Trivial Differences between C and C++

comments /* .... */
Can also use // .... to end of line

Simple input/output

C has three pre-defined data streams

```
stdin, stdout, stderr
```

C++ has four pre-defined data streams

```
cin, cout, cerr and clog

cout << "Enter a number: ";
cout << "Enter a number:\n";
cout << "Enter a number:"; << endl;
```

while for input the converse:

```
cin >> N;     // automatic "format" - type
cin >> C >> I >> R >> Z;      // & not needed.
```

reads C, then I, then R and then Z from standard input, using a format conversion that is appropriate for the declared type of each variable.

```
char C;
int I;
Float R;
double Z;
```

Note that in C the char data type only determines the size of memory allocated, but is otherwise like an 8-bit unsigned int.

In C++, a char remains as a char until it is used in a nontrivial expression, which then forces it to an int.

In both C and C++ there is no difference between a pointer to a character and a pointer to a string of characters.
There are differences in handling i/o of strings.

```
char* string = "nothing but words";
cout << string;
```

would do what you expect, but

```
char s[100];
cin >> s;
```

upon reading the sequence:

nothing but words

would yield `s = "nothing";`

Thus the first blank on input is a data field separator!
Also C++ output is buffered, so output to the screen must be forced with the `flush`; function.

Tags vs. type names

In C++ tags (names identifying a particular kind of structure, union or enumeration) are automatically type names

C:

typedef struct { double real, imag } Complex;

C++

struct Complex { double real, imag };

Functions with no arguments

C++

No need to use the word void when declaring or defining a function with no arguments.

```
void draw (void);        /* no parameters in C */
void draw ();         // no parameters in C++
void draw ();         /* variable parameters */
```

Initializing parameters

C++ allows function arguments to have default values. The following function prints N newline chars, but if no parameter is passed, then a default of 1 is used.

```
void new_line (int N = 1) // default argument
{
    while (N-- > 0)
        putchar ('\n');
}
```

with invocation:

```
new_line (3);// prints 3 blank lines
new_line ();      // 1 blank line by default
```

Call by reference parameters.

This is a significant difference.

In C to pass back a result through a parameter you must provide a pointer to the location where the result is to go (the pointer itself does not change) thus we wrote:

```
scanf ("%d", &N);
```

or looking at the swap function in C we wrote:

```
void swap (int* a, int* b)
{
    int tmp;
    tmp = a;
    a = b;
    b = tmp;
}
```

with invocation

```
swap ( &A, &B );
```

Although this works fine, it is easy to forget and make mistakes. In C++ things are improved, somewhat, by allowing parameters to be declared as references, instead of pointers:

```
void swap (int& a, int& b)
{
    int tmp;
    tmp = a;
    a = b;
    b = tmp;
}
```

with invocation:

```
swap (A, B);     // Just like in Fortran!
                    // but is it any better?
```

Dynamic Storage management

In C programs can dynamically allocate and release blocks of memory by calling `malloc, calloc, realloc` and `free`. Although C++ also has access to these functions it provides two new operators (not functions) called `new` and `delete` (keywords) which allocate and release space. Thus

```
int *int_ptr, *int_array;

int_ptr = new int;    // allocates memory for an int

int_array = new int [100];   // allocates memory for
                             // an array of 100 ints
```

`new` returns a NULL pointer if the requested memory cannot be allocated. `delete` requires a pointer as its operand.

```
delete int_ptr;              // releases memory pointed
                             // to by int_ptr
delete [] int_array;         // de-allocates the array
```

Significant extensions in C++

Classes

Class Definitions

Member Functions

Constructors

Destructors

Overloading

Facilities for Object-Oriented Programming

* **Encapsulation**
* **Inheritance**
* **Polymorphism**

Derivation

Virtual Functions

Templates

Exception handling (we will not cover this topic)