# Introduction to Technical Programming

# Topic 5: Introduction to a high-level programming language

5.1 Creating a Hello World application

Content:

* Geany IDE
* Compiling
* Building
* Running

Learning Outcomes:

Students should be able to:

5.1.1 Create a new project using an IDE

5.1.2 Create a new file with extension CPP

5.1.3 Explain why C++ source files use the .cpp extension and C source files uses the .c extension.

5.1.4 Write the C/C++ code to display output “Hello World”.

5.1.5 Compile a C/C++ application using an IDE

5.1.6 Explain what type of file is produced by the C/C++ compilation process

5.1.7 Show or hide the IDE message window

5.1.8 Investigate the IDE message window to determine if compilation succeeded

5.1.9 Investigate the IDE message window to locate line numbers that contain errors

5.1.10 Build a C/C++ application using an IDE

5.1.11 Explain what file type is produced by the C/C++build process

5.1.12 Run a C/C++ application using the IDE

5.2 C++ application structure

Content:

* C/C++ fundamentals
* Comments

5.2.1 List and explain the different parts of a simple C/C++ application

5.2.1.1 Headers

5.2.1.2 Name space

5.2.1.3 Main

5.2.1.4 return

5.2.2 Explain case sensitivity as it applies to C/C++

5.2.3 Explain the term free-format language and how that applies to C/C++

5.2.4 Explain the term block-structured language and how that applies to C/C++

5.2.5 Explain what a comment is used for in the C/C++ language

5.2.6 Explain how the compiler will treat a comment

5.2.7 List and explain two types of comments allowed by the C/C++ compiler

5.2.8 Discuss the importance of adding comments to a C/C++ application

# Topic 6: Data Types, Variables and Output

6.1. Data types in C/C++

Content:

* Data Types in C/C++
* Arithmetic Operations
* Variables

Learning Outcomes:

Students should be able to

6.1.1 Data Types in C/C++

6.1.2 Define the term data type

6.1.3 Define the term literal value

6.1.4 Name and explain two numerical data types used in C/C++

6.1.5 Explain the difference between Integer data types and floating-point data types

6.1.6 List the different integer data types

6.1.7 List the different floating-point data types

6.1.8 Explain the float, double data types in terms of:

* What the type can consist of
* What the maximum and minimum value can be for the type
* What operations is allowed on the type
* Example literal values allowed for the type
* Example literal values not allowed for the type
* Whether the type is signed or unsigned
* Number of bytes memory used to store type value

6.1.9 Explain the int, char, bool, float double data types in

terms of:

* What the type can consist of
* What the maximum and minimum value is
* What operations is allowed on the type
* Example literal values allowed for the type
* Example literal values not allowed for the type
* Whether the type is signed or unsigned
* Number of bytes memory used to store type value

6.1.10 Use variables as part of a solution

* Define the term variable
* Explain how variables are used in C/C++ applications
* Identify and list C/C++ key words
* List and describe the syntax rules for naming variables
* Write C/C++ code to create variables of different types
* Write C/C++ code to assign literal values to variables of different types
* Write C/C++ code store the result of an arithmetic expression (simple and complex) in a variable
* Write C/C++ code to print the value associated with a variable (combined with string literal for context)
* Write C/C++ code store the result of an arithmetic expression that contains variables on both sides of the operand (simple and complex) in a variable
* Write C/C++ code to assign the value of one variable to a second variable
* Explain the difference between variable declaration and variable initialisation
* Write C/C++ code to initialise variables of different types
* Write C/C++ code to increase or decrease the value of a variable (accumulation statements)
* Write C/C++ code to increase or decrease the value of a variable with 1 using the increment
* Operator
* Write C/C++ code to print the value associatedwith a variable (combined with string literal for context)

6.2. Arithmetic Operations

Content:

* Operators
* Operator Precedence
* Arithmetic expressions

Learning Outcomes:

Students should be able to:

6.2.1 List the different arithmetic operators

6.2.2 Define the term binary operator

6.2.3 Define the term operand

6.2.4 List and explain (with examples) what a binary arithmetic expression consists of

6.2.5 Write C/C++ code that uses printf or cout to display the value of different arithmetic expressions (combined with string literal for context)

6.2.6 List and explain the data type produced based on the data type of the operands involved

6.2.7 List and explain the rules involved when creating complex arithmetic expressions containing multiple operands

6.2.8 List and describe the three levels of precedence

6.2.9 Write C++ code that uses printf or cout to display the value of different complex arithmetic expressions (combined with string literal for context)

6.3. Multiple declarations

Content:

Single statement declarations

Learning Outcomes:

Students should be able to:

6.3.1 Identify or correct the general form for multiple variable declarations

6.3.2 Explain why multiple declarations should be of the same type

6.3.3 Write C/C++ code that declares multiple variables in a single line

6.4. Escape characters and New Lines

Content:

Escape characters

6.4.1 Explain what meaning the backslash character will have in C/C++

6.4.2 Define the term escape character

6.4.3 Define the term escape sequence

6.4.4 Write C/C++ code that uses printf or cout to display/use variable (combined with string literal and variable values for context):

* Backslash
* Question mark
* Single and Double quotation
* Horizontal tab
* Alert

6.5. Formatted Output

Content:

Field with manipulators

Learning Outcomes:

Students should be able to

6.5.1 Give examples of where output in an application could be formatted to be more attractive to the user

6.5.2 Explain what field with manipulators are used for

6.5.3 Write C/C++ code that uses printf and formatting strings to:

* Set the field width
* Set the default leading fill character
* Left justify numbers
* Right justify numbers
* Set floating point precision

# Topic 5: Introduction to a high-level programming language

By the end of the module, students should be able to:

* Create a new project using an IDE
* Create a new file with extension CPP
* Explain why C++ source files use the .cpp extension and C++ source files uses the .c extension.
* Write the C++ code to display output “Hello World”.
* Compile a C++ application using an IDE
* Explain what type of file is produced by the C++ compilation process
* Show or hide the IDE message window
* Investigate the IDE message window to determine if compilation succeeded
* Investigate the IDE message window to locate line numbers that contain errors
* Build a C++ application using an IDE
* Explain what file type is produced by the C++ build process
* Run a C++ application using the IDE
* List and explain the different parts of a simple C/C++ application
* Explain case sensitivity as it applies to C/C++
* Explain the term free-format language and how that applies to C/C++
* Explain the term block-structured language and how that applies to C/C++
* Explain what a comment is used for in the C/C++ language
* Explain how the compiler will treat a comment
* List and explain two types of comments allowed by the C/C++ compiler
* Discuss the importance of adding comments to a C/C++ application

**Introduction**

The Geany IDE supports popular programming languages like C/C++, Java, Python3/x, and others. Geany supports over 50 programming languages and works on Linux, Windows, and macOS.

**How to install Geany**

**Step 1**: Search for Geany downloads. You can enter the following url:

<https://www.geany.org/download/releases/>

Step 2: Select your operating system. In our case we select windows.



Figure 5. 1: Geany downloads

**Step 3**: Double click the downloaded executable file and click Next on the screen indicated on Figure 5.2.

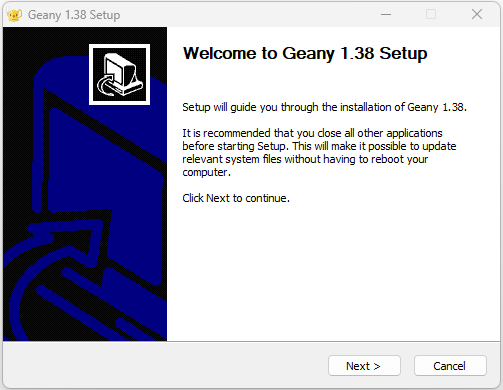


Figure 5. 2: Geany welcome screen

**Step 3**: Accept the license agreement and select full installation

**Step 4**: Select the path from your machine on which Geany must be installed and click next. The screen will look as the one shown in Figure 5.3.

Graphical user interface, text, application, email

Description automatically generated

Figure 5. 3:Choosing installation location

**Step 5**: The last screen will show that Geany has completed the installation process. You can opt to start Geany or show release notes. You will see a screen as the one in Figure 5.4. Click Finish to complete setup process.

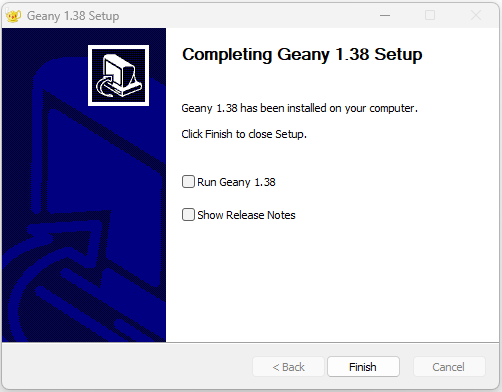


Figure 5. 4: Complete setup

If you do not have the compiler installed you can search for the following address in the browser: <https://jmeubank.github.io/tdm-gcc/download/>

You can then select to download the MinGW which includes GCC C/C++, GNU binutils, mingw32-make, GDB (64-bit), the MinGW-w64 runtime libraries and tools, and the windows-default-manifest package. In our case, you recall we setup our compiler already when we used Code::Blocks. So, you can skip this step.

### Create a new project using an IDE

**Step 1:** Start Geany by tying its name in the search bar or from the task bar alternatively click the Geany icon on the desktop (if one has been created during installation process)

**Step 2:** Click on *Document* from the menu *bar*, select *Set FileType* then Programming Languages and click on *C++ source file.*

**Step 3: By default, you will see some license text in the file called untitled**

### 5.1.2 Create a new file with extension CPP

In section 5.1.1, we learnt how to start Geany program. In this section, we are going to discuss how to create a C++ file with a .cpp file extension.

Step 1: Follow the steps 1 to 3 in section 5.1.1.

Step 2: Select the save icon as indicated in Figure 5.5.

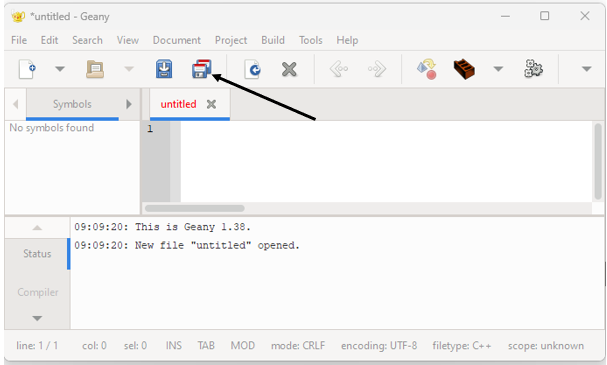


Figure 5. 5:Geany start-up window

Now, you will need to select the path where your program must be saved. You can create a folder where the current program will be saved. In our case, let us create a folder called Helloworld on to the desktop.

Change the file name from untitled to hello\_world.cpp.

If you navigate to the Desktop folder called Helloworld, you will see an image like the one below.

Graphical user interface, application

Description automatically generated

This is a C++ file with extension .cpp.

### 5.1.3 File extension in C and C++

Historically, the first C++ extensions used were .c and .h, just like C. This caused practical issues, particularly with the .c extension, which made it difficult for build systems to distinguish between C++ and C files. Because C language sources typically have the extensions ".c" and ".h," it was common for C++ source files to share the same extensions or use a distinct variation to clearly indicate the C++ code file in the beginning. Today, most C++ implementation files will have the ".cpp" extension and header files will have the ".h" extension (the last one is still shared across most assembler and C compilers).

Other common extension variations for "implementation" code include ".cc", ".C", ".cxx", and ".c++". The same extension variations are used for header files, but the first letter of the extension is usually replaced with a "h," as in ".hh", ".H", ".hxx", ".hpp", ".h++", and so on. For the purposes of this module, we will stick to .cpp

* + 1. Write the C/C++ code to display output “HelloWorld”.

Hello World is commonly used by programmers as the default start program. A "Hello, World!" program is a simple program that displays the phrase "Hello, World!" on the screen.

Using the program called hello\_world.cpp created earlier in section 5.1.2, type the following source code:

**EXAMPLE 5.1**

#include <iostream>

using namespace std;

// main() is where program execution begins.

int main() {

cout << "Hello World"; // prints Hello World

return 0;

}

So let us try to understand the structure of a C++ program. In general, the structure of C++ program is as follows:

|  |
| --- |
| C++ Headers |
| Class definition |
| Member functions definitions |
| Main function |
|  |

Figure 5. 6: General structure of a C++ program

For now, we are not going to dive deeper into the structure. We will try to, compile, and run our project first. We will explain the structure in detail in section 5.2.1. Please note that the Hello world will not display any out even to tell us if there were no errors in the program. The only way to find out will be to compile the source code.

### 5.1.5 Compile a C++ application using an IDE

C++ programs are compiled by converting the source code (.cpp files) into executables or libraries that run on specific platforms. This process can be divided into three key stages:

* **Pre-processing**- Once the pre-processor stage is complete, you will be able to compile the resulting code.
* **Compilation-** C++ compilation is a two-step procedure. First, the compiler converts the source code to assembly language. Second, using an assembler, the assembly language is converted into actual machine code. The resulting output is a collection of files known as an object file. The object code file has .obj or .o file extension
* **Linking**-The final stage results in generation of executable files or libraries.

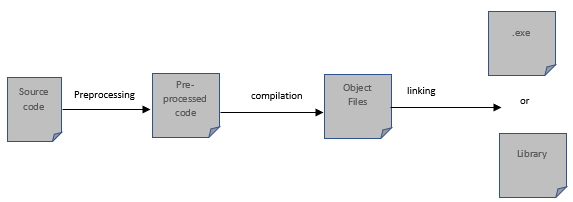


Figure 5. 7: C++ compilation

### 5.1.6: Type of file is produced by the C/C++ compilation process

Let us look in the folder where we saved our project created in 5.1.4. Before compilation, there is one file called hello\_world.cpp

Steps to compile our project

**Step1:** Select *Build* from the menu and select compile. You can click the icon

If you navigate in the folder where we saved our program, you will now see two files as shown in Figure 5.8.

Graphical user interface, text, application

Description automatically generated

Figure 5. 8:C++ Output files

As illustrated in Figure 5.8, the first file is the source code file. The second file (hello.o) is the object file with a .o extension. But for us to see the results, we need to build the program in order to generate an executable file. Executable file will be generated when we select build option from the Build category.

### 5.1.7 Show or hide the IDE message window

Figure 5.11 shows the message window in the Geany IDE.

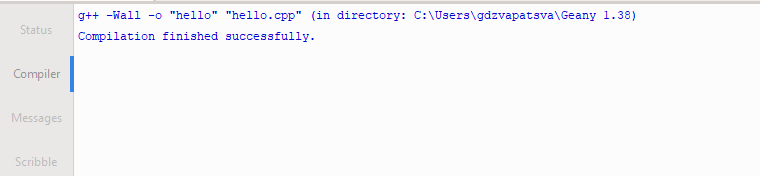


Figure 5. 9:Message window

An optional message window in Geany shows the following tabs:

* Status - A list of status messages. Here is a sample of the status message:

16:34:43: File C:\Users\gdzvapatsva\Geany 1.38\test.cpp saved.

* Compiler - The output of compiling or building programs. In my case, this is what the compiler section of Geany is showing:
* Messages - Results of 'Find Usage', 'Find in Files' and other actions
* Scribble - A text scratchpad for any use.

**Steps to show/hide the message window**

To display the message window, select *View* and check the *Show Message Window* box.

To disable the message window, go to the *View* menu and uncheck the *Show Message Window* checkbox.

5.1.8 Check if compilation succeeded on Message Window

When you compile the C++ program, the Message Window will display if the process was successful or failed. This will be displayed in form a message as shown below:

g++ -Wall -o "hello" "hello.cpp" (in directory: C:\Users\gdzvapatsva\Geany 1.38)

Compilation finished successfully.

5.1.9 Investigate the IDE message window to locate line numbers that contain errors

When using Geany IDE, and your code has errors, these are shown during the build process.

Consider the following code.

EXAMPLE 5.2

// Your First C++ Program

#include <iostream>

using namespace std;

int main() {

<< "Hello World!";

return 0;

}

/\*

Compiling the program causes errors to be displayed in the Message Window. Figure 5.10 shows the errors in the message window:

Text

Description automatically generated

Figure 5. 10:Error messages from the program

As seen in Figure 5.10, Geany does show the line numbers and the suggestion of what the error is about. In the given scenario, it is pointing that an expected primary expression before’<<’ is expected. So, we just type in cout. Recompile, build, and run the program again and the error is not there anymore.

### 5.1.10 Build a C/C++ application using an IDE

Now let us do the practical on performing build on a C++ using the Geany IDE.

**Steps**

Open the hello\_world.cpp file created in section 5.1.2. Just save and compile again in case there were some changes to the file.

**Step1:** Select *Build* from the menu and select compile. You can click the icon

****Step 2:** Select *Build* from the menu bar and click *build.* Alternatively click on the

### 5.1.11 Explain what file type is produced by the C/C++ build process

Let us examine our HelloWorld folder on the Desktop.

Graphical user interface, text, application

Description automatically generated

Figure 5. 11:Build result of C++ program

As you can see in Figure 5.11, we now have an executable file hello\_world with and extension exe. This is the file which we can run to see our output of the program. For now we are going to use an IDE to complete the running of the program.

### 5.1.12 Run a C/C++ application using the IDE

Remember the executable file does not serve its purpose if it has not been run. Users need to see the output of the program and this is achieved by running the program.

Step 1: To run the file, repeat compiling and building in case there were some changes.

Step 2: Click on the  icon to run the program. The output will be as shown in Figure 5.12

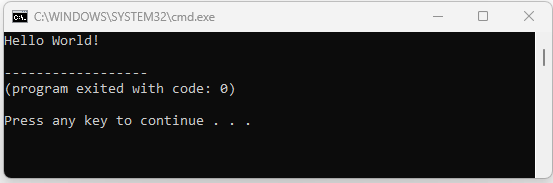


Figure 5. 12:C++ Output

Once you press any key the output disappears. We can also do the same processes using a command line.

Running a program allows users to see the output of the program. When using Geany IDE, running a program is very easy. You will need to follow steps to compile and build the application first. Once done, you need to select Build from the menu bar and click on execute. This will trigger the executable file to be activated and output to be displayed on the command prompt.

It is important to note that, developers can opt for the out to be displayed on different interfaces depending with choice of the organisation. Some of the common interfaces are:

* Command Line Interface (CLI)
* Web Based Interface (WBI)
* Graphical User Interface (GUI)
* Voice user interface ([VUI](https://www.techtarget.com/searcherp/definition/voice-user-interface-VUI))
* Touch user Interface

We briefly discussed CLI and GUI in section1.2.3. We are going to explain them just to keep you up to speed with user interfaces.

