CSE 374: Programming Concepts and Tools

Eric Mullen
Spring 2017
Lecture 20: C++

Administrivia

- Git repositories for HW6: how’s it going?
  - Clone and add *something* ASAP, to make sure it’s working

AMT vulnerability

C++

- C++ is a large language. It contains:
  - All of C
  - Classes and Objects (sorta like Java)
  - Little Conveniences (I/O, new/delete, overloading, pass-by-reference, bigger library)
  - Namespaces
  - Lots we won’t touch (const, more casts, exceptions, templates, multiple inheritance)

Our Focus

Object Oriented programming in a C-like language will help you understand C and Java better.

- Objects can live on the heap or the stack
- Memory management is still manual
- Lots of ways to go wrong still
- Still headers and implementation files
- Allocation and initialization are still separate, but easier to “construct” and “destruct”
- Programmer has more control over how method calls work

Resources

- Lectures and sample code will be enough for your 1 (small) C++ assignment (HW7)
- Book called the C++ Primer is good
- cplusplus.com is a very helpful resource (especially for the standard library)

Hello World in C++

```
#include <iostream>
int main() {
    std::cout << “Hello, World!” << std::endl;
    return 0;
}
```

Differences from C: new style headers, namespaces, I/O via streams

Differences from Java: not everything is a class, any code can go in any file, can write just procedural code

Compiling

- Almost the same as C, with a slightly different compiler

```
g++ -Wall -g -std=c++11 -o hello hello.cc
```

- The .cc extension is just a convention (just like .c for C) but has other options (.cpp, .cxx, and .C are also C++ files)

- Still uses the C preprocessor

I/O

- Operator << takes an ostream and (various things) and outputs it, then returns the stream

```
std::cout << 3 << “hi” << f(x) << ‘\n’;
```

- Operator >> takes an istream and (various things) and reads input into the things

```
int x; std::cin >> x;
```

<< and >>

- We can think of << and >> as keywords, but they are really something else
- We call them “operators”, and we can “overload” them for different pairs of types
- In C they mean “left-shift” and “right-shift” for numeric types, still works in C++
- Use another cool feature of C++ to get input (coming soon)

Namespaces

- In C, all non-static functions in the program need different names
  - Even an OS with 10 million lines
- Namespaces (sorta like Java packages) let you group top level names

```
namespace thespace { <definitions> }
```

- Example: Entire standard library is in namespace `std`
- To access a namespace, use `thespace::some_fun()`

Using

- In order to not have to always write `space::fun()`, you can have a *using declaration*

- Example:

```
#include <iostream>
using namespace std;
int main() {
  cout << “Hello World” << endl;
  return 0;
}
```

Classes and Objects

- Like Java:
    - Fields vs methods, static vs instance, constructors
    - Method overloading (functions, operators, and constructors too)
- Not like Java:
    - access-modifier (public/private) syntax and default
    - declaration separate from implementation
    - funny constructor syntax, default parameters
- Nothing like Java:
    - Objects vs. pointers to objects
    - Destructors and copy-constructors
    - virtual vs. non-virtual (coming soon)

Stack vs Heap

- Java: cannot stack allocate an object
- C: can stack allocate a struct, then initialize
- C++: stack allocate and call a constructor: `Thing t(100)`
- Java: `new Thing(..)` calls constructor, returns pointer to heap allocated object
- C: use malloc, then initialize, must free once later, uses untyped pointers
- C++: Like Java, `new Thing(..)` but can also do `new int(42)`. Like C must deallocate, but must use delete instead of free.

Destructors

- An objects destructor is called just before the space for it is reclaimed
- A common use: reclaim space for heap allocated things pointed to (first calling their destructors)
- Meaning of `delete x`: call destructor, then reclaim space
- Destructors also get called for stack objects, when they leave scope
- Advice: Always make destructors virtual (learn why soon)

Arrays

- Create a heap allocated array of objects: `new A[10]`
- Calls default (0 argument) constructor for each element
- Create a heap-allocated array of pointers: `new A*[10]`
- More like Java, but not initialized
- As in C, `new A()` and `new A[10]` both have type A*
- Unlike C, to delete non-array write `delete e`
- Unlike C, to delete area write `delete[] e`
- Otherwise undefined behavior (sea monsters)

Call by Reference

- In C, arguments get copied
  - copying a pointer means pointer to same thing
- Same in C++, but you can also use a reference parameter (& before name when declaring function)
- `void f(int& x) { x = x+1;}`
- Called: `f(y)`
- Writes to y in callers context

Copy Constructors

- In C, we know x=y or f(y) copies y (if a struct, then member wise copy)
- Same in C++, unless a copy-constructor is defined, then do whatever that code says
- A copy-constructor takes a reference parameter (else we'd need to copy, but that's what we're defining…)
- Copy constructor vs assignment:
    - Copy constructor initializes new space to be a copy
    - Assignment replaces the value in existing space with a new one: may need to clean up old state

const

- const can appear in many places in C++ code: means that value doesn't change (but can be subtle, especially with pointers)

- Examples:

```
const int default_length = 125; //better than
                                //#define

void examine(const thing &t);

int getX() const;
```

- Checked by compiler, strong guarantee (unless you cast)