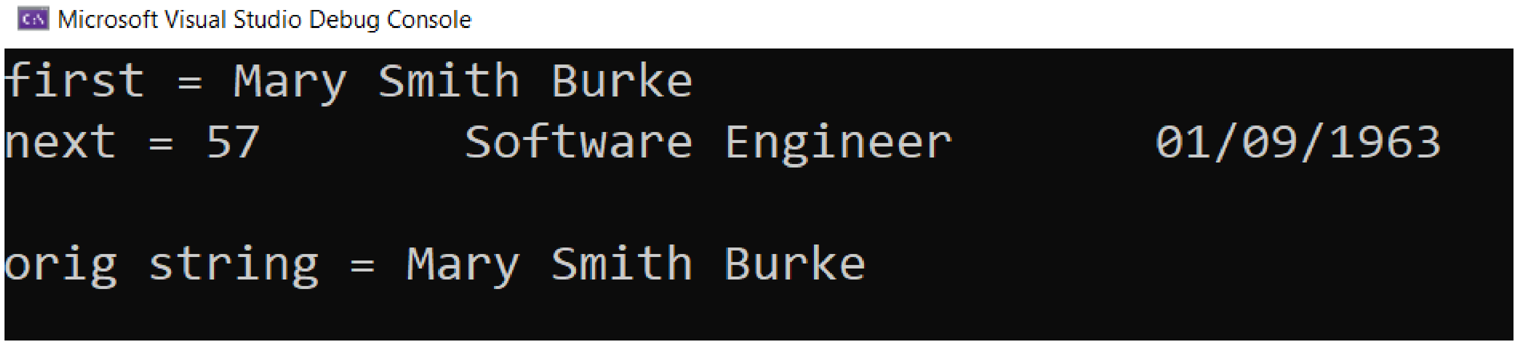

M a r y S m i t h B u r k e \t 5 7 \t S o f t w a r e E n g i n e e r \t 1 / 9 / 1963 \n
M a r y S m i t h B u r k e \0 5 7 \0 S o f t w a r e E n g i n e e r \0 1 \0 9 \0 1963 \n
char* substring1 =
char* substring1 =
char* substring1 =
char* substring1 =
char* substring1 =
char* substring1 =

STRTOK_S Example

```
void main() {
    char stringToParse[] = "Mary Smith Burke\t57\tSoftware Engineer\t01/09/1963\n";
    char delims[] = "\t";
    char* next = NULL;
    char* first = strtok_s(stringToParse, delims, &next);
    printf("first = %s\n", first);
    printf("next = %s\n", next);
    printf("orig string = %s\n", stringToParse);
}
```

STRTOK_S Example

```c
void main() {
    char stringToParse[] = "Mary Smith Burke\t57\tSoftware Engineer\t01/09/1963\n";
    char delims[] = "\t";
    char* next = NULL;
    char* first = strtok_s(stringToParse, delims, &next);
    printf("first = %s\n", first);
    printf("next = %s\n", next);
    printf("orig string = %s\n", stringToParse);
}
```

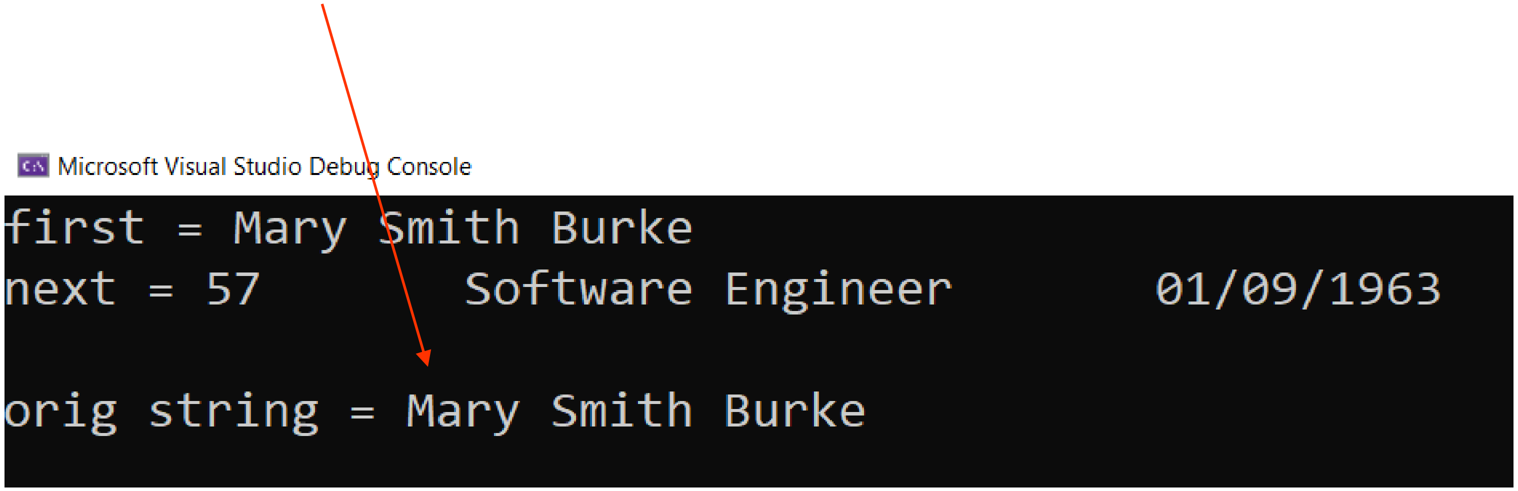


Microsoft Visual Studio Debug Console

```
first = Mary Smith Burke
next = 57       Software Engineer       01/09/1963

orig string = Mary Smith Burke
```

Be Careful with Strtok_s

- Strtok_s is destructive


Microsoft Visual Studio Debug Console

```
first = Mary Smith Burke
next = 57        Software Engineer      01/09/1963

orig string = Mary Smith Burke
```

Next substring

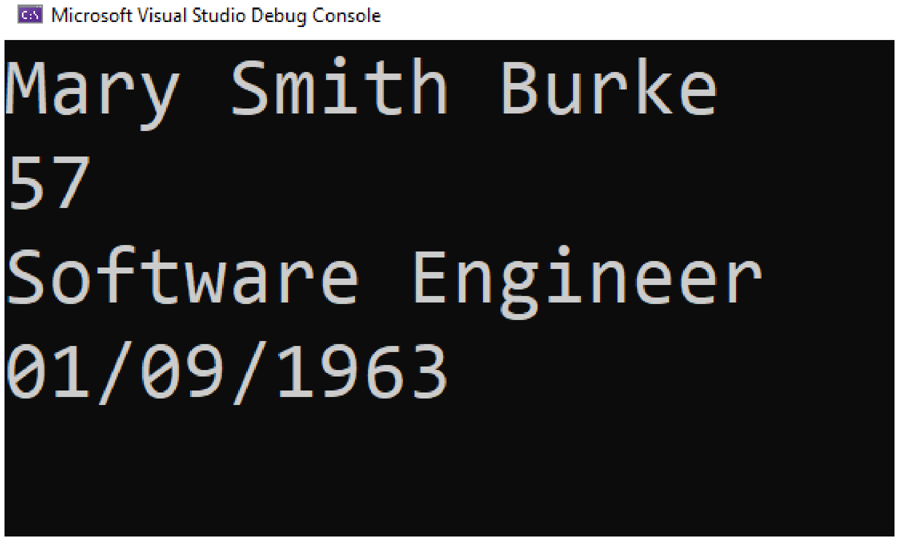
- Next time you call strtok_s, it will replace the second delimiter with ‘\0’ and return the pointer to the second field.
- However you must now pass in NULL as the first argument instead of a string:
  - This is how it knows you are still parsing the same string.

Subsequent Strings

```
void main() {
    char stringToParse[] = "Mary Smith Burke\t57\tSoftware Engineer\t01/09/1963\n";
    char delims[] = "\t";
    char* next = NULL;
    char* first = strtok_s(stringToParse, delims, &next);

    while (first != NULL) {
        printf("%s\n", first);
        first = strtok_s(NULL, delims, &next);
    }
}
```

Subsequent Strings

```c
void main() {
    char stringToParse[] = "Mary Smith Burke\t57\tSoftware Engineer\t01/09/1963\n";
    char delims[] = "\t";
    char* next = NULL;
    char* first = strtok_s(stringToParse, delims, &next);

    while (first != NULL) {
        printf("%s\n", first);
        first = strtok_s(NULL, delims, &next);
    }
}
```


Microsoft Visual Studio Debug Console

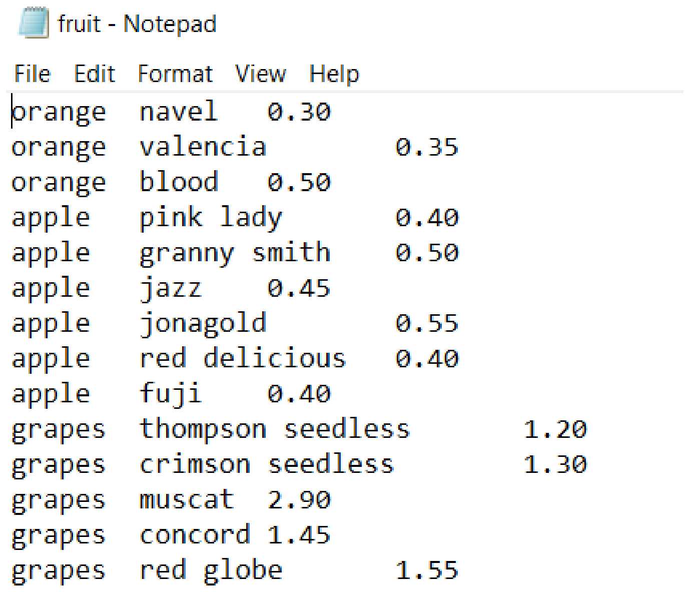
```
Mary Smith Burke
57
Software Engineer
01/09/1963
```

CSV FILES

Scanning from file

- Up until now, we have only considered reading data from .txt files.

- E.g.

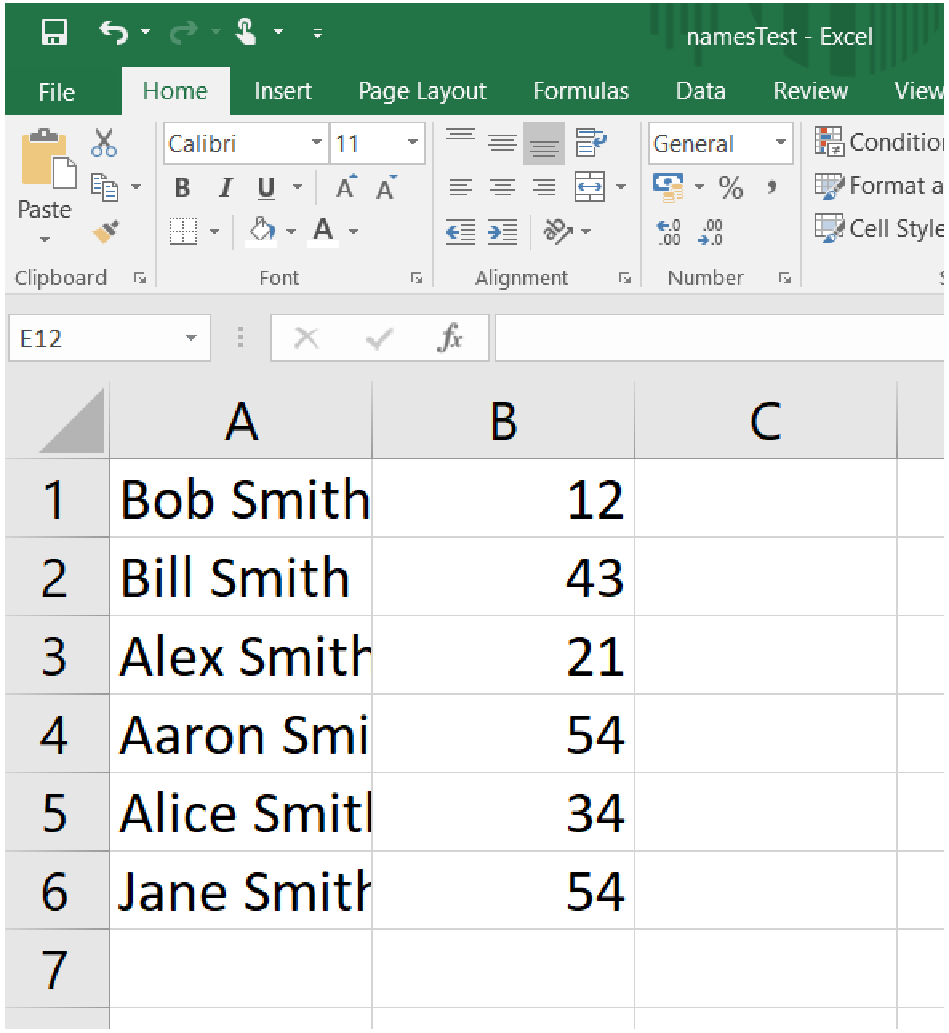


```
orange	navel	0.30
orange	valencia	0.35
orange	blood	0.50
apple	pink lady	0.40
apple	granny smith	0.50
apple	jazz	0.45
apple	jonagold	0.55
apple	red delicious	0.40
apple	fuji	0.40
grapes	thompson seedless	1.20
grapes	crimson seedless	1.30
grapes	muscat	2.90
grapes	concord	1.45
grapes	red globe	1.55
```

Scanning from file

- What if our data is in another file format?
- For example: CSV files
  - CSV = Comma Separated Value

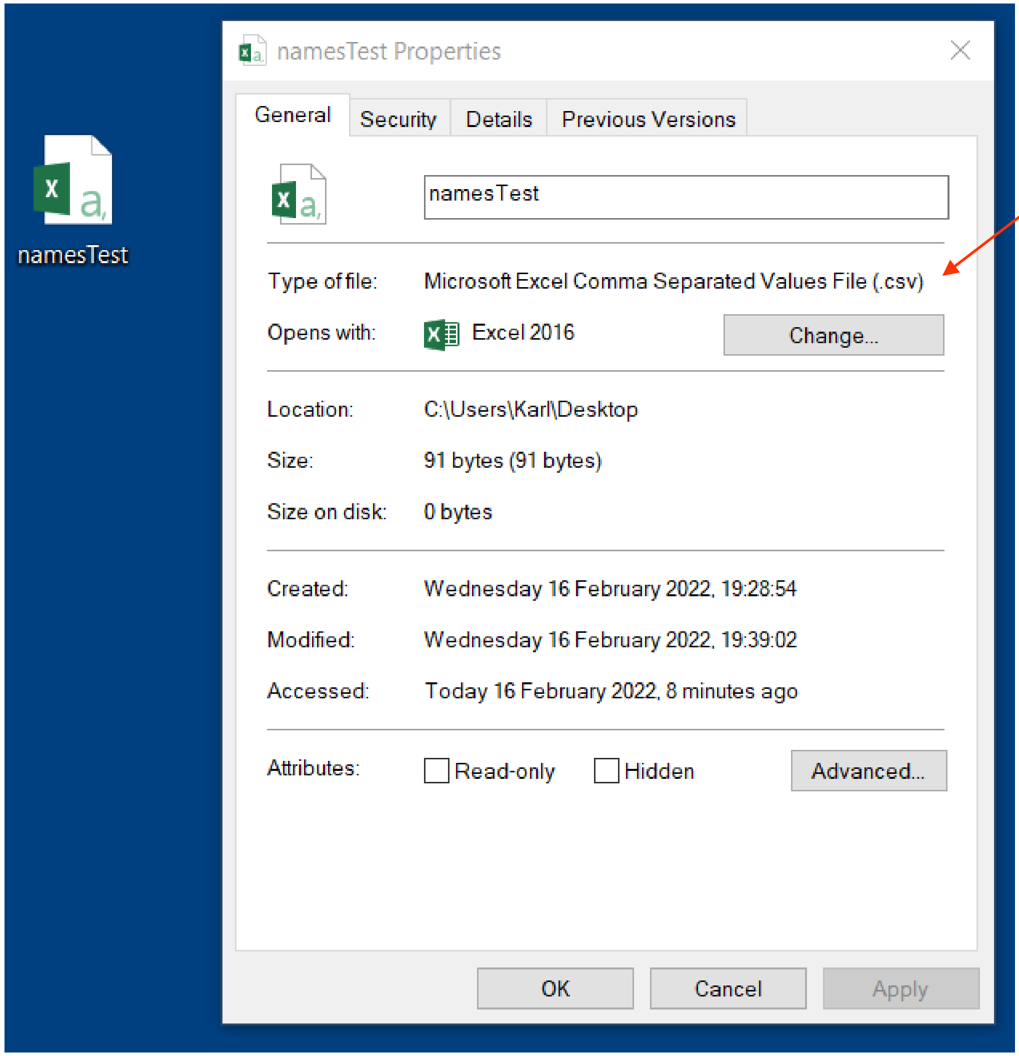
CSV Files



	A	B	C
1	Bob Smith	12	
2	Bill Smith	43	
3	Alex Smith	21	
4	Aaron Smi	54	
5	Alice Smitl	34	
6	Jane Smith	54	
7			

