CRICOS PROVIDER 00123M

School of Computer Science

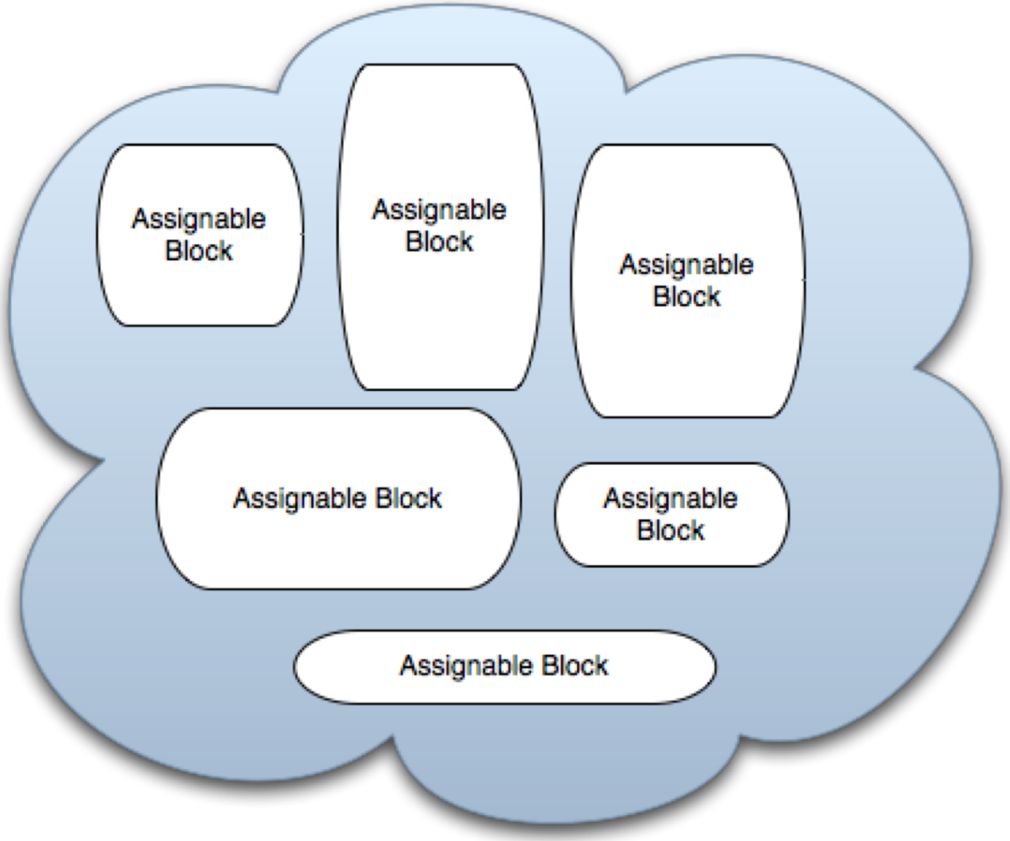
COMP SCI 1103/2103 Algorithm Design & Data Structure

Stack & Heap

Previously on ADDS

- What are pointers?
- How can we create a pointer?
- How can we make use of a pointer?
- Pointer Arithmetic
- Arrays
- C-strings

