Lecture 2, September 4

Intro to C/C++

Instructor: Prashant Shenoy, TA: Shashi Singh

1 Introduction

C++ is an object-oriented language and is one of the most frequently used languages for development due to its efficiency, relative portability, and its large number of libraries. C++ was created to add OO programming to another language, C. C++ is almost a strict superset of C and C programs can be compiled by a C++ compiler. In fact, any of the C++ assignments in 377 can be completed strictly using C, but you will probably be more comfortable using C++.

When comparing C++ and Java, there are more similarities than differences. This document will point out the major differences to get you programming in C++ as quickly as possible. The highlights to look at are C++'s handling of pointers and references; heap and stack memory allocation; and the Standard Template Library (STL).

2 Basic Structure

First, we compare two simple programs written in Java and C++.

Java Hello World (filename: `HelloWorldApp.java`):

```
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

C++ Hello World (filename: `HelloWorld.cpp`):

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

int main (){
    cout << "Hello World!" << endl;
    return 0;
}
```

In Java the entry point for your program is a method named `main` inside a class. You run that particular `main` by invoking the interpreter with that class name. In C++, there is only ONE `main` function. It must be named `main`. It cannot reside within a class. `main` should return an integer - 0 when exiting normally. This points out one major difference, which is C++ can have functions that are not object methods. Main is one of those functions.

The *using* and *include* lines in the C++ program allow us to use external code. This is similar to *import* in Java.

3 Compiling and Running

Java:	C++:
`javac HelloWorldApp.java`	`g++ -o HelloWorld HelloWorld.cpp`
`java HelloWorldApp`	`./HelloWorld`

In Java, you produce bytecode using `javac` and run it using the Java just-in-time compiler, `java`. C++ uses a compiler, which creates a standalone executable. This executable can only be run on the same platform as it was compiled for. For instance, if you compile HelloWorld for an x86 Linux machine, it will not run on a Mac.

The ./ at the beginning of run line says to run the HelloWorld that is in the current directory. This has nothing to do with C++, but many Unix/Linux systems do not include the current directory in the search path. Also, it makes sure you get the *right* HelloWorld. For instance, there is a Unix program called *test* that can be easily confused with your own program named *test*.

4 Intrinsic Types

Java:	C++:
`byte myByte;`	`char myByte;`
`short myShort;`	`short myShort;`
`int myInteger;`	`int myInteger;`
`long myLong;`	`long myLong;`
`float myFloat;`	`float myFloat;`
`double myDouble;`	`double myDouble;`
`char myChar;`	`char myChar;`
`boolean myBoolean;`	`bool myBoolean;`

So the differences between Java and C++ intrinsic types are: C++ does not have a separate type for bytes, just use char; and the boolean type is named bool.

5 Conditionals

Java:

```
boolean temp = true;
boolean temp2 = false;

if (temp)
    System.out.println("Hello World!");

if (temp == true)
    System.out.println("Hello World!");

if (temp = true) // Assigns temp to be true
    System.out.println("Hello World!");
```

C++:

```
bool temp = true;
int i = 1;

if (temp)
    cout << "Hello World!" << endl;

if (temp == true)
    cout << "Hello World!" << endl;

if (i)
    cout << "Hello World!" << endl;
```

Conditionals are almost exactly the same in Java and C++. In C++, conditionals can be applied to integers—anything that is non-zero is true and anything that is zero is false.

NOTE: Be very careful in both C++ and Java with = and ==. In both languages, = does an assignment and == does not - it tests equality. In C++, you can now do this with integers. For instance:

```
if (x = 0){
}
```

sets x to be 0 and evaluates to false. Unless you know what you are doing, always use == in conditionals. In Java, the double form of the operators gives short-circuiting: && and ||. In C++, it does the same thing.

6 Other control flow

For loops, while, and do-while are the same syntax. C++ also has switch statements with the same syntax. Remember break.

7 Pointers and Reference Variables

Pointers and reference variables are the most difficult concept in C++ when moving from Java. Every object (and simple data types), in C++ and Java reside in the memory of the machine. The location in memory is called an address. In Java you have no access to that address. You cannot read or change the address of objects. In C++ you can.

In C++, a pointer contains the address of an object or data type. Pointers point to a specified type and are denoted with *.

```
int *ptr;
```

The variable `ptr` is a pointer to an integer. At the moment ptr does not contain anything, it is uninitialized. However, we can find out the address of some integer, using &, and store it in ptr. For instance:

```
int *ptr, *ptr2;
int x = 5;
int y = 4;

ptr = &x;
ptr2 = &y;
```

At the end of that example, ptr contains the address of x, and ptr2 contains the address of y. Additionally we can *dereference* the get the value of what ptr points to:

```
int *ptr;
int x = 5;

ptr = &x;

cout << *ptr << endl;   //prints 5
```

There are other tricky things you can do with pointers, but that is all you should need for 377. It you want antoehr reference on the subject, take a look at: `http://www.codeproject.com/cpp/pointers.asp`.

There is something else in C++ called a reference variable. These are most useful in function arguments discussed later.