CSV Files

Has .csv file extension



namesTest

Type of file: Microsoft Excel Comma Separated Values File (.csv)

Opens with: Excel 2016

Location: C:\Users\Karl\Desktop

Size: 91 bytes (91 bytes)

Size on disk: 0 bytes

Created: Wednesday 16 February 2022, 19:28:54

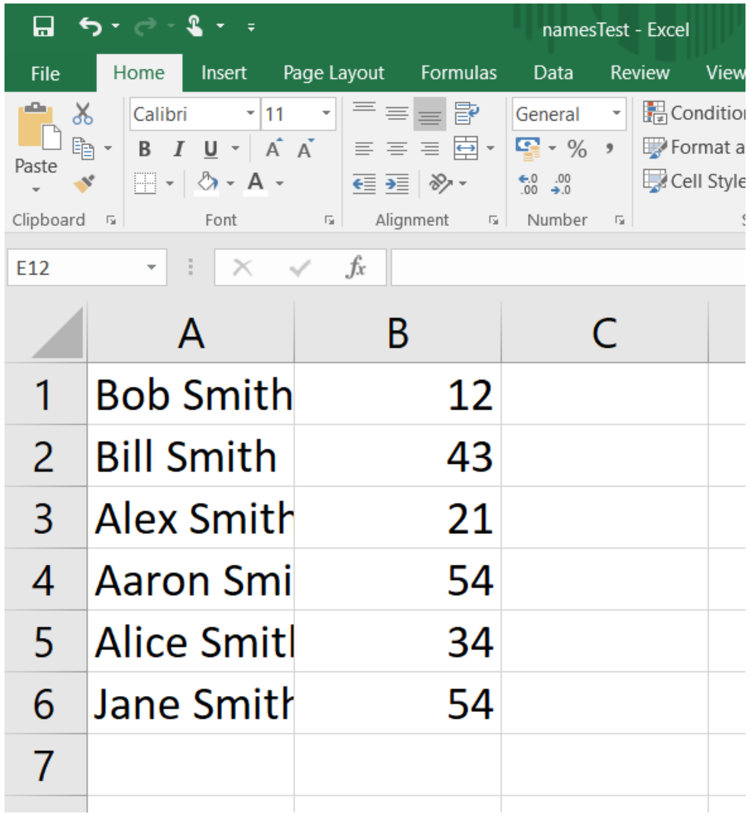
Modified: Wednesday 16 February 2022, 19:39:02

Accessed: Today 16 February 2022, 8 minutes ago

Attributes: ☐ Read-only ☐ Hidden

Scanning CSV files

- How can we read in CSV files?



	A	B	C
1	Bob Smith	12	
2	Bill Smith	43	
3	Alex Smith	21	
4	Aaron Smi	54	
5	Alice Smitl	34	
6	Jane Smith	54	
7			

Scanning CSV files

New file extension

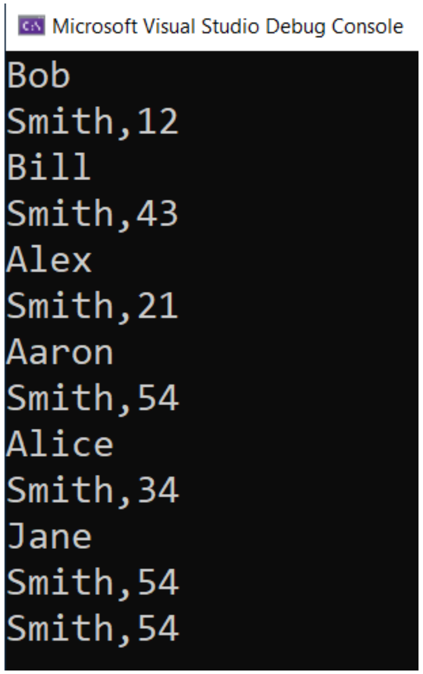
```
void main() {
    FILE* fptr;

    char myfilePath[] = "C:\\Users\\Karl\\Desktop\\namesTest.csv";
    char line[200];
    fopen_s(&fptr, myfilePath, "r");

    if (fptr != NULL) {
        while (!feof(fptr)) {
            fscanf_s(fptr, "%s,", line, 20);
            printf("%s\n", line);
        }
        fclose(fptr);
    }
}
```

Much the same as before!

Scanning CSV files

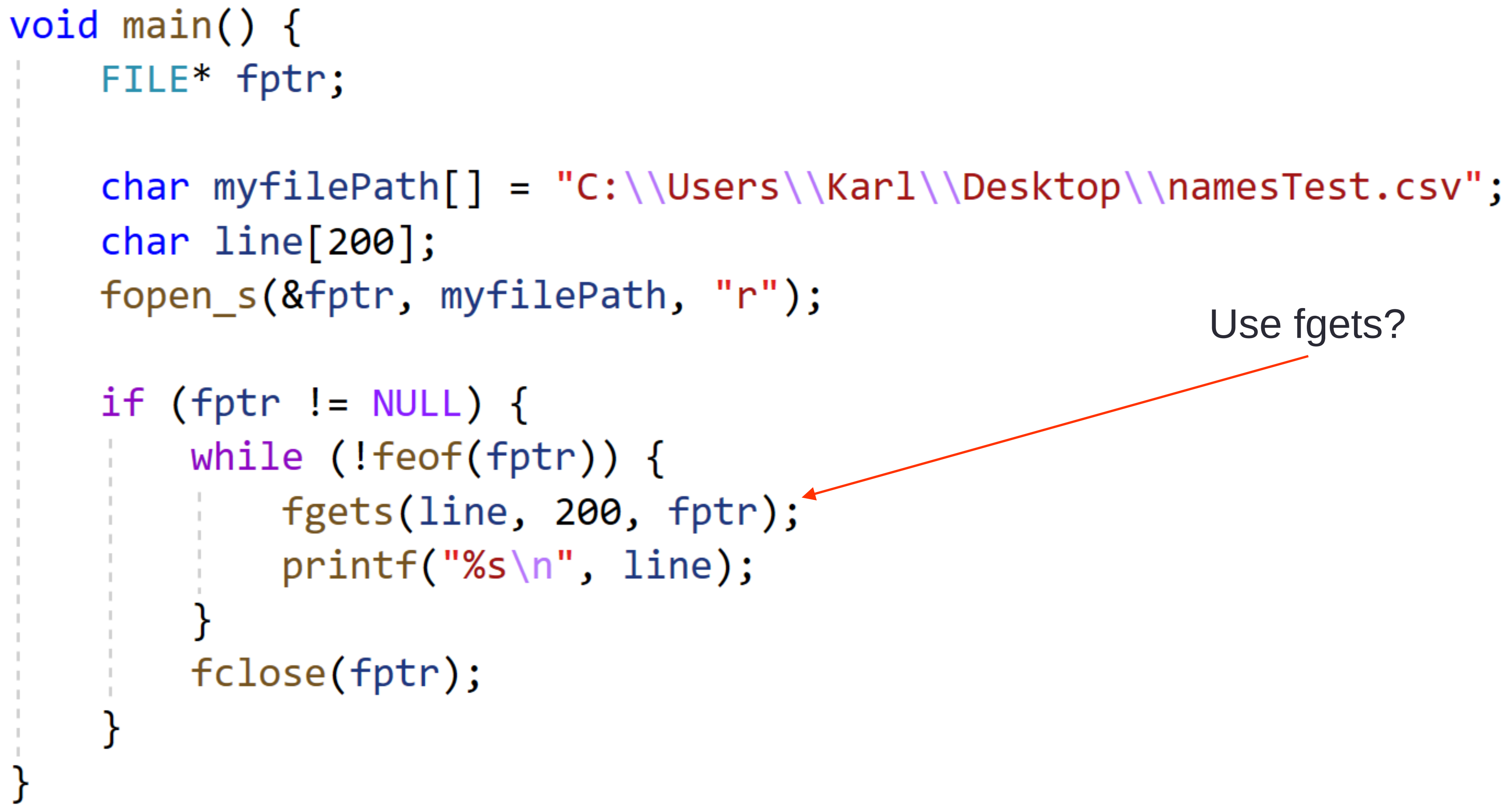
Sort of works…

```
void main() {
    FILE* fptr;

    char myfilePath[] = "C:\\Users\\Karl\\Desktop\\namesTest.csv";
    char line[200];
    fopen_s(&fptr, myfilePath, "r");

    if (fptr != NULL) {
        while (!feof(fptr)) {
            fscanf_s(fptr, "%s,", line, 20);
            printf("%s\n", line);
        }
        fclose(fptr);
    }
}
```


Microsoft Visual Studio Debug Console

```
Bob
Smith,12
Bill
Smith,43
Alex
Smith,21
Aaron
Smith,54
Alice
Smith,34
Jane
Smith,54
Smith,54
```

How about this?



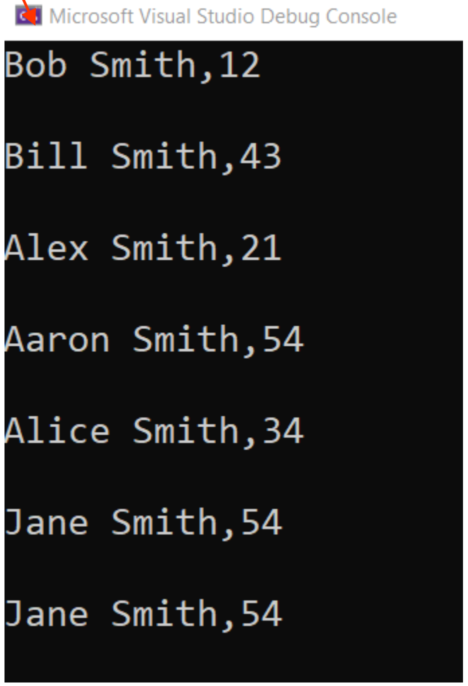
```c
void main() {
    FILE* fptr;

    char myfilePath[] = "C:\\Users\\Karl\\Desktop\\namesTest.csv";
    char line[200];
    fopen_s(&fptr, myfilePath, "r");

    if (fptr != NULL) {
        while (!feof(fptr)) {
            fgets(line, 200, fptr);
            printf("%s\n", line);
        }
        fclose(fptr);
    }
}
```

Use fgets?

How about this?

```c
void main() {
    FILE* fptr;

    char myfilePath[] = "C:\\Users\\Karl\\Desktop\\namesTest.csv";
    char line[200];
    fopen_s(&fptr, myfilePath, "r");

    if (fptr != NULL) {
        while (!feof(fptr)) {
            fgets(line, 200, fptr);
            printf("%s\n", line);
        }
        fclose(fptr);
    }
}
```

Works great!

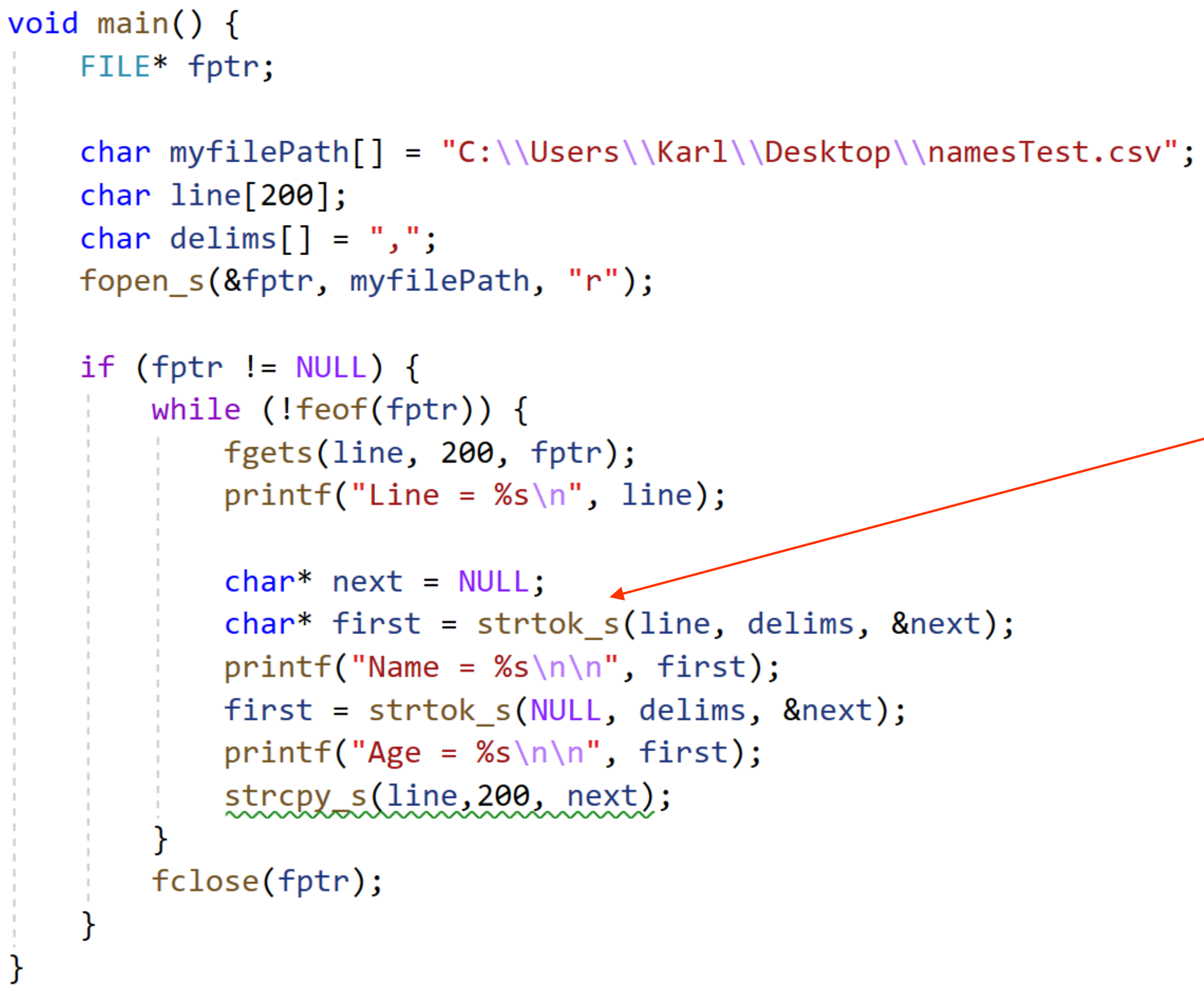


```
Bob Smith,12

Bill Smith,43

Alex Smith,21

Aaron Smith,54

Alice Smith,34

Jane Smith,54

Jane Smith,54
```

Scan Names Only

- Can we use strtok_s to get just the names from our csv file?

