Name: BIT STUDENTS

***MOUNTAINS OF THE MOON UNIVERSITY***

***Faculty of Science, Technology and Innovation***

***DEPARTMENT OF COMPUTER SCIENCE***

Course: BIT

Course unit: STRUCTURED PROGRAMMING

Lecturer: Mr. ANDREW

***GROUP ASSIGNMENT***

***Assignment***

**Bachelor of Information Technology**

1. Explain the any tools used by programmers to develop C++ program.
2. What are the minimum requirement of installing C++Program
3. Demonstrate the procedure/steps of compiling C++Program
4. Write down the procedures of writing and running a C++
5. With aid of illustration describe the Basic structure of C++

Discuss the basic elements of C++ program

***ANSWERS***

**1** Software Development Kits

There are some free C/C++ compilers available, including:

[GNU Compiler Collection](http://gcc.gnu.org/)  
[Minimalist GNU for Windows (MinGW)](http://www.mingw.org/)

2 Graphics/GUI APIs and Libraries

There are some a variety of different C/C++ APIs/libraries available for 2D graphics and/or graphical user interfaces.

The following are for graphics only:

[Cairo](http://cairographics.org)  
[SDL](http://libsdl.org)

The following are for graphics and also provide "lightweight" GUI functionality:

[dlib](http://dlib.net)  
[Fast Light Toolkit (FLTK)](http://www.fltk.org/index.php)  
[Fox Toolkit](http://www.fox-toolkit.org/)  
[IUP](http://www.tecgraf.puc-rio.br/iup/)  
[Ultimate++](http://www.ultimatepp.org/)

The following provide full graphics/GUI functionality:

[GIMP Toolkit (GTK+)](http://www.gtk.org/)  
[Qt Project](http://qt-project.org/)  
[wxWidgets](http://www.wxwidgets.org/)

For 3D graphics, the most popular and widely used cross-platform API is OpenGL (which is often used with the very simple windowing system GLUT and the simple graphical user interface system GLUI):

[OpenGL](http://www.opengl.org/)  
[GLUT](http://www.opengl.org/resources/libraries/glut/)  
[GLUI](http://glui.sourceforge.net/)

3 General APIs and Libraries

There are quite a few C/C++ libraries in the public domain, but you have to be careful. Some sources include:

[Planet Source Code](http://www.planet-source-code.com/)  
[The Free Country](http://www.thefreecountry.com/developercity/freelib.html)

4 Integrated Development Environments

There are a number of commercial Integrated Development Environments (IDEs) that support C/C++. There are also some good free IDEs available, including:

[Bloodshed Dev-C++](http://www.bloodshed.net/devcpp.html)  
[Code::Blocks](http://www.codeblocks.org/)  
[Visual-MinGW](http://visual-mingw.sourceforge.net/)

In addition, some other "generic" IDEs support C/C++, including:

[Eclipse (CDT)](https://eclipse.org/cdt/)  
[jGRASP](http://www.jgrasp.org/)  
[Visual Studio (Community)](https://www.visualstudio.com/vs/community/)

5 Documentation Tools

There are a number of documentation tools available for C/C++. Some are similar to the javadoc tool that comes with Java, including:

[CppDoc](http://www.cppdoc.com/)  
[Doxygen](http://www.stack.nl/~dimitri/doxygen/)

There are also some that use a variety of different approaches, including:

[DOC++](http://docpp.sourceforge.net/)

6 Unit Testing

Several unit testing tools exist for C/C++, including:

[Auto Unit](http://autounit.tigris.org/)  
[Check](https://libcheck.github.io/check/)  
[Google Test](https://github.com/google/googletest)  
[minunit](https://github.com/siu/minunit)

***No2***

## ****System Requirements****

### ****Operating Systems****

Aspose.Page for C++ is a native library. It supports 32 as well as 64-bit operating systems, but not limited to:

* Microsoft Windows desktop (XP, Vista, 7, 8, 10) and server operating systems (2003, 2008, 2012), etc.

### ****Development Environments****

You can use Aspose.Page for C++ to develop applications in any development environment that supports C++, but the following environments are explicitly supported:

* Microsoft Visual C++ version 2017 or higher

***No3***

## The different kinds of files

Compiling C++ programs requires you to work with four kinds of files:

1. Regular **source code** files. These files contain function definitions, and have names which end in ".cc" by convention (although sometimes you will see source code filenames which end in ".cpp" or ".C").
2. **Header** files. These files contain class declarations, function declarations (also known as function prototypes) and various preprocessor statements (see below). They are used to allow source code files to access externally-defined classes and functions. Header files end in ".hh" or ".h" by convention.
3. **Object** files. These files are produced as the output of the compiler. They consist of function definitions in binary form, but they are not executable by themselves. Object files end in ".o" by convention, although on some operating systems (Windows, MS-DOS), they often end in ".obj".
4. **Binary executables**. These are produced as the output of a program called a "linker". The linker links together a number of object files to produce a binary file which can be directly executed. Binary executables have no special suffix on Unix operating systems, although they generally end in ".exe" on Windows.

