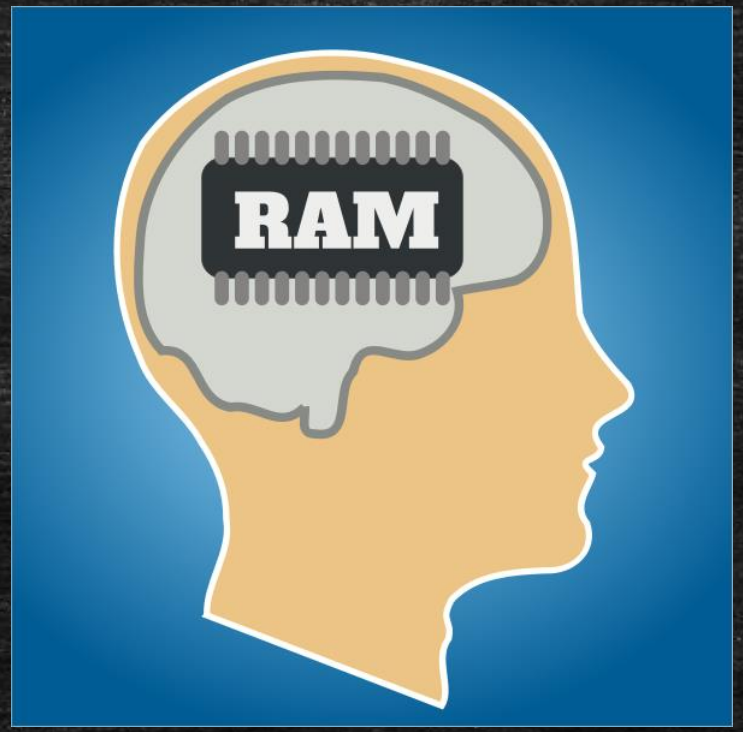
Memory & Pointers



CS 150 – C++ Programming I
Lecture 17

Review: *vector* Basics

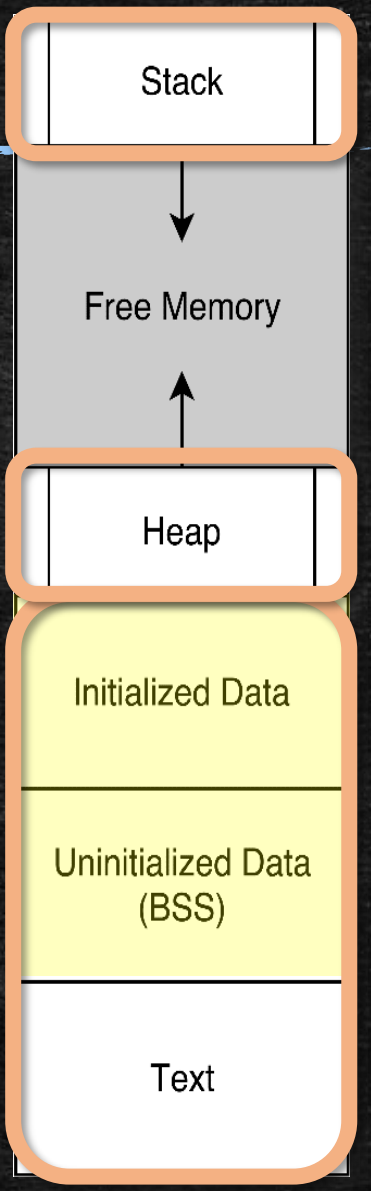
- Create variables by specifying the base type
  -
```
vector<int> v1,  // empty
    v2(10),      // 10 ints 0 initialized
    v3(3, 4),    // 3 ints initialized to 4
    v4{3, 4};    // 2 ints - 3 and 4
```
- Access with `at()`, `front()`, `back()`, or `[]`
- Members: `push_back()`, `pop_back()`, `size()`
- Loops: `for(auto e : v)... // &, etc`

More vector Function Practice

- Passing `vector` as function parameters
  - 1. Input parameters: use `const vector<?>&`
  - 2. In-Out or Output: use `vector<?>&`
  - 3. Never pass a `vector` by value
- You can also return a `vector` from a function
  - Create an empty `vector`, fill it in, then return it
  - Makes a copy; output parameter more efficient
    - Modern compilers may "optimize" this copy away
  - You may return a `vector` of indexes instead
- Exercise: Complete the three functions

Examining Memory

- When you run your program runs:
  - The static area has code (text) and globals
  - The stack has runtime info and locals
  - The freestore (heap) has dynamic storage
- Three related concepts
  - Scope (block, file) where can I see a name?
  - Duration (static, auto, dynamic) how long does an item stay in memory?
  - Linkage (internal, external, none) which files & functions can the item be used in?



Exploring Memory Layout

- Exercise: open *f1.cpp* and *f2.cpp* side-by-side

f1.cpp

```
1  #include <iostream>
2  using namespace std;
3
4  int a = 3;
5  static int b = 7;
6
7  static void d()
8  {
9      int e = 5;
10     static int f = 6;
11     cout << a << b << e << f << endl;
12     f++;
13 }
14
15 void c()
16 {
17     cout << a << b << endl;
18     d();
19 }
20
```

f2.cpp

```
1  extern int a;
2
3  static int b = 8;
4  void c();
5  int main()
6  {
7      c();
8      a = 12;
9      c();
10 }
11
12
```

Variables, Sizes & Addresses

- All variables have three attributes:
  - Name: used instead of memory address
  - Type: determines what can be stored in the variable and the valid operations on the data
  - Value: the data or state stored
- Address-of operator (&) returns location of object
- `sizeof` operator returns the storage size in bytes
- Exercise: Print address & size of each variable
  - `n lives at xxxx and uses xx bytes`

Pointer Basics

- Pointers are variables that store addresses
  - Pointer variables thus point to other variables
- Create a pointer like this:
  - `typeOfPointee * nameOfPointer;`
  - `int * iPtr;`
  - Read (right to left) as: *iPtr is a pointer to int*
- Space before or after * doesn't matter
  - But, every pointer variable needs it's own *
  - `int * iPtr, i; // iPtr is a pointer; i is an int`

Initializing Pointers

- Unless you initialize a pointer, it points to a random address in memory!!! (very, very bad!!!)
- You may initialize using:
  - `nullptr` or `0` - signifies the pointer is "unused"
    - No other literal (integer) values permitted
  - A memory address returned from the `&` operator
  - A memory address returned from `new` operator
  - A function that uses one of these techniques
- Exercise: complete part 2 of *`variables.cpp`*

The Pointers Value(s)

- A pointer's explicit value is the address it contains
  - It's indirect value is the value of the variable it points to
  - Retrieved by the indirection or dereferencing operator (*)
  - `int a = 3; int *pa = &a;`
  - `cout << a << ", or " << *pa << endl;`
  - Note the * means something else when declaring the pointer
- The pointer itself also has an address which can be stored
- Exercise: complete part 3 of *variables.cpp*
  - ip contains xxxx, is stored at xxxx, and points to xxxx.

Pointers as Output Parameters

- A pointer input parameter should be const
  - `int f(const int* p) // can't change *p`
- A pointer output parameter acts like a reference
  - `int f(int * p) // can write to *p`
- Exercise: a function that takes a pointer to `int`
  - Fill the pointer's indirect value with a random number
  - In `main`, create an `int` variable (unititialized)
  - Pass its address to the function
  - Print the value before and after calling the function.