Scan Names Only



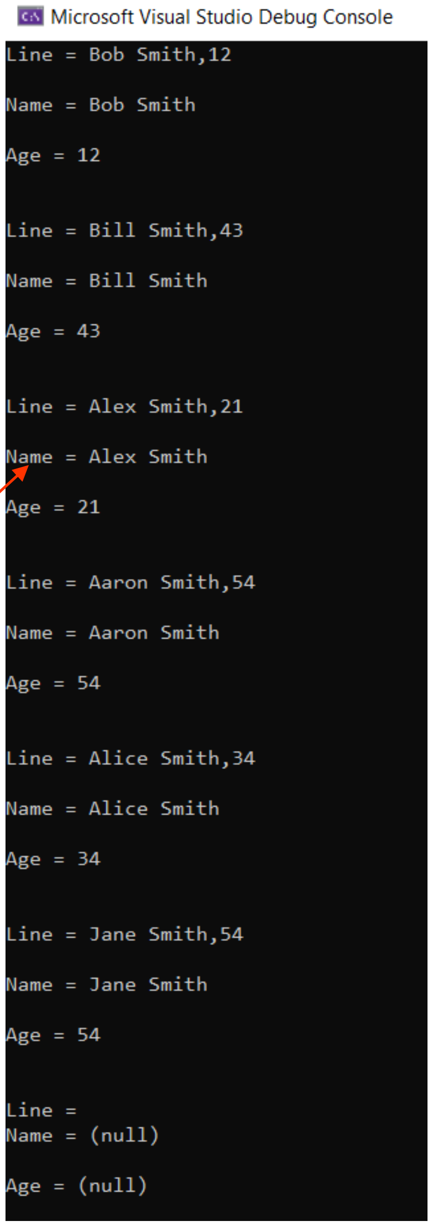
```
void main() {
    FILE* fptr;

    char myfilePath[] = "C:\\Users\\Karl\\Desktop\\namesTest.csv";
    char line[200];
    char delims[] = ",";
    fopen_s(&fptr, myfilePath, "r");

    if (fptr != NULL) {
        while (!feof(fptr)) {
            fgets(line, 200, fptr);
            printf("Line = %s\n", line);

            char* next = NULL;
            char* first = strtok_s(line, delims, &next);
            printf("Name = %s\n\n", first);
            first = strtok_s(NULL, delims, &next);
            printf("Age = %s\n\n", first);
            strcpy_s(line,200, next);
        }
        fclose(fptr);
    }
}
```

Will this work?

Scan Names Only

```
void main() {
    FILE* fptr;

    char myfilePath[] = "C:\\Users\\Karl\\Desktop\\namesTest.csv";
    char line[200];
    char delims[] = ",";
    fopen_s(&fptr, myfilePath, "r");

    if (fptr != NULL) {
        while (!feof(fptr)) {
            fgets(line, 200, fptr);
            printf("Line = %s\n", line);

            char* next = NULL;
            char* first = strtok_s(line, delims, &next);
            printf("Name = %s\n\n", first);
            first = strtok_s(NULL, delims, &next);
            printf("Age = %s\n\n", first);
            strcpy_s(line,200, next);
        }
        fclose(fptr);
    }
}
```

We can easily separate names and ages



Microsoft Visual Studio Debug Console

```
Line = Bob Smith,12

Name = Bob Smith

Age = 12


Line = Bill Smith,43

Name = Bill Smith

Age = 43


Line = Alex Smith,21

Name = Alex Smith

Age = 21


Line = Aaron Smith,54

Name = Aaron Smith

Age = 54


Line = Alice Smith,34

Name = Alice Smith

Age = 34


Line = Jane Smith,54

Name = Jane Smith

Age = 54


Line =
Name = (null)

Age = (null)
```

C CODE

C Code

- Let’s finish today’s lecture by running some C programs in Visual Studio.

CT103 Programming

Semester 2 Week 7

Revision: Loops, Arrays, File Handling

Dr. Sam Redfern

Discord Server: the same one we're using for CT1114

Exercise

- Write a C program which asks the user for a positive integer, and then (*using a loop*) determines and displays the minimum number of terms needed in the summation
- 1 + 3 + 5 + 7 + 9 + …
- for the sum to exceed the user's integer.
- Questions:
  - What's the best kind of loop to use here?
  - What work do we want each loop iteration to do?
  - What variables do we need?

Arrays

Allows a program to store multiple variables under the same name

A first example of a *data structure* or *collection*

Declaration:

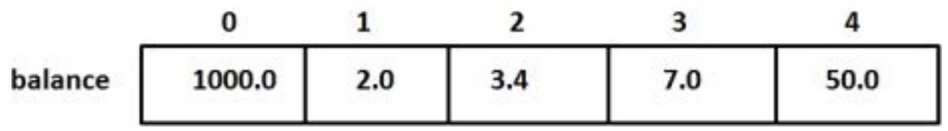
```
type arrayName [ arraySize ];
```

e.g.:

```
double balance[5];
```

Declare and initialising at the same time:

```
double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};
```



Access members of an array (for read or write):

```
double b = balance[0];
double b = balance[i];
```

Arrays are *random access* data structures: you can access any element at any time without needing to access preceding elements first. This is very useful.

Example

07_CountLetters.cpp

- Counting upper case letters
- Random access FTW!
- Also makes clever use of char-as-int in the C language (in C, char variables store ASCII values as 1-byte integers)

```
fsdfsdfJKHHJKHJHJHJJKHBHKJ

   A:  0   B:  1   C:  0   D:  0   E:  0   F:  0
   G:  0   H:  7   I:  0   J:  7   K:  4   L:  0
   M:  0   N:  0   O:  0   P:  0   Q:  0   R:  0
   S:  0   T:  0   U:  0   V:  0   W:  0   X:  0
   Y:  0   Z:  0
```

```c
#include <stdio.h>
#include <ctype.h>

int main(void) {
    int c, i, letter[26];

    for (i = 0; i < 26; ++i)
      letter[i] = 0;

    while ((c = getchar()) != '\n') {
      if (isupper(c))
        letter[c - 'A']++;
    }

    for (i = 0; i < 26; i++) {
      if (i % 6 == 0)
        printf("\n");
      printf("%4c:%3d", 'A' + i, letter[i]);
    }
    printf("\n\n");

    return 0;
}
```

Generating Random Numbers in a range

```
/* Generate random numbers within a range. */
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int rnd(int lower, int upper);

int main(void) {
    srand(time(0));
    printf("Here's some random numbers between 1 and 10:\n");
    for (int i=0; i<20; i++)
        printf("%d ", rnd(1,10));
}

int rnd(int lower, int upper) {
    int range = (upper-lower)+1;
    return (rand()%range) + lower;
}
```

Exercise (Arrays)

- Write a program which generates an array of 10000 random integers, each in the range 0-100
- The program should then identify which of the numbers occurred most often
- How might we do this…? What data do we need?
  - We need an array to store the frequencies of each possible value (0-100)
  - The frequency array is therefore of size 101
  - Loop for each of the 10000 random numbers
    - .. For each number, increment the frequency array at the appropriate index
  - Another loop, after the first one, for each of the values 0-100
    - Check each to see if it's the most frequent (pattern: find the largest value in a list of values).

Nested Loops

- Example: times tables

```
   1   2   3   4   5   6   7   8   9  10  11  12

   1   2   3   4   5   6   7   8   9  10  11  12
   2   4   6   8  10  12  14  16  18  20  22  24
   3   6   9  12  15  18  21  24  27  30  33  36
   4   8  12  16  20  24  28  32  36  40  44  48
   5  10  15  20  25  30  35  40  45  50  55  60
   6  12  18  24  30  36  42  48  54  60  66  72
   7  14  21  28  35  42  49  56  63  70  77  84
   8  16  24  32  40  48  56  64  72  80  88  96
   9  18  27  36  45  54  63  72  81  90  99 108
  10  20  30  40  50  60  70  80  90 100 110 120
```

```c
/* Display times tables using nested loops */

#include <stdio.h>

int main(void) {
   for (int y=1; y<=12; y++)
      printf("%4d", y);
   printf("\n");

   for (int x=1; x<=10; x++) {
      printf("\n");
      for (int y=1; y<=12; y++) {
         printf("%4d", x*y);
      }
   }
   printf("\n");

   return 0;
}
```

Exercise (Nested Loops)

```
Please enter a positive integer: 6
*
**
***
****
*****
******

     *
    **
   ***
  ****
 *****
******
```

```
Please enter a positive integer: 9
*
**
***
****
*****
******
*******
********
*********

        *
       **
      ***
     ****
    *****
   ******
  *******
 ********
*********
```

Exercise: Number guessing game

- Write a program which generates a random number between 1 and 100
- The player has to guess the number in as few guesses as possible
- After each guess, the program tells them if they're too high or too low
- After the correct guess, they're told how many guesses it took.
- Questions:
  - What's the best kind of loop to use here?
  - What work do we want each loop iteration to do?
  - What variables do we need?

Reading from File

- A text file is a collection of ASCII characters
- Text files also contain the newline character - signifying the end of a line
- A simple way to read a file line-by-line is to use fgets()
- See next slide

```c
#include <stdio.h>

#define MAXSTRING 100

int main() {
    // fopen requests a file to be opened obtains a FILE pointer to access it
    FILE *file_ptr;
    file_ptr = fopen("dictionary.txt", "r"); // open for reading

    if (file_ptr == NULL)
        printf("Could not open dictionary.txt");
    else {
        char txt[MAXSTRING]; // string for reading each line into
        int lines = 0;
        while (fgets(txt, MAXSTRING-1, file_ptr)!=NULL) {
                lines++;
        }
        printf("dictionary.txt contained %d lines.", lines);
        fclose (file_ptr); // don't forget to close the file
    }

    return 0;
}
```

Exercise: Read a file and display each line which contains an 's' in it

- (I will supply the file – it's a dictionary)
- Question: how do we find out whether a string contains 's' ?

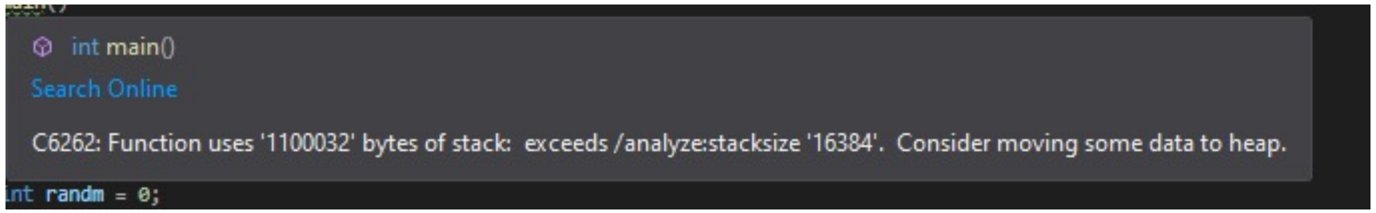
Lab Assignment: Word guessing game

- Write a C program which reads the supplied dictionary file into an array of strings (make sure the array is big enough for all the words.. 100000 is plenty)
- The program should reject words from the file which have less than 4 or more then 7 letters
- It should then randomly pick a word and the user must guess letters in the word and try to get the whole word in as few guesses as possible (a politically incorrect person might call the game 'Hangman')
- Make appropriate use of functions
- For required output, see next slide

```
Loaded 30409 suitable words from the dictionary.

Guess 1.
------
Guess a letter >i

Guess 2.
------
Guess a letter >a

Guess 3.
a-----
Guess a letter >e

Guess 4.
a--e--
Guess a letter >m

Guess 5.
a--e--
Guess a letter >t

Guess 6.
atte-t
Guess a letter >s

Well done, that took you 6 guesses to find attest!
```

Stack and Heap considerations..

- Declaring a very large array inside a function could cause the program to fail with an error, e.g.
    - char dictionary[100000][50];



```
C6262: Function uses '1100032' bytes of stack:  exceeds /analyze:stacksize '16384'.  Consider moving some data to heap.
int randm = 0;
```

- It depends on your compiler whether this happens
- The issue is that some compilers put a severe limit on the allowable memory used by the **Stack**
- If you move your very large array outside of the function, making it global, that should fix it because now it will be allocated on the **Heap**
- The Stack is for short-term working memory within a function, the Heap is for longer-term memory that your program needs to allocate for longer
- See:
    - https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2015/code-quality/c6262?view=vs-2015&redirectedfrom=MSDN
    - https://www.guru99.com/stack-vs-heap.html

CT103 Programming

Semester 2 Week 8

More String Handling, File Handling, and Binary Search

Dr. Sam Redfern

Discord Server (the same one we're using for CT1114)

Separating words in a string (with some pangrams)

- "The quick brown fox jumps over the lazy dog"
- "How vexingly quick daft zebras'... jumpers... jump!"
- "Pack my box... with five   dozen, liquor jugs"
- "Jack's ' dawn loves...my big\t\t sphinx, , of quartz."

- What rules are needed to robustly separate words in all these cases?
  - Alphabetic letters
  - Apostrophes
  - White-space? Dots? Tabs? Commas? Exclamations?
- When we have identified the rules, what's the general process for displaying each word on a separate line?