7.1 Assignment

In C++, the assignment operator works a little different than in Java. For simple data types, it works exactly the same. However, for objects it works differently.
In Java, assignment copies a reference to the object. In C++, it COPIES the object. If you want to copy a reference, then use pointers. For instance, in C++:

```
SomeClass  x,  y;
SomeClass  *a;

x=y;  // This copies object y to x.  Modifying x does NOT modify y.

a=&x;  // This copies a reference to x.
       // Modifying the object a points to modifies x.
```

7.2 Object Instantiation

In Java, if you declare and instantiate an object like this:

```
SomeClass  x;

x = new SomeClass();
```

In C++, if you declare an object, it instantiates it for you:

```
SomeClass  x;
```

7.3 The -> Operator

For pointers to objects you can call methods and modify variables like so:

```
SomeClass  x;
SomeClass  *a;

a=&x;
(*a).SomeMethod();
```

So we have dereferenced a and called its SomeMethod. This is ugly. We can use the -> operator to do the same thing:

```
SomeClass  x;
SomeClass  *a;

a=&x;
a->SomeMethod();
```

8 Global Variables, Functions and Parameters

In C++, you can have variables that are not members of objects. That is called a global variable. That variable is accessible from any function or object in the same source file. **NOTE:** Global variables are generally considered poor programming form. Use them judiciously when necessary.

Similar to global variables, there are functions that are not methods. For instance main is not a method of any object. Similarly you can create new functions that are not contained in objects. These functions can take parameters just like methods. For instance:

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

void foo(int i){
  cout << i << endl; // Prints 1
}

int main (){
  foo (1);

  return 0;
}
```

Similar to Java, C++ functions can return nothing (void), simple data types, or objects. If you return an object in C++, you are reutrning a COPY of that object, not a reference. You can return a pointer to an object, if you want to return a reference.

But here is where Java and C++ have one major difference: parameters. In Java simple data types are passed by value (a copy), and objects are passed by reference. In C++ both simple data types and objects are passed by value! However, functions would be fairly useless in C++ if we couldn't change what we passed. That is where pointers come in. Quite simply:

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

void foo(int *i){
  *i = 6;
}

void bar(int i){
  i = 10;
}

int main (){
  int i = 0;

  foo (&i);
  cout << i << endl; // prints 6

  bar (i);
  cout << i << endl; // prints 6

  bar (&i);   // WOULD NOT COMPILE

  return 0;
}
```

In the above example, we have a function foo that takes a pointer to an integer as a parameter. When we call foo, we have to pass it the address of an integer. So foo is modifying the same i as main. The function bar

takes i by value and any changes made are lost. The last call to bar attempts to call bar with an address to an integer, not an integer, and would not compile.

In C++, there is a slightly simpler way of accomplishing the exact same thing:

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

void foo(int &i){
  i = 6;
}

int main (){
  int i = 0;

  foo (i);
  cout << i << endl;   // prints 6.

  return 0;
}
```

The integer i in foo is a reference variable. It takes care of passing the address, and dereferencing i when it is used in foo. This is cleaner and easier to understand, but both are valid. However, you cannot avoid the uglier form. If I give you a function to call then it will be of the first form.

9 Arrays

Java and C++ both have arrays. Indexes start at 0, and you index into them using **[ ]**. However, arrays behave a little differently in C++. First, the elements of the array do not need to be allocated with new, C++ allocates them for you. For instance, if you create an array of Objects, it is an array of Objects, not an array of references like in Java. Second, C++ arrays do not know how large they are. You have to keep track of this yourself. C++ will not stop you from going past the end or beginning of the array. This is a programming error and will often crash your program. Third, if you refer to an array without a subscript, C++ treats this is a pointer to the first element. For instance, in this program we pass a pointer to qux to foo, which modifies the contents of the first element.

```
#include <iostream>

using namespace std;

void foo (int bar[]){
  bar[0] = 5;
}

int main(){

  int qux[10];

  qux[0] = 10;
  foo (qux);

  cout << qux[0] << endl;   // Prints 5
```

```
}
```

You can also create an array of pointers.

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

void foo (int* bar[]){
  *(bar[0]) = 5;
}

int main(){
  int* qux[10];

  qux[0] = new int;
  *(qux[0]) = 10;
  foo(qux);

  cout << *(qux[0]) << endl;   // Prints 5
}
```

10 Structs and Classes

C++ has something called a struct. Think of it as a class with no methods, only public member variables. You can refer to the variables using '.'. If you hold a pointer to a struct, you can use the **->** operator. For instance:

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

struct foo{
  int a;
};

int main (){
  foo b, *c; // b is a struct, c points to a struct

  b.a = 6;
  c = &b;
  c->a = 7; // Remember c points to the struct b!

  cout << b.a << endl;   // prints 7.

  return 0;
}
```

As for classes, the syntax is a little different, but many things are the same. In Java, you have:

