Michael Brudno
C/C++ Tutorial (week of 8/10/1998)

C++ is what Java is based on. It also happens to be a horror to write anything in it. It is actually losing its vogue in the buiseness: most of the modern systems are actually written in either C, or more recently Java. C++ is a horror to debug, a horror to write in, and just a horror overall. However you have to know it for the final.

These notes will cover the differences between C/C++ and Java. For the purposes of this class it is not important that you know the difference between the two. In general, however, C++ is an Object Oriented Version of C: They took a very good, compact, programming language and added on an OOP part (Classes, templates and the like) to make a monster. So first the parts which they share:

Explicit pointers

In Java you have two basic types: primitive types and reference types. The two are quite distinct. Primitive types are only a few things: ints, shorts, doubles, etc. Anything which you as a programmer define is going to be a reference type. You can never get your hands on the object itself, you always deal with a pointer to it. In C there is no such distinction between primitives and objects: you can get your hands on the object itself, while you can also have pointers to primitive types. consider the following declarations:

```
int a = 5;  //Just like in java, declare a to be equal to 5
int* b;     //This is new: b will be a pointer to an int
b = &c;     //b will now be equal to the memory address of a
*b = 6;     //dereference b, and write 6 into the resulting memory address.
```

After you are done with these operations a will also be equal to 6. Lets consider what the box and pointers will look like after each line is executed:

```
      ___
1. a |5|   originally we have just the variable a, it’s value equal to 5.

2. b[ ]    we add b to our name space, and it initially contains an undefined value
(not null!)

3.      _
    a [5]
       ^
       |
    b [ ]    after line 3 b points to the location of a in memory.

4.      _
    a [6]
       ^
       |
    b [ ]    after line 4 the value to which b pointed was changed, and hence a was
too.
```

this concept of pointers is extended to classes, but this will be covered later.

If you remember early in the course we stressed the idea that in Java the arrays are actually pointers to some object in memory which hold the values of whatever we have in the array. Well, in C++ the story is not any different. When you declare an array, you actually create a pointer to it's first element. Thus the C++ equivalent of the java declaration

```
int[] k = new int[5];
```

is

```
int *k = new int[5];
```

It is important to note that *new* will always return a pointer (to the newly allocated object). However, since k is a pointer, we can do all sorts of weird stuff with it:

instead of writing *k[0]*, for instance, we could write **k*. Instead of *k[p]* we can write **(k+p)*. Thus in C++ you can add to pointers, subtract from them and to other weird stuff. For instance, given the declaration of k above, I could write:

k[0] = 7;
k = k+1;
printf("%d", k[-1]);

this will print out 7, because I readjusted k to point to the second element of itself, thus the 7. I am not sure whether all versions of C++ support negative indeces for arrays, but the one I am using (gcc) does.

Stack v Heap storage

C++ provides you with another way to declare arrays. The syntax

int k[5];

creates an array of 5 integers, but not on the heap (as you are used to), but on the stack. k is declared as a *const int **, a pointer whose value cannot change. Again, you don't have access to the array as is, just to a pointer to it. However the pointer cannot change (if you have another *int n[]*, the statement *k = n* is illegal.) When you try to return k from a proceudure you wrote you areactually returning a pointer to the array. But if the array was declared on the stack it will no longer be there when you exit from the procedure, and the pointer to it is actually pointing to garbage. This is perhaps the most common mistake of the beginning C++ programmer. And as usual, the compiler will not warn you, your program will execute ok, and you will be sitting there, wondering why that element you just assigned to be a 5 is suddenly 4955321 or some other weird number.

However always allocating memory on the heap is also dangerous: perhaps the most common bug in C++ is memory leeks, when you forget to de-allocate some piece of memory you created. As a result your program will run out of memory. Remember, unlike Java there is no automatic garbage collection done for you.

C++ also has classes, which are very similar to Java classes, and templates, which are not in Java. Those are covered pretty well in Hilfinger's notes so I'll let you read those for that information.