An approach (*example*)

- Declare a string to use to build the current word into, e.g.: `char word[50];`
- Iterate through each letter in the user-supplied string
    - Check the letter using `isalpha()` from `<ctype.h>`
        - If it's alphabetic, add it to the end of `word[]`
            - How can we 'add to the end' of a string in C?
        - If it's not alphabetic, then print out `word[]` on a new line (if it has anything in) and reset `word[]` to empty
        - Special treatment of apostrophes? They may be internal or external to a word

- Sample solution: 08_WordsToLines.cpp

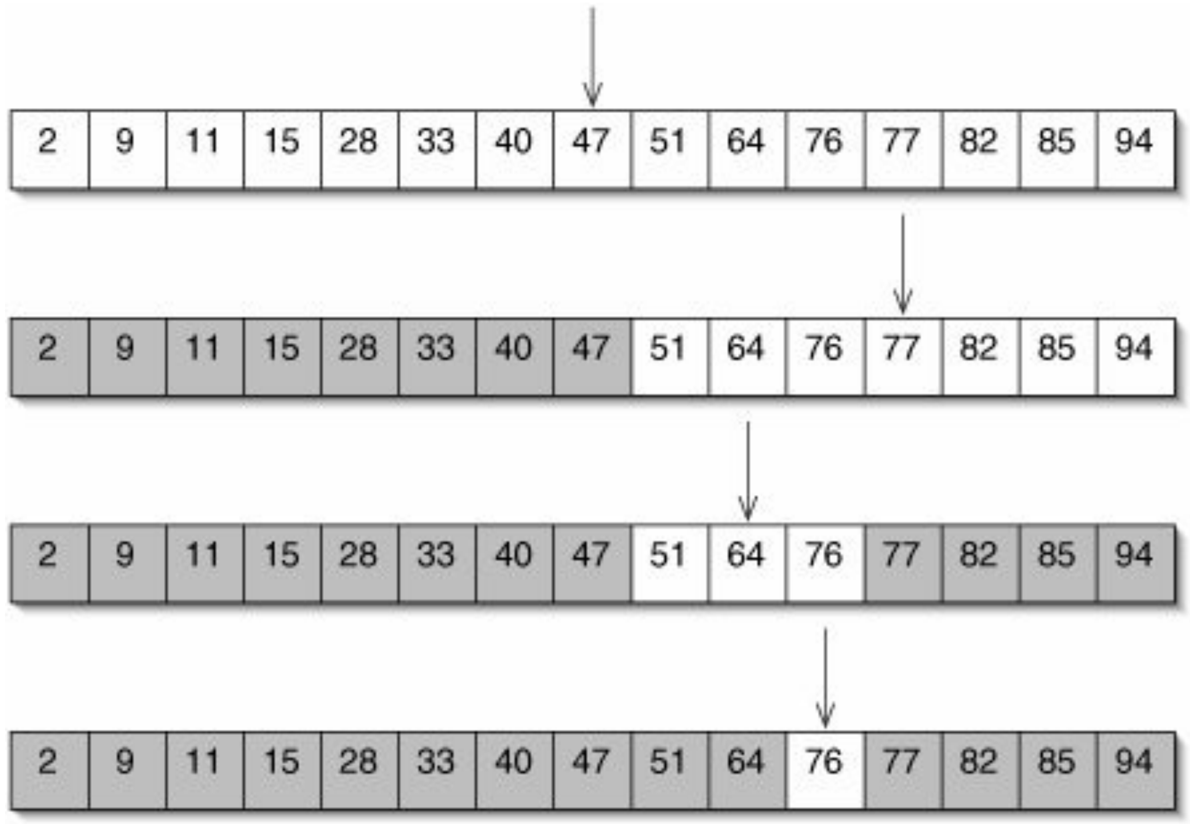
Spellchecking (*exercise*)

- We can use the `dictionary.txt` file again
- Load in dictionary.txt as we did in the hangman game
- Ask the user for a word
- Is the word in the dictionary?
  - Consider Lowercase vs. Uppercase letters?
  - Use tolower() on from <ctype.h> on each character in turn

- How to search the dictionary.. linear search?
- 08_Spellcheck_Linear.cpp

Binary Search

- If our data is stored in sorted sequence, we can use Binary Search rather than Linear Search
- Two variables are needed to record the region of search space still to be considered:
  - Low
  - High
- Each iteration involves checking the Middle item in the remaining search space, with 4 possibilities:
  - Too Low?
  - Too High?
  - Item Found?
  - Failed?
- Each iteration halves the remaining size of the search space
- Solution to previous exercise, now done this way: 08_Spellcheck_Binary.cpp



`strtok()` - split strings into tokens

- A function for splitting strings, where you supply the delimiter which separates your tokens
- Similar to the `.split()` method of strings in Javascript
- *#include <string.h>*

```
char* strtok (char* str, const char* delimiters);
```

- A sequence of calls to this function splits *str* into tokens, which are sequences of contiguous characters separated by any of the characters that are part of *delimiters*.

  On a first call, the function expects a C string as argument for *str*, whose first character is used as the starting location to scan for tokens. In subsequent calls, the function expects a null pointer and uses the position right after the end of the last token as the new starting location for scanning.

- See: http://www.cplusplus.com/reference/cstring/strtok/
- https://www.tutorialspoint.com/c_standard_library/c_function_strtok.htm

strtok() example

```
#include <string.h>
#include <stdio.h>

int main () {
  char str[80] = "This is - www.tutorialspoint.com - website";
  const char s[2] = "-";
  char *token;

  /* get the first token */
  token = strtok(str, s);
  /* loop through other tokens */
  while( token != NULL ) {
     printf( " %s\n", token );
     token = strtok(NULL, s);
  }
}
```

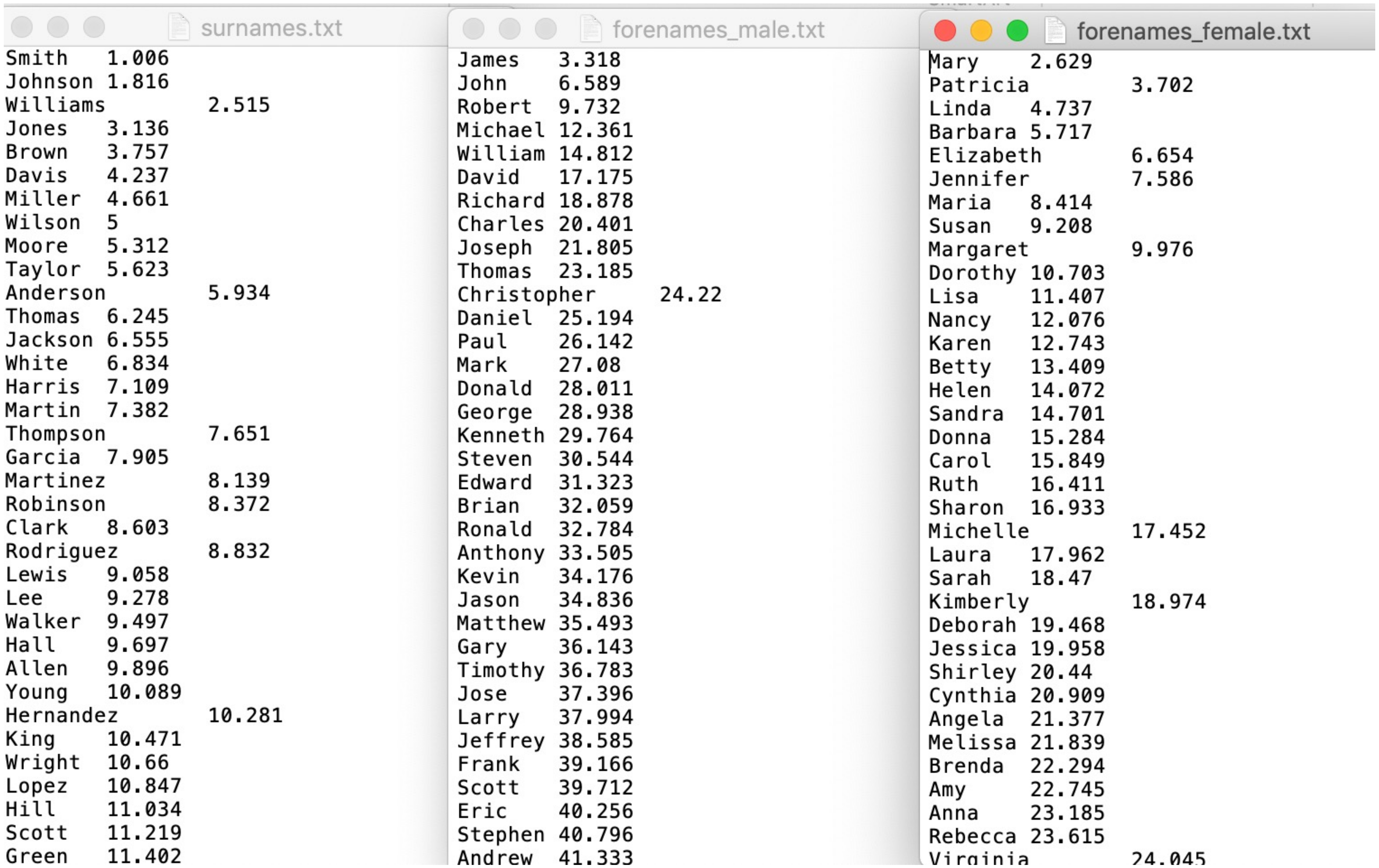
Exercise: Some more fun with files

- 3 files, with the 1000 most common surnames, male forenames, and female forenames from a US county census (see next slide):
    - surnames.txt
    - forenames_male.txt
    - forenames_female.txt
- See sample data from them on the next slide: not just names, but names and cumulative frequencies
- What character is being used as delimiter here?

- **Exercise: write a program to:**
    - open surnames.txt
    - read its contents into an array of strings
    - Remove the cumulative frequency from each name by using strtok
    - Print out **one** of the surnames, selected randomly from the array

My sample solution (08_random_names.cpp) will:

- read all 3 files in line by line, separating the name from the frequency number, and storing them into 3 arrays of strings
- generate 10 random male names and 10 random female names. It doesn't worry about the frequencies yet (other than removing them upon reading).. We'll explore their use next week.



```
Smith   1.006
Johnson 1.816
Williams        2.515
Jones   3.136
Brown   3.757
Davis   4.237
Miller  4.661
Wilson  5
Moore   5.312
Taylor  5.623
Anderson        5.934
Thomas  6.245
Jackson 6.555
White   6.834
Harris  7.109
Martin  7.382
Thompson        7.651
Garcia  7.905
Martinez        8.139
Robinson        8.372
Clark   8.603
Rodriguez       8.832
Lewis   9.058
Lee     9.278
Walker  9.497
Hall    9.697
Allen   9.896
Young   10.089
Hernandez       10.281
King    10.471
Wright  10.66
Lopez   10.847
Hill    11.034
Scott   11.219
Green   11.402
```

```
James   3.318
John    6.589
Robert  9.732
Michael 12.361
William 14.812
David   17.175
Richard 18.878
Charles 20.401
Joseph  21.805
Thomas  23.185
Christopher     24.22
Daniel  25.194
Paul    26.142
Mark    27.08
Donald  28.011
George  28.938
Kenneth 29.764
Steven  30.544
Edward  31.323
Brian   32.059
Ronald  32.784
Anthony 33.505
Kevin   34.176
Jason   34.836
Matthew 35.493
Gary    36.143
Timothy 36.783
Jose    37.396
Larry   37.994
Jeffrey 38.585
Frank   39.166
Scott   39.712
Eric    40.256
Stephen 40.796
Andrew  41.333
```

```
Mary    2.629
Patricia        3.702
Linda   4.737
Barbara 5.717
Elizabeth       6.654
Jennifer        7.586
Maria   8.414
Susan   9.208
Margaret        9.976
Dorothy 10.703
Lisa    11.407
Nancy   12.076
Karen   12.743
Betty   13.409
Helen   14.072
Sandra  14.701
Donna   15.284
Carol   15.849
Ruth    16.411
Sharon  16.933
Michelle        17.452
Laura   17.962
Sarah   18.47
Kimberly        18.974
Deborah 19.468
Jessica 19.958
Shirley 20.44
Cynthia 20.909
Angela  21.377
Melissa 21.839
Brenda  22.294
Amy     22.745
Anna    23.185
Rebecca 23.615
Virginia        24.045
```

Zachary Dartaghan is as common as Mary Smith??

- What is the cumulative frequency data and how we can use it..?
- We’ll see a solution next week, using arrays of structs to keep the names and their frequency data neatly together

Graded Assignment: Flesch Readability Index

- Write a program which reads all the text in a file and computes the *Flesch Readability Index* for it.
- The *Flesch Readability Index* was invented as a simple tool for determining the legibility of a document without linguistic analysis. It may be implemented using the following 4 steps:

1. Count all words. A *word* is any sequence of characters delimited by white space.
2. Count all syllables in each word. Each *group* of adjacent vowels (a, e, i, o, u, y) counts as one syllable (for example, the "ea" in "real" contributes one syllable, but the "e..a" in "regal" counts as two syllables). However, an "e" at the end of a word doesn't count as a syllable. Also, each word has at least one syllable, even if the previous rules give a count of 0.
3. Count all sentences. A sentence is ended by a full stop, colon, semicolon, question mark, or exclamation mark.
4. The index is computed by the following formula:

$$Index = 206.835 - 84.6 * \frac{syllables}{words} - 1.015 * \frac{words}{sentences}$$

Starting code: 08_LAB_START_Flesch.cpp

What are the steps for solving this?

- Separate words
- Then…?

Starting code: 08_LAB_START_Flesch.cpp

CT103 Programming

Semester 2 Week 9

Practice with Structs and Arrays (and Pointers, and Files).

Dr. Sam Redfern

Discord Server (the same one we're using for CT1114)

Recall from last week: names with cumulative frequencies

- Zachary Dartaghan is as common as Mary Smith??
- What is the cumulative frequency data and how we can use it..?
- Example: updating the previous solution to use frequencies, and produce statistically believable sets of names:
  - 09_random_names_with_freq.cpp
- My solution uses linear search.. could it use binary search?

Structs for soccer teams and games

```c
typedef struct {
  char name[30];
  int won, lost, drew;
  int goalsFor, goalsAgainst;
} team;

typedef struct {
  char team1name[30];
  char team2name[30];
  int goals1, goals2;
} game;
```

soccer_teams.txt (newline-delimited strings)

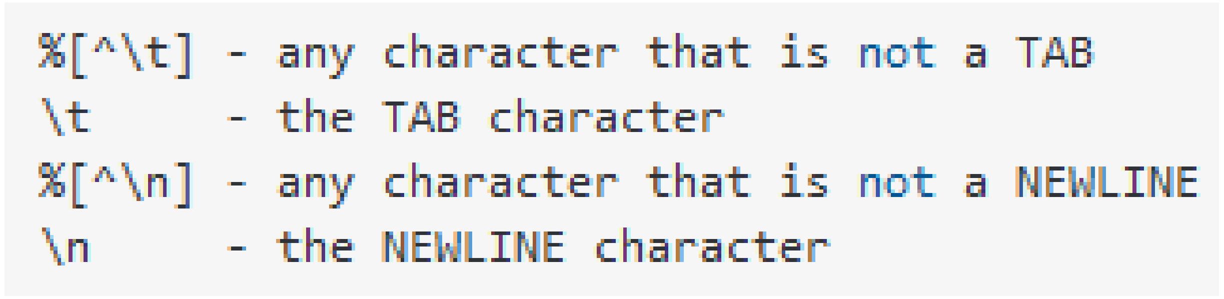
```
Manchester City
Manchester United
Leicester City
West Ham
Chelsea
```

soccer_results.txt (a mix of strings and integers, delimited by tabs and newlines)

```
Manchester City	2	Manchester United	1
Leicester City	3	West Ham	1
Chelsea	0	Manchester City	2
Manchester United	1	Leicester City	1
West Ham	2	Chelsea	1
```

sscanf() is one way of parsing strings delimited by miscellaneous characters

- i.e. we want Manchester City to be read as one string, not two
- Explanation of the fscanf codes
- See: https://stackoverflow.com/questions/10908668/how-do-you-read-tab-delimited-strings-from-a-txt-file-and-put-them-into-variable



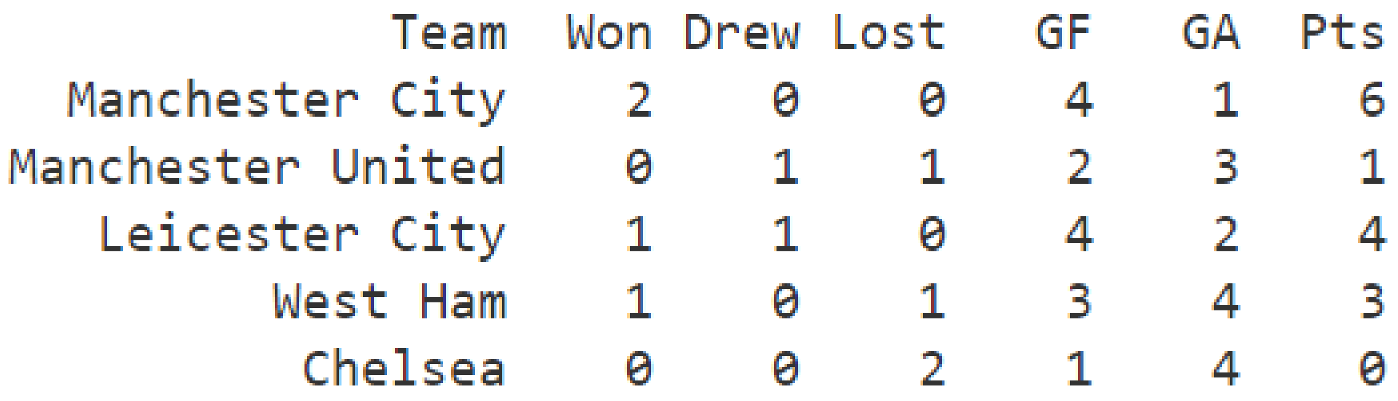
```
%[^\t] - any character that is not a TAB
\t     - the TAB character
%[^\n] - any character that is not a NEWLINE
\n     - the NEWLINE character
```