**Command Line Interface (CLI)**

Command Line Interfaces accept keyboard commands as input; the commands entered at the command prompt are then executed by the computer. Command-line interfaces can be found in the MS-DOS operating system and the command shell in the Windows operating system. For the purposes of this module, we will stick to command line interface.

**Graphical User Interface**

A graphical user interface (GUI) is a graphical (rather than purely textual) user interface to a computer. This is most common for desktop platforms. Users interact with the program through clicking graphical.

**Web Based Interface**

This is common for online programs where users interact with the program through web browsers. Most organisations have resorted to this interface to allow their users to access their programs from any geographical location.

**Voice User Interface**

Voice user interface (VUI) is a type of speech recognition technology that allows people to interact with a computer program by through voices. The VUI can interpret the voice into machine understandable format.

**Touch User Interface**

This is graphic in character, but it responds to user input through touch. Since developers are becoming more aware of the aspects of human computer interaction, this is now incorporated in the majority of banks.

**How to create a C++ file from Notepad and compile from Command Prompt**

**Step 1:** By now you know how to check if the compiler is installed. If you have forgotten, its pretty simple. Simply open the command prompt and type the following command :g++ -- version.

**Step 2:** Create a folder on the Desktop and give it any name for example:

C++\_Programming. You can do all these using the following commands:

C:\Users\gdzvapatsva>cd Desktop

C:\Users\gdzvapatsva\Desktop>mkdir C++\_Programming

C:\Users\gdzvapatsva\Desktop>notepad

C:\Users\gdzvapatsva\Desktop>

**Step 3:** When Notepad opens, save the file as hello\_world.cpp. Type the following code:

#include <iostream>

using namespace std;

int main (){

std::cout<<"Hello programmers";

return 0;

}

Make sure you change the *Save as type* to All types

**Step 4**: If you open the C++\_programming folder, you will notice our source file hello\_world.cpp is there. So navigate to this path in command prompt.

**Step 5**: Type the following command:

g++ hello\_world.cpp and hit enter

**Step 6**: Type a in the command prompt

The output will be as follows:

Text

Description automatically generated

Figure 5. 13:Command Line output for the compilation

So the program has been compiled and run. If you go and check the folder with our program, you will see an executable file call a. a is a default filename assigned to the executable file.

**Note:** You can also assign a name of your choice to the executable file instead of the default file name a.exe. To do so go to step 5 and modify it as follows:

g++ hello\_world.cpp -o hello.exe

Now you can go back to command prompt and type hello.

The output will still be same as in Figure 5.9.

The -o flag indicates that you are assigning a file name.

**eLink**

<https://www.youtube.com/watch?v=GxHLErBLgI8>

# FORMATIVE ASSESSMENT 5.1 INDIVIDUAL TASK

5.1.1 What is a high-level programming language. (2)

5.1.2 Define the term IDE and give two examples of IDE’s which can be used for C++.(2)

5.1.3 What is the most common file extension for C++ source file. (2)

5.1.4 With the aid of the diagram, explain how C++ files are compiled. (4)

5.1.5 Consider the following C++ program

#include <iostream>

using namespace std;

int main (){

/\*This is the main block

I am learning about multiline comments

\*/

cout<<"my name is Donald";

return 0;

}

Identify the following parts on the program:

* Preprocessor directive
* namespace
* Header file
* Comments
* Main function
* Return statement (7)

5.1.6 Create a file called exams using notepad and save it in exams folder on the Desktop.

Enter the following C++ code:

#include <iostream>

using namespace std;

int main (){

/\*This is the main block

I am learning about multiline comments

\*/

cout<<"Compiling C++ using command line";

return 0;

}

Write a command to compile the source file and produce an executable file called exams1.exe.

(10)

5.1.7 List THREE commands you need to select to compile and run C++ code on an IDE such as Geany. (3)

5.1.8 Differentiate compiling and building in C++. (4)

**Total: 34 marks**

## 5.2 C++ application structure

### 5.2.1 List and explain the different parts of a simple C/C++ application

So far, we have learned how to create a basic C++ program. Next, we will identify the different parts of a simple C++ program. A C++ program is structured in a specific and unique way. A program in C++ is divided into the three sections listed below:

1. Pre-processor directives
2. Namespace
3. Main function
4. Body of main function
5. Opening and closing braces

Figure 5.14 shows the different parts of a simple C++ program

Diagram

Description automatically generated with medium confidence

Figure 5. 14: Structure of a C++ program

So let us explain each of the identified parts and explain them

#### 5.2.1.1 Headers

It is important for us to define preprocessor directives first when discussing the structure of a C++ program. Lines beginning with a hash sign (#) are directives read and interpreted by what is known as the preprocessor. An example of preprocessor directive is the #include. These are typically used to make source programs easy to change and easy to compile. The [preprocessor](https://en.wikipedia.org/wiki/preprocessor) is either a separate program invoked by the [compiler](https://en.wikipedia.org/wiki/compiler) or part of the compiler itself. Before the compiler attempts to compile the resulting source code, the preprocessor performs intermediate operations that modify the original source code and internal compiler options.

Headers or .h files are libraries of code you may insert in your program by including them through referencing them after the preprocessor directives. In C++, all the header files may or may not end with the .h extension.

Syntax of Header File in C++

#include<filename.h>

Example:

#include <iostream>

Here are some common examples of headers:

* #include <iostream> input/output interaction with the program
* #include<time.h>(Time header)- Use date and time functions such as setdate() and getdate() (). To change the system date and obtain the CPU time.
* #include<fstream.h> (File stream) – Used to control the data to read from a file as an input and data to write into the file as an output.
* #include<cmath> (Math header )- Perform mathematical operations like sqrt() and pow(). To obtain the square root and the power of a number respectively.
* <ctype.h>(Character type header)- Use character type functions such as isaplha() and isdigit() ().
* #include <string>

C++ programs must include the header file <iostream>, which stands for input and output stream and is used to take input using the "cin>>" function and display the output using the "cout" function.

#### 5.2.1.2 Name space

A namespace refers to the declarative region containing identifiers (e.g., types, functions, variables, etc.) inside. In a code base that includes multiple libraries, namespaces are useful for organizing code into logical groups and preventing name collisions. All identifiers at namespace scope are unqualifiedly visible to one another.

All C++ standard library types and functions are declared in the std namespace or namespaces nested inside std. The using pre-appended to the namespace instructs the compiler that the following code will use names from the specified namespace. A namespace definition begins with the keyword namespace followed by the namespace name as follows:

**EXAMPLE 5.3**

#include <iostream>

using namespace std;

// main() is where program execution begins.

int main() {

std::cout << "Hello World"; // prints Hello World

return 0;

}

Once declared in the top block, you can leave it out in the function block.

Programmers can also define their own namespaces as shown below.

**EXAMPLE 5.4**

#include <iostream>

using namespace std;

// first namespace

namespace ncv\_level\_two

{

void display()

{

cout << "I am in level two" << endl;

}

}

// second namespace

namespace ncv\_level\_three

{

void display()

{

cout << "I am in level three" << endl;

}

}

using namespace ncv\_level\_three;

int main ()

{

// calling from ncv\_level\_three namespace.

display();

return 0;

}

Output

Graphical user interface, text

Description automatically generated

If we change the namespace we are calling in int main() function to ncv\_level\_two, the output will be: “I am in level two”. So, we used the namespaces to arrange our code nicely and separate code. Although we used the same function name *display(),* our program could identify which one we were referring to at any particular point.

How about if we wanted to call the two display() in the different namespaces. This is easy. We could modify our main function as follows:

**EXAMPLE 5.5**

int main ()

{

// calling all the code in different namespaces

ncv\_level\_two::display();

ncv\_level\_three::display();

return 0;

}

Output

Graphical user interface, text

Description automatically generated

In this program we also introduced a new word called *endl. endl* Inserts a new-line character and [flushes](https://cplusplus.com/basic_ostream::flush) the stream. If we remove it, you will notice that our output is printed in the same line. Its behaviour is equivalent to calling (‘\n’) for inserting new line.

#### 5.2.1.3 Main

This line starts the function declaration. A function is basically a collection of code statements that have been given a name; in this case, the code statements after this one are given the name "main." Functions will be discussed in detail in a later topic, but in general, their definition begins with a type (e.g int), a name (main), and a pair of parentheses (()), optionally followed by parameters.

The *main( )* function is a special function in all C++ programs; it is called when the program is executed. In C++, the main function is the first function to be executed, regardless of where it is located in the code. The main function has two brackets in front of it to all adding parameters. We learned about parameters in level 2. We will not discuss them in detail here but in later topics. After the brackets we have opening brace and closing brace. The code will be placed inside the braces. { signifies the start of a block of code, ​and } signifies the end.

int main() {

}

#### 5.2.1 return

Return statement (s) terminates the execution of a function and returns control to the calling function (or, in the case of main function, transfers control back to the operating system). Execution resumes in the calling function at the point immediately following the call. In the case of our example shown in section 5.2, the return is 0 since the default return value of C++ program is an integer unlike in C programming where the main{} function returns void.

### 5.2.2 Explain case sensitivity

C++ is a case sensitive programming language so; all the keywords must be in lowercase. Case sensitive means that the uppercase and lowercase letters are considered differently. Variable names intNumber1 and intnumber1 are different.

Here is an example of an error message the compile will generate if you try to treat same words differently.

**EXAMPLE 5.6**

Graphical user interface, text, application

Description automatically generated

Figure 5. 15: Case-sensitivity

When you compile the above program, an error message will be displayed in the message window as shown in Figure 5.15.

Graphical user interface, text

Description automatically generated

Figure 5. 16:error messages for case-sensitive

The program generated error because of two intNumber1(i.e line 5and 7) declared. However, the compiler did not see an error between *intNumber1* (line5) and *intnumber1* (line 6).

### 5.2.3 Explain the term free-format language and how that applies to C++

A free-format language is a programming language in which character positioning on the page in program text is irrelevant. Program text does not need to be placed in specific columns. C++ is a free-format programming language. The benefit of free format language is that programmers can type in code without worrying about line breaks. Here is an example of a code snippet.

Logo, company name

Description automatically generated

Figure 5. 17: Free-formatting

Line 7 is a continuation of line 6 and the keyword cout is way in the middle column. When compiled and executed, the program runs fine without generating any error message. Other languages such as Python will generate an “IndentationError: unexpected indent”.

### 5.2.4 Explain the term block-structured language and how that applies to C++

**DEFINITION**

A block-structured programming language is one that allows the creation of blocks, including blocks nested within other blocks.

A block consists of a sequence of statements and/or blocks, preceded by declarations of variables. Block structured languages define blocks of scope using some sort of delimiter pair for e.g. braces in C++ { }. A code block can be the body of a function, or it can be controlled by conditional execution (if statement) or repeated execution (while statement, for statement, etc.) We will explain what conditional statements are in later topic 8 and 9. Consider the code shown in example 5.7

**EXAMPLE 5.7**

#include <iostream>

using namespace std;

namespace addition

//This is a block of code

{

void add\_numbers(){

int age=19;

cout<<"The age is "<<age;

}

}

int main ()

//This is the main block

{

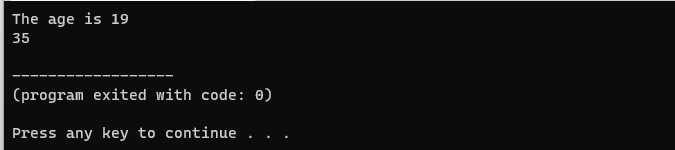
//calling a block of code

addition::add\_numbers();

cout<<endl<<7\*5 ;

}

Output



You can see from example XXX that the code is split into structured block. Entry to a block is seen with braces. In the example, there are two blocks of code: the main block and the addition block. If we write the following statement in the main block:

cout<<age;

We will get an error because it was declared outside the main block and its scope not recognised.  The scope of age is confined to the block in which it has been declared only.

### 5.2.5 Explain what a comment is used for in the C++ language

Comments in C++ are hints that a programmer can add to make their code easier to read and understand. C++ compilers ignore comments completely and treats them as white space. Comments are normally used to annotate code for future reference.

**Syntax of a comment**

//declare variable for first number

Int intnumber1=10;

You can also create a comment that displays at the end of a line of code. But generally, its a better practice to put the comment before the line of code.

### 5.2.6 Explain how the compiler will treat a comment

C++ compiler ignores comments. Preprocessors in C++ compilers analyze each character in a source file sequentially, before any other compilation phase. It ignores the beginning of a comment, either /\* or //. When it does, it continues to consume source file characters, throwing them away until it encounters either a \*/ or the end of line character (for single-line comment). Regardless of how the comment ends, the preprocessor continues processing subsequent characters normally until another comment begins. The preprocessor typically replaces the entire comment with a single whitespace character (e.g., space, tab, newline). In some cases lexical analyser are responsible for detecting and throwing away comments. It is the lexical analyzer's job to break down the source code into tokens, which are then fed into the parser. When the lexical analyser meets // or /\* \*/, it sends nothing to the parser and continues to do so until it finds valid token.

### 5.2.7 Two types of comments allowed by the C++ compiler

There are two ways to add comments to code:

// - Single Line Comments

/\* \*/ -Multi-line Comments

**Single Line comments**

EXAMPLE

In C++, any line that starts with // is a comment as shown in Figure 5.18.

A picture containing graphical user interface

Description automatically generated

Figure 5. 18:Single-line comments

Line 5 and line 7 are never executed when the program is compiled and build.

**Multi-line comments**

In C++, any line between /\* and \*/ is als**o a comment**. For example,

Graphical user interface

Description automatically generated with medium confidence

Figure 5. 19:multi-line comments

Comments can also be used to disable code to prevent it from being executed. As an example, if we encounter an error while running the program, we can use comments to disable the error-prone code from executing; this can be useful for debugging. Comments should describe what the program does at a high level. Reiterating something that is obvious is not a good idea.

### 5.2.8 Importance of adding comments to a C++ application

Comments are import in programming only to the programmers. Users will never see the program comments as they are not executed by the compiler.

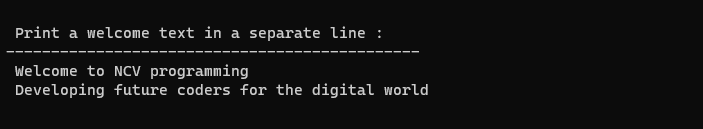
**Importance of comments**

1. Saves time- new programmers who join the organisation can quickly understand the code segments by reading comments
2. Comments helps to identify important blocks of code easily
3. Comments help to understand why programmers used a certain way to get a solution rather than the other option
4. Comments help to add description and clarity of what certain blocks of code do by adding context

# FORMATIVE ASSESSMENT 5.2 INDIVIDUAL TASK

5.2.1 Explain each of the following terms as found in C++ program. (6)

5.2.2 Write a program in C++ to print 3 lines of text each in its on line. The output must look as follows.



(8)

5.2.3 Consider the following C++ program which is supposed to add two numbers hardcoded in the program by the programmer making use of user defined namespaces.

using namespace std;

int main()

{

addition:add\_numbers;

}

namespace addition

//This is a block of code

{

void add\_numbers(){

cout << " The sum of 29 and 30 is : "<< 29+30 <<"\n\n" ;

}

}

(5)

5.2.4 State whether the following statements are true or false.

a) Since C is a subset of C++, all C programs will run under C++ compilers.

b) By merely looking at one or two lines of code, we can easily recognise whether a program is written in C or C++.

c) The main {} function in C++ and in C are the same. (3)

5.2.5 What are comments in C++? (2)

5.2.6 List TWO ways of adding comments in C++ program. (4)

5.2.7 Explain the term free-format language and how that applies to C++ . (2)

**Total:30 Marks**

# SUMMATIVE ACTIVITY 5.3 INDIVIDUAL TASK

5.3.1 Outline the steps to create and run a new C++ project called addition using Geany. (8)

5.3.2 C++ is case sensitive. True or False (1)

5.3.3 Consider the code below and answer the associated questions.

#include <iostream>

using namespace std;

namespace languages

//This is a block of code

{

void display(){

cout << " C++ is an extension of C ";

}

}

namespace jobs

{

void display(){

cout << " I want to work as a backend developer";

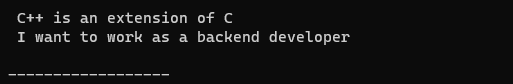
}

}

int main(){

}

Complete the given code so as to produce the following output.



Hint: You have to make all namespaces available in the main function. (5)

5.3.4 At which stage is the object file produced in C++. (2)

5.3.5 Explain the following types of categories of errors. (4)

5.3.6 Define block structured language with examples. (4)

5.3.7 List FOUR importance of adding comments to the code. (4)

5.3.8 What is meant by the term breakpoint as applied in debugging.

5.3.9 With the aid of an example, explain what a syntax error is in C++. (3)

**Total :31 Marks**

# References

<https://learn.microsoft.com/en-us/cpp/cpp/namespaces-cpp?view=msvc-170>

# Topic 6: Data Types, Variables and Output

6.1. Data types in C/C++

After you have completed this module, you should be able to :

* List Data Types in C/C++
* Define the term data type
* Define the term literal value
* Name and explain two numerical data types used in C/C++
* Explain the difference between Integer data types and floating-point data types
* List the different integer data types
* List the different floating-point data types
* Explain the float, double data types in terms of:
* What the type can consist of
* What the maximum and minimum value can be for the type
* What operations is allowed on the type
* Example literal values allowed for the type
* Example literal values not allowed for the type
* Whether the type is signed or unsigned
* Number of bytes memory used to store type value
* Explain the int, char, bool, float double data types in

terms of:

* What the type can consist of
* What the maximum and minimum value is
* What operations is allowed on the type
* Example literal values allowed for the type
* Example literal values not allowed for the type
* Whether the type is signed or unsigned
* Number of bytes memory used to store type value
* Use variables as part of a solution
* Define the term variable
* Explain how variables are used in C/C++ applications
* Identify and list C/C++ key words
* List and describe the syntax rules for naming variables
* Write C/C++ code to create variables of different types
* Write C/C++ code to assign literal values to variables of different types
* Write C/C++ code store the result of an arithmetic expression (simple and complex) in a variable
* Write C/C++ code to print the value associated with a variable (combined with string literal for context)
* Write C/C++ code store the result of an arithmetic expression that contains variables on both sides of the operand (simple and complex) in a variable
* Write C/C++ code to assign the value of one variable to a second variable
* Explain the difference between variable declaration and variable initialisation
* Write C/C++ code to initialise variables of different types
* Write C/C++ code to increase or decrease the value of a variable (accumulation statements)
* Write C/C++ code to increase or decrease the value of a variable with 1 using the increment
* Operator
* Write C/C++ code to print the value associatedwith a variable (combined with string literal for context)
* Arithmetic Operations
* List the different arithmetic operators
* Define the term binary operator
* Define the term operand
* List and explain (with examples) what a binary arithmetic expression consists of
* Write C/C++ code that uses printf or cout to display the value of different arithmetic expressions (combined with string literal for context)
* List and explain the data type produced based on the data type of the operands involved
* List and explain the rules involved when creating complex arithmetic expressions containing multiple operands.
* List and describe the three levels of precedence.
* Write C++ code that uses cout to display the value of different complex arithmetic expressions (combined with string literal for context)
* Multiple declarations
* Identify or correct the general form for multiple variable declarations
* Explain why multiple declarations should be of the same type
* Write C/C++ code that declares multiple variables in a single line
* Escape characters and New Lines
* Escape characters
* Explain what meaning the backslash character will have in C/C++
* Define the term escape character
* Define the term escape sequence
* Write C/C++ code that uses cout to display/use variable (combined with string literal and variable values for context):
* Backslash
* Question mark
* Single and Double quotation
* Horizontal tab
* Alert
* Formatted Output
* Give examples of where output in an application could be formatted to be more attractive to the user
* Explain what field with manipulators are used for
* Write C/C++ code that uses printf and formatting strings to:
* Set the field width
* Set the default leading fill character
* Left justify numbers
* Right justify numbers
* Set floating point precision

## 6.1. Data types in C/C++

**Introduction**

In programming, a data type is a categorization that determines what type of value a variable has and what mathematical, relational, or logical operations can be performed on it without creating an error. An integer (int) is a data type that is used to categorise entire integers.

