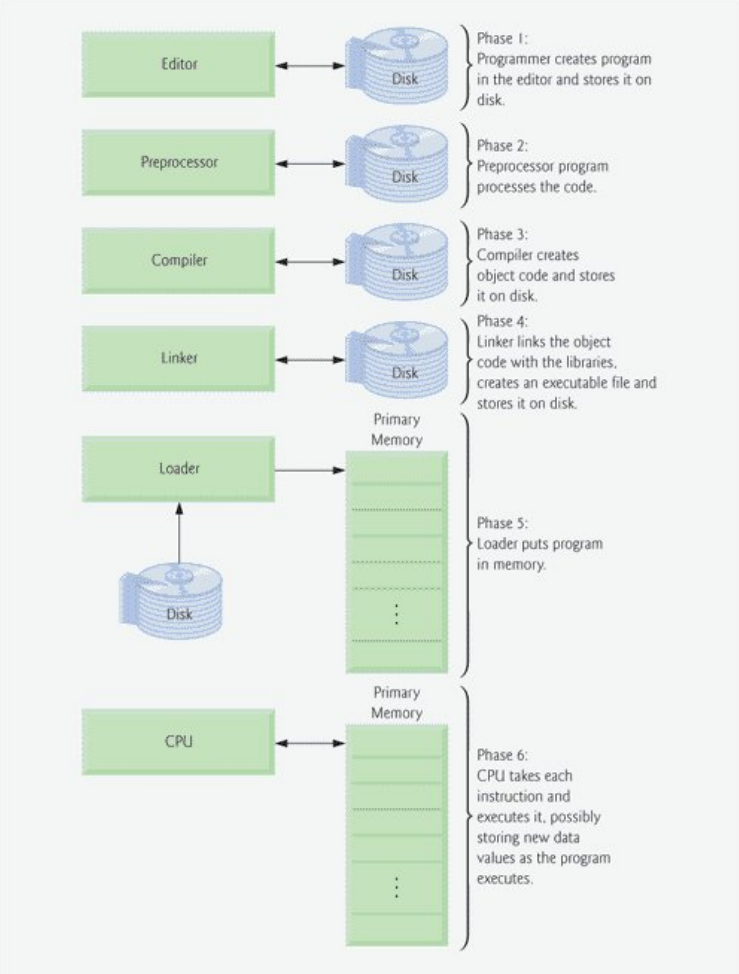
University of Essex

C++ Programming

02- Hello C++

C++ Development Environments

- Development of a C++ program involves six phases:



Source: http://www.gibmonks.com/c_plus/ch01lev1sec14.html

University of Essex

Integrated Development Enviroment

- Popular IDEs include Visual C++, NotePad++. However, in the labs we shall be using Sublime, Visual Studio or TextPad.
- There is a video on how to use Sublime on Moodle.

You first application is the “Hello World”

```
1  // C++
2  #include <iostream>
3  //
4  using namespace std;
5  // could use int main(int argc, char* argv[]) or
6  // int main(int argc, char* argv[], char **env)
7  main() // default return type is int
8  {
9      // cout << "\nHello, World!\n";
10   std::cout << "\nHello, World!\n";
11     // return 0;
12 }
```

University of Essex

You first application is the “Hello World”

```
1  // C++
2  #include <iostream>
3  //
4  using namespace std;
5  // could use int main(int argc, char* argv[]) or
6  // int main(int argc, char* argv[], char **env)
7  main() // default return type is int
8  {
9      // cout << "\nHello, World!\n";
10     std::cout << "\nHello, World!\n";
11     // return 0;
12 }
```

The main function:

- not written inside a class
- may take arguments to access command-line arguments, but optional;
- its return type is **int**. When a function is declared without a return type it is by default given the type **int**
- not regarded as an error if a function fails to return a result

You first application is the “Hello World”

```
1  // C++
2  #include <iostream>
3  //
4  using namespace std;
5  // could use int main(int argc, char* argv[]) or
6  // int main(int argc, char* argv[], char **env)
7  main() // default return type is int
8  {
9      // cout << "\nHello, World!\n";
10     std::cout << "\nHello, World!\n";
11     // return 0;
12 }
```

