Stack and Heap memory

David Croft

Coventry University

david.croft@coventry.ac.uk

2016

Overview

1 Stack and Heap

- Stack
- Heap

Stack and Heap

C

- Memory model used so far is a simplification.
- Actually two places in memory that variables can go.
  - The stack and the heap.
- Both are just regions of the same physical memory.
  - Are managed differently.

The stack

I

- When program is run, block of memory is allocated.
  - Called the stack.
- Each program has it’s own stack.
  - Each instance.
- As variables created and functions called they are put on the stack.
- When variables are destroyed/functions complete they are removed from the stack.
- Has limited size.
  - Recursive functions can fill the stack if not careful.

A

```
⇒   int add( int a, int b)
    {
⇒       int result = a+b;
⇒       return result;
    }

⇒   int sub( int a, int b )
    {
⇒       int result = a-b;
⇒       return result;
    }

⇒   int main()
    {
⇒       int var1 = 42;
⇒       int var2 = 1;
⇒       add(a,b);
⇒       sub(a,b);
⇒       return 0;
    }
```

int main()	
int var1	
int var2	
int add()	
int a	
int b	
int result	
↑ Stack	↑ Heap

The Heap

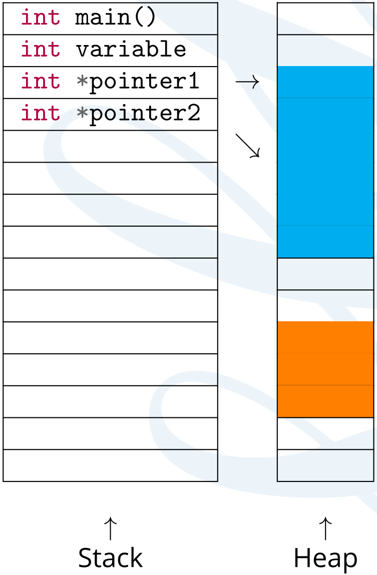
I

- Shared memory between all running programs.
- Very big in comparison to the stack.
- Dangerous, must remember to deallocate our memory.
  - Memory leaks.

```
⇒   int main()
    {
⇒     int variable = 42;
⇒     int *pointer1;
⇒     pointer1 = new int[6];

⇒     int *pointer2;
⇒     pointer2 = new int[3];

⇒     delete [] pointer1;
⇒     return 0;
    }
```



Differences

Stack

- Fast - processors typically have special instructions for dealing with stacks quickly.
- Contiguous - everything in one block, easier to know where to put next variable/function.
- Small - limited size.
  - Trying too variables will fill stack and cause “stack overflow”.

Heap

- Huge - relative to the stack.
- Dangerous - must remember to deallocate otherwise have memory leaks.

The End