The Heap

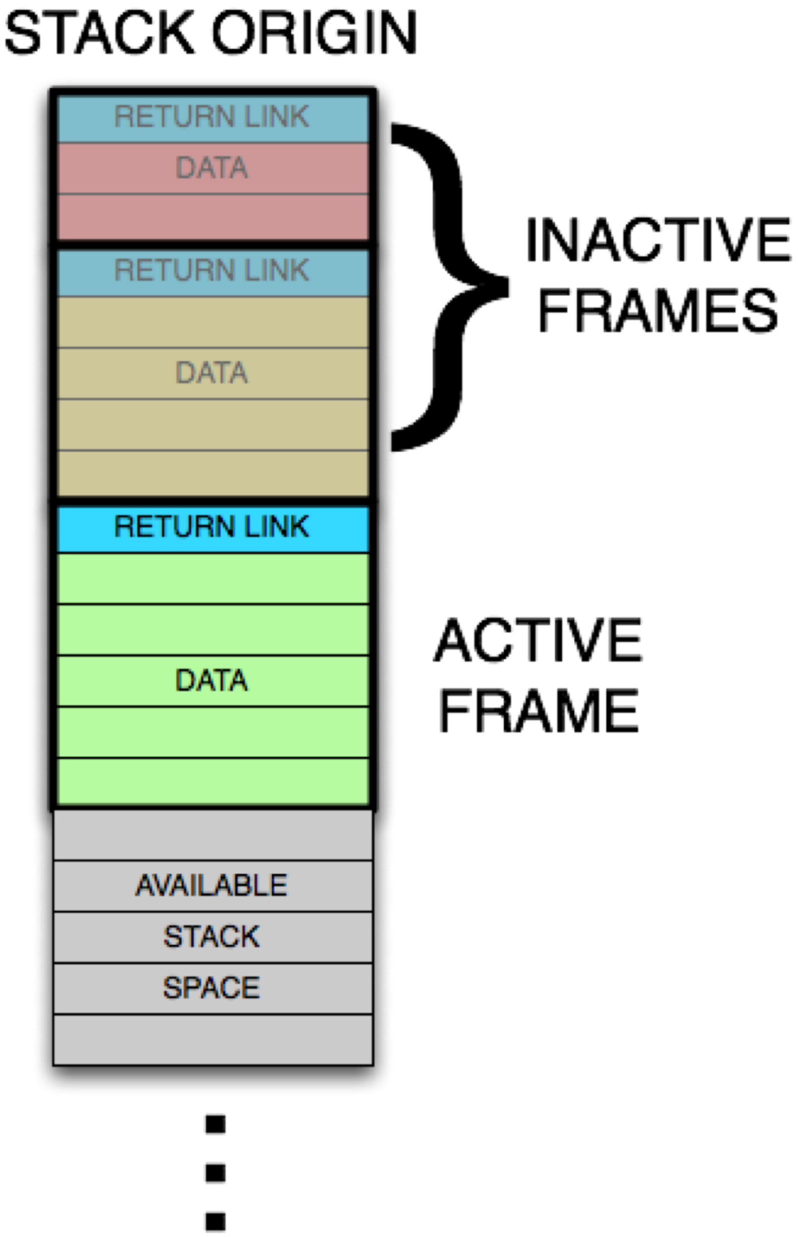


An area reserved for dynamic variables. It is also called the freestore.

Two drawbacks:

1. Searching
2. Heap fragmentation

The Stack



Last In First Out

For allocating memory to some new variable, we just need to keep track of where the free block starts.

Stack and Heap

- The stack keeps track of all of the variables and parameters that you are currently using - also called an activation record.
    - Call too many functions - run out of stack! (stack overflow)
- The heap, or freestore, allows you to create dynamic variables and refer to them with pointers, with the *new* command. Use *delete* to hand it back.
    - Use too much of it and calls to *new* will fail!

Stack

- Variables created on the stack will go out of scope and automatically deallocate when a function returns.
- Much faster to allocate in comparison to variables on the heap.
- Implemented with an actual stack data structure.
- Stores local data, return addresses, used for parameter passing.
- Can have a stack overflow when too much of the stack is used.
- Data created on the stack can/cannot be used without pointers?
- When can you use the stack? If you know exactly how much data you need to allocate before compile and it is not too big.
- Usually has a maximum size already determined when your program starts.

Heap

- Variables on the heap must be destroyed manually after use and there is no scope.
- Slower to allocate.
- Used on demand to allocate a block of memory for use by the program.
- Can have fragmentation after a lot of allocations and deallocations.
- To access heap variables, you need pointers.
- Can have allocation failures.
- You would use the heap if you don't know exactly how much data you will need at runtime or if you need to allocate a lot of data.
- May lead to memory leaks.

Review (not just a review) of stack and heap

- We use *new* to get a chunk of new memory from the stack.
  - **No! From the heap.**
- Heap allows us to make changes to our memory allocation while the program is running, without using pointers.
  - **No! we do need pointers for this.**
- Where are these variables stored?
  - int a;
  - int * ptr= &a;
  - int * ptr= new int;
- Is it possible to store a pointer in Heap? **Double pointers**
  - int ** ptr2= new int *; //which pointer is in heap?
- What can be stored in ptr2?
  - It's address of an integer pointer. So, is it ok to do this: ptr2= &ptr?
- What can be stored in *ptr2?
  - It's address of an integer. So, is it ok to do this: *ptr2= &a?

Variables in C++

- There are three basic descriptions for how C++ handles the memory management of variables:
  - Global
  - Automatic
  - Dynamic
- What do each of these words mean to you?

Global Variables

- Global variables are declared outside of any function definition.
    - When `main` starts executing, these variables are already defined!
- They exist as long as the program is running.

Automatic Variables

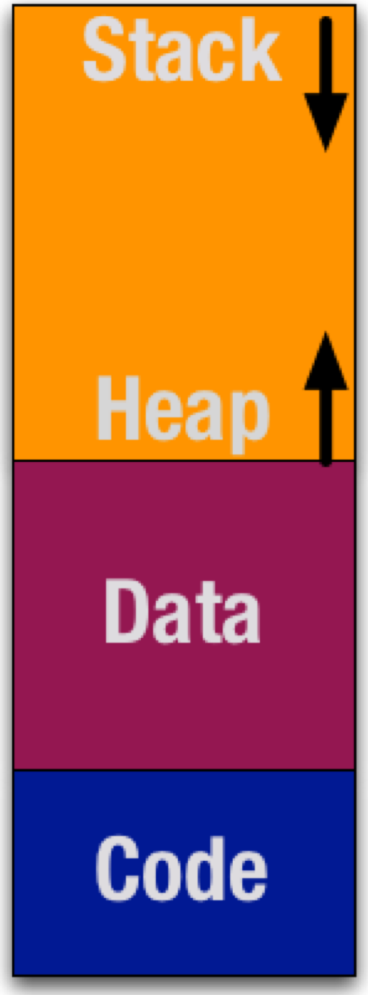
- Automatic variables are created for you to use, automatically, whenever a function is called.
  - Local to a function or the main part of the program
  - Once the function is returned, the variables are destroyed