There are other kinds of files as well, notably libraries (".a" files) and shared libraries (".so" files), but you won't normally need to deal with them directly.

## The preprocessor

Before the C++ compiler starts compiling a source code file, the file is processed by a *preprocessor*. This is a separate program (normally called "cpp", for "C preprocessor"), but it is invoked automatically by the compiler before compilation proper begins. Preprocessor commands start with the pound sign ("#"). There is really only one preprocessor command you need to know for this track:

* **#include**. This is used to access function definitions defined outside of a source code file. For instance:

#include <iostream>

* causes the preprocessor to paste the contents of **<iostream>** into your source code file. **#include** is almost always used to include header files. In this case, we use **#include** in order to be able to use the **cin** and **cout** objects (input/output streams), whose declarations are located in the file **iostream.h**. C++ compilers do not allow you to use a class, function or global object unless it has previously been declared or defined in that file; **#include** statements are thus the way to re-use previously-written code in your C++ programs. Note that, unlike C, you do not have to include the file extension of the header file in the **#include** statement.

There are a number of other preprocessor commands as well, but we won't be needing them. In particular, C programmers should note that you should **never** use **#define** to define a constant! Instead, use **const**:

const int BIGNUM = 1000000;

This is much safer, since the compiler can use the type information to check that BIGNUM is being used correctly. It's good to do this in C code as well.

## Making the object file: the compiler

After the C++ compiler has included all the header files and expanded out all the **#include**statements, it can compile the program. It does this by turning the C source code into an **object code** file, which is a file ending in ".o" which contains the binary version of the source code. Object code is not directly executable, though. In order to make an executable, you also have to add code for all of the library functions that were **#include**d into the file (this is not the same as including the declarations, which is what **#include** does). This is the job of the linker (see the next section).

In general, the compiler is invoked as follows:

% g++ -c foo.cc

where "%" is the unix prompt. This tells the compiler to run the preprocessor on the file **foo.cc** and then compile it into the object code file **foo.o**. The **-c** option means to compile the source code file into an object file but not to invoke the linker. If your entire program is in one source code file, you can instead do this:

% g++ foo.cc -o foo

This tells the compiler to run the preprocessor on **foo.cc**, compile it and then link it to create an executable called **foo**. The **-o** option states that the next word on the line is the name of the binary executable file (program). If you don't specify the **-o**, *i.e.* if you just type **g++ foo.cc**, the executable will be named **a.out** for silly historical reasons.

Note also that the name of the compiler we are using is **g++**, which is related to the GNU C compiler **gcc** (it shares most of its internals with **gcc**).

## Putting it all together: the linker

The job of the linker is to link together a bunch of object files (**.o** files) into a binary executable. This includes both the object files that the compiler created from your source code files as well as object files that have been pre-compiled for you and collected into **library files**. These files have names which end in **.a** or **.so**, and you normally don't need to know about them, as the linker knows where most of them are located and will link them in automatically as needed.

Like the preprocessor, the linker is a separate program called **ld**. Also like the preprocessor, the linker is invoked automatically for you when you use the compiler. The normal way of using the linker is as follows:

% g++ foo.o bar.o baz.o -o myprog

This line tells the compiler to link together three object files (**foo.o**, **bar.o**, and **baz.o**) into a binary executable file named **myprog**. Now you have a file called **myprog** that you can run and which will hopefully do something cool and/or useful.

This is all you need to know to begin compiling your own C++ programs. Generally, we also recommend that you use the **-Wall** command-line option:

% g++ -Wall -c foo.cc

The **-Wall** option causes the compiler to warn you about legal but dubious code constructs, and will help you catch a lot of bugs very early.

