Programming Paradigms

CT331 Week 4 Lecture 2

Finlay Smith

finlay.smith@nuigalway.ie

Memory Allocation

```
typedef struct studentStruct
{
        char name[30];
        int number;
} student;
```

```
student newStudent(char* name, int number){
        student s;
        strcpy(s.name, name);
        s.number = number;
        return s;
}
```

Memory Allocation

```
typedef struct studentStruct
{
        char name[30];
        int number;
} student;
```

```
student newStudent(char* name, int number){
        student s;
        strcpy(s.name, name);
        s.number = number;
        return s;
}
```

Creates a new student on stack.

Pointer to that student can be obtained with `&s;`

What if we also want to store student exam scripts, results etc...? (Just update the struct…)

What if we want to create all 17k thousand students in NUIG? (Stack memory too small)

What if we are creating student records based on CAO and we don't know how many students are going to apply for NUIG?

Memory Allocation

```
typedef struct studentStruct
{
        char name[30];
        int number;
} student;
```

```
student* newStudent(char* name, int number){
        student* s = malloc(sizeof(student));
        strcpy(s->name, name);
        s->number = number;
        return s;
}
```

Memory Allocation

```
typedef struct studentStruct
{
        char name[30];
        int number;
} student;
```

```
student* newStudent(char* name, int number){
        student* s = malloc(sizeof(student));
        strcpy(s->name, name);
        s->number = number;
        return s;
}
```

Creates a new student on heap.

Pointer to that student's allocated memory space is returned from malloc;

The sizeof(student) includes the name and number, so we have allocated enough space for everything at once.

We can call this as often as we like without running out of memory (theoretically)

We can free() memory if we know we don't need it

Memory Allocation

```
typedef struct studentStruct
{
        char name[30];
        int number;
} student;
```

What if a student has a name longer than 30 characters?

Memory Allocation

```
typedef struct studentStruct
{
        char name[30];
        int number;
} student;
```

What if a student has a name longer than 30 characters?

char name[30]; ⟶ char* name;

```
typedef struct studentStruct {
        char* name;
        int number;
} student;
```

Memory Allocation

```
typedef struct studentStruct
{
        char name[30];
        int number;
} student;
```

What if a student has a name longer than 30 characters?

char name[30]; ⟶ char* name;

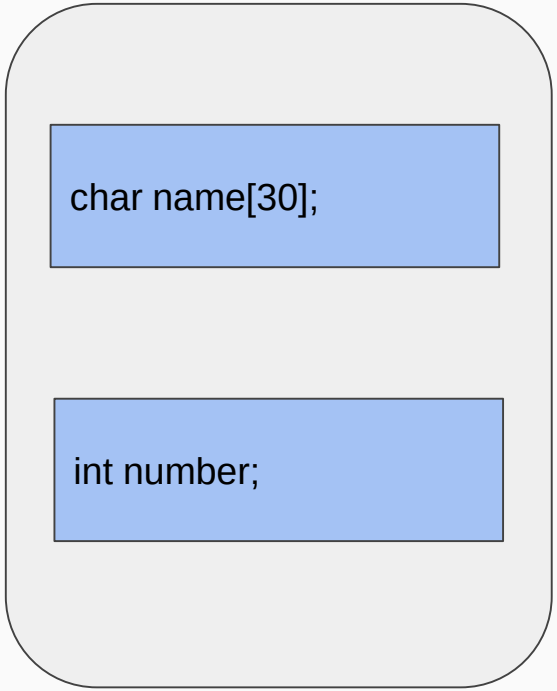
```
typedef struct studentStruct {
        char* name;
        int number;
} student;
```

What will sizeof(student) do?

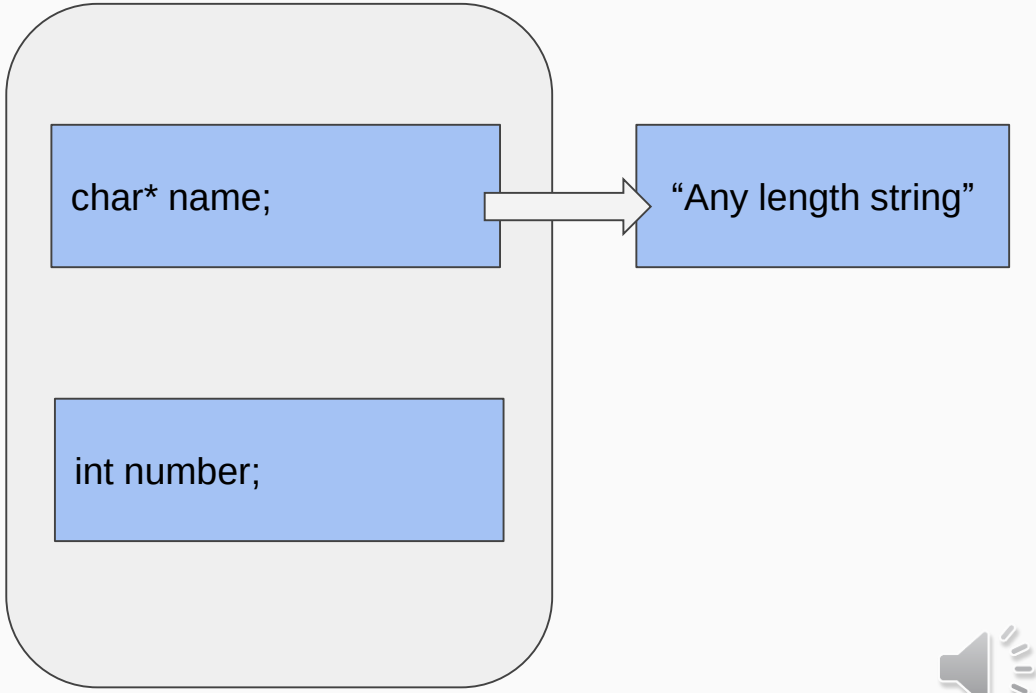
Or malloc(sizeof(student));

