377 Student Guide to C++

©Mark Corner, edited by Emery Berger

January 23, 2006

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.[1] 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 Javam 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. In 377, we will generally tell you the necessary include files. For 377, always use the *std* namespace. We will explain `cout` later.

[1] There are a few things that you can do in C that you can't do in C++, but they are obscure constructions that you will not run into.

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:	C++:
`boolean temp = true;`	`bool temp = true;`
`boolean temp2 = false;`	`int i = 1;`
`if (temp)`	`if (temp)`
`  System.out.println("Hello␣World!");`	`  cout << "Hello␣World!" << endl;`
`if (temp == true)`	`if (temp == true)`
`  System.out.println("Hello␣World!");`	`  cout << "Hello␣World!" << endl;`
`if (temp = true) // Assigns temp to be true`	`if (i)`
`  System.out.println("Hello␣World!");`	`  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();
```

Pretty, huh?

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. However, in this class I will grant you some leeway in using them. Do not abuse this. I consider making loop control variables global a serious abuse.

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![2] 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

[2]This is more consistent than Java, and one of the things that C# fixes. C# passes EVERYTHING by reference including simple data types.

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.
Ugly, huh? 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.[3]

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
```

[3]This maintains compatibility with C.

```
}
```

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.[4]
In C++, public and private are applied to sections inside the class.[5]. 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;
```

[4]This only applies to a top-level class. See next section on inheritance.
[5]C++ also has protected, see the next section on inheritance

```
};

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. Do you need them for 377? Nope.

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 BAD 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 a horrible nasty death.[6]

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 GOOD 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! Great, but there is one slight problem in C++. Java knows when you are no longer using an object and frees it from the heap[7]. C++ does not so you have to tell it when you are done. For instance in Java this is legal:

```
  bar qux;
```

[6] Actually if you compile this program and run it, it will probably work. However, in a more complex program it will NOT

[7] This is called garbage collection

```
  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);
  }
```

Easy, right? Ok, 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;
}
```

14 The Standard Template Library

As you know already, it is tough to program without good data structures. Fortunately, most C++ implementations have them builtin! They are called the Standard Template Library (STL).
The two that you may need for 377 are *queues* and *maps*. You should know what these are already. Using the STL is pretty straightforward. Here is a simple example:

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

queue<int> myQueue;
int main(int argc, char * argv[]){

  myQueue.push(10);
  myQueue.push(11);

  cout << myQueue.front() << endl; // 10
```

```
  myQueue.pop();

  cout << myQueue.front() << endl; // 11
  myQueue.pop();

  cout << myQueue.size() << endl; // Zero
}
```

The type inside of the angle brackets says what the queue myQueue will hold. Push inserts a **COPY** of what you pass it. Front gives you a reference to the object, unless you copy it. I will make this clearer in class. Pop, pops it off the queue and discards it. Often you will have a queue or a map of pointers. Here is a more complex example you should now understand:

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

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;
}

// In a map the first parameter is the key
map<int, IntCell *> myMap;

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

  IntCell *a;
  int i, max = 100;

  for (i = 0; i < max; i++){
    a = new(IntCell);
    a->setValue(max-i);
    myMap[i] = a; // Inserts a copy of the pointer
```

```
  }

  for (i = 0; i < max; i++){
    a = myMap[i];
    cout << a->getValue() << endl;
    delete (a);
    myMap[i] = NULL; // Good idea?
  }
  myMap[0]->setValue(0);  // Quiz: can I do this?
}
```

Think about what the output from this should be. If you know, you are a C++ guru![8]

15 Crashes, Array Bounds, Debugging

C++ programs can crash for a lot of reasons. You will run your program and it will print out some strange message, such as "segfault" or "bus error". This means you have done something wrong in your program. The three most common errors are: accessing an array out of bounds, dereferencing a stale pointer, and memory allocation errors (such as deleting a stale pointer).
There is a tool named `gdb` that can help you diagnose some of these problems. The TAs will show you how to use it in office hours.

16 Misc

16.1 Assert

The assert function is a handy call in C++. If you assert something that is true, nothing will happen. If you assert something that is false your program will exit on that line and tell you it wasn't true.

```
int *ptr = NULL;

assert(ptr != NULL); // Peogram exits
```

16.2 Typedef

Typedef is just some way of defining one type as another.

```
typedef int* p_int;

p_int a;  // a is pointer to an integer
```

16.3 Exceptions

Yes, C++ has exceptions kind of like Java. Do you need them in 377? Nope.

16.4 Comments

Same syntax as Java.

[8] Ok, maybe not a guru, but you can probably pass 377.

16.5 #define

You can can globally replace a set of characters with another[9].

```
#define MAX 100

int baz[MAX]; // baz is an array of 100 integers
```

16.6 static

The keyword static means something in Java classes. It means the same thing in C++ (with some differences). Don't worry about using it in classes, you won't need to in 377.
However, it also means several other things in C++. If you put static before a global variable or function, it means it is only visible in this file. You will only be writing programs with one file, but you will be including files the instructors wrote. To ensure there are no names that conflict, we ask that you declare all your functions (except main) and global variables with static.
If you declare a variable inside a function to be static, it is just like a global variable, but only visible inside the function. In other words it keeps its value between calls to that function.[10]
Here is an example of both:

```
static int b;

static void foo(int a){
  static int x;
}
```

16.7 unsigned variables

There are other datatypes, such as "unsigned int". It is a purely positive integer (ie. no twos-complement). Don't do this:

```
void foo(unsigned int a){
  cout << a << endl;
}

int main(){

  int a = -1;

  foo(a);
}
```

Actually the compiler will stop you from doing this. You are trying to give it the wrong type.

16.8 typecasting

However, there is something you can do about it. You can change any type into any other type. As you can tell this is more dangerous than running with scissors. For instance, this WILL compile:

[9] This is done before compilation by the preprocessor
[10] This is because static variables are allocated in the data segment, not the stack.

```
void foo(unsigned int a){
  cout << a << endl;
}

int main(){

  int a = -1;

  foo((unsigned int)a);
}
```

The int in parentheses says make this an unsigned int.
You can also do this:

```
int main(){

  int *a;
  float b;

  a = (int *)&b;  // Take the address of b and pretend it is a pointer to an int
}
```

Don't.