Dynamic Variables

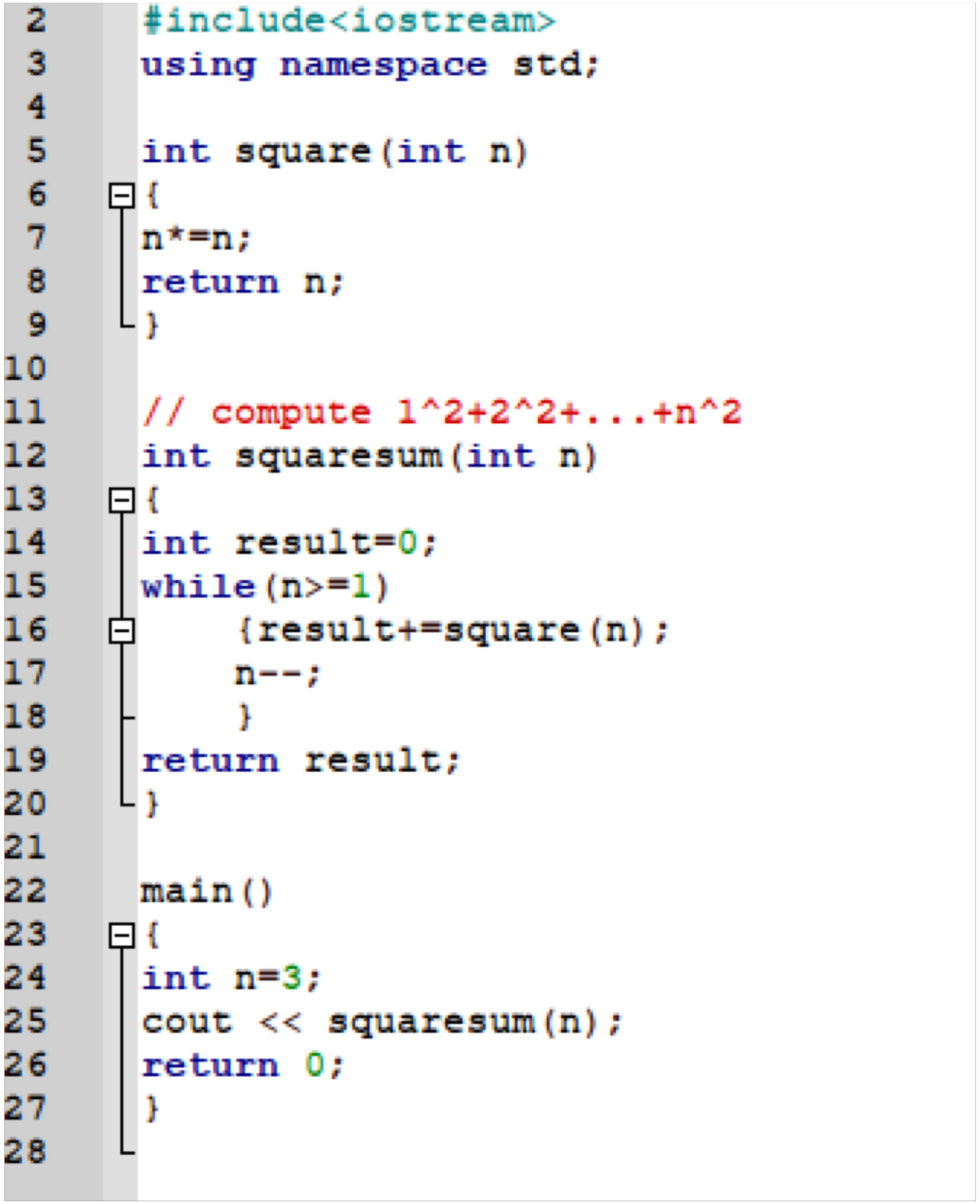
- Dynamic variables are created and destroyed as the program is running.
  - These variables are created with *new* and destroyed with *delete*.
  - No automatic deletion during runtime!

Where are they stored?

- The stack and heap are in a shared area.
- Where do we store each of these types of variables?
- Pre-process data area
  - Each process has a special data area set aside for variables that are global.
  - Other information for the process is also stored here.



Example



```
#include<iostream>
using namespace std;

int square(int n)
{
n*=n;
return n;
}

// compute 1^2+2^2+...+n^2
int squaresum(int n)
{
int result=0;
while(n>=1)
    {result+=square(n);
    n--;
    }
return result;
}

main()
{
int n=3;
cout << squaresum(n);
return 0;
}
```

More about stack

- All automatic variables are stored in here.
- As we call a function, another frame is put onto the stack to hold the space for the variables that we are currently using.
- When the function ends, it is removed. How?
- What if the function calls itself, without stopping?
  - recursion