Memory Allocation

Old studentStruct



New studentStruct



Memory Allocation

```
typedef struct studentStruct
{
        char* name;
        int number;
} student;
```

```
student* newStudent(char* name, size_t nameSize,
                                    int number)
{
        student* s = malloc(sizeof(student));
        char* namePointer = malloc(nameSize);
        strcpy(namePointer, name);
        s->name = namePointer;
        s->number = number;
        return s;
}
```

Memory Allocation

```
typedef struct studentStruct
{
        char* name;
        int number;
} student;
```

```
student* newStudent(char* name, size_t nameSize,
                                    int number)
{
        student* s = malloc(sizeof(student));
        char* namePointer = malloc(nameSize);
        strcpy(namePointer, name);
        s->name = namePointer;
        s->number = number;
        return s;
}
```

Creates new student on heap.

Student name pointer is included...not the string itself.

Creates new student namePointer on heap.

Copies name into namePointer
Copies namePointer into student.

Memory Allocation

```
typedef struct studentStruct
{
        char* name;
        int number;
} student;
```

Create new student.

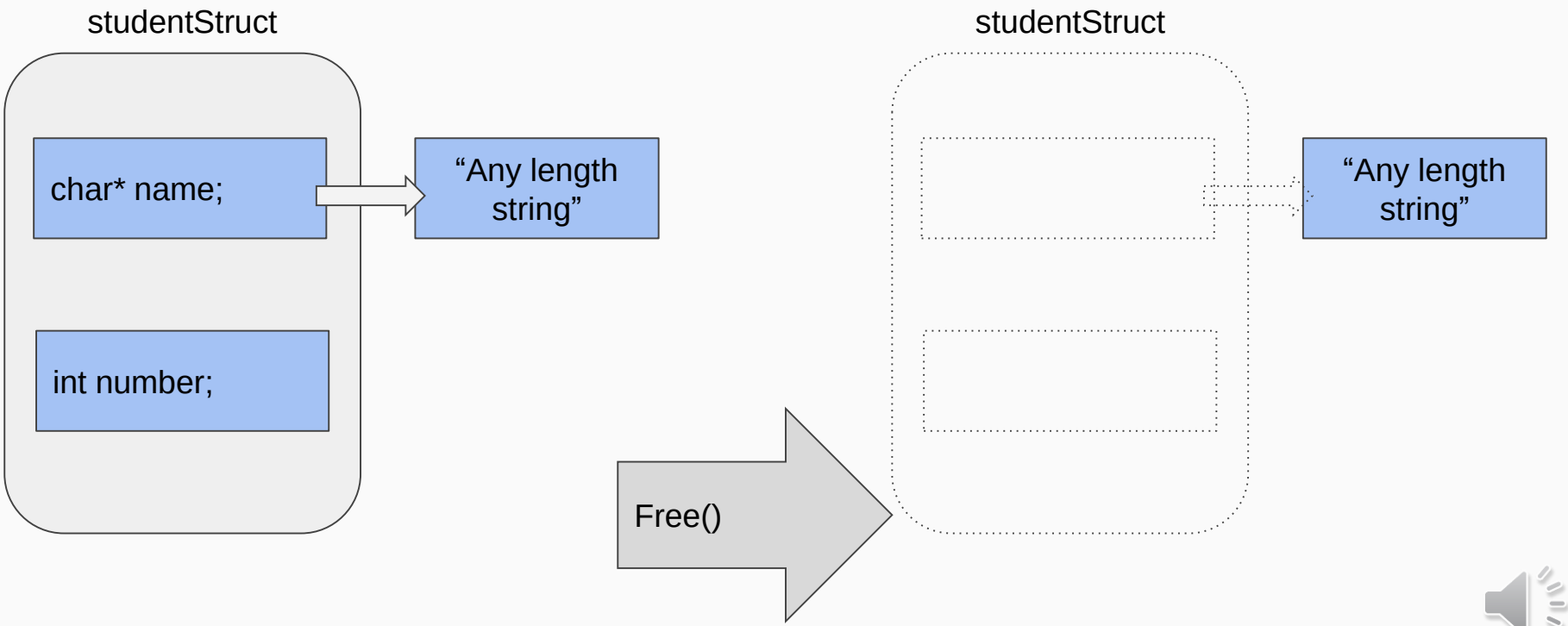
```
int i = 123456;
student* a = newStudent("Finlay",7, i);
```

Free student

```
free(a);
```

What about student->name?

Memory Allocation



Memory Allocation

```
typedef struct studentStruct
{
        char* name;
        int number;
} student;
```

```
void freeStudent(student* s){
        free(s->name);
        free(s);
}
```

Free heap memory for struct and struct pointer members (name in this case)

Preventing a memory leak where the student->name would not be freed.

You can only use free() on pointers from malloc(), calloc() and realloc().

Memory Allocation

```
typedef struct studentStruct
{
        char* name;
        int number;
} student;
```

```
void deleteStudent(student* s, size_t nameSize){
  memset(s->name, 0, nameSize);
  s->number = 0x0;
  free(s->name);
  free(s);
  s = NULL;
}
```

Free only flags the memory address as available.

The data may still exist at an address after free is called.

If data security is a concern, or to avoid bugs associated with accessing freed memory, we can manually change each value to NULL.

(an int can't be NULL, so we set number to 0)