C++
Basics

CO650 Advanced Programming

Slide Format

- Text within a green box illustrates the syntax using pseudo code.

```
if (condition) {
    statment
}
```

- Text within a blue box is an example of C++ code.

```
if (age > 65){
        cout << “You are retired” << endl;
}
```

Topics

- The main function
- include
- namespaces
- Comments
- Output & Input
- Variables, Constants & Types
- Addresses, Pointers & References
- Scope
- Operators
- Casting
- Time
- Glossary

The main function

- The entry point for C++ programs
- Invoked by the operating system
- The default main function generated by Visual Studio

```
int _tmain(int argc, _TCHAR* argv[]) {
        return 0;
}
```

- This can be simplified, if you do not wish to process the command line arguments.

```
int main() {
        return 0;
}
```

- Main returns an integer value. 0 indicates the program terminated normally and any other value indicates an error. How the return value is handled is Operating System dependant.

Pausing The Console Window

- Selecting the **Debug->Start Debugging** menu option or **Start Debugging** button from within VS, will execute the program but not pause it on completion.
- This results in the window closing after the code within main has executed and you not being able to see the output.
- To avoid this either select **Debug->Start Without Debugging** or add the system pause instruction to main, just above return 0.

```
int main() {

        system("pause");
        return 0;
}
```

#include

- A mechanism that allows code located in one file to access code located in other files.
- A pre processor directive that precedes a file name
- The contents of the included file are treated as if they were embedded within file in which the directive is placed.

```
#include <path_filename>
#include “path_filename”
```

- <> search for the file in the directory designated as the include directory within the IDE
- “” instructs processor to look for file in the same directory as the source file containing the directive and if not present the designated IDE include directory.
- “” if a full path is specified then the preprocessor will only look in that directory

#include header files

- Your project will contain many files.
- Some of these will be header files (.h) that contain code that may be required by other files in the project.
- You must explicitly indicate that a file requires the code in a header file.

```
#include “fileName.h”
```

- The include directive should be invoked within the file that requires access to the definitions within the header file.

```
#include “myvars.h”
```

- A semicolon after the file name is optional.

namespaces

- An identified declarative region that avoids naming conflicts
- Identifiers are accessed using the scope operator ::

```
namespace myFirstNamespace{
        int a = 10;
}

namespace mySecondNamespace{
        int a = 12;
}

int main() {
        int c = myFirstNamespace::a + mySecondNamespace::a;
        cout << c << endl;
        using myFirstNamespace::a;
        c = a + mySecondNamespace::a;
        cout << c << endl;
}
```

- The **using** keyword avoids having to scope each object within the namespace.

namespaces

- Classes, standalone functions etc can be placed in a namespace.
- General rule is to only use namespaces when the code is to be used by third parties. Thus avoiding potential naming conflicts (name pollution).
- All components in the C++ Standard Library are located within the **std** namespace.
- To avoid qualifying each object with std:: we can state which namespace we are using. In the example below this is placed in global scope.

```
#include "iostream";

using namespace std;
```

Comments

// Single line

/* Paired comments for
multiple lines ***/**

Output

- cout is an object of the ostream class
- ostream being the standard output stream
- The standard output usually being the console
- Data is output using the **insertion operator <<**

```
#include "iostream";

int main() {
        int c =10;
        std::cout << c;
        return 0;
}
```

- **iostream** is an OO library that implements I/O operations.
- cout ostream is declared within the **std** namespace

Input

- cin is an object of the istream class. Standard input stream
- The standard input usually being the keyboard
- Data is retrieved using the **extraction operator >>**

```
#include "iostream";

using namespace std;

int main() {
        int c;
        cin >> c;
        cout << c << endl;
}
```

- The using statement avoids having to prefix each cin / cout with std::
- endl generates a new line (flushing the buffers).
- When using cin with strings only the first word is captured.

Variables & Types

- Valid identifier's name consists of one or more digits, letters or the underscore character.
- Must begin with a letter or underscore.
- Not be a reserved word.
- They are **case sensitive**.
- A variable has an associated type
- Some common types are listed below
- There are unsigned equivalents also

Name	Description	Size	Range
char	Character or small int	1byte	-128 to 127
int	Integer	2bytes	-32768 to 32767
bool	Boolean value	1byte	True or false
float	Floating point number	4bytes	+/-3.4e+/-38

typedef

- Allows the creation of an alias for a type
- The alias can be used as if it was a C++ type

```
typedef type aliasName;
```

- This is not creating a new type. Simply providing an alternative name for the type..
- Commonly used to improve the readability of the code.

```
typedef int Mark;

Mark studentMark = 75;
```

- Very useful tool when used with function pointers that have a complex type.