### 6.1.1 Data Types

When working with computer programs, data must be stored in the computer's memory so that you can refer to it for future use and allocate it appropriate space. The data that our programs deal differs depending on what you want to store. In this section we are going to list the different data types. It's worth noting that the data types we discussed in Topic 3 working with Arduino are still the same data types we're discussing and reinforcing here. It is important to mention that there are three categories of data types and these are:

* Primitive/Built-in data types e.g int, float, char, wide character, Boolean, double, void
* Abstract data types e.g functions, Array, Pointers and Reference data types
* Derived data types e.g class, structure, union, enumeration, Typedef defined data types

Primitive data types will be the main topic of this section. A primitive data type is one that can only hold values that are of a very basic nature, such as a number, character, or truth value. Primitive data types serve as the foundation for more complicated data types and are the most fundamental building blocks for all programming languages. The four modifiers for the data types are as follows:

* Signed
* Unsigned
* Short
* Long

The data type modifiers can be used to change some of the core data types further e.g. short int, long int etc. Table 6.1 below shows the different primitive data types.

Table 6. 1: Different primitive data types in C++

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Type** | **Size in Bytes** | **Explanation** | **Example** |
| int | 2 Bytes | Stores a 2 byte(16 bits) signed integer value that is in range of -32,768 to 32,767. | 96; 57; |
| char | 1 Byte | The char datatype can store any number of character set. The literals are written inside a single quote like ‘a’, ‘#’ etc. Ranges from 4-127 to 127 or 0 to 255 | ‘M’; |
| Float | 4 Bytes | Stores a signed 4-byte(32-bit) value that is integer or a value with decimal point (say 12.15) that is in range of -3.4028235E+38 to 3.4028235E+38. | 45E-2; 56.8 |
| Double | 4 Bytes | The double data type is also used for handling the decimal or floating-point numbers. | 45.67788 |
| Unsigned int | 2 Bytes | Stores an unsigned integer value of 2 bytes(16 bits) that is in range of 0 to 65,536 | 15 |
| Wide character | 8 bits | char takes 1-byte space, but wide character takes 2-bytes (sometimes 4-byte depending on compiler) of space in memory. wide char can take on 65536 values which corresponds to UNICODE values | L'x’ |
| short | 2 bytes | The short is an integer data type that stores two bytes. Range is from -32768 to 32767 | 125 |
| long | 4 Bytes | Stores a 4 byte (32 bit) signed integer value that is in range of -2,147,483,648 to 2,147,483,647 | 12 365 427 |
| void |  | The void keyword is used only in function declarations. It indicates that the function is expected to return no information to the function from which it was called. |  |

### 6.1.2 Defining data types

In C++, data types are declarations for variables. This determines the type and size of data associated with variables. For example,

int mark\_test1=56;

The above declaration shows the data type of int for the variable mark\_test1. You will notice that we have been working with these, but we did not go into detail to explain the data types.

### 6.1.3 Define the term literal value

**VOCABULARY**

Literals are data used for representing fixed values. For example, 3.8, 9, ‘b’ etc.

Literals can be used directly in the code, and you cannot assign different values to literals.

Literals contain memory but they do not have references as variables. Generally, both terms, constants, and literals are used interchangeably. There are 5 types of literals in C++ and these are:

* Integer literal
* Float literal
* Char literal
* String literal
* Boolean literal

**Integer literals**

An integer literal is a literal number without any fractional or exponential parts. An integer literal can be a decimal, octal, or hexadecimal constant. A prefix specifies the base or radix: 0x or 0X for hexadecimal, 0 for octal, 0B, and nothing for decimal. Table 6.2 explains the different integer literals.

Table 6. 2: Integer literals

|  |  |  |
| --- | --- | --- |
| **Type of integer literals** | **Description** | **Example** |
| Binary-literal (base 2): | 0b or 0B followed by one or more binary digits(0, 1). | 0b1001, 0B1010 |
| **Octal-literal (base 8):** | a **0** followed by zero or more octal digits(0, 1, 2, 3, 4, 5, 6, 7). | 045, 076, 06210 |
| **Decimal-literal(base 10** | A **non-zero decimal digit** followed by zero or more decimal digits(0, 1, 2, 3, 4, 5, 6, 7, 8, 9). These are the common integer numbers which you are all familiar with. | 87, 43 |
| Hexadecimal literal(base 16) | 0x or 0X followed by one or more hexadecimal digits(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, A, b, B, c, C, d, D, e, E, f, F). | 0x7f, 0x2A |

Another way to represent integer literals is by using a suffix appended at the end of a number. The prefix of the integer literal indicates the type into which it is to be read. The common suffixes are:

LL -long long integer

U or U- Unsigned integer values

L or l -long integer

ULL or ull- unsigned long long integer

Example 6.1 shows a sample code and output for some integer literals

**EXAMPLE 6.1**

#include <iostream>

using namespace std;

const int num\_1 =75;

const int num\_2=123456765465653LL;

const int num\_3=0b1001;

const int num\_4=045;

const int num\_5=0x7f;

int main()

{

cout << num\_1 << endl; //Printing a integer literal

cout << num\_2 << endl; //Printing a Long Long Integer

cout<<num\_3<<endl; //printing Binary

cout<<num\_4<<endl; //printing octal number

cout<<num\_5<<endl; //printing hexadecimal

return 0;

}

**Output**

**75**

**-2069457867**

**9**

**37**

**127**

**FLOAT LiTERALs**

A floating-point literal is a numeric literal that has either a fractional form or an exponent form. For example: -3.2, 0.00008, -0.44E-5. Take note: E-5 is the same as 10-5. The following are examples of invalid float literals:

823E is invalid because of missing exponent.

3250f is invalid because of missing decimal or exponent.

.e879 is invalid because of missing integer before the decimal comma.

**EXAMPLE 6.2**

#include <iostream>

using namespace std;

const float height=1.7;

const float distance1=0.0008E-2;

const float value\_1=10.5E-3; // equivalent to 10.5 x 10 pow -3

int main()

{

cout << height << endl;

cout << distance1<< endl;

cout << value\_1 << endl;

return 0;

}

**Output**

**1.7**

**8e-06**

**0.0105**

Value\_1 output is printed as 0.0105 after evaluating 10.5 x 10-3.

**NOTE**

So, you notice that although we declare some floating-point literals, some of the answers where powers or exponents are involved as a result of evaluating the expression.

**Character literals**

A character is turned into a literal by being enclosed in single quotation marks. For example:

char letter = 'm';

char country=L 'Z';

If the literal begins with L (uppercase only), it is a wide character literal (wchar)(e.g., L'x') and should be stored in wchar\_t type of variable. wchar\_t is only used in C++.

**STRING LITERALS**

String literals are enclosed in double quotes. A string may contain combinations of characters that are similar to character literals: plain characters, escape sequences, and universal characters.

*Examples of string literals*

"fair " represents string constant

" " represents empty spaces

" x"-This is a string literal with one character and is not treated a s a character

" I am a \ South African " represents sting in two lines.

Here is sample code for string literals.

**EXAMPLE 6.3**

#include <iostream>

using namespace std;

const string grade="fair";

const string citizen="I am a \nSouth African";

const string empty\_space=" ";

int main()

{

cout << grade << endl; //Printing a string literal constant"

cout << citizen << endl; //Printing a string in two lines

cout<<empty\_space<<endl; //printing empty space

return 0;

}

**Output**

fair

I am a

South African

**Basic Operations with strings**

Let’s go into specifics about the string manipulations you’ll be doing the most.

**Counting the number of characters in a string**

**.length()**-This method returns the number of characters in a string, including spaces and punctuation. We invoke member functions using dot notation as we do with many string operations.

Example

string firstname= "South Africa";

cout<<firstname.length()<<endl;

OUTPUT

12

The output above gives 12 including the space between South and Africa.

**Accessing individual characters**

Using square brackets, you can access individual characters within a string as if it were a char array. String str positions are numbered from 0 to str.length() - 1.

string firstname= "South Africa";

cout<<firstname[4]<<endl;

OUTPUT

h

**Comparing two strings**

You can compare two strings for equality using the == and != operators.

Example

string country1= "South Africa";

string country2="Zambia";

bool result=(country1==country2);

cout<<result<<endl;

OUTPUT

0

The output of 0 means they are not equal. If we replace the word Zambia with South Africa and run the program, the output will be a 1. We will discuss this in detail in Topic 8. But just know that 0 represents false and 1 represents true. We are also going to discuss more about string manipulation in topic 12. For now, you should have grasped the fundamentals of working with strings and a bit of string manipulation.

**BOOLEAN LITERALS**

Boolean literals are used to represent the boolean data types which can only be either true (1) or false (0). C programming does not have boolean literals. Example 6.4 is a sample code and output for boolean literal usage.

**EXAMPLE 6.4**

#include <iostream>

using namespace std;

const bool isLate=true;

const bool isDegree=false;

string msg1="isLate is ";

string msg2= "isDegree is ";

int main()

{

cout << isLate<< endl; //prints a 1 for true

cout <<isDegree<< endl; //prints a 0 for false

return 0;

}

**Output**

**1**

**0**

**NOTE**

Literals contain memory but they do not have references as variables.

### 6.1.4 Name and explain two numerical data types used in C++

C++ has two numerical data types as explained in section 6.1.3 and these are:

* Integer data type- holds only integer values
* Floating point data type- holds floating point values.

To distinguish the two above, let us look at the sample code in example 6.5 where:

* first\_number holds 56 which is an integer
* second\_number holds 65.23 which is a float though declared as integer

**EXAMPLE 6.5**

#include <iostream>

using namespace std;

int first\_number=56;

int second\_number=65.23;

int total;

int main(){

total=first\_number+second\_number;

cout << total<< endl; //printing the result

return 0;

}

**Output**

**121**

result holds the sum of first\_number and second\_number and the value is an integer. So because result is an int data type, the value after the decimal point is not printed. If we try to print the value of second\_number, 65 will be printed instead of 65.23. If we change the data type of result without changing the data type of second\_number the output will still remain as an integer. However, if we change the data type of second\_number and result to float, the output will be 121.23.

### 6.1.5 Key difference between Integer data types and floating-point data types

* Integers do not have any fractional parts, while a float is a number that can only be expressed as a decimal.
* Integer datatypes can be represented in four forms i.e., hexadecimal, octal, decimal, or binary form.
* Floating point can be represented using decimal number system.
* When it comes to memory requirement, **generally** int require 2 bytes while floats require 4 bytes.
* Any combination of digit from the set 0 through 9 consist of Integer data type. A decimal or exponent must be present in floating point data type.
* Floats have only good precision up to 6 digits. Beyond that float is not the best since  it will truncate anything after that although it takes less memory.

### 6.1.6 List the different integer data types

Table 6.3 shows the different integer variants and their sizes.

Table 6. 3: int data type variants

|  |  |  |
| --- | --- | --- |
| **int variant** | **Typical bits width** | **Typical Range** |
| int | 4bytes | -2147483648 to 2147483647 |
| unsigned int | 4bytes | 0 to 4294967295 |
| signed int | 4bytes | -2147483648 to 2147483647 |
| short int | 2bytes | -32768 to 32767 |
| unsigned short int | 2bytes | 0 to 65,535 |
| signed short int | 2bytes | -32768 to 32767 |
| long int | 8bytes | -9223372036854775808 to 9223372036854775807 |
| unsigned long int | 8bytes | 0 to 18446744073709551615 |
| long long int | 8bytes | -(2^63) to (2^63)-1 |
| unsigned long long int | 8bytes | 0 to 18,446,744,073,709,551,615 |

\* The size might be different from those shown in the above table, depending on the compiler and the computer you are using.

We can make use of sizeof to check the size for a specific architecture using the following sample code:

**Example 6.6**

#include <iostream>

using namespace std;

int main()

{

cout << "Size of int : " << sizeof(int) << endl;

return 0;

}

Output

Size of int : 4

**Task 6.1**

Write a program to show the following output making use of **sizeof** keyword

The following are the sizes of different data types :

------------------------------------------

The sizeof(char) is :1 bytes

The sizeof( short) is :2 bytes

The sizeof(int) is :4 bytes

The sizeof(long) is :4 bytes

The sizeof(long long) is :8 bytes

The sizeof(float) is :4 bytes

The sizeof(double) is :8 bytes

The sizeof(long double) is :16 bytes

The sizeof(bool) is :1 bytes

### 6.1.7 List the different floating-point data types

There are three different types of floating point data types: **float, double, and long double**. On modern architectures, floating point representation almost always follows the IEEE 754 binary format. In this format, a float is 4 bytes, a double is 8, and a long double can be equivalent to a double (8 bytes), 80-bits (often padded to 12 bytes), or 16 bytes.

### 6.1.8 Explain the float, double data types in terms of:

Float and double data types are used to store floating point numbers. Float has a size of 4 bytes, double accommodates 8 bytes, and long double has a capacity of 16 bytes of storage size. The precision of double is twice that of float. The double data type is also used for exponentials. e.g.

double distance = 45E12 // 45E12 is the same as 45 \* 10 ^ 12.

These decimal-oriented data types can hold values with up to 15 digits of numbers and can either have the decimal point prefixed or suffixed. The range of the data type can be varied with values from 0 × 10−345to 1.7 × 10308.  When dealing with a large decimal number, it is preferable to use the double data type. Here is an example of double data type implementation. By converting degrees Celsius to degrees Fahrenheit, we will demonstrate the use of the double data type in Example 6.7.

**EXAMPLE 6.7**

#include <iostream>

//Coverting celcius to fahrenheight

using namespace std;

//declaring the variables

double celcius, fahrenheight;

int main()

{

//accepting input

cout << "Please enter the temperature in degrees celcius" << endl;

cin>>celcius;

//processing

fahrenheight=(celcius \* 1.8) + 32;

//displaying output.

cout<<"The temperature in fahrenheight is :" <<fahrenheight <<"degrees fahrenheight"<< endl;

return 0;

}

**Output**

**Please enter the temperature in degrees Celsius**

**27**

**The temperature in fahrenheight is :80.6 degrees Fahrenheit**

Remember, C++ doesn't support unsigned floating-point types because most floating-point hardware doesn't support unsigned floating-point types. The values that can be stored in a double-type variable range from 1.7E - 308 to 1.7E + 308.

### 6.1.9 Explain the int, char, bool, float double data types

Table 6.1 has outlined the different data types such as int, char, bool, float and double with all the allowed values and whether the data type is signed or not. In this section we are going to summarise the data types for the purposes of emphasis.

int- is used to refer to integers meaning whole numbers.

Example

int age = 35;

char-The keyword char represent characters and are enclosed in single quotes.

Example

char dress\_size= 'm';

If you use double quotes, then this becomes a string.

**NOTE**

String is not a primitive data type but rather a derived data type.

float-float is the keyword used to hold floating-point numbers (decimals and exponentials).

Example

float temp = 37.5;

Double Floating Point-double is the keyword used to hold floating-point numbers (decimals and exponentials) with double precision.

Example

double val = 1527.1251;

bool -The boolean data type's keyword is bool holds True or False values and mainly used in conditional and loop statements.

Example

bool is\_registered = true;

So how can we print out the data type of a variable. To get the datatype of variable, use typeid(x).name() of typeinfo library. It returns the type name of the variable as a string.

**Syntax**

The syntax to get the type name of a variable **x** using typeid() is:

typeid(x).name()

Example 6.8 code demonstrates the use of typeid().name to display the different data types.

**EXAMPLE 6.8**

Here is an example

#include <iostream>

using namespace std;

int main()

{

//declaring variables and assigning values

int num\_1=56;

char dress\_size='m';

bool is\_registered=true;

double value\_1=1256.1365;

float answer=45E-3;

//printing the data types

cout << "Type of num\_1 : " << typeid(num\_1).name() << endl;

cout << "Type of dress\_size : " << typeid(dress\_size).name() << endl;

cout << "Type of is\_registered : " << typeid(is\_registered).name() << endl;

cout << "Type of value\_1 : " << typeid(value\_1).name() << endl;

cout << "Type of answer : " << typeid(answer).name() << endl;

return 0;

}

**Output**

**Type of num\_1 : i**

**Type of dress\_size : c**

**Type of is\_registered : b**

**Type of value\_1 : d**

**Type of answer : f**

So, our output does not print the full data type name but just an initial of the data type like i for integer, c for char, b for bool, d for double, and f for float.

### 6.1.10 Use variables as part of a solution

**Defining the term variable**

**VOCABULARY**

A variable is a place to store data and values in memory location.

Variables are crucial because they determine how much space they take up in the storage and how the stored bit pattern is interpreted. Memory is set aside for storing the variable and the variable is given a name that allows us to access it at a given time. Variables consist of a name, a value, and a type.

Here is an example of a variable declaration:

int x =13;

int is the data type, x is the variable name and holds a value 13.

**NOTE**

Values of variables can change.

Variables can be defined using a combination of letters, digits, or special symbols like underscore (\_) and assigned different data types like char, int, float, and double. Variables can be anything except the reserved keyword; the first letter of the variables must start with the letter only.