- Screen output is sent to a stream called **cout**, defined in a ***namespace*** called **std**.
- We have to inform the compiler of the namespace: we can do this explicitly using **std::cout**,
- or we can use a **using** statement to indicate that the namespace **std** resolves any otherwise undeclared identifiers and then simply use **cout**.
- The **<<** operator is used to send values to be output to a stream; we may use a chain of these

You first application is the “Hello World”

```
1  // C++
2  #include <iostream>
3  //
4  using namespace std;
5  // could use int main(int argc, char* argv[]) or
6  // int main(int argc, char* argv[], char **env)
7  main() // default return type is int
8  {
9      // cout << "\nHello, World!\n";
10     std::cout << "\nHello, World!\n";
11     // return 0;
12 }
```

In C++ (and C) information about external classes is stored in the form of declarations in a text file called a ***header*** and to make these available to the compiler we must include the contents of appropriate header files within the source code using **#include**.

C++ vs JAVA

```cpp
// C++
#include <iostream>
//
using namespace std;
// could use int main(int argc, char* argv[]) or
// int main(int argc, char* argv[], char **env)
main() // default return type is int
{
    // cout << "\nHello, World!\n";
  std::cout << "\nHello, World!\n";
    // return 0;
}
```

```java
// Java
public class HelloWorld
{
    public static void main(String args[])
    {
        System.out.print("\nHello, World!\n");
    }
}
```

University of Essex

“Hello World” using a class

```
1  #include <iostream>
2  using namespace std;
3  // HelloWorld class definition // (a user-defined type)
4  class HelloWorld
5  {
6      public:
7      void displayMessage()
8      {
9          cout << endl << "Hello, World!" << endl;
10     }
11 }; // semicolon to terminate class definition
12
13 int main()
14 {
15     HelloWorld my_hello;
16     my_hello.displayMessage();
17 }
```

“Hello World” using a class

“Hello World” program in which the output is produced in a class and the main function makes a call to a function in this class

```
1  #include <iostream>
2  using namespace std;
3  // HelloWorld class definition // (a user-defined type)
4  class HelloWorld
5  {
6      public:
7      void displayMessage()
8      {
9          cout << endl << "Hello, World!" << endl;
10     }
11 }; // semicolon to terminate class definition
12
13 int main()
14 {
15     HelloWorld my_hello;
16     my_hello.displayMessage();
17 }
```

“Hello World” using a class

conventional to start class names with upper-case letters and variables and function names with lower-case letters

```
1  #include <iostream>
2  using namespace std;
3  // HelloWorld class definition // (a user-defined type)
4  class HelloWorld
5  {
6      public:
7      void displayMessage()
8      {
9          cout << endl << "Hello, World!" << endl;
10     }
11 }; // semicolon to terminate class definition
12
13 int main()
14 {
15     HelloWorld my_hello;
16     my_hello.displayMessage();
17 }
```

“Hello World” using a class

```
1  #include <iostream>
2  using namespace std;
3  // HelloWorld class definition // (a user-defined type)
4  class HelloWorld
5  {
6      public:
7      void displayMessage()
8      {
9          cout << endl << "Hello, World!" << endl;
10     }
11 }; // semicolon to terminate class definition
12
13 int main()
14 {
15     HelloWorld my_hello;
16     my_hello.displayMessage();
17 }
```

We have used **endl** instead of **'\n'**; this makes the program easier to read.

“Hello World” using a class

```
1  #include <iostream>
2  using namespace std;
3  // HelloWorld class definition // (a user-defined type)
4  class HelloWorld
5  {
6      public:
7      void displayMessage()
8      {
9          cout << endl << "Hello, World!" << endl;
10     }
11 }; // semicolon to terminate class definition
12
13 int main()
14 {
15     HelloWorld my_hello;
16     my_hello.displayMessage();
17 }
```

- This is the class instantiation
- You can also use:

  HelloWorld my_hello = HellowWorld();
- my_hello is this the object

Using Separate Files

- Most classes written in C++ will be expected to be reused
- It is common practice to write each class in a separate file, into
- A a header file with a **.h** extension
- And the respective **.cpp** extension.