Defining a Variable

- C++ is strongly typed
- All variables must be defined

```
char gender;
int age;
bool married;
float height;
```

- Integer types can be **unsigned** or **signed**. By default an integer type is signed.

```
unsigned int age;
signed int score;
```

- Variables of the same type can be defined within a single statement

```
int a,b,c;
```

Initialising a Variable

- The initial value of a local variable is undetermined
- Variables can be initialised immediately after definition. Initialization using the = symbol is known as copy initialization or explicit assignment.

```
char gender = 'M';
int age = 21, children = 2;
bool married = false;
float height = 1.85;
```

- Alternatively using direct initialization syntax (implicit assignment)

```
char gender('M');
int age(21);
bool married(false);
float height(1.85);
```

Scope

- Global variables are defined outside of a function / class.
- By prefixing the definition with static the scope is restricted to the file.
- Local variable have automatic duration. They are destroyed when the block of code they are defined in goes out of scope.

```
int n1 = 30;                // Global scope

static int n2 = 40;         // File scope

void f(){
    int n3 = 50;            // Local scope
    cout << n1 << endl;
    cout << n3 << endl;
}

int main(){
    cout << n1 << endl;
    cout << n2 << endl;
    return 0;
}
```

String Type

- A Class defined within the <string> library
- Within the std namespace

```
#include <string>
using namespace std;

string name = "Guy Walker";
```

- Strings can be treated as arrays of characters

```
char firstLetter = name[0];
```

- The number of characters in the string is returned by the length() member function

```
int numberChars = name.length();
```

- Compare strings using the comparison operator ==

Constants

- A value inserted into the code is a **literal constant**

```
int age;
age = 21;
```

- In the above code 21 is a literal constant.
- Constants can be defined by prefixing the definition with the **const** key word

```
const int RETIRE_AGE = 65;
```

- By convention constants names are written in upper case.

Addresses

- The computers memory consists of memory cells in which the variable's value is stored.
- Each memory cell has a unique address
- The cells are given consecutive addresses
- The data value stored in the address is known as the **rvalue**
- The address itself is known as the **lvalue.**
- The OS decides at what address the variable will be stored when the variable is defined.
- We can access this address through the **reference operator** &

```
int age(21);
cout << &age << endl;  // displays 0013FF54 on my PC
```

Pointers

- A pointer is a variable that contains the memory address of a variable.
- Pointers are defined using the asterisk symbol.
- The type of the pointer is the type of the variable the pointer points to.

or

```
type* PointerName;
```

or

```
type  *PointerName;
```

```
type * PointerName;
```

- After definition, the pointer does not contain a valid address. Its contents are undetermined.

Pointers

- Pointers should always be initialised

```
Type * PointerName = 0;
```

- If the address is unknown at definition, assign 0 as there is no address 0

```
int age(21);
int *ptrToAge = 0;

ptrToAge = &age;

cout << ptrToAge << endl; // displays 0013FF54 on my PC
```

- When declaring more than one pointer of the same type, repeat the *

```
int * a, * b, * c;
```

Pointers

- Through the pointer we can access the value within the variable whose address is stored in the pointer.
- The **dereference** operator * precedes the pointer to return the variables value.
- The dereference operator translates to “value pointed by”

```
int age(21);
int *ptrToAge;

ptrToAge = &age;

cout << *ptrToAge << endl;     // displays 21
```

Pointers

Computer Memory

Address	Type	Identifier	Value
1000	int	age	21
1001	float	height	1.8
1002	bool	married	true
1003	int*	pAge	1000

```
int age = 21;
float height = 1.8;
bool married = true;
int* pAge = &age;
cout << &age << " " << pAge << " " << *pAge << endl; // Displays 1000  1000   21
```

References

- An alias for a variable, through which the data within the variable can be modified.
- The reference holds the address of a variable.
- The reference is declared using the reference operator &
- The reference must be initialise during definition

```
type& referencename = value;
```

- Once initialised the reference address can’t change (unlike a pointer).
- After declaration the reference can be treated as a normal variable.

```
char gender = 'M';
char& ref = gender;
ref = 'F';
cout << ref << endl;          // Outputs F
cout << gender << endl;   // Outputs F
```

Scope

- **Global** variables are defined outside of a function.
- Global variables are visible within all functions that are located after the variable definition.
- **Local** variables defined inside of functions
- Local variables are only visible within the function in which they are defined.

```
int globalAge = 20;

int main(){
        int localAge = 10;
        cout << "Global : " << globalAge << " Local : " << localAge << endl;
        system("pause");
        return 0;
}
```

Operators

- Assignment operator =
- Mathematical operators /, -,+, *,
- Modulus (integer div remainder) %
- Decrement / increment --, ++
- Compound assignment +=, *=, -=, /=
- Prefix – increments value before assignment

  firstValue = ++secondValue;
- Postfix – assigns value and then increments

  firstValue = secondValue++;
- Relational and equality operators ==, !=, >, >=, <, <=

Time

- Access time related functions by including **<time.h>**
- The time / date is stored within a variable of type **_time32_t**
- The **_time32** function populates a variable of type _time32_t (passed as a parameter) with the current system time as an integer. The number of seconds elapsed since 1st Jan 1970.
- The **_localtime32_s** function is passed the time in seconds and populates a **tm** structure with the appropriate s,h,days,month and year values. Access individual values using dereference operator ->
- To display the time as a string pass the tm structure to the **asctime_s** function along with a character array which is populated with the time as a textual description.

Time Example

```
__time32_t rawtime;
struct tm timeinfo;
char buffer[32];
_time32(&rawtime);
_localtime32_s(&timeinfo, &rawtime);
asctime_s(buffer,32,&timeinfo);

cout << “The current time is “ << buffer << endl;
```

Summary

- C++ is an Object Oriented version of C
- The syntax has much in common with Java & C#
- The main difference being pointers.
- All the projects created will use a console window as output.