**Explain how variables are used in C++ applications**

The declaration of variable tells the compiler what type of data variables the program will use. Whenever you declare variables, you tell the compiler how much storage they need. The compiler does not have to worry about the storage until it is declared.

**C++ key words**

Keywords, also known as **reserved words,** have special meanings to the C++ compiler and are always written in lower case. Keywords cannot be used when naming variables. If you declare your variables as follows:

float int =34.1;

Because the word "int" is already understood by the C++ compiler, you will receive a compilation error. Some examples of keywords include:

namespace, void, long, char, typeid, for, if, false, class, switch, sizeof, and, using etc.

For a full list of reserved words, see the link below:

<https://en.cppreference.com/w/cpp/keyword>

**Rules for naming variables**

**These have been covered already in Topic 3. It is important to reinforce the concept to solidify your understanding.**

* Variables can consist of both uppercase (A-Z) and lowercase(a-z) letters.
* Variables can contain numbers 0 to 9, but cannot start with a number.
* Variables may not be keywords, e.g. you cannot have a variable named float.
* Variables must have unique names, i.e. you cannot have two variables with the same name.
* Variable names are case sensitive, so **Count** and **count** are two *different* variables.
* Variables may not contain any special characters, except the underscore (\_)
* Variables names must be meaningful such that who ever reads the code can quickly get an idea of what it is for.

**Examples of valid variables**

* **result**
* **Test\_2**
* **\_total**
* **first\_4\_marks**

**Examples of invalid variables**

* Sum-1   // containing special character '-'.
* 2data    // the first letter is a digit.
* break    // use of a keyword.
* %name //can not start with a special character %

**Difference between variable declaration and variable initialisation**

Variable declarations tell the compiler where a certain entity is located in a program.  Declaring a variable and initializing it are good practices. Initializing a variable involves assigning it a value. The example below demonstrates the difference between variable declaration and initialisation.

//variable declaration

int age;

//declaring initialisation

age=0b10;

As you may notice, the two lines of code can be shortened by declaring and initialising in the same line as:

//variable declaration and initialisation

int age=0b10;

Example 6.9 shows declarations of different data types

**EXAMPLE 6.9**

// C++ program to show difference between

// definition and declaration of a

// variable

#include <iostream>

using namespace std;

int main()

{

// this is declaration of variable a

int num1;

// this is initialisation of a

num1 = 10;

// this is definition = declaration + initialisation

int num\_2 = 20;

// declaration and definition

// of variable 'a123'

char a123 = 'a';

//declaring a wide character

wchar\_t=L’x’

// This is also both declaration and definition

// as 'c' is allocated memory and

// assigned some garbage value.

float c;

// multiple declarations and definitions

int \_c, \_d45, e;

//declaring Boolean

bool is\_painted=false;

// Let us print a variable

cout << a123 << endl;

return 0;

}

We are going to declare two variables of integer types and assign decimal and binary literals and then add them.

**EXAMPLE 6.10**

#include <iostream>

using namespace std;

int main()

{

//declaring variables and assigning values

int first\_number=45;

int second\_number=0b10;

int sum=first\_number+second\_number;

//printing the data types

cout << "sum of " <<first\_number << "+"<< second\_number<< "="<<sum<< endl;

return 0;

}

Output

sum of 45+2=47

In Example 6.10, second\_number is assigned a value of 0b10, which is a binary literal representing 2. When added together, the compiler added 45 and 2, which is why the answer is 47. We also combined  a string when printing our values for the sum by separating them with the << symbol. It is also possible to have the string literal in one line and the actual value for the final answer in another. We will simply have to modify our code as shown in Example 11.

**EXAMPLE 6.11**

#include <iostream>

using namespace std;

int main()

{

//declaring variables and assigning values

int first\_number=45;

int second\_number=0b10;

int sum=first\_number+second\_number;

//printing the data types

cout << "sum of " <<first\_number << "+"<< second\_number<< "=\n"<<sum<< endl;

return 0;

}

**Output**

**sum of 45+2=**

**47**

So, we used the \n for making splitting the string literal into another line.

It is also possible to declare more than one variable in one line if they are of the same data type as follows:

//declaring more than one variables and assigning values

int first\_number=45, second\_number=0b10, sum;

**Task 6.2**

What is the output of the following code below:

#include <iostream>

using namespace std;

int main()

{

//declaring variables and assigning values

int age=0b10;

int age1=0b01;

int sum=age1+age;

//printing the data types

cout << "sum is " <<sum << endl;

return 0;

}

We can also assign a value to another second variable. For instance, if we declare and initialise first number, we can then declare and assign the value of the second number to be the first variable, as shown in example 6.12:

**EXAMPLE 6.12**

#include <iostream>

using namespace std;

int main()

{

//declaring variables and assigning values

int first\_number=45;

int second\_number =first\_number;

//printing the data types

cout << "The value of second\_number is :" << second\_number<< endl;

cout << "The value of first\_number is :" << first\_number<< endl;

return 0;

}

**Output**

**The value of second\_number is :45**

**The value of first\_number is :45**

**Incrementing values of variables by using ++**

To increase the value of a variable by one, use the increment operator ++ in C++. If we use the increment operator in our print statement for first\_number, the result will be 46, although the value of second\_number will remain at 45. The same applies with the decrement operator. To illustrate this, look at the code in example 6.13 modified by adding the increment operator.

**EXAMPLE 6.13**

#include <iostream>

#include <cmath>

using namespace std;

int main()

{

//declaring variables and assigning values

int first\_number=45;

cout << "The initial value of first\_number is :" << first\_number<< endl;

//assigning an incremented value

int second\_number =++first\_number;

//printing the data types

cout << "The value of second\_number is :" << second\_number<< endl;

cout << "The new value of first\_number is :" << first\_number<< endl;

return 0;

}

**Output**

The value of first\_number is :45

The value of second\_number is :46

The value of first\_number is :46

Let us explain the code. First, we printed the value of first\_number, which is 45. When we assigned first\_number to the variable second\_number, we added an increment operator to the effect that it added a 1 to the previous value of first\_number and printed 46. When we printed first\_number for the second time after the increment, the value 46 was printed.

Let us consider the code in example 6.14.

**EXAMPLE 6.14**

#include <iostream>

using namespace std;

int main()

{

int a=5;

int b=8;

int c=++a;

int d=b;

int z=++b+a;

cout<<"The value of a is :"<< a <<endl;

cout<<"The value of b is :"<< b<<endl;

cout<<"The value of c is :"<< c<<endl;

cout<<"The value of d is :"<< d<<endl;

cout<<"The value of z is :"<< z<<endl;

return 0;

}

After running the program, the output is printed as follows:

The value of a is :6

The value of b is :9

The value of c is :6

The value of d is :8

The value of z is :15

Explain each line of output. How can you modify c to make sure that it holds the original value of a but with the increment sign.

* The first output is 6 because the variable a was only printed after being incremented.
* Value of b is 9 because it was printed after being incremented in the statement z=++b+a;
* c is 6 because the variable a is incremented and assigned to C.
* Value of d has the original value because it has not been incremented at the time of being printed.
* z is 15 because the value of b is incremented first to give 9 and a has been incremented already to 6. 9+6=15.
* c=a++

**Decrementing values of variables by using ++**

Similarly, the decrement operator -- decreases the value of a variable by 1.

Consider the following code in example 6.15:

**EXAMPLE 6.15**

#include <iostream>

using namespace std;

int main()

{

int a=5;

int b=8;

cout<<"The value of a is :"<< --a <<endl;

cout<<"The value of b is :"<< b--<<endl;

cout<<"The new value of b is :"<< b<<endl;

return 0;

}

**Output**

The value of a is :4

The value of b is :8

The new value of b is :7

The first output line generates 4 because of the decrement operator being positioned before the variable. The second output line displays the same value because the decrement sign was printed after the variable. Despite this, the middle sentence causes the value to be decreased by 1 when we try to print b again.

**Complex calculations**

We now want to do complex evaluation of arithmetic expressions. In this case we are going to solve a quadratic equation. Our quadratic equation is given as:

0b01x2 + 14x + 0x2D = 0

What is important is to understand the simplest form of the quadratic equation and identify the values of a, b and c.

In the given equation:

a=0b01

b=14

c=0x2D

Take note that in the given example, our expression is using binary literal for the value of a and hexadecimal literal for the value of c. in simplest form, the decimal equivalent for a is 1 and 45 for c.

In addition, as you will note, we will use to use some mathematical functions such as power and square root. To do that, we will add a c++ header called cmath to allow us to use sqrt function as shown in example 6.16

**EXAMPLE 6.16**

#include <iostream>

#include <cmath>

using namespace std;

//declaring the variables

float x1, x2;

int main()

{

// x2 + 14x + 45 = 0 a=0b01 b=14 c=0x2D

//calculating the roots of x1 and x2

x1=(-14+ sqrt((14\*14)-4\*1\*0x2D))/(2\*0b01);

x2=(-14- sqrt((14\*14)-4\*1\*0x2D))/(2\*0b01);

//Displaying the output

cout << "The value of x1 is :"<<x1<< endl;

cout << "The value of x2 is :"<<x2<< endl;

return 0;

}

**Output**

The value of x1 is :-5

The value of x2 is :-9

**TASK 6.3**

Evaluate the following quadratic equation and find the roots (x1,x2).

0b110x 2 – 0b110000x – 0b110110 = 0 a=6, b=48, c=54

C++ code store the result of an arithmetic expression that contains variables on both sides of the operand (simple and complex) in a variable

In mathematics, there are expressions which needs to be solved and will be having variables in both sides. It is important to know how you can solve the problem using C++. For example,

In some cases, we may want to swap the values of the variables without redoing the program. How can we do this using C++. We are going to demonstrate two way to swap values:

* Using a temporary variable
* Addition and difference.

**Using a temporary variable**

Swapping variables using a temporary variable is classic. As the name suggests, this approach requires an additional temporary variable. Let's swap the values of variables a and b using a temporary variable temp:

#include <iostream>

using namespace std;

int main()

{

cout << " Swap two numbers :\n";

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

int num1=50, num2=12, temp=0;

//Swapping values

temp=num1;

num1=num2;

num2=temp;

cout <<"Num1 is now"<< num1 <<endl ;

cout <<"Num2 is now"<< num2 <<endl ;

}

Output

Swap two numbers :

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

12

50

You can swap variables without the use of additional memory (like a temporary array or variable).

The following example swaps the variables num1 and num2 using the addition + and difference - arithmetic operators:

Here is an example

using namespace std;

int main()

{

cout << " Swap two numbers :\n";

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

int num1=50, num2=12;

//Swapping values

num1=num1+num2;

num2=num1-num2;

num1=num1-num2;

cout <<"Num1 is now "<< num1 <<endl ;

cout <<"Num2 is now "<< num2 <<endl ;

}

The initial values are num1=50, num2=12. Let us examine the three statements:

1. num1 = num1 + num2 assigns to num1 the value 50 + 12.
2. num2 = num1 – num2 assigns to num2 the value 50+12-12= 50 (num2 is now 50).
3. num1 = num1 – num2 assigns to num1 the value 50 + 12 - 50 = 12 (a is now 12).

I trust this is clear. Swapping numbers using addition and subtraction is the best when you do not want to waste space.

**TASK 6.4**

Write a program in C++ to swap two numbers. Your output should look as follows:  
Swap two numbers :

Input 1st number : 25  
Input 2nd number : 39

After swapping the 1st number is : 39  
After swapping the 2nd number is : 25

# FORMATIVE ASSESSMENT 6.1 GROUP/INDIVIDUAL TASK

6.1.1 Define the term data types. (2)

6.1.2 List 3 categories of C++ and give examples of each. (9)

6.1.3 Define the term literal as applied in C++. (2)

6.1.4 Complete the following table by representing the different decimal numbers into specified literals.

|  |  |  |
| --- | --- | --- |
| **Decimal Number** | **Type of integer literals** | **Example** |
| 45 | Binary-literal (base 2) |  |
| **87** | Binary-literal (base 2) |  |
| **45** | **Octal-literal (base 8)** |  |
| **124** | **Octal literal (base 8)** |  |
| **197** | Hexadecimal literal(base 16) |  |
| 350 | Hexadecimal literal(base 16) |  |

(6)

6.1.5 Which of the following is the correct way of declaring a constant? Give Explanation for your answer.

1. const keyword
2. #define preprocessor
3. both a and b
4. None of these (2)

6.1.6 Which type is best suited to represent the logical values? Give Explanation for your answer.

1. Integer
2. Boolean
3. Character
4. Float (2)

6.1.7 Consider the code below.

#include <iostream>

using namespace std;

int main()

{

int data = "98";

int val=50;

cout<<"After Adding value with val the result is "<<data<<"+"<<val<<" we get "<<data+val;

return 0;

}

What will be the output of the following program. Explain your answer? (2)

6.1.8 Explain the use of the keyword sizeof() when used in C++. (2)

6.1.9 Define the term variable as used in programming. (2)

**[Total=27 Marks]**

## 6.2. Arithmetic Operations

**VOCABULARY**

Operators perform specific mathematical or logical computations on values.

At this point, the concept of operators is not new; it was covered in Topic 3 while working with Arduino. So, we are going to repeat them to reinforce the concept and solidify your understanding. Operators, just like variables, form the foundation of any programming language. For example, + is an operator used for addition, while - is an operator used for subtraction. We can only evaluate expressions most of the time by including an operator or operators. Expressions can have more than one operator, depending on the complexity of the problem.

**Operators in C++ can be classified into 6 types:**

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Bitwise Operators
5. Assignment Operators
6. Ternary or Conditional Operators

**Arithmetic Operators**

These operators are used to perform arithmetic or mathematical operations on the operands for example multiplication, addition, and subtraction and others.

**Arithmetic Operators can be classified into 2 Types:**

1. **Unary operators**
2. **binary operators**

**Unary operators-**These operators operate or work with a single operand. For example: Increment(++) and Decrement(–) Operators.

Example 6.17 demonstrates use of unary operators.

**EXAMPLE 6.17**

#include <iostream>

using namespace std;

int num1 = 20;

int num2 = 35;

int main()

{

cout << "num1++ is " << num1++ << endl;

cout << "++num1 is " << ++num1 << endl;

cout << "num2-- is " << num2-- << endl;

cout << "--num2 is " << --num2 << endl;

return 0;

}

**Output**

num1++ is 20

++num1 is 22

num2-- is 35

--num2 is 33

**Explanation of the increment ++ and decrement operator**

The first output is 20 because the value is printed first then incremented leaving the new value of num1 as 21. In the second print line, the variable is incremented first before printing the value from 21 to 22. So, the result is also dependent on where the incrementor is put.

A similar condition applies for the decrement operator. First, the variable is called before the decrement operator and 35 is printed but the new value of num2 is now 34. In the last cout lines, the decrement or is called before the variable causing the new value to change from 34 to 33.

**Binary Operator-** These are operators that operate on two operands. For example, add two numbers. Binary arithmetic operators include addition, subtraction, division, and modulo operations. The code below illustrates how each is used. Pay attention to the code comments.

**EXAMPLE 6.18**

#include <iostream>

using namespace std;

int a = 25;

int b= 5;

int answer = 0;

int main()

{

//addition-Adds two operands and store answer in the third

answer =a +b;

cout << a <<" + "<<b<<" = "<<answer<< endl;

//subtraction-Subtracts second operand from the first

answer =a -b;

cout << a <<" - "<<b<<" = "<<answer<< endl;

//multiplication-Multiplies two operands

answer =a \* b;

cout << a <<" x "<<b<<" = "<<answer<< endl;

//division-Divides first operand by the second operand

answer =a / b;

cout << a <<" / "<<b<<" = "<<answer<< endl;

//division-Returns the remainder an integer division

answer =a % b;

cout << a <<" % "<<b<<" = "<<answer<< endl;

return 0;

}

**Output**

**25 + 5 = 30**

**25 - 5 = 20**

**25 x 5 = 125**

**25 / 5 = 5**

**25 % 5 = 0**

**Note**

The "/" operator is the division operator. If an integer is divided by another integer, we will get the quotient. However, if either the divisor or the dividend is a floating-point number, we will get the result in decimals, as illustrated in the example 6.19.

**EXAMPLE 6.19**

#include <iostream>

using namespace std;

int main()

{

cout << 9/2 << endl;

cout << 9/2.0 << endl;

cout << 9.0/2 << endl;

return 0;

}

**Output**

**4**

**4.5**

**4.5**

**Relational Operators-** **Also** referred to as comparison operators. Relational operators are used to compare the values of two operands. The result of the relational operator is boolean, i.e., true or false.  We are going to explain the different operators using the integer values x=9, y=2. See table 6. 4.

Table 6. 4: Relational Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Output** |
| = = Is equal to | Checks if both operands are equal | cout <<(x==y) << endl;  return 0 (false) |
| > greater than | Checks if first operand is greater than the second operand | cout <<(x>y) << endl; returns 1 because 9 is greater than 2 |
| >= greater than or equal to | Checks if first operand is greater than or equal to the second operand | cout <<(x>=y) << endl;  returns 1 |
| < less than | Tests whether the first operand is less than the second operand | cout <<(x<y) << endl;  returns 0 false |
| <= less than or equal to | Tests if the first operand is smaller or equal to the second operand | cout <<(x<=y) << endl; returns 0 (false) |
| != not equal to | Tests for equality between two operands | cout <<(x!=y) << endl;  returns 1 (true) |

**Logical Operators-** These are used to integrate two or more conditions or to supplement the original condition's evaluation. If the expression is true, it yields 1, otherwise it returns 0.

Table 6. 5: Logical operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Output** |
| **&&** | Logical AND. True only if all the operands are true. | cout <<(5&&6) << endl;  returns true |
| **||** | Logical OR. True if either operand is true or non-zero | cout <<(5||6) << endl;  returns 1 (true) |
| ! | Logical NOT. Returns true if the operand is false or zero | cout <<(!5) << endl;  returns 0 |

**Assignment Operators-** Variables can be assigned values using these operators. In our explanation on assignment operator given in table 6.6, we use x=9, y=2.

Table 6. 6: Assignment Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Output** |
| = Assignment Operator | The value on the right is assigned to the variable on the left | x=9 |
| += | It first adds the left variable's value to the right variable's value, then assigns the result to the left variable | cout <<(x+=y) << endl;  returns 11. |
| -= | The right value is subtracted from the left variable's current value, and then the result is assigned to the left variable | cout <<(x-=y) << endl;  returns 7 as value for x. |
| \*= | Multiplies the left variable's value by the right variable's value, then assigns the result to the left variable | cout <<(x\*=y) << endl;  returns 18 |
| /= | Divide the variable on the left's current value by the variable on the right, and then assign the resulting value to the variable on the left. | cout <<(x/=y) << endl;  returns 4 |
|  |  |  |

**Bitwise Operators**-  Bitwise operators are used to perform operations on individual bits. Table 6.7 lists and describes the different bitwise operators.

