CSE 374: Lecture 23

Introduction to C++

Object Oriented Programming

- **Encapsulation**
  - Discrete portions of code keep state and implementation private while providing public interfaces
- **Abstraction**
  - The high-level interface is exposed to users without detailing underlying code.
- **Inheritance**
  - Classes can be derived from other classes allowing for shared code.
- **Polymorphism**
  - Subclasses implement methods of superclasses to allow for a consistent interface.

C++ Hello, World!

```
#include <cstdlib>
#include <iostream>

const int CURRENT_YEAR = 2019;

using namespace std;

// REFERENCE
void pig(string& s) {
  char first = s[0];
  s = s.substr(1);
  s += first;
  s += "ay";
}
```

```
int main() {
  // stack-allocated array: int arr[100];
  // C++ style heap allocation:
  int* arr = new int[100];

  // C++ style array deletion:
  delete [] arr;
  // Use "delete x;" for things non-arrays.

  cout << "What is your name? ";
  string name;
  cin >> name;
  pig(name);
  cout << "What year were you born? ";
  int year;
  cin >> year;
  const int age = CURRENT_YEAR - year;
  cout << "Hello, " << name << "!" << endl;
  cout <<"You're "<<age<< " years old"<< endl;
  return EXIT_SUCCESS;
}
```

So, what different with C++?

- File Names (instead of *.c)
  - *.cc or *.cpp or *.cxx
- Compiler (instead of gcc)
  - $g++
- Preprocessor (still uses C preprocessor)
  - But #include <cstdlib>

- Still use *.h for header files

- Basically does the same thing as <stdlib.h>

Namespaces

- Group code logically
- Can re-use names for each namespace
- Can nest namespaces
- Disambiguate with :: syntax
- Can avoid using the prefix with `using namespace foo`
  `doSomething(3)`
- If you are using a namespace in a header, you must also use the namespace in the source code (.cpp)

```
namespace foo {
    int doSomething(int x);
}

namespace bar {
    int doSomething(int x);
}

int main() {
    foo::doSomething(3);
    bar::doSomething(3);
}
```

I/O in CPP

Std library include a `cout` and a `cin` function

Operators ‘>>’ and ‘<<’ act like shell redirection

Operators ‘>>’ and ‘<<’ take left and right operands and return a stream

Use namespace std or

use `std::cout` & `std::cin`

```
using namespace std

cout << "What is your name? ";
string name;
cin >> name;

cout << "When were you born? ";
int year;
cin >> year;
```

Pass by reference

- In C: all function arguments are copies
  - Pointer arguments pass a copy of the address value
- In C++: Can do the above
  - but can also use a “reference parameter” (& character before var name)
  - As though the calling line wrote pig(&name) and in ‘pig’ every ‘s’ is a ‘*s’

```
void pig(string& s) {
  char first = s[0];
  s = s.substr(1);
  s += first;
  s += "ay";
}
```

```
string name;
cin >> name;
pig(name);
```

Const

In C++ we also have the new "const" keyword, which says "this thing must not change". We can use this to declare global constants:

```
const int CURRENT_YEAR = 2018;
```

Global constants look a lot like global variables, but they are OK stylistically whereas regular global variables are not because the "const" keyword says that this value CANNOT CHANGE.

```
// This won't compile.
 CURRENT_YEAR = 2038;
```

New / delete

In C:

```
int* arr = (int*) malloc(sizeof(int) * 100);
free(arr);
```

In C++, we have a nicer syntax for this that does the same thing:

```
int* arr = new int[100];
delete [] arr;
```

We can also do this for non-array types:

```
int* x = new int(4);        // x stores the value 4.
delete x;
```

Arrays

- Create a heap-allocated array of objects: `new A[10];`
  - Calls default (zero-argument) constructor for each element
  - Convenient if there's a good default initialization
- Create heap-allocated array of pointers to objects: `new A*[10];`
  - More like Java (but not initialized?)
- As in C, `new A()` and `new A[10]` have type `A*`
- `new A*` and `new A*[10]` both have type `A**`
- Unlike C, to delete a non-array, you must write `delete e`
- Unlike C, to delete an array, you must write `delete [] e`

Resources

Best place to start: C++ Primer, Lippman, Lajoie, Moo, 5th ed., Addison-Wesley, 2013

Every serious C++ programmer should also read: Effective C++, Meyers, 3rd ed., Addison-Wesley, 2005

Best practices for standard C++

Effective Modern C++, Meyers, O'Reilly, 2014

Additional "best practices" for C++11/C++14

Good online source: cplusplus.com