09_SoccerStructs.cpp

- Functions:
- `FILE* openFileForReading(char* filename)`
  - a helper function for opening a file
- bool readInTeams()
  - Reads team names in from soccer_teams.txt, and populates an array of teams (with other team struct data zeroed)
- int readInResults()
  - Reads in and parses data from soccer_results.txt, and updates the data of appropriate teams identified in each game
- team* findTeamByName(char* name)
  - A helper function which searches the array of teams for one with name matching the argument (returns NULL on failure)
- int getPoints(team* t)
  - A helper function which calculates the points of a team (i.e. 3*won + drew)
- int getSortValue(team* t)
  - A helper function which returns a value for a team, which is suitable for sorting teams into a league table

Exercise

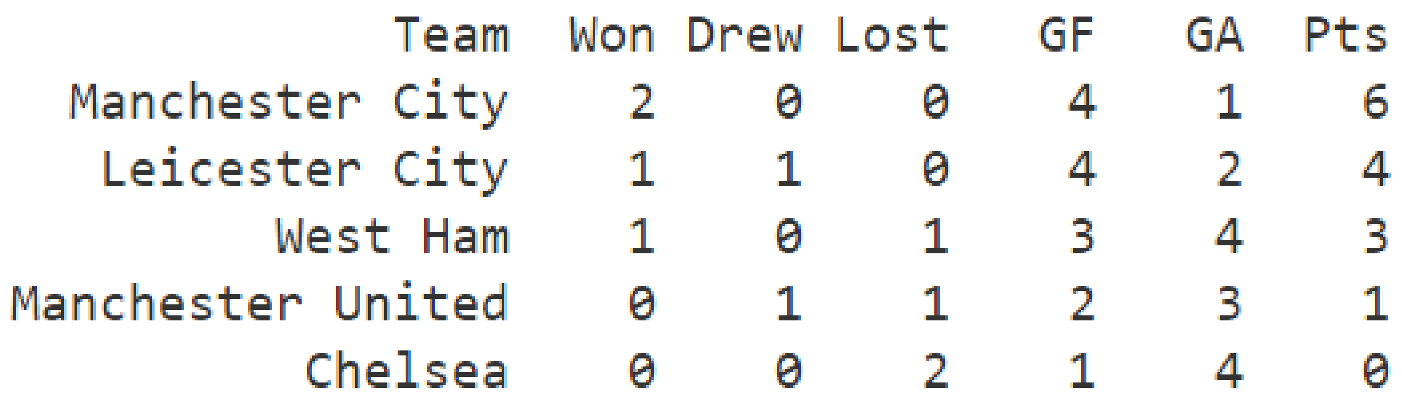
- Modify 09_SoccerStructs.cpp so that it displays the teams in a league table format (but not yet sorted)



```
             Team  Won Drew Lost   GF   GA  Pts
  Manchester City    2    0    0    4    1    6
Manchester United    0    1    1    2    3    1
   Leicester City    1    1    0    4    2    4
         West Ham    1    0    1    3    4    3
          Chelsea    0    0    2    1    4    0
```

Exercise

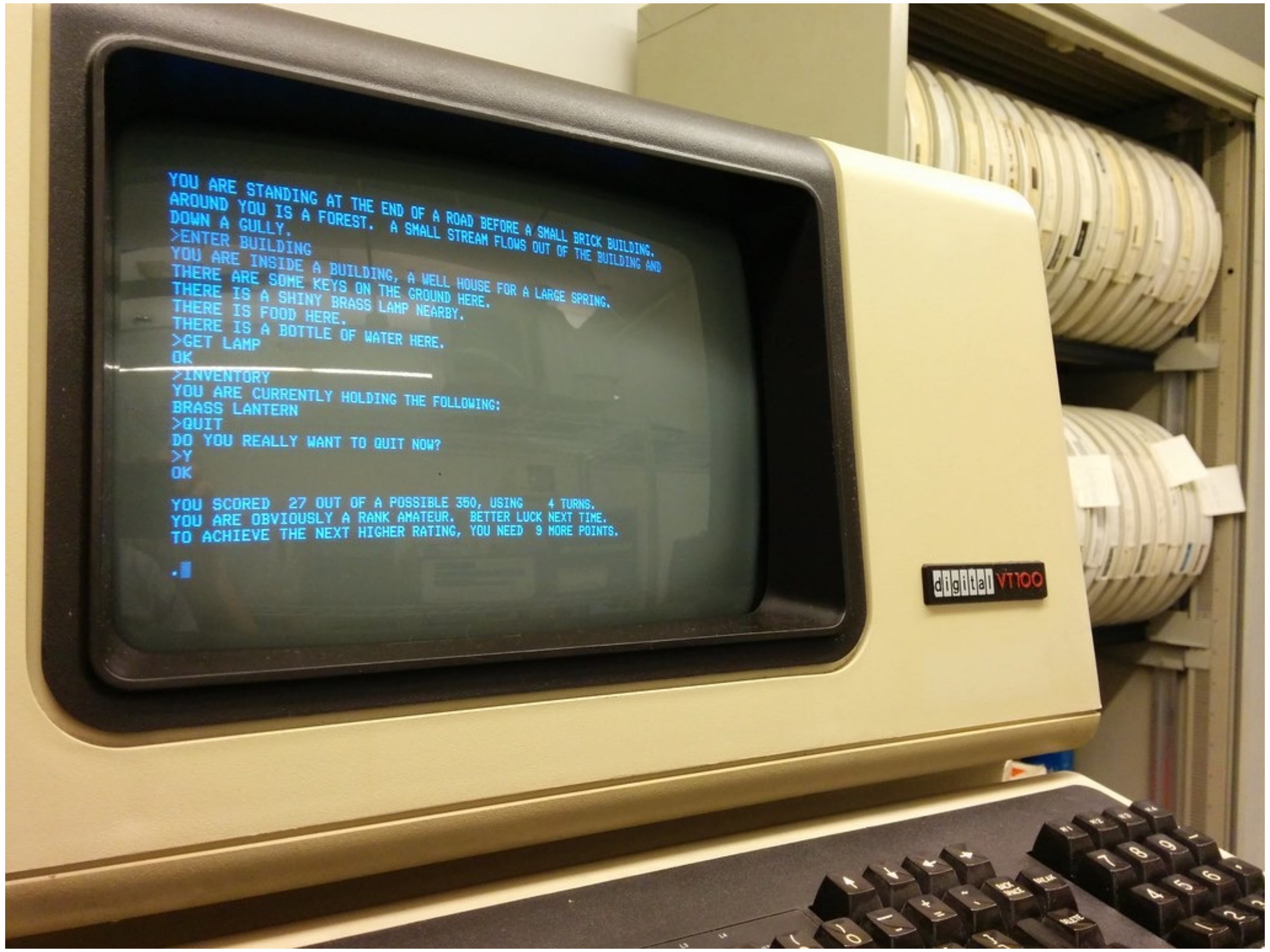
- Modify the previous program so that it now sorts the league table



```
             Team  Won Drew Lost   GF   GA  Pts
  Manchester City    2    0    0    4    1    6
   Leicester City    1    1    0    4    2    4
         West Ham    1    0    1    3    4    3
Manchester United    0    1    1    2    3    1
          Chelsea    0    0    2    1    4    0
```

- Suggested ‘sortvalue’ calc: 1000*points + GF - GA

"Colossal Cave Adventure"



```
YOU ARE STANDING AT THE END OF A ROAD BEFORE A SMALL BRICK BUILDING.
AROUND YOU IS A FOREST.  A SMALL STREAM FLOWS OUT OF THE BUILDING AND
DOWN A GULLY.
>ENTER BUILDING
YOU ARE INSIDE A BUILDING, A WELL HOUSE FOR A LARGE SPRING.
THERE ARE SOME KEYS ON THE GROUND HERE.
THERE IS A SHINY BRASS LAMP NEARBY.
THERE IS FOOD HERE.
THERE IS A BOTTLE OF WATER HERE.
>GET LAMP
OK
>INVENTORY
YOU ARE CURRENTLY HOLDING THE FOLLOWING:
BRASS LANTERN
>QUIT
DO YOU REALLY WANT TO QUIT NOW?
>Y
OK

YOU SCORED  27 OUT OF A POSSIBLE 350, USING   4 TURNS.
YOU ARE OBVIOUSLY A RANK AMATEUR.  BETTER LUCK NEXT TIME.
TO ACHIEVE THE NEXT HIGHER RATING, YOU NEED  9 MORE POINTS.

.
```

- Written in FORTRAN for the PDP-11 mainframe
- We can play it here: https://grack.com/demos/adventure/

Colossal Cave: Some Commands

```
HELP
N, S, E, W, IN, OUT
TAKE [OBJECT]
DROP [OBJECT]
EXAMINE [OBJECT]
INVENTORY
LOOK
QUIT
```

- You will implement the above over 2 weeks of labs

adventure_locations.txt

```
ID      N       S       E       W       IN      OUT     Description
-----------------------------------------------------------------------------------
1       5       4       6       7       2       0       On the NUIG campus, beside the CS building.
2       0       0       0       0       3       1       In the Computer Science Building, beside a computer lab.
3       0       0       0       0       0       2       In a computer Lab.
4       1       0       8       7       9       0       On the Salthill prom. The sea looks inviting (but cold).
5       0       1       6       7       0       0       In Newcastle.
6       0       8       0       1       0       0       In Terryland.
7       0       0       1       0       0       0       In Knocknacarra.
8       1       0       0       4       0       0       On Quay Street. Everywhere is shut (stupid pandemic).
9       0       0       0       0       0       4       In the Sea. It's freezing!
```

This file is tab-delimited
Note the presence of the two header lines at the start of the file, which you'll need to deal with

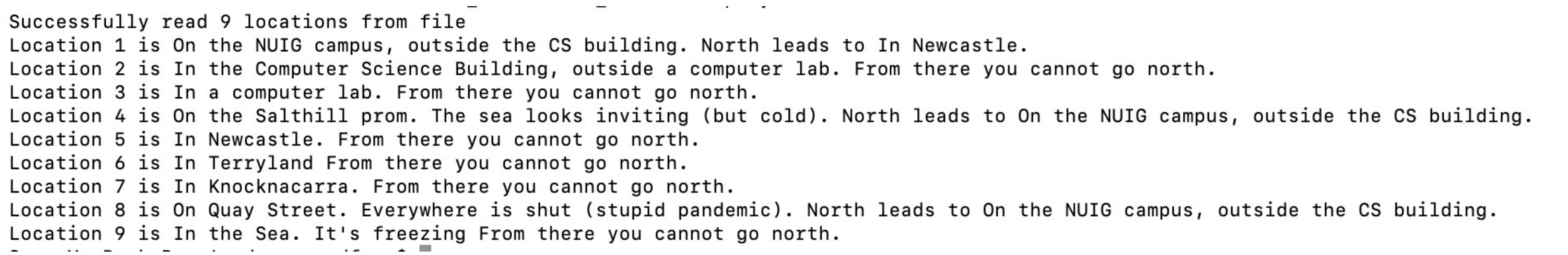
Exercise

- Write a program which defines a `struct` suitable of storing the locations defined in `adventure_locations.txt`
- Now have the program read `adventure_locations.txt` into an array of these structs

- (09_AdventureA_ReadLocations.cpp)

Exercise

- Modify your previous program so that it displays each location (after reading them all in), including where each location leads to if you travel North from it.
- Sample output:



```
Successfully read 9 locations from file
Location 1 is On the NUIG campus, outside the CS building. North leads to In Newcastle.
Location 2 is In the Computer Science Building, outside a computer lab. From there you cannot go north.
Location 3 is In a computer lab. From there you cannot go north.
Location 4 is On the Salthill prom. The sea looks inviting (but cold). North leads to On the NUIG campus, outside the CS building.
Location 5 is In Newcastle. From there you cannot go north.
Location 6 is In Terryland From there you cannot go north.
Location 7 is In Knocknacarra. From there you cannot go north.
Location 8 is On Quay Street. Everywhere is shut (stupid pandemic). North leads to On the NUIG campus, outside the CS building.
Location 9 is In the Sea. It's freezing From there you cannot go north.
```

Lab Assignment

- Using the file `adventure_locations.txt`, implement the movement commands (`N, S, E, W, IN, OUT`) as well as LOOK, `HELP` and `QUIT`
- Display the description of each location as the player moves to it
- You can start with the code from the previous exercise (your own code, or my sample solution).
- See sample input/output on next slide

```
Welcome to Galway Adventure. Type 'help' for help.

On the NUIG campus, outside the CS building.
warning: this program uses gets(), which is unsafe.
> help
I know these commands:
n, s, e, w, in, out, look, help, quit.

> w

In Knocknacarra.
> s

You can't go that way.

> e

On the NUIG campus, outside the CS building.
> in

In the Computer Science Building, outside a computer lab.
> look

In the Computer Science Building, outside a computer lab.
> out

On the NUIG campus, outside the CS building.
> s

On the Salthill prom. The sea looks inviting (but cold).
> quit
Bye!
```