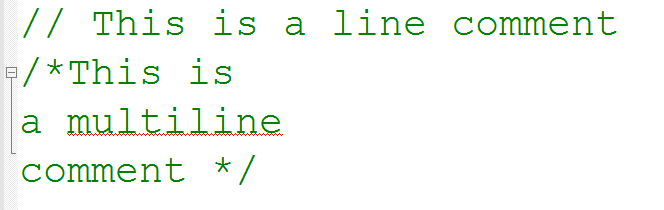
***No 4***

## **STEP 1: DOWNLOAD AND INSTALL AN IDE**



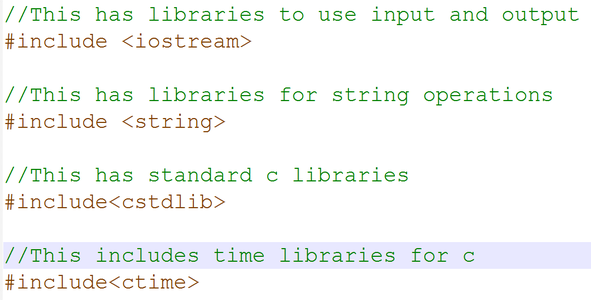
The first step to developing your C++ program will be to download an IDE (Integrated Developing Environment). An IDE often includes a compiler, text editor, and often includes a debugger.  Using IDEs makes programming simpler. Three IDEs that I have used and would recommend are the following;  
  
Dev C++  
Visual Studio  
Eclipse  
  
Here's a link to install DevC++ if you choose to use it:  
http://www.youtube.com/watch?v=Y8So6Hh-ZSs

## **STEP 2: COMMENTING**



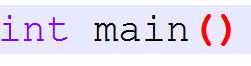
Though comments in programming don't change how the code works, it is important in communicating what a program does to future developers. Commenting is more important in larger programs, but is also good to use for smaller programs to develop good habits. There are two basic ways to comment. The first is the line comment. Any line that starts with \\ is a comment. Also any code between /\* and \*/ are comments. This is shown in the picture corresponding to this step.

## **STEP 3: #INCLUDE DIRECTIVES**



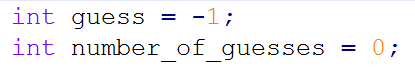
After comments, #include statements are written. These lines allow us to specify libraries, or to use code we have written in other files. In the example program, we include a library to use C++ input and output streams, a library to be able to use strings, the c standard library, and a time library. These libraries will enable us to use more operations further on in the program.

## **STEP 4: MAIN FUNCTION**



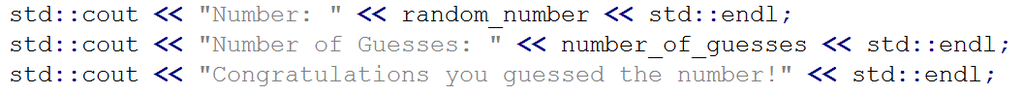
The main function line will be in almost any program you will write. This is where the program will start to run. The main function often is written in the form int main((int argc, char \*\*argv)). This would allow us to pass arguments to our main function, but can be ignored for this program.

## **STEP 5: VARIABLES AND VARIABLE TYPES**



In C++, depending on what type of data is being dealt with, different data types might be necessary. The data type used in the picture on this page shows two variables, guess and number\_of\_guesses, both of type int. They can hold any integer value as their name indicates. There are different other types of variables. Other basic kind of variables include float, double and char. A char can hold a single character, while a float and double can hold decimal values. An example of a char would be the character 'c'. A value that a float or double could store could be the value 1.5. The example program for this tutorial, in addition to using int, uses the type Std::string. This type can hold a sequence of characters.  
  
In the example the value -1 is stored in guess and 0 is stores in number\_of\_guesses.

## **STEP 6: PRINTING TO CONSOLE**



In C++, text can be printed to the console by sending data to std::cout. This can include basic data types. The std::endl adds a new line to the output. This is the C++ way to do this. C++ supports most functionality from C. This includes the printf function. Instead of the following code, it could instead be written as the following:  
  
printf("Number: %d \n",random\_number);  
printf("Number of Guesses: %d \n",number\_of\_guesses);  
printf("Congratulations you guessed the number! \n");  
  
In the printf function, the text entered between the quotes is the text displayed. After the quotes and the comma, the variables printed out are listed. They are printed out in order and must correspond to a %d, %c or other sequence starting with the percent sign. The \n character displays a new line.

## **STEP 7: READING FROM THE CONSOLE**

IMG_262

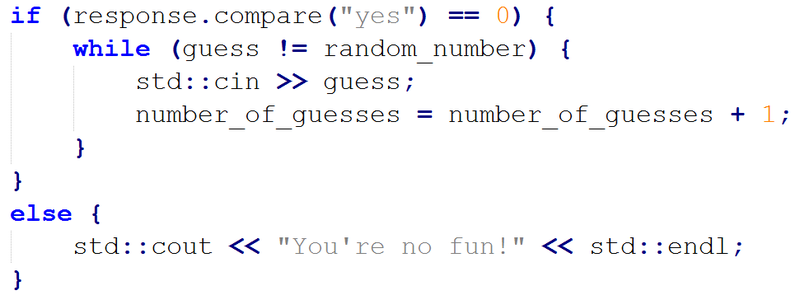
In C++, text can be read from the console by sending data from std::cin and storing in a variable. The console waits until user input when the std::cin function is called. After the user types something in, the program will attempt to store it in guess. In this example, no error checking is done, so if something other than an integer were typed in, the program would likely crash.

