Programming Paradigms

CT331 Week 4 Lecture 1

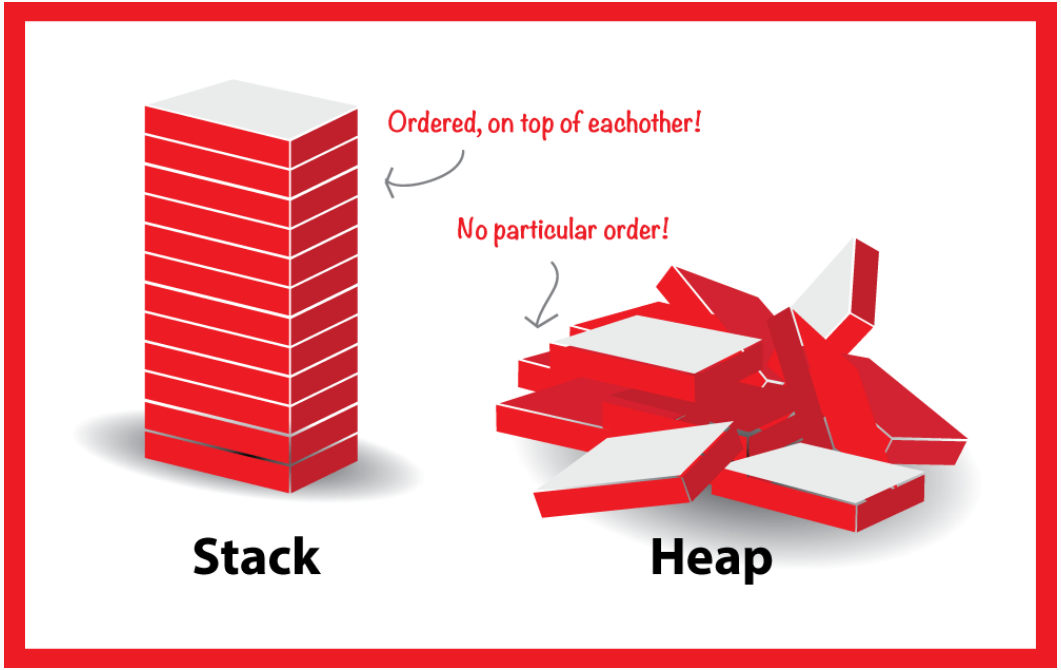
Finlay Smith

finlay.smith@nuigalway.ie

More Memory Allocation

Heap vs. Stack

Stack:



Heap vs. Stack

Stack:

- LIFO
- Push / Pop
- Limited size
- Limited access (scope)
- Very Fast

Heap vs. Stack

Stack:

- LIFO
- Push / Pop
- Limited size
- Limited access (scope)
- Very Fast

```
char a = ‘a’;

int b = 100;
Int c = 50;

void swap(int* x, int* y){
                int temp = *x;
                *x = *y;
                *y = temp;
}

swap(b, c);
```

First, a, b and c are pushed onto the stack.

When swap() is called,
x , y and temp are pushed onto the stack.

When swap returns, x, y and temp are popped from the stack.

Their memory is no longer in use.

Heap vs. Stack

Stack:

- LIFO
- Push / Pop
- Limited size
- Limited access (scope)
- Very Fast

```
char a = ‘a’;

int b = 100;
Int c = 50;

void swap(int* x, int* y){
                int temp = *x;
                *x = *y;
                *y = temp;
}

swap(b, c);
```

First, a, b and c are pushed onto the stack.

When swap() is called, x , y and temp are pushed onto the stack.

When swap returns, x, y and temp are popped from the stack.

Their memory is no longer in use.

What if we wanted to keep track of temp and use it later?

Heap vs. Stack

Heap:

- Unordered
- allocate / free
- unlimited size*
- global access (threads)
- Slower

Heap vs. Stack

Heap:

- Unordered
- allocate / free
- unlimited size*
- global access (threads)
- Slower

```
int b = 100;
Int c = 50;

void* swap(int* x, int* y){
        Int temp = *x;
        int* perm = malloc(int);
        perm = &temp;
        x = y;
        y = *perm;
    return perm;
}

void* p = swap(b, c);
…
free(p);
```

First, b and c are pushed onto the stack.

When swap() is called, x , y and temp are pushed onto the stack.

We allocate space in memory for perm using malloc.

When swap returns, x, y and temp are popped from the stack.

The memory allocated to perm is still in use!

Heap vs. Stack

Heap:

- Unordered
- allocate / free
- unlimited size*
- global access (threads)
- Slower

Ok, so we could just return temp in the same way but...

- Even when this function terminates, another function can access perm using that pointer.
- If we need to store a large amount of data (or an undeterminable amount of data) we can safely use heap.
    - No risk of stack overflow.
    - No risk of losing reference or accidental deallocation of memory.

Heap vs. Stack

Stack Pros:

- Fast
- Easy to manage

Stack cons:

- Limited size (stack overflow)
- Limited access (scope / closures)
- Cannot free memory

Heap Pros:

- Unlimited size
- Unlimited access

Heap Cons:

- Harder to manage (memory leaks!)
- Slower