Topic 2-1
Stack vs Heap

Welcome!

- **In this lecture we will discuss:**
  - How the memory is allocated for your C++ program.

Review

- **We've discussed many aspects of programming so far:**
  - Computer architecture
  - Data representation
  - Memory
  - First look at the stack and heap

Basic layout

THE STACK AND THE HEAP
(Growing from opposite ends)

WHERE VARIABLES THAT ARE ACCESSIBLE TO EVERYTHING (GLOBAL) RESIDE

WHERE THE COMPILED CODE RESIDES, USUALLY READ ONLY

Why do we need the additional memory

- Program behaviour changes as it runs.
- Variables are allocated and destroyed.
- Functions are called and returned from.

The Stack in Operation

- **When we call a function**
  - We allocate space for its return value on the stack.
  - We *push* function arguments onto the stack.
  - We start executing inside the function.
  - Local variables are *pushed* onto the stack as they are created.
  - We save the return value, jump back to the caller and free stack space.

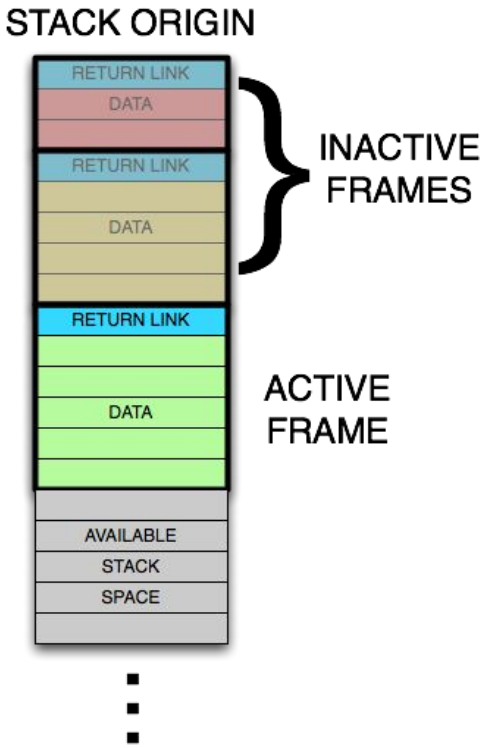
Push? Pop?

- **Think of the stack like pile of plates.**
- **As we call functions, we add plates.**
- **As we return from a function call, we remove a plate.**
- **We *push* things on to a stack and *pop* them off.**

How the Stack Works

- **Because we push *activation records* on to the stack for each function call, we can:**
  - Keep track of which variables are defined
  - Be sure what the parameters are
  - Work out where to deliver the result (if any)

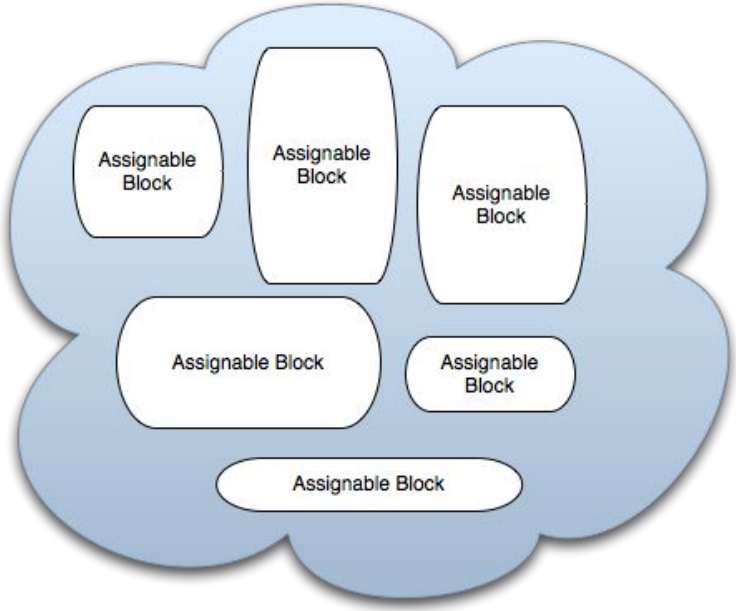
Stack Example



Memory Allocation

- **We use the stack to keep track of parameters and variables, but sometimes we want to allocate blocks of memory but do not know the size until our program runs.**
- **We can allocate chunks of memory from the *heap*.**

The Heap


Assignable Block
Assignable Block
Assignable Block
Assignable Block
Assignable Block
Assignable Block

Shared Space

Memory Leaks

- **Allocating from the heap is great for large things (arrays, structures and classes)**
- **We don't know what the location of memory will be (no fixed structure like the stack), so we have to use pointers.**
- **If we forget about heap allocations then we can *leak* memory and it becomes unavailable to us.**

Good Practice

- **Keep track of your memory!**
- **Don’t forget to destroy your heap allocations when you’re done.**
- **Only use as much memory as you need.**
- **Only call functions as deeply as you need and return as soon as you can.**

Memory Organization

- **Knowing how memory is organized and how programs work gives you great control over what your program can do**

Next Topic

Accessing/allocating memory using pointers.