Using separate files

The header file for our **HelloWorld** class will be

```
1  // HelloWorld.h
2  class HelloWorld
3  {
4      public:
5        void displayMessage();
6   };
```

The file containing the **main** function will be

```
1  // HelloMain.cpp
2  #include "HelloWorld.h"
3  int main()
4  {
5      HelloWorld myHelloWorld;
6      myHelloWorld.displayMessage();
7      return 0;
8  }
```

Using Separate Files

- We must specify the filename(s), including the **.h** extension, inside quotes in the **#include** directive(s).
- The angled brackets are used for header files that are found in the C++ system folders.
- Assuming the **HelloWorld.h** file is to be included in the **HelloWorld.cpp** file its contents will be copied into the the latter file at compilation time.

Using Separate Files

- We have to write the function body outside of the class definition

```
1  #include <iostream>
2  #include "HelloWorld.h"
3
4  using namespace std;
5
6  void HelloWorld::displayMessage()
7  {
8      cout << endl << "Hello, World!" << endl;
9  }
```

Avoiding Duplicate Header Inclusion

- Many header files need to include other header files
- With multiple files from multiple sources, it is reasonable to assume that there will be repetition of includes.
- This do NOT result in errors!
- The header file for the string class has been written in such a way that if the class declaration has already been included it will not be included again.

Avoid Duplicate Header Inclusion

- The contents of header files are inserted into the temporary copy of the source file by the ***preprocessor***
- To prevent class declarations being duplicated, header files should use preprocessor ***directives*** :

```
1  // myClass.h
2  #ifndef _MYCLASS_H_
3  #define _MYCLASS_H_
4  // use names like _MYCLASS_H_ to avoid
5  // potential clashes with program variable
6  // names
7  // body of header file comes here
8  #endif
```

Avoiding Duplicate Header Inclusion

- Here is a version of **HelloWorld.h** that uses preprocessor directives to avoid any possibility of duplicate inclusion.

```
1  // HelloWorld.h
2  #ifndef _HELLOWORLD_H_
3  #define _HELLOWORLD_H_
4  class HelloWorld
5  {
6      public:
7          void displayMessage();
8  };
9  #endif
```

A Typical C++ Program Framework

```
1  // Class1.h
2  #ifndef _CLASS1_H_
3  #define _CLASS1_H_ // include headers
4
5  class Class1
6  {
7      public:
8          Class1(/* args */);
9          // other members
10     private:
11         // members
12 };
13 #endif
```

```
1  // Class1.cpp
2  // include headers
3  Class1::Class1(/* args */)
4  {
5    // constructor
6    //    implementation
7  }
8  // implementation of other
9  //    members
```

```
1  // main.cpp
2  // include headers
3  main(int argc, char *argv[])
4  {
5      // create objects, use them
6  }
```

Primitive Types

- C++ has a large number of primitive types:

  bool char

  short int (or **short**) **int**

  long int (or **long**)
  long long int (or **long long**)
  unsigned char
  unsigned short int (or **unsigned short**) **unsigned int** (or **unsigned**)
  unsigned long int (or **unsigned long**)
  unsigned long long int (or **unsigned long long**)

  float double long double

Primitive Types

- The value of a data item of type **bool** is either **true** or **false**.
- Any numeric value may be used in a context where a boolean is required; non-zero values are interpreted as **true** and 0 as **false**.
- The next 10 types listed on the previous slide are all regarded as integer types and it is possible to freely assign between them – overflow may of course occur.
- The unsigned versions cannot hold negative values: an initialisation such as **unsigned int i = -1;** will result in the value of **i** being the integer that has the same value as the two's complement representation of -1, i.e. the maximum value for the type **int**.

Primitive types

- The range of values that may be stored in a data item of each type is hardware-dependent.
- A **char** will almost invariably have 8 bits, a **short** will have at least 16 bits and a **long** will have at least 32 bits.
- The unsigned versions will always have the same number of bits as their signed counterparts.
- In most modern implementations the **short** type uses 16 bits, both **int** and **long** use 32 bits and **long long** uses 64 bits.
- The **float** type usually uses 32 bits, **double** 64 bits and **long double** 128 bits.
- Since in some older implementations the **int** type has only 16 bits it is wise to use **long** for any variable or class member that is expected to sometimes take values larger than 32767.

