A SHORT COURSE ON C++

Dr. Johnson

School of Mathematics

Semester 1 2008
Outline

1. Introduction to C++
   - Object-Orientated Programming
   - Syntax
   - Handling Data and Variables
   - Input/Output

2. Flow Control and Functions
   - If Else
   - Looping
   - Functions
   - Cmath Library
   - Prototyping
A structured language can *hide* information from the rest of the program.

Structuring code and data allows

- easy upgrades
- many programmers to work on a large project

Object-oriented programming imposes a high level of structure

Problems are broken down into subproblems, and then into self-contained units called objects

Common traits of object-oriented languages are:

- encapsulation
- polymorphism
- inheritance
Using Objects

1. **Encapsulation:**
   functions and data inside an object have restricted access.

2. **Polymorphism:**
   represents the concept of “one interface, multiple method”. The same interface can be used to do different things for different objects: i.e. define + to add numbers, but perform string concatenation on characters and strings, ‘a’ + ‘b’ = ‘ab’.

3. **Inheritance:**
   allows one object to acquire the properties of another. An example would be to define a generic object “car” that has a steering wheel, four wheels and an engine. The new object “sports car” inherits all these properties and adds a sun roof, go-faster stripes and a huge stereo.
**Writing C++**

The key elements of C/C++ syntax are:

- Semicolon used to mark end of statements
- Case is important
- Totally free form, lines and names can be as long as you like!
- Comments take the form /* C style comment */ or // C++ style comment
- Code blocks are surrounded by braces {}
The following is a C++ program.

```c++
main()
{
}
```

There are no commands to execute.

If we save it in the file “simple_prog.cc”,
we can compile and run it with the commands:

```
> c++ simple_prog.cc
> ./a.out
```
Unlike Fortran, there are almost no intrinsic functions in C++. This includes the ability to print to screen. We can include standard libraries for:

- Input/Output
- Advanced Storage
- Strings
- Mathematical functions

The syntax for including libraries is:

```
#include <library_name>
```

Include statements must appear before any other statements.
A simple example of the standard input/output library:

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

main()
{
    cout << 'Hello World!' << endl;
}
```

The output at a terminal will look like:

```
> c++ hello_world.cc
> ./a.out
Hello World!
```
There are six basic data types in C++:

- char – CHARACTER
- int – INTEGER
- float – REAL
- double – REAL(dp)
- bool – LOGICAL
- void

Corresponding fortran variables are shown in red.

We use void for functions that do not return a value (SUBROUTINE).
Declaring Variables

- We may declare variables anywhere in the code.
- Variables will be localised to the block in which they are declared.
- What is the output from the following?

```cpp
#include <iostream>
using namespace std
main()
{
    int i=0;
    cout << " i= " << i << endl;
    {
        int i=10;
        cout << " i= " << i << endl;
        }
    cout << " i= " << i << endl;
}
```
Arrays

- We declare and reference arrays using square brackets `[]`.

```cpp
int array[100]; // 100 integer array
array[0] = 0;
array[1] = 1 + array[0]
```

- Arrays are indexed from 0, and this **cannot** easily be changed.
- Multidimensional arrays are declared in the obvious way

```cpp
int array_2D[5][5]; // 2D array
array_2D[0][0] = 0;
```
Operators

- We have the same simple operators +, -, *, / like Fortran.
- There is **no** equivalent to the power **.**
- There are three extra operators:
  - % is the modulus operator, giving the remainder of integer division.
  - ++ adds 1 to its operand.
  - -- takes 1 away from its operand.
- We can write the code \( x = x + 1 \) as \( x++ \).
Simple Input and Output

- We use stream variables to access the screen, keyboard and files.
- They are like UNITS in Fortran.
- We need to include stream libraries at the top of the program

```cpp
#include<iostream>
using namespace std
main(){
    int i
    cout << " Enter a number. " << endl;
    cin >> i;    //read in a number
    cout << " Your number is " << i << endl;
}
```
Simple Input and Output

- `cout` is the standard screen variable, and `cin` the standard keyboard variable.
- To pass data to and from the stream we use the `<<` and `>>` operators.
  - `<<` data is passed right to left, in the example the string is passed to `cout`.
  - `>>` data is passed left to right, in the example the integer is passed from `cin` to `i`.
- Multiple bits of data can be passed to the stream by stringing them together in the same command.
- Use `endl` to finish a line.
To read and write to files we must include the \texttt{fstream} library.

Input streams have type \texttt{ifstream}, and output streams \texttt{ofstream}

```cpp
ifstream file_input; // an input file stream
ofstream file_output; // an output file stream
```

- \texttt{ifstream} and \texttt{ofstream} have intrinsic functions to open and close files.
- We can also check if the file is open with the \texttt{is\_open()} function.

```cpp
file_input.open("input.in"); // open file input.in
if(file_input.is\_open()) // check file is open
```
We can use if, else if, and else to control flow through the program.

```cpp
int i;
cout << " Enter a number " << endl;
cin >> i;
if(i<0)cout << " i is negative" << endl;
else if(i==0)cout << " i is zero" << endl;
else cout << " i is positive" << endl;
```
To execute more than one command on an if condition use blocks

```cpp
if(condition){
// lots of commands in here
}
else {
// and in here too.
}
```
FOR LOOPS

- The general form for a loop is

```cpp
for(initialisation; condition; increment) statement;
```

- We can loop over multiple commands using a block

```cpp
for(int i=0; i<10; i++){
    temp = i*10;
    cout << " value " << temp << endl;
}
```
EXITING A LOOP

The command `break` can be used like the command `EXIT` in Fortran.

```cpp
for(int loop=0; loop<iter_max; loop++){
    solve_for_U(u, y, U);
    if(residual(x, y, U) < tolerance) break;
}
```
The general syntax for a function is:

```cpp
data type function name(arguments)  
{  function statements  }  
```
#include<iostream>
using namespace std

// square an integer
int square(int i) 
{ return i*i; }

// Main Program
main(){
    int number=5
    cout << square(number) << endl;
}
Accessing the Math Library

Simply include the library at the top of your code:

```cpp
#include<cmath>
```

- All of the trigonometric, hyperbolic and exponential functions are present.
- There is also a `pow(x, y)` to raise x to the power y.
- and a `sqrt()` function.
A function must be defined before it can be called.
Use prototypes to declare functions before they are used.

```cpp
data type function_name(arguments)
```

- The main body of the function can be placed somewhere else in the code (or even a separate file)
- This is like the `EXTERNAL` declaration in Fortran.