CT103 Programming

Semester 2 Week 10

Command-Line Arguments

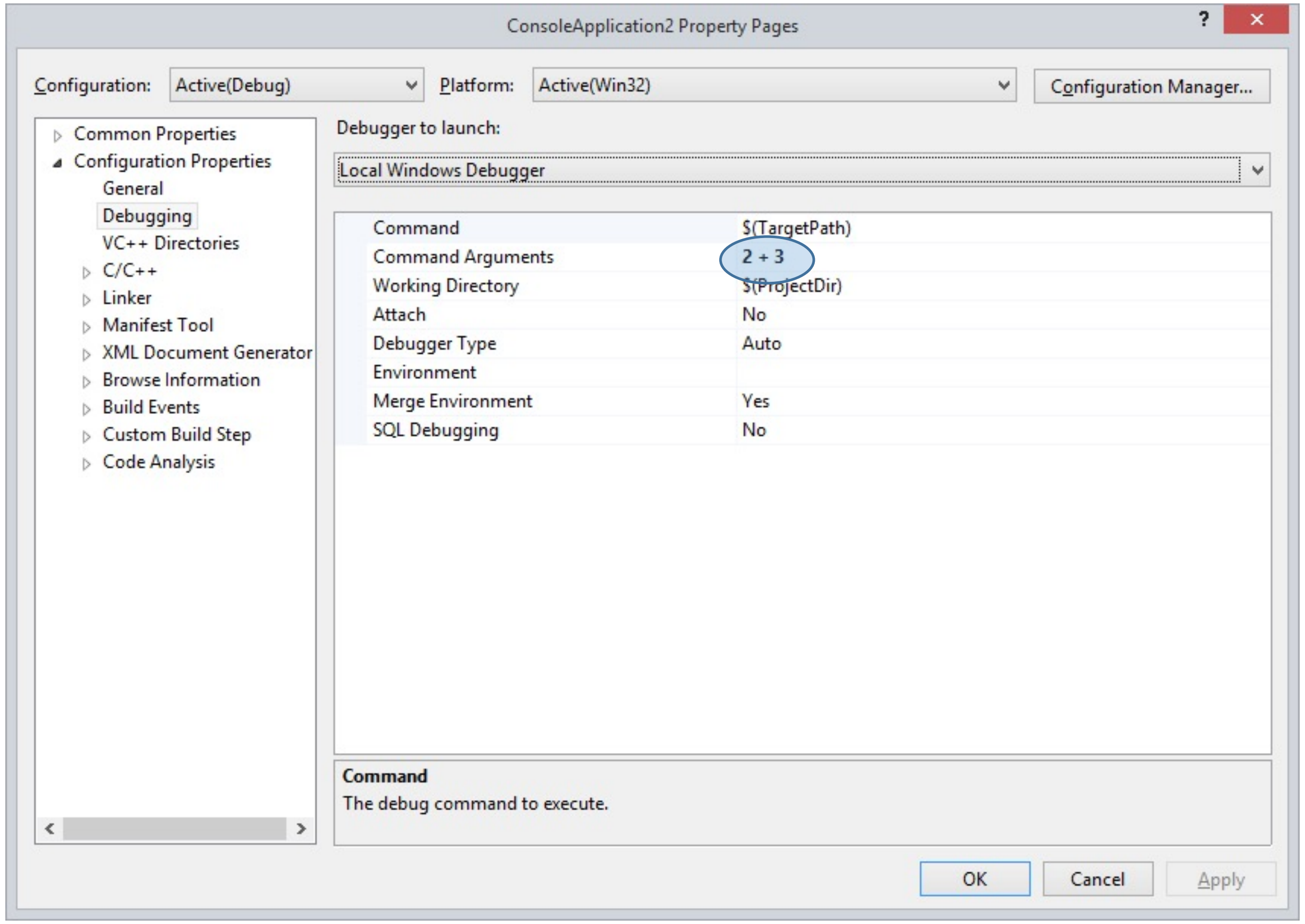
and Fun with Calendars

Dr. Sam Redfern

Discord Server (the same one we're using for CT1114)

Command Line Arguments

- You can set the inputs (command line arguments) for your .exe in the option
- Project Properties
  - Configuration Properties
    - Debugging
      - Command Arguments

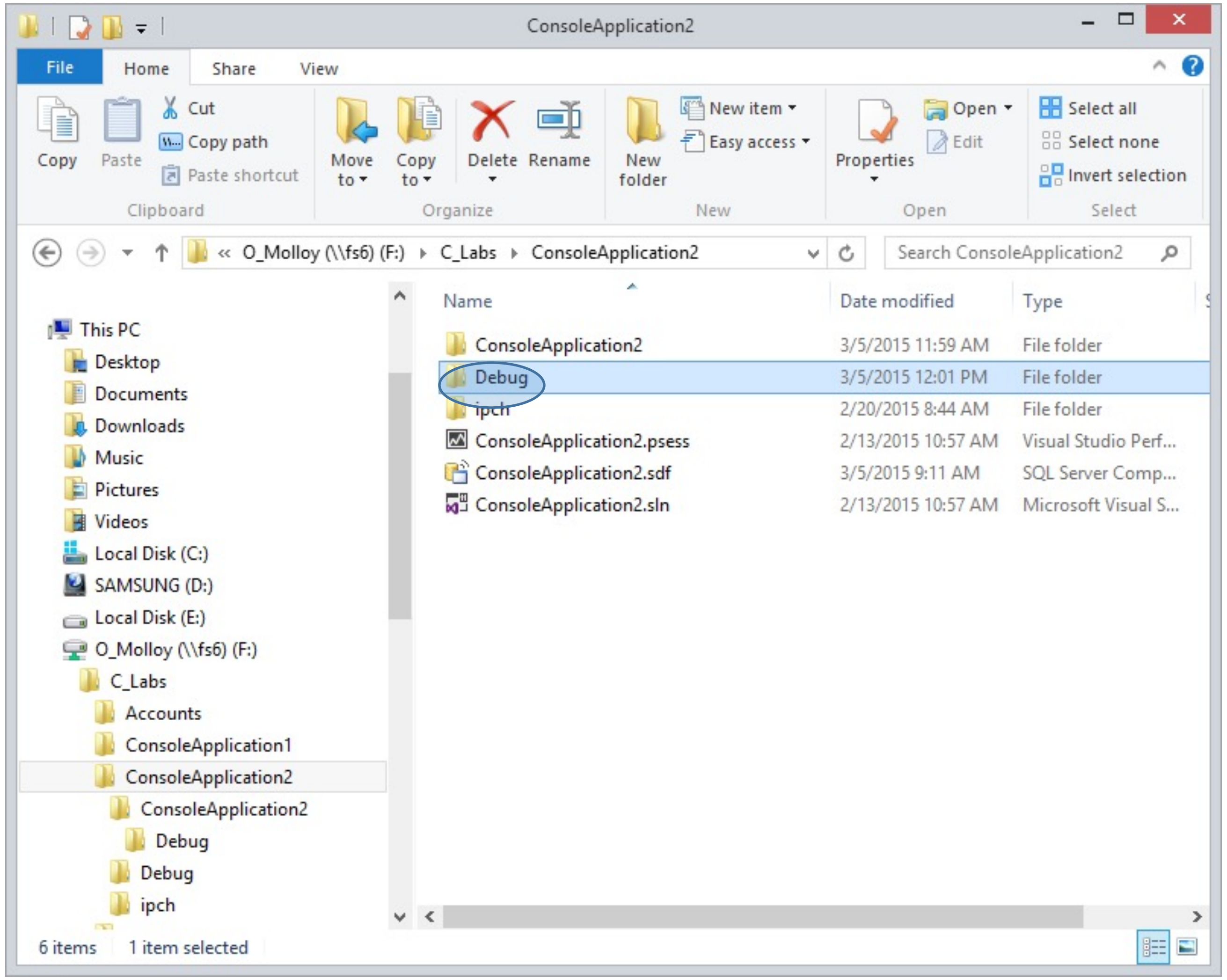


Command	$(TargetPath)
Command Arguments	2 + 3
Working Directory	$(ProjectDir)
Attach	No
Debugger Type	Auto
Environment	
Merge Environment	Yes
SQL Debugging	No

Command
The debug command to execute.

Or alternatively, run from a command prompt

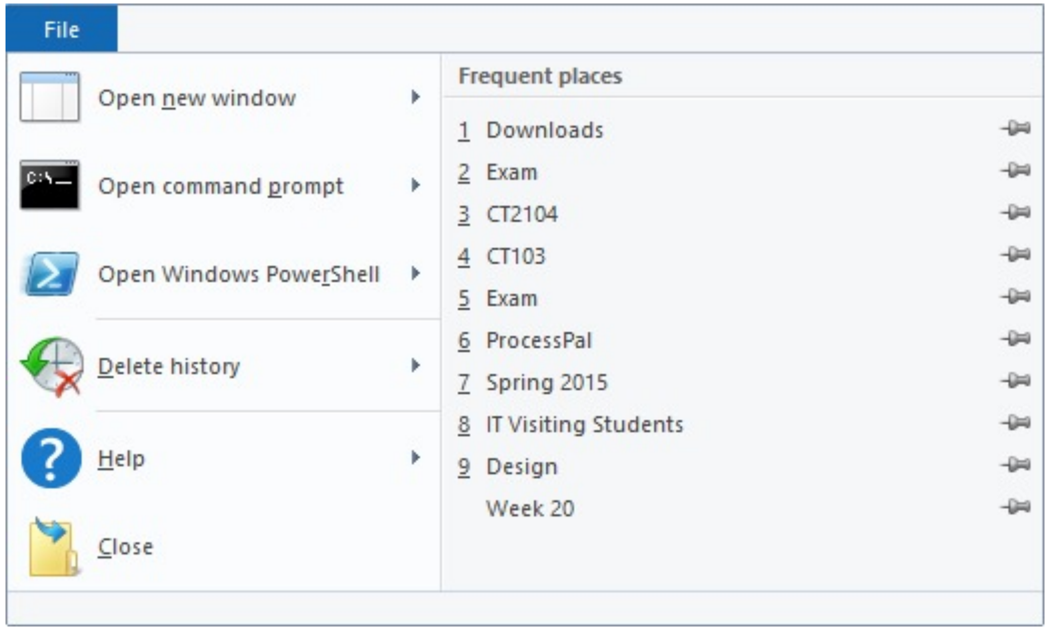
- Open the folder containing your solution
- There should be a Debug folder – open it
- It should contain the .exe



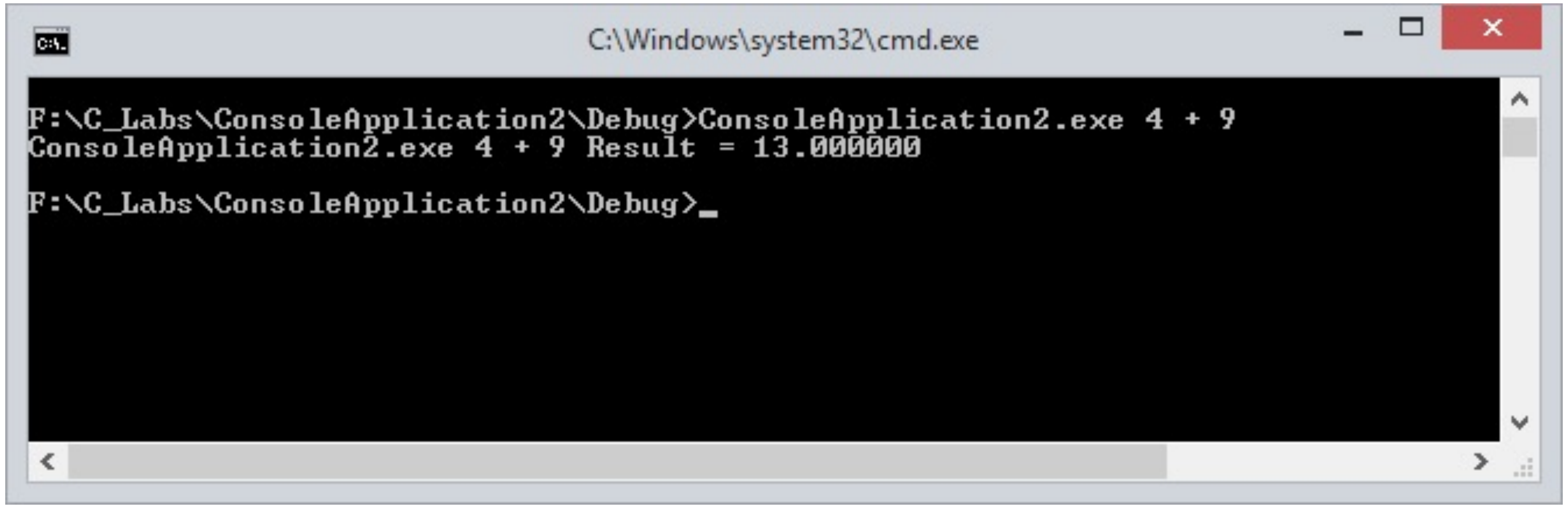
Name	Date modified	Type
ConsoleApplication2	3/5/2015 11:59 AM	File folder
Debug	3/5/2015 12:01 PM	File folder
ipch	2/20/2015 8:44 AM	File folder
ConsoleApplication2.psess	2/13/2015 10:57 AM	Visual Studio Perf...
ConsoleApplication2.sdf	3/5/2015 9:11 AM	SQL Server Comp...
ConsoleApplication2.sln	2/13/2015 10:57 AM	Microsoft Visual S...

Visual Studio can open a command prompt at the correct folder for you

- File
  - > Open Command Prompt
  - > In this Debug folder



- Now you can run the .exe by typing in the full name of the executable, followed by the arguments



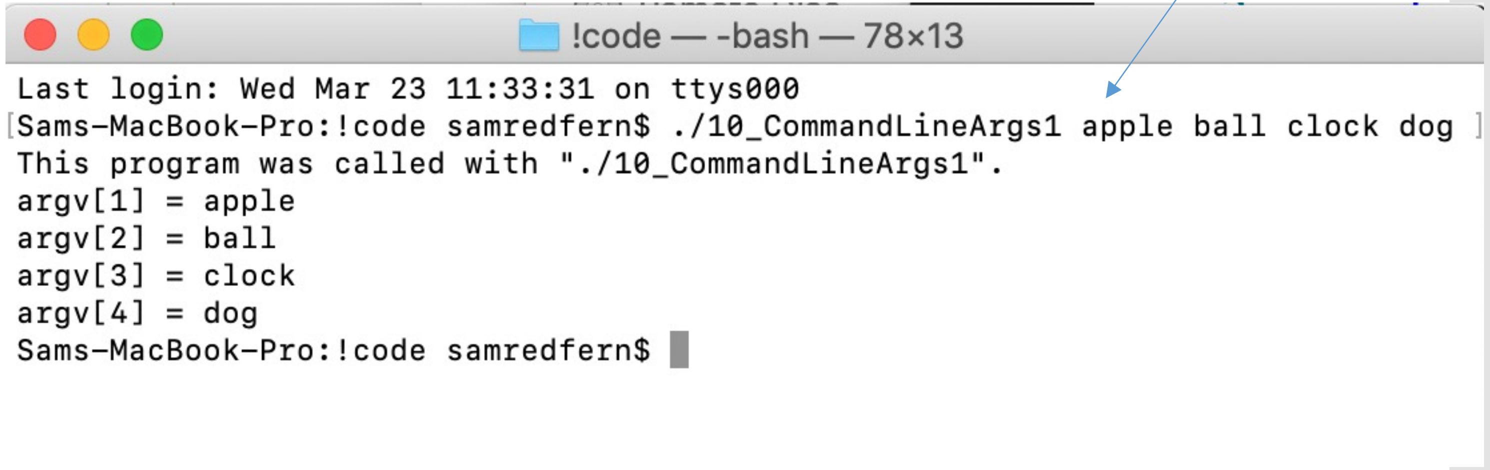
```
F:\C_Labs\ConsoleApplication2\Debug>ConsoleApplication2.exe 4 + 9
ConsoleApplication2.exe 4 + 9 Result = 13.000000

F:\C_Labs\ConsoleApplication2\Debug>_
```