## **STEP 8: ARITHMETIC OPERATIONS AND ASSIGNMENT OPERATOR**

IMG_263

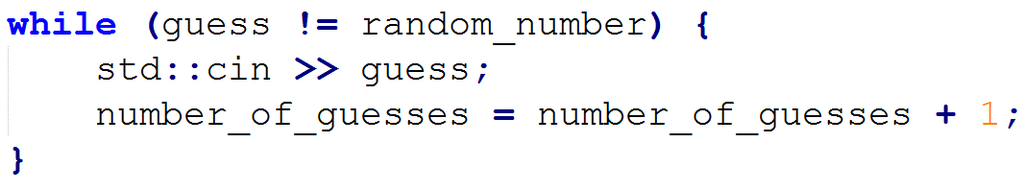
The assignment operator (= sign) assigns the value from the right side of the equals to the variable on the left side of the equals. For this to work properly, the left side must be a variable.  
Arithmetic operations allow mathematical operations to be performed on numbers. There are many operators that can be used to operate on numbers. They include addition (+ sign), subtraction ( - sign), multiplication (x sign) among others. In the line of code, the number\_of\_guesses gets assigned its previous value plus 1.

## **STEP 9: CONDITIONAL (IF) STATEMENTS**



Conditional statements (if statements) change what code runs next depending on what is inside the parenthesis next to an if statement. First the inside of the parenthesis is evaluated. In this instance, if response.compare ( a function from the string library) return 0, the code following it is executed. The compare function returns 0 when the string calling it (response in this instance) is equivalent. Note that strings and basic types use different comparisons. If response is anything other than "yes", "You're no fun!" will be printed to the console. Case does matter.

## **STEP 10: LOOPS**



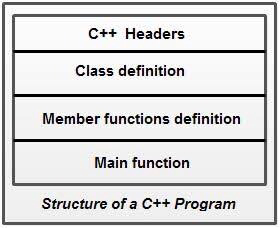
Loops are almost always used with if statements and run until a certain condition is met. Inside the parenthesis next to the while loop is essentially an if statement. If the statement is true, the program runs until the closing bracket of the while loop and the condition is evaluated again and if it is true, the program runs until the closing bracket of the while loop. This cycle continues until the condition (guess != random\_number) isn't true anymore. This loop runs until guess doesn't equal the random\_number variable.

***No5***

## **Structure of a C++ program**

# **Structure of a C++ Program**

Programs refer to a sequence of instructions or statements. These statements are what form the structure of a C++ program. Moreover, the C++ program structure divides into several sections which are namely headers, class definition, member functions definitions and main function. So, this article will let us learn about struct C++ in detail.



It is essential to remember that C++ offers the flexibility of writing a program with or without a class and its member functions definitions. A simple C++ struct program (without a class) will include comments, namespace, headers, main() and input/output statements.

***No 6***

Function: collection of statements; when executed, accomplishes something o May be predefined or standard

Syntax: rules that specify which statements (instructions) are legal

o Programming language: a set of rules, symbols, and special words

Semantic rule: meaning of the instruction Comments are for the reader, not the compiler

o Two types:

􏰁 Single line

//This is a C++ program. It prints the sentence: // Welcome to C++ Programming.

􏰁 Multiple line

/\*

You can include comments that can occupy several lines.

\*/

Reserved words, keywords, or word symbols

o Include:

􏰁 int

􏰁 float

􏰁 double 􏰁 char

􏰁 const 􏰁 void

􏰁 return

Identifiers and variables:

o Consist of letters, digits, and the underscore character ( \_ ) o Must begin with a letter or underscore

o C++ is case sensitive

􏰁

***REF***

https://w3.cs.jmu.edu/bernstdh/web/common/tools/c.php

* The man page for g++. Type:  **man g++ | more**   at the unix prompt. This is actually the same as the man page for **gcc**. Just look for the material specific to **g++**.

The GNU Info documentation on gcc. This also includes full documentation of g++. **Warning!** This is far more information than most people could possibly absorb in the average millenium.

https://www.instructables.com/How-to-Write-a-Simple-C-Program/?amp\_page=true

https://www.toppr.com/guides/computer-science/introduction-to-c/getting-started-with-c/structure-of-a-c-program/