Memory and Storage

- The computer memory for a C++ program has four areas.
- There is an area which stores the code and constants. This is pretty much invariant – the code shouldn't change.
- Data items are stored in three areas:
  - a ***static*** area for items whose lifetime is the duration of the program.
  - a ***stack*** area for local variables in functions, whose lifetime is the invocation of a function; this area is also used to pass arguments to functions and return results, and to store return addresses.
  - a ***heap*** area for dynamic items, whose lifetime is controlled by the programmer

Variables

- A variable in a programming language has three major attributes.
- It has a ***type*** (e.g. **int**, **bool**, **string**, **Student**). Types may be primitive or classes.
- It has ***lifetime*** (or extent). It is born, it is used, it dies. It has ***scope*** (or visibility). When it is in existence, what other things can see it?
- It also has a name, a size (i.e. how much memory it occupies), and a value.

Storage Classes and Variable Lifetime

- There are five ***storage classes***:
  - ***automatic***: for local variables only, existing only when the block in which the variables are defined is active.
  - ***register***: for local variables only, for fast access, rarely used nowadays.
  - ***mutable***: for class members only, to make them always modifiable even in a **const** member function or as a member of a **const** object.
  - ***external***: for identifier linkage of global variables and functions, lasting for the duration of the program.
  - ***static***: for variables and class members that last for the duration of the program and can be initialised once only
- There are five keywords that can (or could) be used to indicate that a variable should have one of these classes: **auto**, **register**, **mutable**, **extern** and **static**.

Storage Classes and Variable Lifetime

- Since local variables are usually stored on the stack by default the use of **auto** keyword to a variable should be automatic is obsolete and since C++11 is no longer used for this purpose but has a totally different meaning.
- The register class is used to indicate that a frequently-accessed local variable should be stored in a register instead of memory in order to optimise performance
- Such optimisations are rarely necessary on modern machines.
- A **const** member function cannot normally change the values of members of the object to which it is applied.
- There may be rare occasions when we wish to allow the value of a particular member to be changed in such a function (for example, during program development it may be desirable to add an access counter when analysing for possible optimisations), hence the need for mutable variables.

Storage Classes and Variable Lifetime

- A static data item exists for the duration of the program and can be initialised only once.
- Classes can have static members; in this case one copy is shared by all of the objects of that class.
- If a local variable in a function is declared to be static it is initialised when the program starts running and retains its value between calls to that function.
- A static variable must be initialised with a value when it is declared, e.g.

static int x = 77; **// *not* static int x;**

Storage Classes and Variable Lifetime

- A ***global*** variable is one that is defined outside any class or function.
- If a program is split between several files it may be necessary to access the same global variable in more than one file.
- The variable can however be declared in only one of the files so in all of the others it must be declared as external.
- For example, if one file contains the global declaration

int a = 5;

- and we need to access **a** in another file we would need to include the declaration

extern int a;

- in that file. This if often done by placing the line in a header file associated with the first file and included in the second file.

Scope

- The scope of a variable specifies where in the program it can be accessed.
- A variable declared locally in a block of code can be accessed only between its declaration and the end of that block.
- If, within the block we make a call to a function, the variable cannot be accessed within the function.
- A private member of a class can be accessed only within that class and its member functions and ***friend*** functions (we shall see what a friend function is later).
- The member functions may be defined outside the class body as seen in our example on slide 20.
- If our **HelloWorld** class had any private members they would have been accessible in the **displayMessage** function.

Scope

- A local variable may have the same name as a variable that is already in scope; the following code fragment is valid.

```
1  int a = 0; // global ......
2
3  int myFunc()
4  {
5      char a = 'x';
6      ......
7  };
```

- Within the body of **myFunc** any occurrence of **a** will refer to the local variable. If we need to access the global variable we must use **::a**.

Exercise

```
 1  #include <iostream>
 2
 3  using namespace std;
 4  int a = 1;
 5
 6  void f()
 7  {
 8      int b = 1;
 9      static int c = a; // initialised once only
10      int d = a; // initialised at each call of f()
11      cout << "a=" << a++ << " b=" << b++ <<
12      " c=" << c << " d=" << d << endl;
13      c+=2;
14  }
15
16  int main()
17  {
18      while (a<4)
19      f();
20  }
```

What are the scope and lifetime of **a**, **b**, **c** and **d**?

What would be the outputs?

University of Essex

CONTACT YOUR LECTURER

m.barros@essex.ac.uk

University of Essex