On the Mac

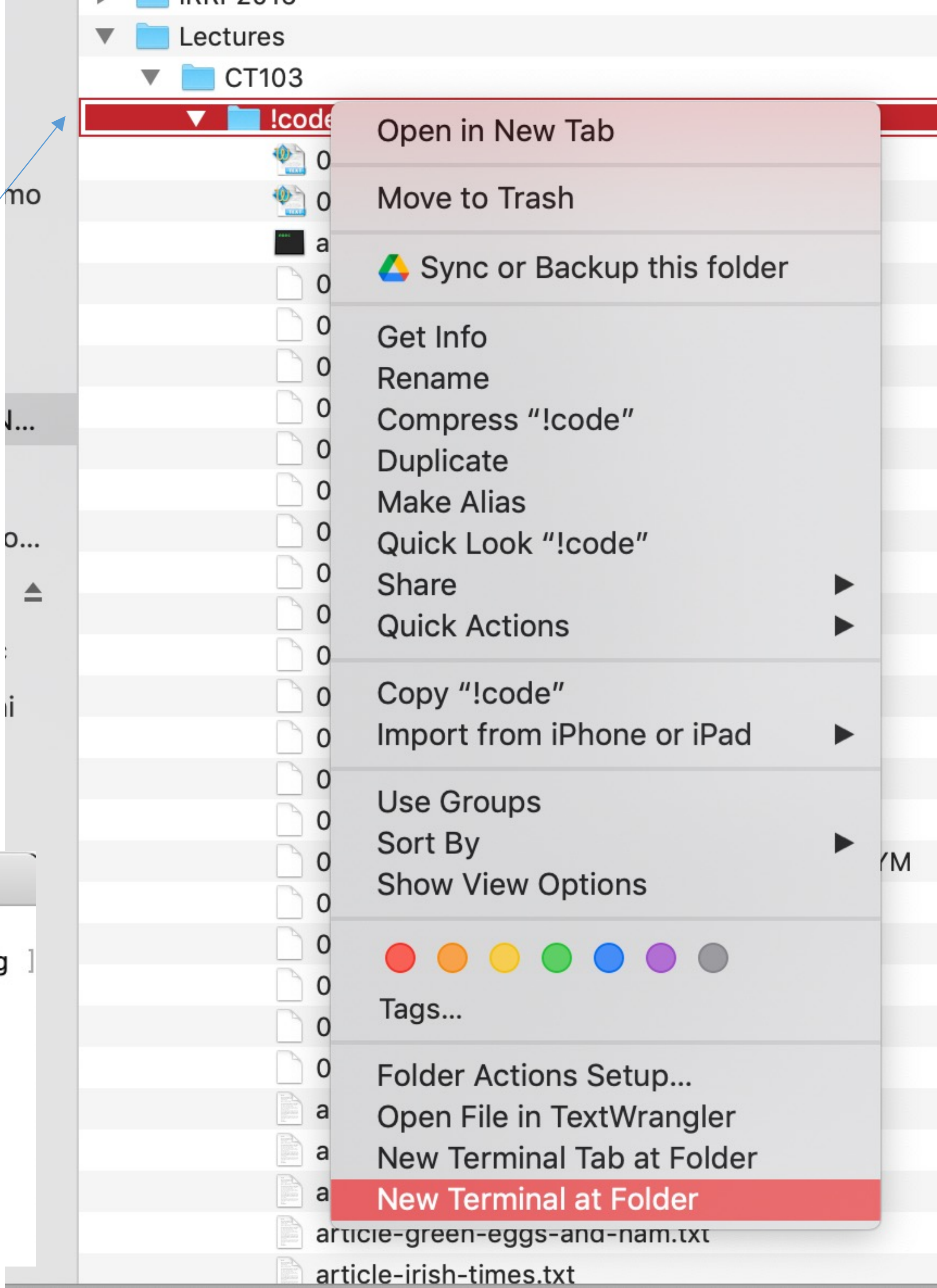
- In Xcode
- Product
    - Scheme
        - Edit Scheme
            - Run
                - Arguments
                    - + (add arguments)

On the Mac (without Xcode)

- Right click the folder that contains your executable
- Select "New Terminal at Folder"
- Type the name of your executable, with "./" in front of it, and command line args as desired



```
Last login: Wed Mar 23 11:33:31 on ttys000
Sams-MacBook-Pro:!code samredfern$ ./10_CommandLineArgs1 apple ball clock dog
This program was called with "./10_CommandLineArgs1".
argv[1] = apple
argv[2] = ball
argv[3] = clock
argv[4] = dog
Sams-MacBook-Pro:!code samredfern$ 
```



Reading Command Line Arguments in your code

```
#include <stdio.h>

int main (int argc, char *argv[]) {
  int count;

  // argc is the number of command-line args (including exe name)
  // argv[0] is the exe name (including path)
  printf ("This program was called with \"%s\".\n", argv[0]);

  if (argc > 1) {
    // argv[1], argv[2] etc. are the "actual" arguments
    for (count = 1; count < argc; count++)
      printf("argv[%d] = %s\n", count, argv[count]);
    }
    else {
      printf("Called with no command-line arguments.\n");
    }

  return 0;
}
```

Exercise

- Add together all of the numbers supplied at the command-line
- You can convert a string to a number using atoi() or atof() from <stdlib.h>
    - **`int atoi(const char *str)`**
    - **`double atof(const char *str)`**

Some Exercises with Calendars

A function that returns the number of days in a month

```
int no_of_days(int year, int month) {
    if (month == 9 || month == 4 || month == 6 || month
== 11)
        return 30;

    if (month != 2)
        return 31;

    // but what about February?
}
```

Exercise: Leap Years

- Write a C function which receives a year number as an argument.
- The function should return 1 if the year is a leap year, and return 0 if it is not.
- Start with the code provided on the next slide
- When you have finished, modify the program so that it receives the year number as a command-line argument rather than using scanf()

```
#include <stdio.h>

int is_leap(int year);
int no_of_days(int year, int month);

int main() {
  int y;
  printf("Enter a year number > ");
  scanf(" %d", &y);
  if (is_leap(y)==1)
    printf("It's a leap year!");
  else
    printf("It's not a leap year!");
  printf(" ... and February has %d days.",
                         no_of_days(y,2));
}
```

```
int no_of_days(int year, int month) {
  if (month == 9 || month == 4 || month
== 6 || month == 11)
    return 30;

  if (month != 2)
    return 31;

  return 28 + is_leap(year);
}


int is_leap(int year) {
  return 1;
  // to do: change this so that leap
years return 1
  // and others return 0
}
```

What day of the week does a month start on?

- Fact: January 1st, 1900 was a Monday

- So, what day of the week was February 1st, 1900?
  - And how do we calculate that?
- What day of the week was May 1st, 1900?
  - And how do we calculate that?
- What about May 1st, 1901?

Exercise

- Write a command-line-driven program which accepts a year number as an argument, and one or more month numbers as subsequent arguments
- The program should print out calendars for the specified months
- See starting code:

`10_CommandLineCalendarsStart.cpp`

```
.\calendars.exe 2021 3 4


3/2021

 Sun Mon Tue Wed Thu Fri Sat
           1   2   3   4   5   6
   7   8   9  10  11  12  13
  14  15  16  17  18  19  20
  21  22  23  24  25  26  27
  28  29  30  31

4/2021

 Sun Mon Tue Wed Thu Fri Sat
                       1   2   3
   4   5   6   7   8   9  10
  11  12  13  14  15  16  17
  18  19  20  21  22  23  24
  25  26  27  28  29  30
```

Assignment

- In this assignment, you will be adding some more features to the text adventure game which you started last week
- You may use your own code or my sample solution from last week as a starting point for this week

```
HELP
N, S, E, W, IN, OUT
TAKE [OBJECT]
DROP [OBJECT]
EXAMINE [OBJECT]
INVENTORY
LOOK
QUIT
```

- The 4 commands you're adding this time all relate to objects which may be picked up from one location and dropped in another
- When displaying the description of a location, also tell the players which objects (if any) are there
- 'INVENTORY' lists the objects currently being carried by the player
- 'EXAMINE' displays an object's description – but only if it's being carried or in the current location
- For TAKE, EXAMINE, and DROP you can have the player enter the command and then them ask for the object – this is easier than trying to separate them from one string

adventure_objects.txt

```
Name                 Location                      Description
------------------------------------------------------------------
USB Drive            3                             A small USB drive which holds a whopping 1TB
Ice Cream            4                             An ice cream, inexplicably found on Salthill Prom
```

This file is tab-delimited
Note the presence of the two header lines at the start of the file, which you'll need to deal with

```
On the Salthill prom. The sea looks inviting (but cold).
Objects here: Ice Cream
> take
Take what? > Ice Cream
You take Ice Cream.
> inventory
You are carrying: USB Drive, Ice Cream
> w

In Knocknacarra.
Objects here: nothing
> drop
Drop what? > Ice Cream
You drop Ice Cream.
```

CT103 Programming

Semester 2 Week 11

Dynamic Memory Allocation

Recursion

Introduction to some more Data Structures

Dr. Sam Redfern

Discord Server (the same one we're using for CT1114)

Static vs. Dynamic memory allocation

- Static arrays – size defined at compile time
  - Memory stored on the stack (if declared inside a function) or the heap (if declared globally)
  - Stack grows when entering new blocks (branches, loops, functions)
  - Stack shrinks when leaving blocks
- Dynamic array – size defined at run time
  - Memory stored on the heap
  - Stays available until removed
    - In C – removed manually with function calls
    - In Java, or C#, or Javascript – removed automatically with garbage collection => no risk of memory leak

- Why have dynamic memory?
  - Input of unknown size
  - Data structures that require dynamic memory allocation
    - Linked lists, trees, etc.
- Flexibility and Efficiency of memory consumption

sizeof

- The **sizeof** operator will return the number of bytes reserved for a variable or data type.

- Determines:
  - The byte length of a simple data type (int, float, char etc.)
  - Number of bytes required for a structure (user defined type)
  - Byte length of an array

sizeof example

```c
#include <stdio.h>

struct {
  int a;
  int b;
  float d;
} myStruct;

int main() {
  char myString[20];

  printf("An int uses %d bytes\n", sizeof(int));
  printf("A float uses %d bytes\n", sizeof(float));
  printf("A char uses %d bytes\n", sizeof(char));
  printf("myStruct uses %d bytes\n", sizeof(myStruct));
  printf("myString uses %d bytes\n", sizeof(myString));
}
```

As expected, myStruct is reported at 12 bytes

Why is this happening?

```c
struct {
  int a;
  int b;
  char c1;
  float d;
} myStruct;
```

This is reported at 16 bytes (.. but 13 was expected?)

```c
struct {
  int a;
  int b;
  char c1, c2;
  float d;
} myStruct;
```

And this is also 16 bytes! (.. why?)

Dynamic memory functions in <stdlib.h>

- malloc()
  - Allocate a memory block
- free()
  - De-allocate a previously allocated memory block
- calloc()
  - Allocate space for an array
- realloc()
  - Change the size of a previously allocated memory
- Each function is used to initialize a pointer with memory from the heap’s free store (a section of memory available to all programs)

malloc

- The function malloc() will allocate a block of memory that is size x bytes large. If the requested memory can be allocated a pointer is returned to the beginning of the memory block.
- **Note:** the content of the received block of memory is not initialized.

- **malloc()** prototype**:**
  - void * malloc (size);
- **Parameters:**
  - Size of the memory block in bytes.
- **Return value:**
  - If the request is successful then a pointer to the memory block is returned.
  - If the function failed to allocate the requested block of memory, a null pointer is returned.

malloc usage

```
int *ptr = (int*) malloc( sizeof (int) );
```

- Note that malloc() does not know what the memory will be used for, it only knows how many bytes (contiguously allocated) are required
- Therefore, the return type is **void*** and you will need to cast it to the correct type e.g. **int***

At what value for numInts does this fail, and why?

11_malloc_huge.cpp

```
#include <stdio.h>
#include <stdlib.h>

int main() {
  int* buffer;
  int numInts = 1;

  while (true) {
    int bytesRequired = numInts * sizeof(int);
    buffer = (int*) malloc ( bytesRequired );
    if (buffer==NULL) {
      printf("Failed to allocate %d bytes.\n", bytesRequired);
      return 0;
    }
    else {
      free(buffer);
      printf("Succeeded in allocating %d bytes.\n", bytesRequired);
      numInts *= 10;
    }
  }
}
```

Dynamically Allocated Arrays

- Allows you to avoid declaring array size at declaration.
- Use malloc to allocate memory for array when needed:

```
int *dynamic_array;
dynamic_array = malloc( sizeof( int ) * 10 );
dynamic_array[0]=1;
```

Question: explain why we can declare an **int*** and then treat it like an array, with square-bracket access to elements?

Exercise

- Write a program which asks the user for a number (call it x)
- It then uses malloc to create an array of size x, containing floats
- It should populate each array entry with a random float between 1 and 1000
- Finally, it calculates and displays the average of these values

Deallocation of memory

- As already seen, **free(void*)** is used to release memory back to the heap
  - The operating system knows how large the block of allocated memory is
- But what if we forget to do that?

```
int *ptr;
ptr = (int *)malloc(sizeof(int));
ptr = (int *)malloc(sizeof(int));
```

- This is a "Memory Leak"
- This is one of the things that garbage collectors in more modern languages avoid
- Note: after using **free()** you should set the pointer to NULL, otherwise it will still be pointing to the same address, *which your program no longer owns*
  - Modern operating systems will stop programs from accessing memory they don’t own (it will crash them instead)

Recursion

- A recursive function is one which calls itself
- This gives an alternative to loops
- Generally recursion is less efficient than loops, but certain types of problem are much easier to write using recursion
  - We'll use recursion a bit later on, when loading and saving a binary decision tree

- Question: why might recursion be less efficient than loops?

Recursion Example: sum of 1-N

```c
#include <stdio.h>
int sum(int n);

int main() {
  int number, result;

  printf("Enter a positive integer: ");
  scanf("%d", &number);

  result = sum(number);

  printf("sum = %d", result);
}

int sum(int n) {
  if (n != 0)
    return n + sum(n-1);
  else
    return n;
}
```

There is no benefit to using recursion in this specific case; it's just a simple example

Infinite recursion should be avoided (just the same as infinite loops)

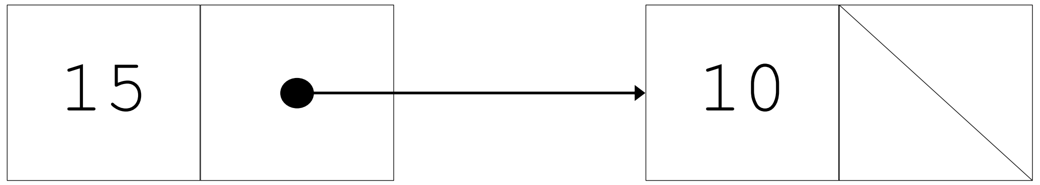
Too many recursive calls leads to 'Stack Overflow’

- Any ideas what this means, precisely?