Table 6. 7:Bitwise Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Output** |
| & Binary AND | Copies a bit to the evaluated result if it exists in both operands | cout <<(x&y) << endl;  returns 0. 9=1001, 2=0010 The only value copied is 0 |
| | Binary OR | Copies a bit to the evaluated result if it exists in any of the operand | cout <<(x|y) << endl; returns 11 (1011) |
| << Left shift | Shifts the value to left by the number of bits specified by the right operand. | cout <<(x<<1) << endl;  x was 15 (01111). Returns 30 |
| >> Shift | Shifts the value to right by the number of bits specified by the right operand. | cout <<(x<<1) << endl;  x was 15 (01111). Returns 7 (0111) |

**Ternary or Conditional Operators**

In certain scenarios, C++'s ternary operator can replace if...else. A ternary operator evaluates the test condition and executes a block of code based on its result.

**Syntax**

**condition? Expression1: expression2**

Expression1 is executed if condition is true. And expression 2 is executed if condition is false. Here is an example to print if a student has passed an exam if the mark entered is 50 otherwise it’s a fail.

**EXAMPLE 6.20**

#include <iostream>

using namespace std;

int main() {

int test1\_mark;

// take input from users

cout << "Enter your mark in % for test1: ";

cin >> test1\_mark;

//condition

string result = (test1\_mark >= 50) ? "passed" : "failed";

cout << "You " << result << " the test 1";

return 0;

}

**Output**

Enter your mark in % for test1: 49

You failed the test 1

**Task 6.5**

Use a ternary operator to write a program to check if a person qualifies to vote or not. If a person’s age is 18 or above, the program must display a message:

"Based on the analysis of your age, you can vote";

If the person is below 18 the program should display a message:

"Based on the analysis of your age, you cannot vote";

### 6.2.1 List the different arithmetic operators

We have discussed several arithmetic operators, and we have been using them since we started this module. Arithmetic operators are also used in our day to day lives as we make decisions. For the purposes of reinforcement, here is a list of arithmetic operators:

++ increment

* + - * - decrement

+ addition

* + - * subtraction

/ division

\* multiplication

% modulo

### 6.2.2 Define the term binary operator

**VOCABULARY**

Binary operator is an operator which contains two operands to perform a mathematical or operation.

Table 6.8 summarises the categories of operators in C++.

Table 6. 8: Categories of operators in C++

|  |  |  |
| --- | --- | --- |
| **Categories** | **Type** | **Example of** |
| Unary Operator | Arithmetic Operators | --, ++ |
| Binary Operator | Arithmetic Operator | +, -, /, \*, % |
| Relational Operator | <, <=, >, =>, = =, != |
| Logical oper | &&, ||, ! |
| Bitwise Operator | &, |, <<, >>, |
| Assignment Operator | =, +=, -=, /=, \*=, %= |
| Ternary Operator | Conditional Operator | ?: |

### 6.2.3 Operand

Operands are objects operated on by an operator. Operators and operands are combined to form mathematical expressions that produce values based on the positioning of the operators and operands. An operand is a number, a variable that represents a number or a function that returns a number. For example, the addition operator (+) is used to add two numbers together, and the [multiplication operator](https://www.ibm.com/docs/en/xl-c-aix/13.1.2?topic=expressions-multiplication-operator) (x or \*) is used to multiply two numbers together, as in the following equation:

6x2 + 3x + (4 x 5)

The equation contains four operands – 6x2, 3x , 4 and 5 -- along with one addition operator and one multiplication operator.

### 6.2.4 Binary arithmetic expression

Binary refers to an operation which is performed on two values, such as the addition or subtraction of two numbers. The order in which the operations in an expression are performed can be controlled using parentheses. A binary arithmetic expression consists of :

* Operand
* Operator (Arithmetic and Assignment Operator)

**Example**

int expr1=3

int expr2=5

int product=0

product=expr1 \* expr2

Parenthesis can be used to regulate the order in which the actions in an expression are done. Expressions enclosed in parentheses will be evaluated before expressions not enclosed in parentheses. Also, if parentheses are nested within parenthesis, the expression in the innermost set of parentheses is evaluated first. The usage of parenthesis in evaluating numerical expressions is seen below.

4 + 2 \* 5 - 12 / 3 evaluates to 10

( 4 + 2 ) \* 5 - 12 / 3 evaluates to 26

**Try this out**

1. 4+ 2 \* ( 5 - 12 / 3 )
2. 4 + 2 \* ( ( 5 - 12 ) / 3 )

### 6.2.5 C++ code that uses cout to display the value of different arithmetic expressions (combined with string literal for context)

**cout**

For input and output operations on sequential media such as files, keyboards, or screens, C++ provides streams. A stream is a container into which characters can be put or removed. C++'s output stream is cout, while its input stream is cin. The iostream header file defines it. cout is an abbreviation of character output. To display the output on a console, use the cout command in conjunction with the stream insertion operator (<<).

**cout Syntax**

cout<<variable\_name

**Example**

//accepting input

int age;

//displaying output

cout << "Please enter your age: !" << endl;

//entering age

cin>>age;

It is possible to have a string and a variable in one line using cout keyword as shown in Example 6.21.

**EXAMPLE 6.21**

#include <iostream>

using namespace std;

int main()

{

string first\_name= " ";

int age =0;

cout << "Please enter your name :" << endl;

//entering name

cin>>first\_name;

cout << "Please enter your age :" << endl;

//entering age

cin>>age;

//displaying name

cout<<"My name is " << first\_name<<endl;

//displaying age

cout<< "I am " << age << " years old " <<endl;

return 0;

}

**Output**

Please enter your name :

Godwin

Please enter your age :

56

My name is Tshepo

I am 18 years old

Notice how we used cout to display a string and a variable in one line.

cout<<"My name is " << first\_name<<endl;

When literals and variables are mixed together, chaining insertions is especially useful. Line breaks are not used in the output.

We also used the cin stream in getting input. cin stands for character input. The **cin** is a predefined object of the **input stream** class. cin relates to the standard input device, which is usually a keyboard. The cin is used in conjunction with the stream extraction operator (>>) to read the input from a console.

endl is a C++ manipulator or command. As a result, when it is encountered, the operating system will flush the output buffer and insert a new line. In C++, every endl essentially does two things:

* Insert a new line
* Flush the output buffer

**TASK 6.6**

Write a program in C++ to print the sum of two numbers using variables. The program must ask user to input the values of the two numbers and display the output as:

The sum of 67 and 23 is : 90

### 6.2.6 Data type produced based on the data type of the operands involved

Rules (operator) that are applied to the operands in an operation determine the types of data in a result. In addition, the  output of an expression depends on the data type of the inputs and in some instances, the data type of the output. For example, if we have a program to add two integers and the output data type is declared as int, then the output will be an integer. However, if the data type of the answer is declared as float, the output will also be float. Data types for arithmetic expressions will always be numbers.

As for relational expressions, the result is a Boolean data type. For example, the following code snippet will produce a boolean answer.

cout << (5>7) << endl;

The answer is 0 which represents false.

### 6.2.7 Rules involved when creating complex arithmetic expressions containing multiple operands.

1. **Addition:**You can write complex numbers as follows for addition:
   1. (a + ib) + (c + id) = (a + c) + i(b + d).
2. **Subtraction:**You can write complex numbers as follows for subtraction:
   1. (a + ib) – (c + id) = (a – c) + i(b – d).
3. **Multiplication:**You can write complex numbers as follows for multiplication
   1. (a + ib). (c + id) = (ac – bd) + i(ad + bc).
4. **Division:**You can write complex numbers as follows for division
   1. (a + ib) / (c + id) = (ac + bd)/ (c2+ d2) + i(bc – ad) / (c2 + d2)
5. **Additive identity:**For the additive identity, it can be represented as:
   1. (a + bi) + (0 + 0i) = a + bi

Example: Simplify  the value of: 20i + 5i(6 – i)

Given, 20i + 5i(6 – i)

= 20i + 30i – 5i2

= 50i – 5 × (-1)

= 50i +5

Example: Evaluate (2 + 3i)(4 – 6i)2 and write the end result in the form of (a + bi)

In the expression above, we need to simply the second bracket first and this evaluates to:

(4 – 6i) (4 – 6i)= (-48i -20)

Add the first expression

(2+3i) (-48i-20)

= -40 – 96i – 60i + 144

= – 156i +104 rewrite in standard complex form

=104-156i

### 6.2.8 Operator Precedence

With the help of examples, we will learn about the precedence of operators in C++. Operator precedence determines the grouping of terms in an expression. Rather, operators with higher **precedence** have their operations evaluated first.

Let us consider an example:

Int num1=5-17\*8

The expression will be evaluated as follows:

5-(17\*8) // brackets will be cleared first to give 136 and then 5 -136 = -131.

 An expression that has multiple operators is called a ***compound expression***. To evaluate compound expression, we must understand both what the operators do, and the correct order to apply them. The order in which operators are evaluated in a compound expression is determined by an operator’s precedence. If two operators with the same precedence level are adjacent to each other in an expression, the operator’s **associativity** tells the compiler whether to evaluate the operators from left to right or from right to left.  Table 6.9 shows the different levels of precedence and associativity.

Table 6. 9: Operator precedence

|  |  |  |  |
| --- | --- | --- | --- |
| **Precedence** | **Operator** | **Description** | **Associativity** |
| 1 | : : | scope | Left-to-right |
| 2 | () [ ] . -> ++ -- dynamic\_cast static\_cast reinterpret\_cast const\_cast typeid | postfix | Left-to-right |
| 3 | ++ -- ~ ! | unary (prefix) | Right-to-left |
| \* & | indirection and reference (pointers) |
| + - | unary sign operator |
| 4 | \* / % | multiplicative | Left-to-right |
| 5 | + - | additive | Left-to-right |
| 6 | << >> | shift | Left-to-right |
| 7 | < > <= >= | relational | Left-to-right |
| 8 | == != | Equality | Left-to-right |
| 9 | & | Bitwise AND | Left-to-right |
| 10 | ^ | bitwise XOR | Left-to-right |
| 11 | | |  | bitwise OR |
| 12 | && | Logical AND | Left to right |
| 13 | || | Logical OR | Left to right |
| 14 | ? | Conditional/ternary | Right to left |
| 15 | = \*= /= %= += -= >>= <<= &= ^= |= | assignment | Right-to-left |
| 16 | , | comma | Left-to-right |

For simpler expressions, a common technique for remembering the order of operations is the abbreviation "PEMDAS". This phrase stands for, and helps one remember the order of:

1. Parentheses,
2. Exponents,
3. Multiplication and Division,( (going from left to right) and
4. Addition and Subtraction (going from left to right)

**Example**

15 ÷ 3 × 4 ..is not 15 ÷ (3 × 4) = 15 ÷ 12, but is actually:

* 1. /3) × 4 = 5 × 4 ...because, going from left to right, you get to the division sign first.

**TASK 6.7**

Simplify 4 + (2 + 1)2.

**Typecasting**

Casting is a conversion process wherein data can be changed from one type to another.  Type casting is also known as Type Conversion.

C++ has two types of conversions:

* Implicit conversion
* Explicit conversion

**Implicit conversion**: Conversions are performed automatically by the compiler without the programmer's intervention.

**Explicit conversion:** Conversions are performed only when explicitly specified by the programmer.

Example 6.22 program demonstrates implicit and explicit conversion in C++.

**EXAMPLE 6.22**

#include <iostream>

using namespace std;

int main(){

cout<<"implicit conversion"<<endl;

cout<<87/4<<endl;

cout<<"explicit conversion"<<endl;

cout<<(float)87/4<<endl;

}

Output

implicit conversion

21

explicit conversion

21.75

In the example above, we divided the number 87 by 4 and the answer was automatically given as an integer and is 21.  For explicit conversion, we prefixed the calculation with the data type that we wanted our output to be, and in our case, we used float. The answer is given as 21.75.

### 6.2.9 Using cout to display the value of different complex arithmetic expressions.

**Complex numbers in C++**

One of the most important notions in modern science is the concept of complex numbers. The complex library implements the complex class, which contains complex numbers , as well as a set of functions and overloads for working with them..

**VOCABULARY**

Numbers that are complex are those with the form a + ib, in which a and b are real numbers. In complex numbers, a is known as the real part and ib is known as the imaginary part.

* real() – It returns the real part of the complex number.
* imag() – It returns the imaginary part of the complex number.

To work with complex numbers, you will need to add the #include complex header. Using this header makes sure you do not need to use the std::complex statement.

The code shown in example 6.23 shows how to declare complex numbers.

**EXAMPLE 6.23**

#include <iostream>

#include <complex>

using namespace std;

int main(){

//declaring complex numbers

complex<double> complex\_one; // value 0 + 0i

complex<double> complex\_two(3.14); // value 3.14 + 0i

complex<double> complex\_three(1.5, 3.14); // value 1.5 + 3.14i

//assigning values of a complex number to another variable using a variable

complex<double> complex\_four(complex\_two); // value is also 3.14 + 0i

cout<<complex\_one<<endl;

cout<<complex\_two<<endl;

cout<<complex\_three<<endl;

cout<<complex\_four<<endl;

return 0;}

**Output**

(0,0)

(3.14,0)

(1.5,3.14)

(3.14,0)

You will notice that the output has two values, i.e., the real and the imaginary value. For example, the output for variable complex\_two is (3.14, 0). 3.14 is the real number, and 0 is the imaginary number. A complex number can be assigned the value of another complex number. e.g.

complex<double> com\_four(complex\_two);

Since the one-argument constructor is also used for a conversion operator, a complex number can also be assigned the value of a real number. The real field is changed to the right-hand side, while the imaginary field is set to zero:

To access the real and imaginary values from a complex number, we can use the member functions real() and imag().

**Example 6.24**

#include <iostream>

#include <complex>

using namespace std;

int main()

{

//declaring complex numbers

complex<double> complex\_one; // value 0 + 0i

complex<double> complex\_two(3.14); // value 3.14 + 0i

complex<double> complex\_three(1.5, 3.14); // value 1.5 + 3.14i

//assigning values of a complex number to another variable using a variable

complex<double> complex\_four(complex\_two); // value is also 3.14 + 0i

cout<<complex\_one<<endl;

cout <<"The real number is "<< complex\_one.real() << " and the imaginary number is " << complex\_one.imag()<<endl;

cout<<complex\_two<<endl;

cout <<"The real number is "<< complex\_two.real() << " and the imaginary number is " << complex\_two.imag()<<endl;

cout<<complex\_three<<endl;

cout <<"The real number is "<< complex\_three.real() << " and the imaginary number is " << complex\_three.imag()<<endl;

cout<<complex\_four<<endl;

cout <<"The real number is "<< complex\_four.real() << " and the imaginary number is " << complex\_four.imag()<<endl;

return 0;

}

**Output**

(0,0)

The real number is **0** and the imaginary number is **0**

(3.14,0)

The real number is **3.14** and the imaginary number is **0**

(1.5,3.14)

The real number is **1.5** and the imaginary number is **3.14**

(3.14,0)

The real number is **3.14** and the imaginary number is **0**

The highlighted numbers are the real and imaginary, respectively.

**Arithmetic Operations on complex numbers**

The arithmetic operators +, -, \*, and / can be used to perform addition, subtraction, multiplication, and division of complex numbers. Each of the four works with a complex number and a real number or with a complex number and a real number. Assignment operators are also defined for all four. Example 6.26 shows code to calculate complex numbers in C++.

**EXAMPLE 6.25**

#include <iostream>

#include <complex>

using namespace std;

int main()

{

//declaring the complex number variables and assigning values

complex<double> complex\_two(3.14);

complex<double> complex\_three(1.5, 3.14);

//adding complex numbers

complex<double> complex\_four =complex\_two + complex\_three;

//displaying output

cout <<"The sum of " << complex\_two<< " + " <<complex\_three<< " = "<< complex\_four<<endl;

return 0;

}

**Output**

The sum of (3.14,0) + (1.5,3.14) = (4.64,3.14)

The operators = = () and = = () can be used to compare two complex numbers for equality or inequality. Two values are equal if their corresponding fields are equal. Complex numbers do not have a natural ordering, and thus cannot be compared using any other relational operator. The result of the comparison is of the boolean type.

**Task 6.8**

Write a program to compare the following complex numbers.

complex\_two(3.14, 1.5);

complex\_three(1.5, 3.14);

# FORMATIVE ASSESSMENT 6.2 GROUP TASK

6.2.1 Demonstrate how to declare multiple variables and assign values:

x, y, z = 53 (2)

6.2.2 Define the term operators. (2)

6.2.3 List the THREE categories of operators. (3)

6.2.4 Using examples, explain what is meant by the term binary expression. (4)

6.2.5 Use the expression in 6.2.4 with the following values num2=35, num3=17, answer=0; use the cout keyword to give the following result

35 + 17 = 52

Use multiple declaration for the variables. (6)

6.2.6 Write a program to input the value of the radius of a circle from keyboard and then calculate its perimeter and area. (9)

6.2.7 Write a program to enter the values of two variables 'a' and 'b' from keyboard and then check if both the conditions 'a < 50' **and** 'a < b' are true. Use values 45 and 55 for a and b respectively. What is the answer? (10)

6.2.8 Define the term typecasting. (2)

6.2.9 What is the output of the program below?

#include <iostream>

using namespace std;

int main()

{

int n;

cout <<"Output 1 is "<< (n = 4) << endl;

cout << "Output 2 is" <<(n == 4) << endl;

cout <<"Output 3 is "<<(n > 3) << endl;

cout <<"Output 4 is "<< (n < 4) << endl;

cout <<"Output 5 is "<< (n = 0) << endl;

cout <<"Output 6 is "<< (n == 0) << endl;

cout <<"Output 7 is "<< (n > 0) << endl;

cout <<"Output 8 is "<< (n && 4) << endl;

cout << "Output 9 is "<<(n || 4) << endl;

cout << "Output 10 is "<<(!n) << endl;

return 0;

}

(10x1=10)

**[Total =51 Marks]**

## 6.3. Multiple declarations

### 6.3.1 General form for multiple variable declarations

Each variable stores one value at a time. When declaring variables, programmers usually declare each variable on a single line for readability and if they are of the same data type. However, nothing stops one from declaring multiple variables in the same line if they are of the same data type. Example 6.26 is an illustration of multiple variable declaration.

