

# C++

## Differences between C and C++

(In all the following examples, the first block of code is C, and the next block is C++.)

# Memory allocation

Use **new** and **delete** instead of **malloc()** and **free()**.

---

```
int *a;  
a = (int *)malloc(20*sizeof(int));  
a[19] = 10;  
free(a);
```

---

```
int *a;  
a = new int[20];  
a[19] = 10;  
delete[] a;
```

---

These operators are typed, and they also call constructors as necessary. You should always use **new** and **delete** with objects in C++.

## Pass by reference

C++ has pass-by-reference, using the `&` to mean the same thing `var` does in Pascal.

---

```
void swap( int *a, int *b )
{
    int t;
    t = *a;
    *a = *b;
    *b = t;
}
int x=3, y=4;
swap( &x, &y );
```

---

```
void swap( int &a, int &b )
{
    int t;
    t = a;
    a = b;
    b = t;
}
int x=3, y=4;
swap( x, y );
```

---

This means no more using crazy pointers and address-of operators when all you want to do is pass-by-reference.

# Casts

C++ has funky casts which you should become familiar with.

---

```
int i;  
i = (int)3.14159;
```

---

```
int i;  
i = static_cast<int>( 3.14159 );
```

---

You should use `static_cast<>()` to do any type conversions among the basic types. There are also `dynamic`, `const`, and `reinterpret` casts, which we may run into later.

# Basic I/O, strings, assertions

C++ has some much nicer I/O libraries, as well as real strings.

---

```
#include <stdio.h>
#include <string.h>
char s[30];
printf( "Hello world, i is %d and
      %d.", i++, i++ ); /*undefined*/
strcpy( s, "Bye!" );
strcat( s, s );        /* bad! */
printf( "%s\n", s );
```

---

```
#include <iostream.h>
#include <assert.h>
#include <cstring.h>
cout << "Hello world, i is " <<
      i++ << " and " << i++ << "." <<
      endl;           // well defined
string s="Bye!";
s += s;              // fine
cout << s << endl;
assert( s.length() > 4 );
```

---

**cout** is used to output stuff to the screen. The **<<** is sometimes called the "put to" operator, it's the binary left-shift operator of C that's been overloaded (we'll talk about overloading later) for stream output. It is a nice, typesafe, unambiguous way to send output to the screen. **endl** is the "end of line" string ("**\n**" on a unix machine, "**\r\n**" on DOS, etc.).

**strings** work like you expect. They are actually part of the standard class library--**strings** were written in C++; they're not "built in" to the language.

**assert()** is a nice macro to assert that something is true.

# Booleans

C++ has booleans.

```
-----  
#include <bool.h>  
  
bool my_bool = TRUE;  
if( !my_bool )  
    do_something();  
-----
```

The only legal values of a boolean variable are **TRUE** and **FALSE**. Of course, conditionals still accept integers, where zero is false and non-zero is true, to be backwards compatible with C.

# Comments

C++ introduces single-line comments with `//`.

```
// This is a single-line comment.
```

# Declaration and initialization

C++ has new rules and syntax about declaration and initialization.

---

```
int i;  
int j, k=3;  
...  
j = k;  
for( i=0; i<N; i++ )  
    blah();
```

---

```
int k(3);  
...  
int j(k);  
for( int i(0); i<N; i++ )  
    blah();
```

---

In C++, you should use the new style, for reasons we will cover soon which involve constructors and assignment operators.

# Constants

For those of you who do stuff like

```
#define MAX 20
```

quit it. ANSI C has `const`, and

```
const int MAX=20;
```

is preferred (it's typesafe). Just a reminder. `const` has other meanings in C++, which we'll see later.

# Templates

C++ has templates, a way to make functions work for many different types of arguments in a typesafe way.

---

```
#define MAX(x,y) ((x)>(y)?(x):(y))
int x=3, y=4;
int z = MAX( x, y++ );
printf( "%d\n", y ); /* y==6 !? */
```

---

```
template< class T >
T max( const T& x, const T& y )
{
    if( x > y )
        return x;
    return y;
}
int x=3, y=4;
int z=max(x,y++);
cout << y << endl; // y==5
```

---

## More templates

Here's a C++ example of using a template to write a generic bubble-sort on an array:

```
-----  
template< class T >  
void bubble_sort( T* a, int N )  
// a is array, N is num elements  
{  
    for( int i(0); i<N; i++ )  
        for( int j(i+1); j<N; j++ )  
            if( a[i] > a[j] )  
                swap( a[i], a[j] );  
}  
const int N=4;  
int *ia = new int[ N ];  
ia[0] = 1;  
ia[1] = 4;  
ia[2] = 3;  
ia[3] = 2;  
bubble_sort( ia, N );  
for( int i(0); i<N; i++ )  
    cout << ia[i] << " ";  
cout << endl;  
-----
```

This function works on arrays of **int**, **float**, **char**, **string**—anything with **operator >** defined. Groovy.