```
public class IntCell
{
  public IntCell( )
     { this(0); }
```

```
  public  IntCell(  int  initialValue  )
     {  storedValue  =  initialValue;  }

  public  int  getValue(  )
     {  return  storedValue;  }

  public  int  setValue(  int  val  )
     {  storedValue  =  val;  }

  private  int  storedValue;
}
```

and in C++, you have:

```
class  IntCell
{
  public:

  IntCell(  int  initialValue  =  0)
     {  storedValue  =  initialValue;  }

  int  getValue(  )
     {  return  storedValue;  }

  int  setValue(  int  val  )
     {  storedValue  =  val;  }

  private:

  int  storedValue;
};
```

The end of a C++ class has a semicolon. DO NOT FORGET IT. The compilation errors will not tell you it is missing. The compiler will give you strange errors you don't understand.

In C++, there is no visibility specifier for the class.

In C++, public and private are applied to sections inside the class. In Java, one constructor can call another. You can't do this in C++. The above syntax says that the default value is 0 if one is not passed. This must be a fixed, compile time value.

There is also a slightly nicer way to declare classes in C++. This is exactly the same as the previous definition, we have just split the interface and the implementation.

```
class  IntCell
{
  public:

  IntCell(  int  initialValue  );
  int  getValue(  );
  int  setValue(  int  val  );

  private:

  int  storedValue;
```

```
};

IntCell::IntCell   (int   initialValue   =   0){
  storedValue  =  initialValue;
}

int  IntCell::getValue(  ){
  return  storedValue;
}

int  IntCell::setValue(  int  val  )
{
  storedValue  =  val;
}
```

11 Operator Overloading, Inheritance

Yep, C++ has both.

12 Stack and Heap Memory Allocation

This is the second most difficult thing to get a handle on besides pointers.

12.1 Stack Memory

In a C++ function, local variables and parameters are allocated on the stack. The contents of the stack is FREED (read destroyed) when the function ends. For instance in this example:

```
int  foo  (int  a){
  int  b  =  10;

  return  0;
}
```

After foo ends you can't access a or b anymore. This shouldn't surprise you at all, Java works the same way. However, now that we have pointers you can do something really bad:

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

//  BAD

int*  foo  (){
  int  b  =  10;

  return  &b;
}

int  main(){
  int  *a;

  a  =  foo();
```

```
  cout << *a << endl;  // Print out 10?

  return 0;
}
```

In this example, we have a function foo that returns a pointer to a stack allocated variable b. However, remember when foo returns, the memory location of b is freed. We try to dereference the pointer to b in the cout statement. Dereferencing a *stale* pointer to freed memory, makes your program die.

12.2 Heap Memory

However, what if we want a variable to live on after the end of a function? There are two ways to do it. First is to make it global. Then it will live for the entire lifetime of the program. This is poor programming practice as well as not being dynamic. (How many objects do you need?)

Instead we allocate from the heap using new, just like in Java. However, new returns a pointer (a reference, just like in Java!). For instance:

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

// GOOD

int* foo (){
  int *b;

  b = new (int);

  *b = 10;

  return b;
}

int main(){

  int *a;

  a = foo();

  cout << *a << endl;    // Print out 10!

  return 0;
}
```

Now b is allocated on the heap and lives forever! But there is one slight problem in C++. Java knows when you are no longer using an object and frees it from the heap. C++ does not so you have to tell it when you are done. For instance in Java this is legal:

```
  bar qux;
```

```
  for (int i=0; i < 1000000; i++)
      qux = new bar();
```

Well it is legal in C++ too, but you have what is called a *memory leak*. You are allocating objects and never freeing them. In C++ you must do this:

```
  bar *qux;

  for (int i=0; i < 1000000; i++){
      qux = new (bar);
      delete (qux);
  }
```

Just remember that for **EVERY** new, you must do a delete exactly once. If you delete memory that has already been deleted once, you are deleting a stale pointer. Again, your program will die a horrible, nasty death. One strategy to diagnose this is to comment out your calls to delete and see if your program still works (but sucks up memory). Another good idea is to set pointers to NULL after you delete them and always check that the pointer is NOT NULL before calling delete.

13 Input, Output, Command Line Parameters

13.1 Input

Say you want to read something from a file.

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

int main(){

  int a;
  ifstream input;

  input.open("myfile");
  while (input >> a);
  input.close();
  input.clear();

  return 0;
}
```

This allows you to read a bunch of integers from myfile. You only need clear if you intend to use the input object again. We will give you most of the code you need for input.

13.2 Output

Say you want to print two integers. Very easy:

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

int main(){
```

```
  int a = 5, b = 6;

  cout << a << " " << b << endl; // Prints 5 6

  return 0;
}
```

The endl is an endofline character.

13.3 Command line parameters

Just follow the example:

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

int main(int argc, char * argv[]){

  int a;
  ifstream input;

  if (argc != 3){ // One for the program name and two parameters
     cout << "Invalid usage" << endl;
     return -1;
  }

  int a = atoi (argv[1]); // First parameter is a number
                          // atoi converts from characters to int

  input.open(argv[2]);     /// Second parameter is the name of a file
  input.close();

  return 0;
}
```