C++ for C Programmers

See Allen Supynuk's on-line notes.

* C201 Objectives and how to get there

Why Object Oriented Programming?
   * The crisis in software engineering
   * Does Object Oriented Programming (OOP) help?
   * OOP and Client/Server computing
   * OOP and the Job Market
   * OOP requires notion of Abstract Data Type

C++? encompasses all essential C

* Incompatibilities between ANSI C and C++
* New Identifiers
* Function declarations required
* Global data
* Structs Simplified
   * Types
   * Assignment statements
   * Initializations

Miscellaneous
* How to share header files between C++ and C
* C++ comments
* I/O, I/O and Off to Work We Go...
   * Standard places to read and write
   * Somewhat simpler i/o offered

New features

* Pass by reference
* Default arguments **and more much more**

Objectives

At the end of this semester you will:

* Have seen most of the features in C++ that are not in ANSI C
* Know about the (minor) incompatibilities between C++ and C
* Have been introduced to abstract data types (ADTs) and have seen how C++ facilitates their use
* Have been introduced to object oriented programming and have seen how C++ supports this paradigm

However, you will **NOT**:

* Be comfortable with C++
* Be able to write even 10 line programs without a manual! Non-trivial programs, that is.
* Be an expert on ADTs (that would take another course or two and experience)
* Be an expert in object oriented programming (ditto, or lots of experience)
* Be an expert in object design (ditto)

As an analogy, you will have been shown how to do things, but will not have much actual practice at doing them in a shared (team) programming environment.

There are only three known ways to get proficient at C++:

* Practice, Practice, Practice

How to get there

Why Object Oriented Programming?

Offers the promise of greater code re-use by allowing a new class of objects to be "derived" from an existing class, instead of from scratch.

The crisis in software engineering

It is becoming too expensive to maintain old software for older machines using out-of-date specifications that were appropriate for limiting languages.

A rule of thumb in software engineering is:

As the size of a project doubles you need four times as many programmers, in order to complete the task in the same time **Jobs, jobs, jobs!**

The crisis occurs because the size of large programs doubles every 2-5 years.

For example, rumor has it that programs like MS Word and WordPerfect currently require 20-30 people to produce a new version a year, and that these programs are 250-300,000 lines of code.

This may be an underestimate, it is said that the current version of Excel is more like 5 million lines of code!

Assuming new versions will double in size every 4 years, it is clear that we will quickly reach a state where, either nobody will be able to afford to write the next version, or it will take 4 times as long to produce another version with the same staff.

How does Object Oriented Programming (OOP) help?

The above rule assumed each program was one large piece of code. Breaking a program up into separate functions doesn't help because of interactions between functions. The only types of thing that can work is if a program can be constructed from smaller pieces, each of which is completely independent.

[This was the driving philosophy behind UNIX in 1970, and look where UNIX is now!]

This is exactly what object oriented programming is about. Programs are constructed out of objects. Each object contains data and code and is completely self-contained. There are ways of manipulating these objects (operators) so that all the "real world" manipulations can be done in such a way that:

* objects can be used as-is

* objects can be used mostly as-is, but with some extra code for the problem at hand (**inheritance** and **derived objects**)

* objects can be selected based on criteria known only at run-time (**virtual objects**)

* objects can be generalized to work on other objects whose type is not known until link-time.

Consider sorting an array. You should be able to write an object that sorts arrays no matter what type of object is in each element. (**templates**)

OOP and Client/Server computing

Each object is an independent entity that communicates with other objects by sending messages.
(To: Sort-array, Message: sort this array of integers.) This maps very nicely onto Client/Server computing which has client programs (objects) on various machines interacting with server programs (objects) on other machines via some kind of protocol (messages).

OOP and the Job Market
Microsoft, Borland, Word Perfect, and many other large programming firms are now insisting on object oriented programming experience, usually in C++, for all their programming positions. This applies equally at Nortel (BNR), Corel, IBM and others.

Why C++?

C++ is the world's most successful object oriented programming language. Bjarne Stroustrup, the original author of C++ says:

``C++ did three things [...]

1. It produced code with run-time and space characteristics that competed head-on with the perceived leader in that field: C. Anything that matches or beats C must be fast enough. Anything that doesn't can and will--out of need or mere prejudice--be ignored.

2. It allowed such code to be integrated into conventional systems and produced on traditional systems. A conventional degree of portability, the ability to coexist with existing code, and the ability to coexist with traditional tools, such as debuggers and editors, was essential.

3. It allowed a gradual transition to these new programming techniques. It takes time to learn new techniques. Companies simply cannot afford to have significant numbers of programmers unproductive while they are learning. Nor can they afford the cost of failed projects caused by programmers poorly trained and inexperienced in the new techniques failing by over enthusiastically misapplying ideas.''

By way of comparison it was once noted: "this system took 45,000 lines of Ada, not counting the comments. The C++ version was 18,000 lines of fully commented code."
This may reflect the verbosity of Ada, compared to the obscure terseness of C (e.g. the **for** statement)

Trivial Differences between C and C++

comments `/* ........ */`
Can use `// ........... to end of line in C++`

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;     //no & necessary!
```

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;
```

would, upon reading the sequence:
nothing but words
would be equivalent to **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 versus 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 arguments in C/C++ */
void draw ( );         /* variable parameters in C*/
void draw ( );         // default arguments in C++
```

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

or looking at the swap function in C we write:

```
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, and so we write:

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

with invocation:

```
   swap (A, B);    // Just like in Fortran!
                   // is this 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* 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. The `delete` operator 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 and Class Definitions

Member Functions

Constructors and Destructors

Overloading

Facilities for Object-Oriented Programming

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

Derivation

Virtual Functions

Templates

Exception handling (we won't cover this topic)