**EXAMPLE 6.26**

#include <iostream>

using namespace std;

int main()

{

//declaring multiple variables

int num1=5, num2=56, sum=0;

//displaying values of each variable

cout<<num1<<endl;

cout<<num2<<endl;

//displaying sum

sum=num1+num2/8;

cout<<num1<<endl;

return 0;

}

In the example above, num1, num2 and sum have been declared and initialised in one line.

**NOTE**

A common mistake made is declaring the variables and then assign a value to the third variable as follows:

int num1, num2, num3=45;

If the declaration is done that way, only the third variable will hold the value 45. So, make sure you avoid that mistake if you want to do multiple variable declarations.

It is also possible to declare multiple variables and assign them one value:

**EXAMPLE 6.27**

#include <iostream>

using namespace std;

int main()

{

//declaring multiple variables

int age\_T, age\_S, age\_R;

age\_T= age\_S= age\_R=19;

//displaying values of each variable

cout<<"Tinashe is " <<age\_T<<" years old"<<endl;

cout<<"Samantha is "<<age\_S<<" years old"<<endl;

cout<<"Rutendo is "<<age\_R<<" years old"<<endl;

return 0;

}

**Output**

Tinashe is 19 years old

Samantha is 19 years old

Rutendo is 19 years old

**Task 6.9**

The following program produces errors when compiled.

#include <iostream>

using namespace std;

int main()

{

//declaring multiple variables

int cpt\_temp=jbg\_temp=26;

cout<<"Cape Town temperature is "<<cpt\_temp<<"degrees celcius"

cout<<"Joburg temperature is "<<jbg\_temp<<"degrees celcius"

cout<<"Durban temperature is "<<durb\_temp<<"degrees celcius"

return 0;

}

Correct the program to produce the following output:

Cape Town temperature is 26 degrees celcius

Joburg temperature is 26 degrees celcius

Durban temperature is 26 degrees celcius

### 6.3.2 Importance of having same data type for multiple declarations

Each variable has its own data type. So, when declaring multiple variables in one line-they should all be of the same data type to share the same memory space. While this is a common practice, declaring multiple variables in a single declaration can cause confusion regarding the types of the variables and their initial values.

### 6.3.3 Write C++ code that declares multiple variables in a single line

The program in example 6.27 illustrates multiple declaration of variables of different data types in a single line.

**EXAMPLE 6.27**

#include <iostream>

using namespace std;

int main()

{

//multiple floating point variables

float temp\_1=25.5, temp\_2=28.6;

cout<<"Printing multiple declared float varibles"<<endl;

cout<<temp\_1<<endl;

cout<<temp\_2<<endl;

//multiple ,character variables

char letter='M', age='O';

cout<<"Printing multiple declared character variables"<<endl;

cout<<letter<<endl;

cout<<age<<endl;

// multiple boolean variables

bool is\_late=true, is\_done=false, is\_programmable=true;

cout<<"Printing multiple declared boolean variables"<<endl;

cout<<is\_late<<endl;

cout<<is\_done<<endl;

cout<<is\_programmable<<endl;

return 0;

}

**Output**

**Printing multiple declared float variables**

**25.5**

**28.6**

**Printing multiple declared character variables**

**M**

**O**

**Printing multiple declared boolean variables**

**1**

**0**

**1**

# FORMATIVE ASSESSMENT 6.3 INDIVIDUAL TASK

6.3.1 List FIVE rules to be followed when naming variables in C++. (5)

6.3.2 Given integer variables a, b, c, d, and e, where a = 1, b = 2, c = 3, d = 4,

evaluate the following expressions: assume the variable for output is declared as float.

1. sum\_1=a + b - c + d
2. sum\_2=a \* b/c
3. sum\_3=1 + a \* b % c
4. sum\_4=a +d%b -c (8)

6.3.3 Explain why it is important to declare same data type for multiple declarations. (2)

6.3.4 What will be the output of the following code: (3)

#include <iostream>

using namespace std;

int main()

{

int amount, count ;

count = 3 ;

amount = 2 \* count++ -count ;

cout << "amount is " <<amount<<endl;

return 0;

}

**[Total =18 marks]**

## 6.4. Escape characters and New Lines

### 6.4.1 Backslash character will have in C++.

The backslash character (\) is a line-continuation character when it's placed at the end of a line. If you want a backslash character to appear as a character literal, you must type two backslashes in a row (\\).

**EXAMPLE 6.28**

#include <iostream>

using namespace std;

int main()

{

char backlash='\\';

cout<<"Here is illustration of backlash" << backlash<< "in C++";

return 0;

}

Output

This is a backlash: \ending

Backslashes are handy when dealing with strings. Strings must be enclosed in double quotes, and if there is a double quoted string inside a string declaration, the compilation will fail. For example:

string students="He said "WSU students" are hardworking";

|error: unable to find string literal operator 'operator""WSU' with 'const char [9]', 'long long unsigned int' arguments|

To correct the error, we can use a backlash. Our code will then look as follows:

#include <iostream>

using namespace std;

int main()

{

string students="He said \"WSU students\" are hardworking";

cout<<students<<endl;

return 0;

}

Output

He said "WSU students" are hard working

The sequence \'  inserts a single quote in a string.

string students="That\'s fine"; will give output as That's fine

If we use \’’ then double quotes will be inserted into the string. So, as you can see from the given examples, the backslash (\) escape character turns special characters into string characters.

### 6.4.2 Define the term escape character

**Vocabulary**

Escape sequences in [C++](https://geekonpeak.com/tag/c/) are character combinations that comprise a backslash (\) followed by some character or symbol. They give results such as getting to the following line or a TAB space.

### 6.4.3 Define the term escape sequence

**Vocabulary**

Escape sequences in [C++](https://geekonpeak.com/tag/c/) are character combinations that comprise a backslash (\) followed by some character or symbol. They give results such as getting to the following line or a TAB space.

Escape sequences control the printing behaviour of output stream objects (such as 'cout'). These characters are not displayed in the output. An escape sequence is prefixed with a backslash () and a coded character is used to control the printing behaviour. The backslash () is called an escape character. So, the escape sequence looks like two characters.

Table 6.10 gives a list of some common escape sequence characters

Table 6. 10: Escape sequence

|  |  |
| --- | --- |
| **Character** | **Description** |
| \b | used to delete the previous character. |
| \n | is used to jump to the next line. |
| \t | Horizontal tab |
| \v | Vertical tab |
| \? | Question mark |
| \a | Beep sound |

Here is an illustration of implementation of escape sequence characters

**EXAMPLE 6.29**

#include <iostream>

using namespace std;

int main(){

//new line

cout << "Hi all NCV3\n programmers";

//tab

cout << "\nHi all NCV3\t programmers";

//question mark

cout << "\nHi all NCV3\? programmers";

return 0;

}

Sample output

Hi all NCV3

programmers

Hi all NCV3 programmers

Hi all NCV3? programmers

### 6.4.4 C++ code that uses cout to display/use variable (combined with string literal and variable values for context):

We have discussed what escape characters and escape sequence are. We are going to implement a program to incorporate the following for reinforcement purposes. (See example 6.30)

* Backslash
* Question mark
* Single and Double quotation
* Horizontal tab
* Alert

**EXAMPLE 6.30**

#include <iostream>

using namespace std;

int main()

{

char backlash='\\';

char question='\?';

char single='\'';

char tab= '\t';

string firstname= " ";

char alert='\a';

//backslash

cout << "Select your gender "<<backlash<<"M"<<backlash<<"F"<<backlash<<"Other"<<endl;

//question mark

cout << "What do you plan to do after NCV"<<question<<endl;

//single quotation and question mark

cout << "How"<<single<<"re you"<<question<<endl;

char double\_quotes='"\\"';

//double qoutes

cout<<"He said "<<double\_quotes<<" NCV students "<<double\_quotes<<" are hardworking"<<endl;

//Question mark

cout<<"What is your name "<<question<<endl;

cin>>firstname;

//new line and horizontal tab

cout<<"\nMy name is "<<tab<<firstname<<endl;

//alert sound

cout<<"\nNice learning the escape sequence characters "<<alert<<endl;

return 0;

}

Output

Select your gender \M\F\Other

What do you plan to do after NCV?

How're you?

He said " NCV students " are hardworking

What is your name ?

Minentle

My name is Minentle

Nice learning the escape sequence characters

**Task 6.10** Write a program in C++ to compute quotient and remainder. Your output should look as shown below:

Calculating quotient and remainder :

-------------------------------------

Input the dividend : 58

Input the divisor : 7

The quotient of the division is : 8

The remainder of the division is : 2

# FORMATIVE ASSESSMENT 6.4 INDIVIDUAL TASK

6.4.1 The following code snippet produces errors when compiled.

cout << " The town of "Mthatha" is in the Eastern Cape "<<"\n" ;

Correct the code to give the following output

The town of "Mthatha" is in the Eastern Cape

(2)

6.4.2 Define the term escape sequence as used in C++. (2)

6.4.3 Consider the code below

#include <iostream>

using namespace std;

int main()

{

bool is\_a\_programmer=true;

cout << is\_a\_programmer << '\n';

cout << is\_a\_programmer << '\n';

}

Rewrite the code to produce the following output:

true

1

(4)

6.4.3

#include <iostream>

#include <iomanip>

using namespace std;

int main()

{

bool is\_a\_programmer=true;

cout << boolalpha << is\_a\_programmer << '\n';

cout << noboolalpha << is\_a\_programmer << '\n';

}

(4)

6.4.4 Complete the table below by describing what each escape sequence does.

|  |  |
| --- | --- |
| **Character** | **Description** |
| \b |  |
| \n |  |
| \t |  |
| \v |  |
| \? |  |
| \a |  |
| \’ |  |
| \” |  |

(8) **[Total =20 marks]**

## 6.5. Formatted Output

### 6.5.1 Formatting output

Clean output is always needed by programmers to enhance user interfaces and make debugging messages easier to understand. Using C++ functions like cout, you may send data to streams that is properly formatted. You must either precede your calls with "std::" or include "using namespace std;" before using any of the functions in the iomanip header because they are all located inside the std namespace. One of the concepts is the use of space to make sure that everything is aligned correctly, that there are no excessively long or short text columns, and that everything appears good. In general, there are three ways of formatting ostream-based streams:

1. Using the format flags provided by ios\_base.
2. Stream modifying functions defined in the header <iomanip> and <ios>.
3. By invoking a specific overload of the *<<*-operator.

Each method has its pros and cons, and the choice depends on the situation.

### 6.5.2 Use of field with manipulators

**VOCABULARY**

Manipulators are functions specifically designed to be used in conjunction with the insertion (<<) and extraction (>>) operators on stream objects. Manipulators are used to change formatting parameters on streams and to insert or extract certain special characters.

Here is a example of a stream manipulator.

bool is\_registered=true;

cout<<boolalpha<<is\_registered<<endl;

cout<<noboolalpha<<is\_registered<<endl;

Output

true

1

So, you notice how handy these formatting techniques are. Remember, when we started working with data types and variables, all our boolean variables were outputting a 1 or 0 for true or false, respectively, irrespective of our declarations.

There are various types of manipulators:

**Manipulators with no arguments**

* endl: A new line is entered and the output stream is flushed after each new line.
* **ends**: It is also defined in ostream and it inserts a null character into the output stream.
* flush:  it forces all the output written on the screen or in the file.

**EXAMPLE 6.31**

#include <iostream>

#include <iomanip>

using namespace std;

int main()

{

// illustrating ends and endl

cout << "first";

cout << "second" << ends;

cout << "third" << endl;

cout << "fourth" << endl;

return 0;

}

Output

firstsecondthird

fourth

In example 6.31 you will notice that the first three outputs are grouped together. This is because ends command does not send text to the new line but just inserts a null character at the end. Only when endl, is encountered then the text is sent to the new line.

**Manipulators with Arguments:** These all are defined in the header file. If we want to use these manipulators, then we must include this header file in our program.

Examples of manipulators with arguments includes:

* setw (val)
* setfill (c)
* setprecision (val)
* setbase(val)

**Setting the field width with setw**

The std::setw() function allows you to set the minimum width of the next output via the insertion operator. setw() takes, one argument, the width of the next output (insertion), an integer. if the next output is too short, then spaces will be used for padding. There is no effect if the output is longer than the width--note that the output won't be truncated. Make sure to include the #include <iomanip> header to use setw().

**Syntax**

*cout<<setw(spaces)<<string/variable<<endl;*

The code in example 6.32 shows the usage of setw().

**EXAMPLE 6.32**

#include <iostream>

#include <iomanip>

using namespace std;

int main()

{

cout<<setw(10)<<"ten"<<setw(10)<<"four"<<setw(10)<<"four";

return 0;

}

**Output**

Graphical user interface, text, application

Description automatically generated

**setfill()-**It replaces setw(whitespaces )’s with a different character. It’s like setw() in that it manipulates output, but the only parameter required is a single character.

**EXAMPLE 6.33**

#include <iostream>

#include <iomanip>

using namespace std;

int main()

{

int num1,num2;

num1=19; num2=16;

//using setfill to format output

cout<< setfill('\*') << endl;

cout << setw(4) << num1 << setw(5) << num2<< endl;

return 0;

}

Text

Description automatically generated

As can be seen in the example, the first variable had the first output spaces filled with 2 \* because the character size is 4. As for num2, three \*s are printed first and then the remaining 2- spaces reserved for the value 16.

**setprecision():** The setprecision() utility is often used to alter the sequence of numbers displayed inside a floating-point integer output sequence. It works the same as the round-off functionality. See how the setprecision() function has been implemented in the example 6.34.

**EXAMPLE 6.34**

#include <iostream>

#include <iomanip>

using namespace std;

int main()

{

float number=123.456f;

cout << "Value of number with different setprecision parameters:" << endl;

cout << setprecision( 3) << number << endl;

cout << setprecision( 4) << number << endl;

cout << setprecision( 5) << number << endl;

cout << setprecision(7) << number << endl;

return 0;

}

Output

Text

Description automatically generated

As can be seen in the example above, setprecision() will determine the length of the output value. For example: the variable number was assigned 123.456f and when printed with setprecision(3), the output is 123. It ignores the other digits that come after. Also note that if the value after the comma had been greater or equal to 5, the output would have been 124.

**Adjustment format flags**- when implemented, these flags cause the output to be adjusted to a specified position e.g left, right, and internal

The ***field width*** determines the minimum number of characters to be written in some output representations. If the standard width of the representation is shorter than the field width, the representation is padded with fill characters at a point determined by the format flag adjustfield (one of left, right or internal).

Table 6. 11:Adjustment format flags

|  |  |
| --- | --- |
| **Flag value** | **effect** |
| left | the output is padded to the [field width](https://cplusplus.com/ios_base::width) appending [fill characters](https://cplusplus.com/basic_ios::fill) at the end |
| right | the output is padded to the [field width](https://cplusplus.com/ios_base::width) by inserting [fill characters](https://cplusplus.com/basic_ios::fill) at the beginning. |
| internal | the output is padded to the [field width](https://cplusplus.com/ios_base::width) by inserting [fill characters](https://cplusplus.com/basic_ios::fill) at a specified internal point |

The program shown in example 6.35 illustrates the implementation of left,right and internal adjustment format flags.

**EXAMPLE 6.35**

#include <iostream>

using namespace std;

int main () {

int temp = -23;

// modify adjustfield using manipulators

cout.width(7); cout << internal <<temp<< '\n';

cout.width(7); cout << left << temp<< '\n';

cout.width(7); cout << right <<temp<< '\n';

return 0;

}

Output



### 6.5.3 Write C++ code that uses cout and formatting strings to:

The program below (example 6.36)illustrates the use of field width, left, right and internal justify, setprecision() set to 4 and default leading fill character.

**EXAMPLE 6.36**

#include <iomanip>

using namespace std;

int main () {

float temp=27.4567f;

cout<<"The temperature for the day is " <<temp<<endl;

// modify adjustfield using manipulators

cout.width(7); cout <<setprecision(4)<<internal <<temp<< '\n';

cout.fill('=');

cout.width(15); cout << left << temp<< '\n';

cout.fill('\*');

cout.width(7); cout << right <<temp<< '\n';

return 0;

}

Text

Description automatically generated

**Task 6.11**

Write a program in C++ to show the different values of an integer using the setbase() function. In this program define an int variable called number and store the value 12349. Show the hex, oct equivalent of the number. Use the manipulators to produce output as shown in figure 6.1.

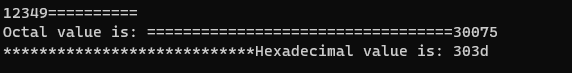


Figure 6. 1: Manipulator output

# FORMATIVE ASSESSMENT 6.5 INDIVIDUAL TASK

6.5.1 Define what manipulators are in C++. (2)

6.5.2 Explain the function of the following manipulators:

1. endl
2. ends
3. flush (6)

6.5.3 With the help of some code snippets, explain the use of the following manipulators:

1. setw (val)
2. setfill (c)
3. setprecision (val) (9)

**[Total =17 marks]**

# SUMMATIVE ACTIVITY 6.6 INDIVIDUAL TASK

6.6.1 Complete the table below.

|  |  |  |
| --- | --- | --- |
| **Item** | **Value** | **Data Type** |
| a | is\_a\_student=false; |  |
| b | "We are learning about data types"; |  |
| c | 15; |  |
| d | 0b11; |  |
| e | L'x'; |  |
| f | 233434.56343; |  |
| g | 277.65 |  |
| h | 45e-2 |  |

(8)

6.6.2 Write a program to accept two numbers through the keyboard and swap the numbers in the variables. Do not use a temporary variable. The output should look as follows:

Swap two numbers :

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Input 1st number : 55

Input 2nd number : 66

After swapping the 1st number is : 66

After swapping the 2nd number is : 55

(10)

6.6.3 Answer Tue or False to the following statements

1. An arithmetic expression that has a value of zero is false.
2. An arithmetic expression that has a value other than zero is true. (2)

6.6.4 Complete the table by rewriting the statements on the right-hand side of the table using assignment operators,

|  |  |  |
| --- | --- | --- |
|  | Simple expression | Expression using assignment operators |
| i | a = a+2 ; |  |
| ii | a = a-3 ; |  |
| iii | a = a\*2 ; |  |
| iv | a = a%2 ; |  |
| v | b = b+(c+2); |  |
| vi | d =d (e \* -5); |  |

(6)

6.6.4 Assignment operators

|  |  |  |
| --- | --- | --- |
|  | Simple expression | Expression using assignment operators |
| i | a = a+2 ; | a += 2 ; |
| ii | a = a-3 ; | a -= 3 ; |
| iii | a = a\*2 ; | a \*= 2 ; |
| iv | a = a%2 ; | a %= 2 ; |
| v | b = b+(c+2); | b += c + 2 ; |
| vi | d =d (e \* -5); | d \*= e - 5 ; |

6.6.5 What is the Difference between ++x and x++. (4)

6.6.6 With the help of some code snippets, explain the use of the following manipulators:

1. setbase(val) (implement setbase to 8, 10 and 16) (8)

6.6.7 Define the term complex numbers as used in C++. (2)

6.6.8 Write a program to declare two complex numbers and multiply them together.

Complex numbers(2.2+0i), (3.6+1.14i) (10)

**[Total=50 marks]**

# References

Bjarne, S., 2013. The C++ Programming Language.-4th

Grimes, R., 2017. *Beginning C++ Programming*. Packt Publishing Ltd.