Self-referential structs

- Structs that contains a pointer to a struct of the same type
- Can be linked together to form useful data structures such as lists, queues, stacks and trees
- Terminated with a `NULL` pointer (**`0`**)

```
struct node {
   int data;
   node *nextPtr;
}
```

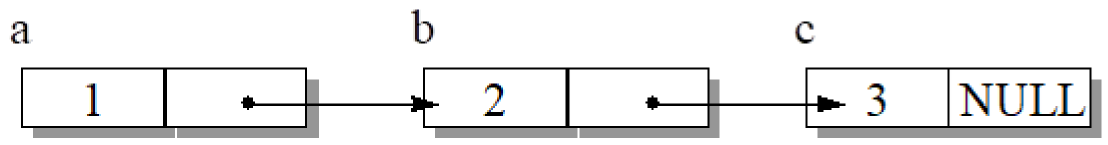


- `nextPtr`
  - Points to an object of type **node**
  - Referred to as a link
    - Ties one **`node`** to another **`node`**

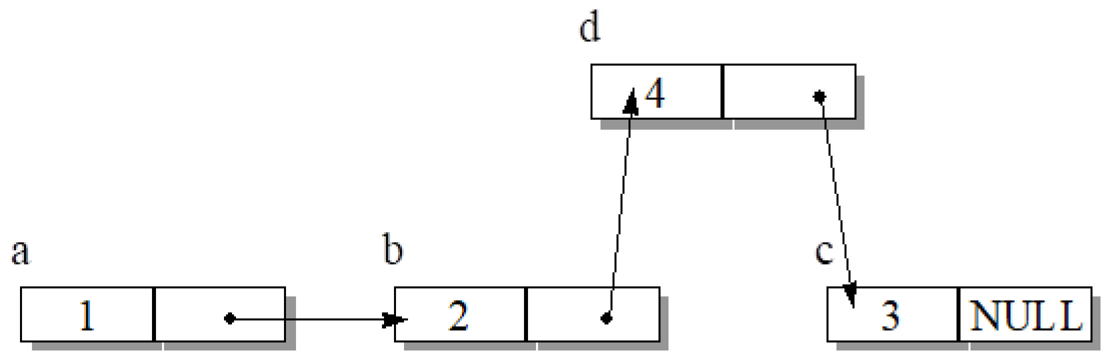
Linked Lists

- Linked list
  - Linear collection of self-referential struct objects, called nodes
  - Connected by pointers
  - Accessed via a pointer to the first node of the list ('head')
  - Subsequent nodes are accessed via the link-pointer member of the current node
  - Link pointer in the last node is set to `NULL` to mark the list’s end
- Strengths vs. arrays:
  - Good for collections of data which grow and shrink at runtime
  - Excellent for efficient insertion or deletion at any point in the middle of the list (just needs some changing of pointers: see next slide) – e.g. if you want to keep a sorted list
- Weakness vs. arrays:
  - Linked lists are sequential access (not random access), i.e. to get to element number 10000 you have to read through the preceding 9999 items

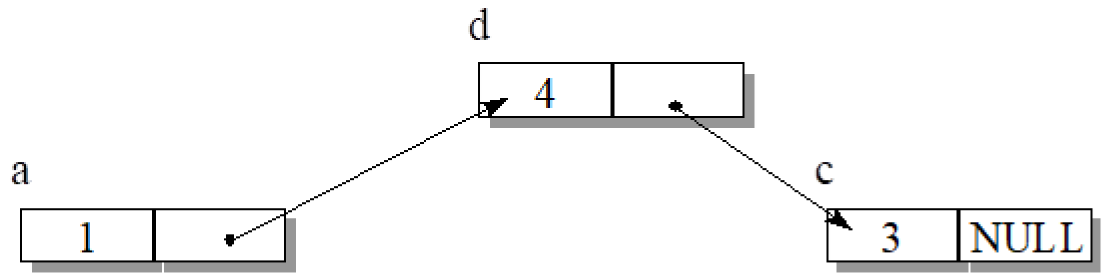
Linked Lists



A linked list with 3 nodes a, b, c



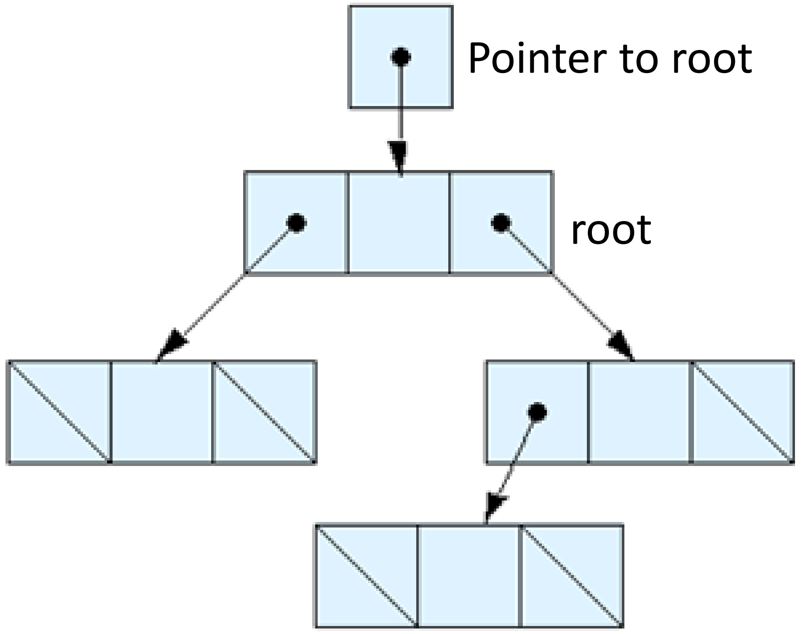
Inserting a new node d between b and c



Removing node b

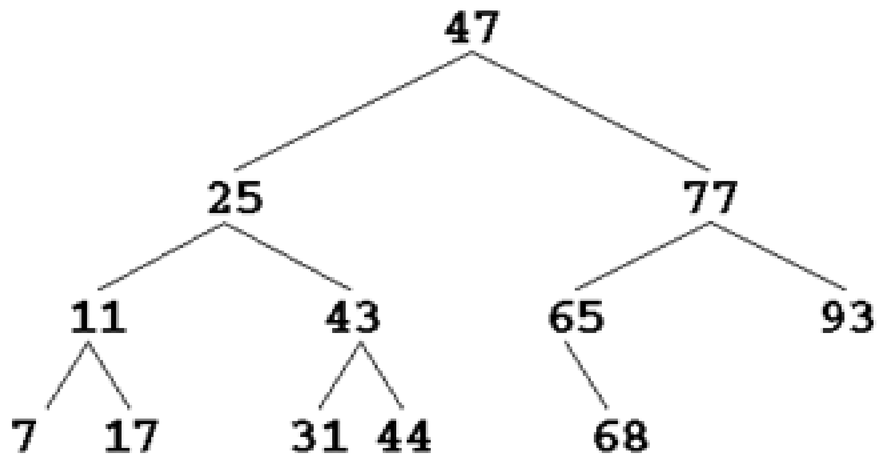
Trees

- Tree nodes contain two or more links (pointers) to other nodes
- Binary trees are a particular (very useful) type of tree
  - All nodes contain two links
    - None, one, or both of which may be NULL
  - The **root node** is the first node in a tree.
  - Each link in the root node refers to a **child**
  - A node with no children is called a **leaf** node
- Recursion is a very natural way of operating on trees



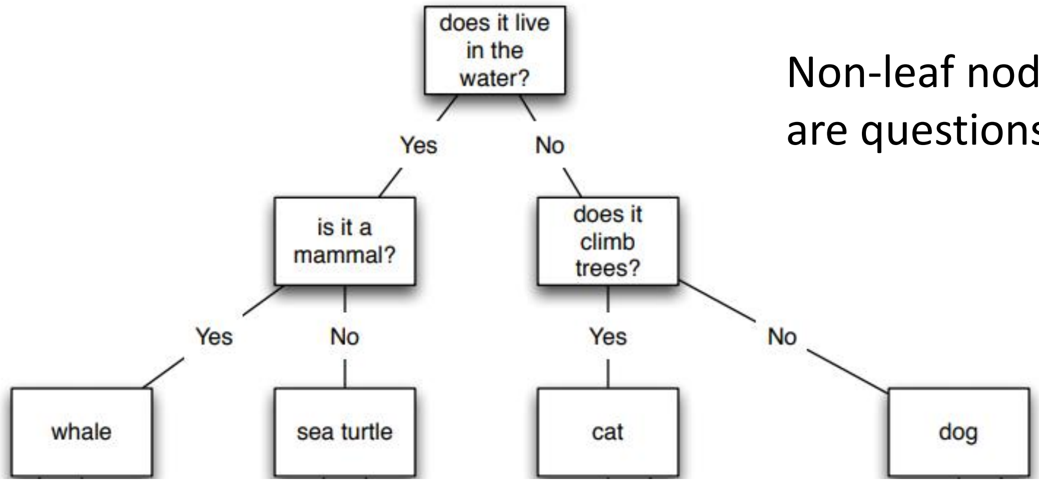
Binary search tree

- Data is inserted in a particular way to facilitate rapid searching
- Values in left subtree are less than parent’s value
- Values in right subtree are greater than parent’s value
- Question: this is similar to using an array with binary search.. but is a binary search tree superior in any way?



'Guess the Animal' game

- The player thinks of an animal and the computer has to try to guess it through a series of yes/no questions
- This data structure is called a 'Decision Tree'
- When a leaf node is met, this is the computer's guess
- If the guess is wrong, the player is asked for the animal they were actually thinking of, plus a yes/no question to differentiate between the (incorrect) guess and that animal
- See sample on next slide



Non-leaf nodes are questions

Leaf nodes are animals

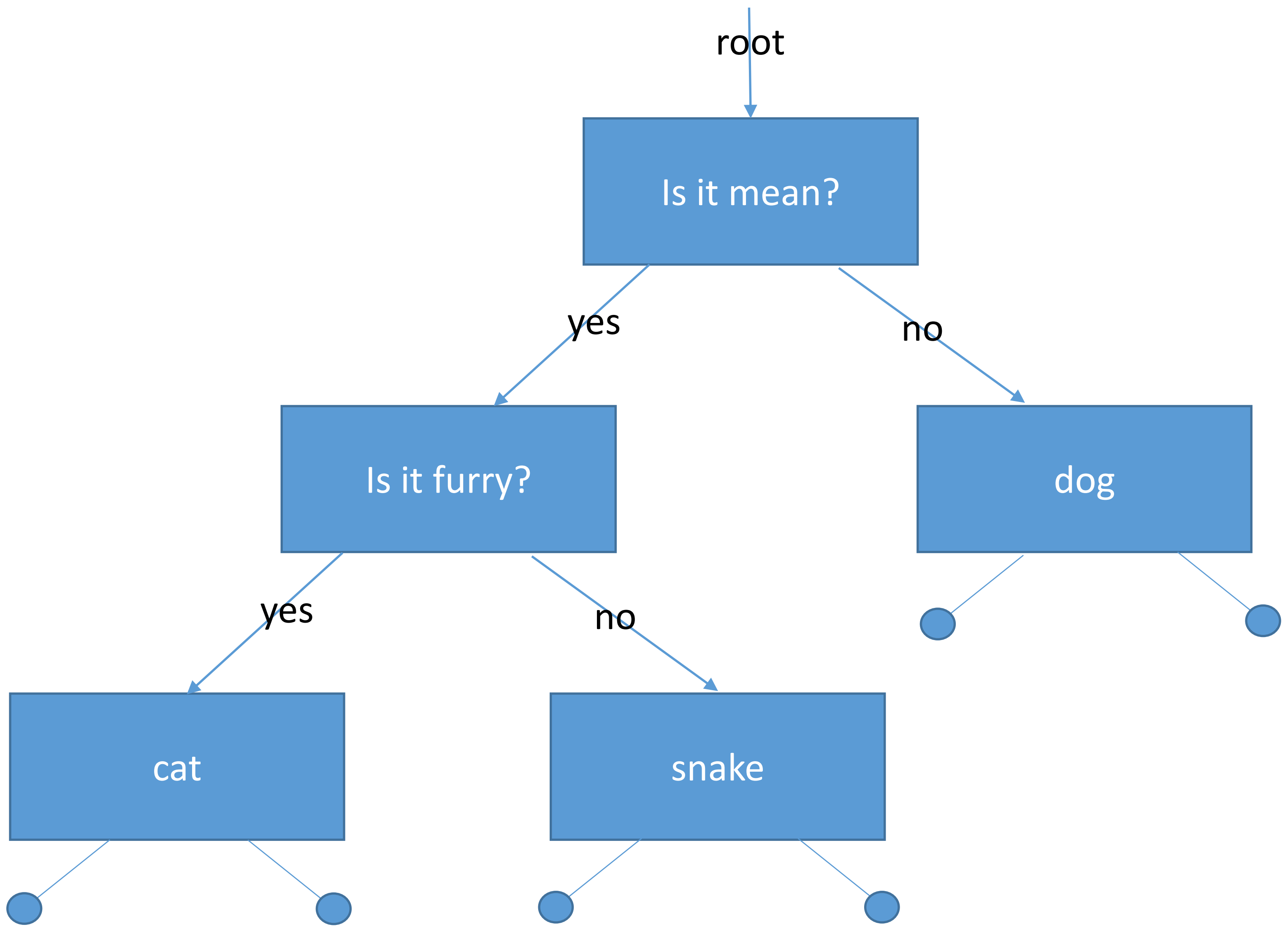
```
Think of an animal. I will amazingly guess your animal!
Press any key to start >
Were you thinking of: dog?
n

Oops. What animal were you thinking of? >cat

Please give me a question to distinguish dog from cat.
>is it mean?

For cat, would the answer be Yes or No? >y

Thank you! Now I know another animal
Do you want to play again? >y
```

```
Think of an animal. I will amazingly guess your animal!
Press any key to start >

is it mean? >y
Were you thinking of: cat?
n

Oops. What animal were you thinking of? >snake

Please give me a question to distinguish cat from snake. >is it furry?

For snake, would the answer be Yes or No? >n

Thank you! Now I know another animal
Do you want to play again? >n
```

node

```
// binary decision tree node for the Animal game
struct node {
  char txt[100];
  node* yes;
  node* no;
};
```

The Binary Tree after adding dog, cat, snake



11_GuessTheAnimal_START.cpp

- This is the full game without loading and saving
- It restarts each time with just one node in the tree (i.e. one animal and zero questions)

Using recursion to free() all the nodes that were allocated using malloc() in GuessTheAnimal game..

This is initially called with `root` as the argument, and cleans up all nodes

```
void destroynode(node* n) {
  if (n->yes!=NULL)
    destroynode(n->yes);

  if (n->no!=NULL)
    destroynode(n->no);

  free(n);
}
```

Sequence:

- ROOT: destroynode(Is it mean?)
    - Y: destroynode(is it furry?)
        - Y: destroynode(cat)
            - Cat destroyed
        - N: destroynode(snake)
            - Snake destroyed
        - Is it furry? destroyed
    - N: destroynode(dog)
        - Dog destroyed
    - Is it mean? destroyed

Saving the data in a binary search tree

- How can we save the data stored in the Guess the Animal game?
- Here's some pseudocode for a recursive function:

```
savenode(n) {
   write n->txt to file
   write a value indicating whether n->yes is NULL or not
   if n->yes is not null, then call savenode(n->yes)
   write a value indicating whether n->no is NULL or not
   if n->no is not null, then call savenode(n->no)
}
```

Data file: dog, cat, snake

is it mean?	(‘is it mean’ node, txt)
NOTNULL	(‘is it mean’ node, is ‘yes’ pointer null?)
is it furry?	(‘is it furry?’ node, txt)
NOTNULL	(‘is it furry?’ node, is ‘yes’ pointer null?)
cat	(‘cat’ node, txt)
NULL	(‘cat’ node, is ‘yes’ pointer null?)
NULL	(‘cat’ node, is ‘no’ pointer null?)
NOTNULL	(‘is it furry?’ node, is ‘no’ pointer null?)
snake	(‘snake’ node, txt)
NULL	(‘snake’ node, is ‘yes’ pointer null?)
NULL	(‘snake’ node, is ‘no’ pointer null?)
NOTNULL	(‘is it mean’ node, is ‘no’ pointer null?)
dog	(‘dog’ node, txt)
NULL	(‘dog’ node, is ‘yes’ pointer null?)
NULL	(‘dog’ node, is ‘no’ pointer null?)

Loading the data in a binary search tree

- Use recursion to load back a tree as saved above
- Pseudocode:

```
readnode(n) {
  read n->txt from file
  read the value which indicates whether n->yes is NULL or not
  if the n->yes is not NULL then instantiate a new node to hold the next piece of data, assign it to n->yes and call readnode(n->yes)
  read the value which indicates whether n->no is NULL or not
  if the n->no is not NULL then instantiate a new node to hold the next piece of data, assign it to n->no and call readnode(n->no)
}
```

Example

- Adding load and save to 'Guess the Animal' game
    - Starting point discussed in class: **11_GuessTheAnimal_START.cpp**

- Loading and saving of the data in the binary search tree, allows a more fun database of questions and animals to be built up each time you play