Oualline, S., 2003. *Practical C++ programming*. " O'Reilly Media, Inc.".

<https://learn.microsoft.com/en-us/cpp/cpp/namespaces-cpp?view=msvc-170>

<https://www.programiz.com/cpp-programming>

<https://www.javatpoint.com/cpp-tutorial>

<https://www.geeksforgeeks.org/cpp-tutorial/>

# Topic 7:Math, Interactive Input, Constants and Errors

## C++ Keyboard Input

After you have completed this module, you should be able to :

* Explain the process that happens when the computer encounters the scanf or cin statement
* Use the C++ cin to read user input (numeric) and store the result in a variable
* Use the C++ cin object to read user input (single character) and store the result in a variable
* Define the term validation
* Explain why user input validation is important.
* C++ Errors
* Define the term syntax
* Define the term syntax error
* Identify and correct syntax errors in C++ code
* Explain the relationship between syntax and compile time errors
* Define the term typographical error
* Explain when a typographical error is a syntax error
* Define the term logic error
* List and explain three common side effects of logic errors
* Math Library
* Explain the concept of a preprogramed function
* Explain the need for preprogramed math functions
* Explain the purpose of the math.h header file
* Write C++ code that includes the header file math.h
* C++ Errors
* Define the term syntax error
* Identify and correct syntax errors in C/C++ code
* Define the term logic error
* List and explain three common side effects of logic errors
* Write C++ code that makes use of the following common mathematical functions
* abs(n)
* pow(n,n2)
* sqrt(n)
* sin(n)
* cos(n,n2)
* tan(n)
* List the mathematical functions covered and the return type they will produce
* Write C++ code containing complex mathematical expressions and math functions
* Validation
* Define the term validation
* Explain why user input validation is important
* Write C++ code that will validate numeric input ensuring the input is within a pre-defined range
* Write C++ code that will validate character input ensuring the expected input is one of 3 (max) characters
* Symbolic Constants
* Define the concept of a constant value
* List common constants found in everyday life
* Define the term magic numbers as used by programmers
* Explain the advantage of using a constant for magic numbers when programming
* Define the term symbolic name
* Write C++ code that uses the const declaration qualifier to mark a value as constant

7.1. C++ Keyboard Input

**Introduction**

Programs are expected to communicate with users via input and output. Computers require user interaction in order to function. Users must enter data using the keyboard or other input devices such as barcode readers. When the program receives input, it acts on it to complete the process and offer results to the users. Of course, we used input and output statements in previous topics, but we didn't delve into detail about the notion. We'll talk about the cin command as part of the input command.

### 7.1.1 Explain the process that happens when the computer encounters the cin statement

In C++, input and output occur in streams, which are collections of bytes. It is referred to as an input operation when data flows into main memory from a keyboard, disk drive, network connection, etc., and as an output operation when data flows out of main memory to a display screen, printer, disk drive, network connection, etc.

C++ has three important header files for input and output operation, and these are:

* <iostream>:
* <iomanip>
* <fstream>

We have been working with iostream and iomanip header files already. We are going to reinforce the concepts again in this topic as part of input.

<iostream>: In this file, the cin, cout, cerr, and clog objects are defined. These objects stand for standard input stream, standard output stream, unbuffered standard error stream, and buffered standard error stream, respectively.

<iomanip>: This is useful for performing formatted I/O such as setw() setprecision() and others.

<fstream>: Used for user-controlled file processing.

For getting user input through the keyboard, we will use cin. cin stands for character input. cin is an object of the input stream and is used to take input from input streams like files, console, etc. A cin operator is used together with an extraction/exertion operator, which is written as >>, which is two greater than symbols. The << is known as the insertion operator in C++. If, for whatever reason, you forget to include the <iostream> header file, you will encounter a compilation error as follows:

error: 'cin' was not declared in this scope|

Whenever the compiler encounters the cin>>, it will wait for input from the user and enter key presses, then display the entered text onto the console.

**Syntax of cin**

cin>>variable

Let us look at the code in example 7.1 to accept name and age.

**EXAMPLE 7.1**

#include <iostream>

using namespace std;

int main()

{

string f\_name="";

int age=0;

cout<<"Please enter your firstname"<<"\n";

cin>>f\_name;

cout<<"Please enter your age"<<"\n";

cin>>age;

return 0;

}

Output

Please enter your firstname

Tshepo

Please enter your age

19

In example 7.1, you will notice that the program asks for the user's input. The first two statements in the main block declare variables of type string and int called f\_name and age. After the cout statement, the cin statement extracts from cin a value to be stored in it, which is f\_name. The same happens to cin>>age; and when it meets cin functions, it waits for the user to type in some text until the enter key is pressed.

scanf- Reads data from stdin and stores them according to the parameter format into the locations pointed by the additional arguments. scanf performs the same function as cin and is defined in the [cstdio](https://www.programiz.com/cpp-programming/library-function/cstdio) header file.

**Syntax**

scanf (const char\* format, ….)

The format parameter of scanf() can contain format specifiers that begin with %. The format string has the following parts:

Non-whitespace characters except % each of which consumes one identical character from the input stream. It can cause the function to fail if the next character on the stream does not compare equal.

**Example 7.2**

#include <cstdio>

#include <iostream>

using namespace std;

int main()

{

char f\_name [20];

int age=0;

cout<<"Please enter your firstname"<<"\n";

scanf("%15s", &f\_name);

cout<<"Please enter your age"<<"\n";

scanf("%d", &age);

cout << "Firsname = " << f\_name<<"\n";;

cout << "Age = " << age;

return 0;

}

**OUTPUT**

Please enter your firstname

Angie

Please enter your age

18

Firstname = Angie

Age = 18

**scanf vs cin**

The C++ standard library has cin, which is used to read input. The C standard library contains the scanf function, which is used to read input. With scanf(), the input type must be explicitly defined, but with cin, the redirection operation is overloaded using templates. This is the only obvious difference. If you later alter a variable type, scanf generates more work. A mess comes from combining the two. We shall only use the C++ function cin for the duration of this course.

### 7.1.2 Reading numeric user input and storing in a variable using cin

Reading numeric input is easy with cin. Let us take for example, accepting age. Earlier, you will have noticed that we managed to declare multiple variables. It is also possible to accept multiple values and store them in their respective variables. When accepting multiple values using cin, the variables are separated by the >> sign as shown in example 7.3.

**EXAMPLE 7.3**

using namespace std;

//This program declares multiple variables and accepts multiple values using one cin function

int main()

{

//declaring multiple variables

int num1, num2,answer=0;

cout << "Please enter number" << endl;

//accepting multiple variables

cin>>num1>>num2;

//displaying the values from the two respective variables

cout<<"The first number is "<<num1<<endl;

cout<<"The second number is "<<num2<<endl;

return 0;

}

Output

**Please enter the two numbers**

**98 65**

**The firs number is 98**

**The second number is 65**

After compiling the program, the user will be prompted to enter the two numbers. After entering each number, the user must press enter for the program to detect that a second value is being captured. Alternatively, the user can press the spacebar to separate the values in the respective variables. If the user forgets to press the enter key after capturing the first value, the two values will be stored in one variable.

### 7.1.3 Use the C++ cin object to read user input (single character) and store the result in a variable

Earlier in topic 6, we discussed the different data types, one of which is a char. As discussed, char is a C++ data type designed for the storage of letters and has a memory size of 1 byte. It also stores a single character as shown in example 7.4.

**EXAMPLE 7.4**

#include <iostream>

using namespace std;

int main() {

char grade = 'A';

cout << "I scored a/an: \t"<<grade<<" grade for the past test"<<endl;

return 0;

}

Output

I scored a/an: A grade for the past test

So, what would happen if you store a sting in the variable like :

char grade = 'Ann-Mary';

Because the data type is a char which takes one character, the result will be the last letter of the variable in this case it will be a letter y.

Remember, we mentioned that the char is interpreted as an ASCII character; we can convert the value of the variable by casting it to an int as shown below:

cout << "I scored a/an: \t"<<grade<<" grade for the past test represented as "<<int(grade)<<" in ASCII"<< endl;

### 7.1.4 Define the term validation

**VOCABULARY**

Validation or data validation means checking or verifying any data before it is allowed into a computer system. This process helps ensure data was not compromised or [corrupted](https://www.computerhope.com/jargon/c/corrupt.htm) during input

### 7.1.5 Explain why user input validation is important.

Data validation guarantees that the information received is accurate, comprehensive, and correct. If the data contains faulty or missing information, any analysis or processing of the data may be incorrect. When users enter data improperly, the software application must be able to detect inaccuracies and notify the user to make the necessary adjustments. Furthermore, fraudulent data that was provided erroneously on purpose should be rejected and never forwarded to the software's data processing stages. Data validation can also detect unintended data corruption during storage or transfer.

Imagine you want to write a program that deals with the ages of people and a user enters 200 as their age. This makes it obvious that the data is invalid as no one has lived up to that age.

**Types of data validation**

**Format validation**- makes sure the information is submitted in the proper format. Such as MM-DD-YYYY for date.

**Data type validation**-Makes sure that data is captured in correct data type defined.

**Range validation**- Ensures the values fall within a [range](https://www.computerhope.com/jargon/r/range.htm) limit eg age for a person is at most is 122 at the time of writing this book.

# FORMATIVE ASSESSMENT 7.1 INDIVIDUAL TASK

7.1.1 Consider the following code snippet below:

int n; float x = 3.8;

n = int(x);

cout << "n = " << n << endl;

What will be the value of 3. Explain your answer. (3)

7.1.2 Write a C++ code snippet to declare three variables and one statement to accept multiple values of variables namely firstname, surname and age. (4)

7.1.3 Which is the correct answer regarding '\n' and endl? (1)

1. [Both are same.](javascript:void(0);)
2. ['\n' and endl both are used to print new line but endl flushes the buffer after printing new line.](javascript:void(0);)
3. ['\n' and endl both are used to print new line but '\n' flushes the buffer after printing new line.](javascript:void(0);)
4. ['\n' used in C programming while endl used in C++ programming.](javascript:void(0);)

7.1.4 What will be the impact of leaving out namespace std is used in C++ ? (2)

7.1.5 Explain with examples what would happen if you store a string in the variable declared as char.. (3)

**[Total =13 marks]**

## 7.2. C++ Errors

A program error is a problem or fault that occurs in the program, which makes the program behave in an abnormal manner. Programming errors frequently go unnoticed until the program is compiled or run. It is possible for even experienced developers to make these errors. In programming, bugs and faults are also known as errors, and debugging is the process of fixing these errors.

**List of different types of errors**

* Syntax Error
* Run-Time Error
* Linker Error
* Logical Error
* Semantic Error

### 7.2.1 Define the term syntax

**VOCABULARY**

Syntax is the set of rules that define a language's structure. A programming language's syntax refers to the rules governing symbols, punctuation, and words.

Syntax improves code readability. Syntax ensures that the **four C’s of coding** are maintained:

* Communication
* Code integration
* Consistency
* Clarity

Syntax errors are easily identified during compilation of the program. The

### 7.2.2 Define the term syntax error

**VOCABULARY**

Syntax errors are bugs in a program which results from code violating the rule of C++ writing techniques or languages.

### 7.2.3 Identify and correct syntax errors in C++ code

The code presented in example 7.5 illustrates some syntax errors.

**EXAMPLE 7.5**

// C++ program to illustrate syntax error

#include <iostream>

using namespace std;

int main()

{

int x = 10;

integer y = 15; //integer is not valid

cout << "Value of x is : "<< x// semicolon missed

cout << "Value of y is : "<< y // semicolon missed

}

**When compiled, the output section under the build tabb will display the following:**

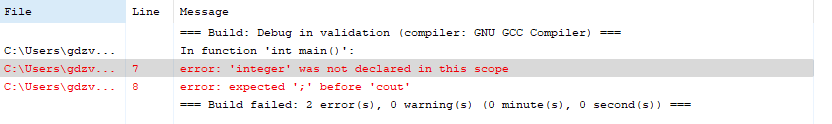
****

Figure 7. 1: Syntax errors

In the example above, the compiler will pick up errors in the code. First, the word integer does not exist within the C++ syntax. Another problem with the cout statements is that a semi-colon is missing. C++ statements end with a semicolon. Syntax errors can be caused by:

* missing the parenthesis (}) while writing the code.
* Displaying the value of a variable without its declaration.
* Missing a semicolon (;) at the end of the statement
* mistakes in the basic construct e.g. loop or conditional structure

### 7.2.4 Explain the relationship between syntax and compile time errors

Compile-time errors are the errors that occur when we write the wrong syntax. Compile time is the period when the programming code is converted to machine code. If the compiler encounters errors at the time of converting high level instructions to machine code, we call this a compilation error. There are mainly two types of compile-time errors: Syntax and Semantic errors. We are going to discuss semantic errors as we already know what syntax errors are.Semantic errors-This kind of error occurs when it is syntactically correct but has no meaning. For example:

int num1=8, num2=6, answer=0;

num1+num2=answer;

**Syntax errors**

The above snippet will generate the following error:

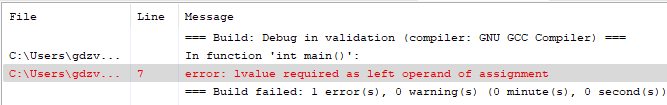


Figure 7. 2:Semantic error

Syntax errors occur at compile time, making them a subset of compile time errors.

### 7.2.5 Define the term typographical error

A typographical error is a mistake made in the typing of keywords and statements or expressions.

### 7.2.6 Explain when a typographical error is a syntax error

Also, it can be useful to consider the typical reasons for syntax errors, which include programming mistakes due to typos or forgetting words, commands, orders, or formats. Imagine when you are declaring a variable of boolean type and mistakenly type "boolean." This will result in it being a syntax error.

### 7.2.7 Define the term logic error

VOCABULARY

A logical error in a program is an error were the instructions given in the program do not accomplish the intended goal.

A logic error is a type of error that affects the way a program works. The program can run but does not do what it is expected to do. For example, let’s assume you want to write a program to accept four numbers and find the average of the four numbers. Here is a sample of the logic error:

**EXAMPLE 7.6**

// C++ program to illustrate syntax error

#include <iostream>

using namespace std;

int main()

{

int x = 15, y=26, z=19, k=18, average=0;

//error in calculating average

average=x+y+z+k/4;

cout << "The average of "<<x<<","<<y<<","<<z<<","<< k<<"is : "<<average<<endl;

}

**Output**

**The average of 15,26,19,18is : 64**

From the given example, the program runs fine when compiled and gives output which is incorrect. The program has been evaluated as follows:

Average=15+26+19+(18/4)

=15+26+19+4

Average =64 This answer is wrong

Clearly, operator precedence rules have not been followed during the implementation of the code.

The program should be entered as follows:

**EXAMPLE 7.7**

// C++ program to illustrate syntax error

#include <iostream>

using namespace std;

int main()

{

int x = 15, y=26, z=19, k=18;

float average;

average=(x+y+z+k)/4.0;

cout << "The average of "<<x<<","<<y<<","<<z<<","<< k<<"is : "<<average<<endl;

}

Output

The average of 15,26,19,18is : 19.5

**NOTE**

A program with a syntax error will not run. A program with a logic error will run but it will not perform as expected.

### 7.2.8 List and explain three common side effects of logic errors

It is important to understand what a side-effect is in relation to C++ programming. A side effect is a result of an operator, expression, statement, or function that persists even after the operator, expression, statement, or function has finished being evaluated. Common examples of side effects include changing the value of objects, doing input or output, or updating a graphical user interface (e.g. enabling or disabling a button).

Side effects can cause undefined behaviour. Consider the code in example 7.8.

**EXAMPLE 7.8**

#include <iostream>

using namespace std;

int main()

{

int a = 1;

cout<< ++a \* ++a;

return 0;

}

Here we can tell that it is using the final updated value of a, which is 3 in both operands, rather, it should have replaced one of them with the value 2 during runtime and the answer would've been 6 in any case (2x3 or 3x2). But no, the compiler uses 3 as the value of a. This is an undefined behaviour.

* side effects can also lead to unexpected results
* Logical errors  can lead to making poor decisions based on improper evaluation of expressions or incorrect decisions based on wrong evaluation

Let us consider the following example.

**EXAMPLE 7.9**

int add(int y, int x)

{

return x + y;

}

int main()

{

int x{ 7 };

int value{ add(x, ++x) }; // is this 7+8, or 8+8?

// It depends on what order your compiler evaluates the function arguments in

cout << value << '\n'; // value could be 11 or 12, depending on how the above line evaluates!

return 0;

}

The method by which function arguments are assessed is not specified by the C++ standard. This becomes a call to add(7,8), which equals 15 if the left argument is processed first. If the right argument is evaluated first, these calls add(8,8), which returns 16! Keep in mind that just one of the arguments to the function add() has a side effect, which causes this to be a concern.

So far we explained syntax and logic errors. We will move on to discuss runtime errors and linker errors.

**Define the term runtime error**

A runtime error in a program is an error that occurs while the program is running after being successfully compiled. Runtime errors are not compilation error, so the compilation will be successfully done. We can check this error if we try to divide a number with 0.

**EXAMPLE 7.10**

#include <iostream>

#include <cmath>

using namespace std;

int main()

{

int x = 52;

int y = 0;

cout<<x<<"divided by "<<y<< " is " << x/y<<"\n";

return 0;

}

**OUTPUT**

**52 divided by 0 is**

**Process returned -1073741676 (0xC0000094) execution time : 4.008 s**

So, as you will notice, the program compiles successfully but encounters issues when it tries to do the division. The error code (0xC0000094) signifies a division by zero type of error. Again, just like logical errors, runtime errors are hard to find as the compiler doesn’t point to the line at which the error occurs.

Other causes of runtime errors include:

* Invalid memory access during run-time.
* Large allocation of memory together/Large Static Memory Allocation

**Invalid memory access during run-time.**

The code shown in example 7.11 illustrates circumstances which cause invalid memory access.

**EXAMPLE 7.11**

#include <iostream>

using namespace std;

int arr[5]; //declaring an array

int main()

{

int ans=arr[-1]; //specifying position which does not exist.

cout<<ans<<endl;

return 0;

}

**Large allocation of memory together/Large Static Memory Allocation**

Here is an example:

#include <iostream>

using namespace std;

int main() {

int arr[1000000000];

return 0;

}

After running the program, you get the following error:

Process returned -1073741571 (0xC00000FD) execution time : 3.534 s

0xC00000FD is a stack overflow error. The size allocated for array is way too big. Usually, array size is up to 10^8.

**Linker Error**

This type of error happens when the application is successfully compiled and attempts are made to connect the various object files with the main object file. The executable is not generated when this error occurs.

**EXAMPLE 7.12**

// C++ program to illustrate linker error

#include <iostream>

#include <bits/stdc++.h>

using namespace std;

int Main() // Here Main() should be main()

{

int number\_1;

cin>>number\_1;

cout << " "<< number\_1;

}

With a linker error, the program doesn’t run. Instead, it produces a compilation error:



Figure 7. 4:Linker Error

# FORMATIVE ASSESSMENT 7.2 INDIVIDUAL TASK

7.2.1 Define the term bug as applied in programming. (2)

7.2.2 Define the term syntax as applied in C++ programming. (2)

7.2.3 Consider the code below:

#include <iostream>

#include <cmath>

using namespace std;

int main()

{

int length=0;/\* this is the length

int width=0; /\*this is the width\*/

int area=0;

area=length \* width;

cout<<"The area of a shape whose length is" <<length<< "and width is "<<width<< "is " <<area<<"/n";

return 0;

}

The program above gives two errors. Identify the two errors. (2)

7.2.4 Following the two errors in the code presented in question 7.2.3, the program gives the following output.

The area of a shape whose length is0and width is 0is 0

Modify the program to give the result of area of a shape whose length is 5cm and width is 3 cm without altering the declarations. (4)

7.2.5 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_are errors that occur when you violate the rules of writing syntax and indicates that code must be fixed before being compiled. (1)

7.2.6 List two types of compile time errors. (2)

7.2.7 List ANY three possible causes of runtime errors. (3)

**[Total =16 marks]**

## 7.3. Math library

C++ has many functions that allows users to perform mathematical tasks on numbers.

### 7.3.1 Math Library

#### 7.3.1.1 Explain the concept of a preprogramed function

A preprogramed function is built into the software and does not need to be created by a programmer. Pre-defined functions often exist to carry out common tasks, such as: finding an average number. determining the length of a string. The opposite of pre-programmed functions would be user-defined functions. There are pre-programmed functions that work without including the cmath or math.h library e.g., max and min. max is used to find the largest number between two numbers.

**Syntax**

max(x,y)

Min is used to find the smallest number between the two.

Syntax

min(x,y)

**EXAMPLE 7.13**

#include <iostream>

using namespace std;

int main()

{

cout<<"The largest number between "<<11<<" and " << 12<< " is "<<max(11,12)<<"\n";

cout<<"The smallest number between "<<11<<" and " << 12<< " is "<<min(11,12)<<"\n";

return 0;

}

**OUTPUT**

**The largest number between 11 and 12 is 12**

**The smallest number between 11 and 12 is 11**

In the above example, we used min and max function without calling cmath or math.h header file.

**NOTE**

User-defined functions are functions that are not pre-defined but are created by a programmer.

#### 7.3.1.2 Explain the need for preprogramed math functions

Preprogramed functions are quite helpful to the programmer as they save a lot of time is software development. Instead of the programmer recoding the mathematical expressions to do a simple complex calculation like power, square root and average minimum and maximum, to mention a few, the programmer can only call this function from a defined class. In C++, the programmer has to include the cmath library in order to be able to do these calculations. Let us look at the following example to calculate the square root of a number: C++ uses the keyword sqrt when calling the pre-programmed function for square root.

**EXAMPLE 7.14**

#include <iostream>

#include <cmath> //math.h has been depreciated in C++

using namespace std;

int main()

{

double number=0;

cout<<"Please enter the number to calculate the square root"<<"\n";

cin>>number;

cout<<"The square root of "<<number<< " is " <<sqrt(number)<<"\n";

return 0;

}

**Output**

**Please enter the number to calculate the square root**

**81**

**The square root of 81 is 9**

We could have written a longer mathematical expression to find the square root but we simply called sqrt.

#### 7.3.1.3 Explain the purpose of the math.h header file

The math.h header defines various mathematical functions and one macro. All the functions available in this library take double as an argument and return double as the result. math. h is the deprecated C header. cmath is the C++ header. The difference is that cmath puts all the names in the std namespace. For the purposes of this tutorial, we are going to stick to cmath.

#### 7.3.1.4 Write C++ code that includes the header file cmath

We are going to make use of cmath header file and apply pre-programmed functions to solve some mathematical expressions.

We are going to calculate volume of a cube.

Volume of a cube is the total cubic units occupied by it, in a three-dimensional space. A cube is a 3d-shape, that has six faces, twelve edges and eight vertices. Hence, the volume of a cube is the space enclosed by its six faces. Therefore, the volume of cube is equal to product of its length, width and height. It is measured in cubic units. Figure 7.3 shows how a cube looks like.

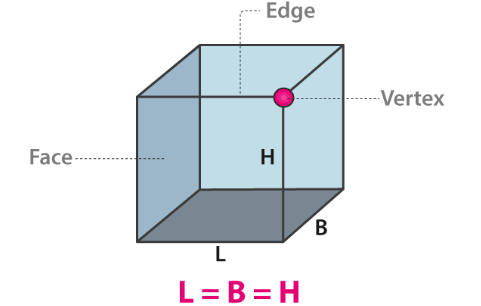


Figure 7. 3: A cuboid [https://byjus.com/volume-of-cuboid-calculator/]

**Volume of a Cube = Length × Width × Height**

Volume = a × a × a

Volume = a3

**Write a program to accept one side of the cube and calculate its volume.**

**EXAMPLE 7.15**

Solution

#include <iostream>

#include <cmath>

//program to calculate volume of a cube.

using namespace std;

int main()

{

double side=0, volume=0;

//Accepting input

cout<<"Please enter size of one side of the cube"<<"\n";

cin>>side;

if (cin.fail()){

cout<<"Enter a valid size of the side"<<"\n";

}

else{

//processing

volume = pow(side, 3);

cout<<"The volume of a cube whose side is "<<side<< "cm is " <<volume<<"cm3"<<"\n";

return 0;

}

}

**Sample Output**

**Please enter size of one side of the cube**

**9**

**The volume of a cube whose side is 9cm is 729cm3**

**Please enter size of one side of the cube**

**bbbbbb**

**Enter a valid size of the side**

In the example, we implemented cmath header file which allowed us to use pow function.

**TASK 7.1**

Write a C++ program using inbuilt function for cube root to **f**ind the length of the edges of the cube, if its volume is equal to 125 cm3. Use cin.fail( ) to trap invalid entry for volume.

**TASK 7.2** :**Write a C++ Program to raise any number X to power N.**

### 7.3.2 Write C++ code that makes use of the following common mathematical functions

So, we have covered some mathematical functions found in the cmath library. Let us explain more functions.

7.3.2.1 abs(n)- The abs() function in C++ returns the absolute value of the argument. Mathematically, abs(num) = |num|.

**EXAMPLE 7.16**

#include <iostream>

#include <cmath>

using namespace std;

int main() {

// get absolute value of -8.5

cout << abs(-8.5);

return 0;

}

// Output: 8.5

abs(x) Parameters

The abs() function takes the following parameter:

x - a floating point number whose absolute value is returned. It can be of the following types:

* + double
  + float
  + long double

#### 7.3.2.2 pow(x,y)

#### The pow() method returns the result of raising the first argument by the second argument's power.

pow(x, y) = xy

**Syntax**

pow(double x, double y)

pow( ) Parameters

The pow() function takes two parameters:

x - the base value

y - exponent of the base

The pow( ) function returns: 1.0 if exponent is zero and 0.0 if base is zero

**EXAMPLE 7.17**

#include <iostream>

#include <cmath>

using namespace std;

int main() {

cout << pow(8,2);

return 0;

}

**Output**

64

#### 7.3.2.3 sqrt(x)

#### The sqrt() function in C++ returns the square root of a number. Mathematically, sqrt(x) = √x

**EXAMPLE 7.18**

#include <iostream>

#include <cmath>

using namespace std;

int main() {

cout << "Square root of 100 = ";

// print the square root of 100

cout << sqrt(100);

return 0;

}

Output

10

#### 7.3.2.4 sin(x)

The sine of an angle (argument) provided in radians is returned by the sin() function in C++. The sin() function takes a single mandatory argument in radians.

**EXAMPLE 7.19**

#include <iostream>

#include <cmath>

using namespace std;

int main()

{

double x = 0.475965, result;

result = sin(x);

cout << "sin(x) = " << result << endl;

return 0;

}

OUTPUT

sin(x) = 0.458196

#### 7.3.2.5 cos(x,y)

When given an angle (argument) in radians, the cos() function in C++ gives the angle's cosine.

**EXAMPLE 7.20**

#include <iostream>

#include <cmath>

using namespace std;

int main()

{

double x = 0.8, result;

result = cos(x);

cout << "cos(x) = " << result << endl;

return 0;

}

**OUTPUT**

**cos(x) = 0.877583**

As can be seen from the example, the cos() function takes one parameter and returns the value in the range of **[-1, 1]**. The returned value is either in double, float, or long double.

#### 7.3.2.6 tan(n)

The tan() function in C++ returns the tangent of an angle (argument) given in radians. The only required argument for the tan() method is in radians (can be positive, negative, or 0). The tan() function returns the value in the range of **[-∞, ∞]**.

**EXAMPLE 7.21**

#include <iostream>

#include <cmath>

using namespace std;

int main()

{

long double x = 0.89947, result;

result = tan(x);

cout << "tan(x) = " << result << endl;

return 0;

}

OUTPUT

tan(x) = 1.25879

### 7.3.4 List the mathematical functions covered and the return type they will produce

Table 7.1 summarises the mathematical functions covered in this

Table 7. 1::Mathematical functions

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Description** | **Example** | **Return Type** |
| abs(x) | Returns the absolute value of x | abs(-5.5)=5 | int |
| pow(x) | returns the value of x to the power of y (xy): | pow(6,3)=216 | int |
| sqrt(x) | Returns the value of square root of x | sqrt(81)=9 | int |
| ceil(x) | rounds x number upwards to its nearest integer | ceil(5.1)=6 | int |
| floor(x) | rounds x number downwards to its nearest integer, and returns the result | floor(5.1)=5 | int |
| sin(x) | Returns the sine of x (x is in radians) | sin(0.956421)= 0.817134 | double |
| cos(x) | Returns the cosine of x | cos(0.7)= 0.764842 | double |
| tan(n) | Returns the tangent of an angle | tan(0.54754)= 0.609726 | double |
| cbrt(x) | returns the cube root of the given argument | cbrt(125)=5 | int |
| remainder() | computes the floating-point remainder of numerator/denominator (rounded to nearest). | remainder(5/2.13) =0.74 | float |

NOTE

The return type depends largely on the data type of result and the input. For example:

cout << cbrt(512.6) << endl;

output

8.00312

If we declare our output as int then the output that will be displayed is of int data type.

int result=cbrt(512.6);

cout << result << endl;

output

8

### 7.3.5 Write C/C++ code containing complex mathematical expressions and math functions

In this section, we are going to use one of the common problems which require complex mathematical expressions. We are going to calculate compound interest.

Question: **Write a C++ Program to Calculate Compound Interest**

So, what is compound interest?

It is the addition of interest to the principal sum of a loan or deposit, or in other words, interest on interest. It is the result of reinvesting interest, rather than paying it out, so that interest in the next period is then earned on the principal sum plus previously accumulated interest.

It may be contrasted with simple interest, where interest is not added to the principal, so there is no compounding.

**Annual compound interest formula**

The formula for annual compound interest, including principal sum, is:

A = P (1 + r/n) (nt)

Where:

*A = the future value of the investment/loan, including interest  
P = the principal investment amount (the initial deposit or loan amount)  
r = the annual interest rate (decimal)  
n = the number of times that interest is compounded per year  
t = the number of years the money is invested or borrowed for*

**NOTE**

Note that this formula gives you the future value of an investment or loan, which is compound interest plus the principal. Should you wish to calculate the compound interest only, you need this:

*Total compounded interest = P (1 + r/n) (nt)– P*

This program will read principal, rate and time in years and then print compound interest on entered principal for given time period.

**EXAMPLE 7.22**

Solution

/\* C++ Program to Calculate Compound Interest \*/

#include<iostream>

#include<cmath>

using namespace std;

int main()

{

//declaring variables

float principal,rate,time,comp\_int;

//Accepting input

cout<<"Enter Principle (Amount) in Rands:: ";

cin>>principal;

cout<<"\nEnter Rate of Interest :: ";

cin>>rate;

cout<<"\nEnter Time Period in years:: ";

cin>>time;

//processing

comp\_int = principal\*pow((1+rate/100),time);

//output

cout<<"\nThe total compound interest is = R"<< comp\_int<<"\n";

return 0;

}

SAMPLE OUTPUT

Enter Principle (Amount) in Rands:: 285000

Enter Rate of Interest :: 12.75

Enter Time Period in years:: 4

The total compound interest is = R460586

# FORMATIVE ASSESSMENT 7.3 INDIVIDUAL TASK

7.3.1 What is the difference between the cmath and math,h header files. (4)

7.3.2 Complete the table below with pre-programmed functions

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Example** |
| abs(x) |  |  |
| pow(x) |  |  |
| sqrt(x) |  |  |
| ceil(x) |  |  |
| floor(x) |  |  |
| cbrt(x) |  |  |
| remainder() |  |  |

(14)

7.3.3 Write a C++ code to solve the following expression:

(a+b)^2

Accept a and b from the keyboard.

Use a=2.4, b=4.5

Display the output. (10)

7.3.4 What is the value of answer in the following C++ program?

int x=4;

int answer=pow(x,2);

cout<<answer<<endl;

(2)

**[Total =30 marks]**

## 7.4. Validation

### 7.4.1 Define the term validation

### 7.4.2 Explain why user input validation is important

Please refer to Section 7.14 and 7.1.5

### 7.4.3 Write C/C++ code that will validate numeric input ensuring the input is within a pre- defined range

In section 7.1.5, we listed the different types of validation as:

* Range check validation
* Form validation
* Data type validation.

In this section, we are going to implement data type validation and range validation.

**Range validation check**

Let’s assume we want to run a program that allows users to capture their age. We do know that age should be greater than 0 and less than 123. We can implement a while loop. We have not covered loops in detail, but using your knowledge of loops covered when working with Arduino should be sufficient. Example 7.23 illustrates range validation check.

**EXAMPLE 7.23**

#include <iostream>

using namespace std;

int main() {

//variable declaration

int age=0;

cout<<"Please enter your age"<<endl;

cin>>age;

//range check

while(age < 0 || age > 123) {

cout << "Invalid age, re-enter: ";

cin >> age;

}

return 0;

}

**Output**

**Please enter your age**

**-23**

**Invalid age, re-enter: -23**

**Invalid age, re-enter: 45**

Looking at the code above, you will notice that if the age entered is less than 0 and greater than 123 the system will prompt the user to capture the age again. So, we can say we are sorted with the range check but another problem arises when a user captures a string. You for the variable age. You notice the program continues. Here is how we can resolve this problem.

**cin Functions**

**cin.fail()** - This function returns true when an input failure occurs. In this case it would be an input that is not an integer. If the cin fails, then the input buffer is kept in an error state.

The code in example 7.24 shows the use of cin.fail().

**EXAMPLE 7.24**

#include <iostream>

using namespace std;

int main()

{

int age;

cout << "Enter number :" << endl;

cin >> age;

if(cin.fail())

{

cout << "Not a number " << endl;

}

else{

cout << "is a number " << endl;

}

return 0;

}

**Output**

**Enter number :**

**A**

**Not a number**

**Enter number :**

**45**

**is a number**

**So, we can modify our earlier program on age range as shown in example 7.25:**

**EXAMPLE 7.25**

#include <iostream>

using namespace std;

int main()

{

int age;

cout << "Enter number :" << endl;

cin >> age;

if(cin.fail())

{

cout << "Not a number " << endl;

}

else{

cout << "is a number " << endl;

//range check

while(age < 0 || age > 123) {

cout << "Invalid age, re-enter: ";

cin >> age;

}

}

return 0;

}

**cin.clear()** - This is used to clear the error state of the buffer so that further processing of input can take place. This ensures that the input does not lead to an infinite loop of error message display.

**cin.ignore()** - This function is used to ignore the rest of the line after the first instance of error that has occurred, and it skips to or moves to the next line.

Example 7.26 shows the implementation of cin.clear(), cin.ignore()

**EXAMPLE 7.26**

#include <iostream>

using namespace std;

int main()

{

int age;

cout<<"Enter the number "<<endl;

while(!(cin >> age)) {

cin.clear();

cin.ignore(1000, '\n');

cout << "The data entry failed to meet the requirements"<<endl;

}

return 0;

}

Output

Enter the number

fgg

The data entry failed to meet the requirements

What is important is that if we capture a value like “12abc”, the program will discard “abc” and extract the number. However, if the user captures the value as abc12, an error message will be displayed.

7.4.4 Write C++ code that will validate character input ensuring the expected input is one of 3 (max) characters

The char data type is used to store a single character. The character must be surrounded by single quotes, like 'A' or 'c':

**isalpha()**

We can start by implementing the isalpha() function. The isalpha() function in C++ checks if the given character is an alphabet or not.  Here is an example to validate if a character entered through the keyboard is an alphabet.

**EXAMPLE 7.27**

/\* isalpha example \*/

#include <iostream>

using namespace std;

int main ()

{

char chars, chars2;

cout<<"Enter alphabet"<<endl;

cin>>chars;

//checking for alphabetic character

if (isalpha(chars)){

cout<<chars<< "is an alphabetic character"<<endl;

}

else {

cout<<chars<< "is not an alphabetic character"<<endl;

}

//displaying the ASCII equivalent of the entered value

cout<<int(chars)<< " is the ASCII equivalent of "<<chars<<endl;

return 0;

}

**OUTPUT**

**Enter alphabet**

**M**

**Mis an alphabetic character**

**77 is the ASCII equivalent of M**

In the above example, we also casted the char entered using int to check its ASCII equivalent. This is quite important if you want specific character you can specify their ASCII values as well.

**ispunct()-** Checks if character is a punctuation character. See example 7.28.

**EXAMPLE 7.28**

/\* ispucnt example \*/

#include <iostream>

using namespace std;

int main ()

{

char ch;

cout<<"Enter the character"<<endl;

cin>>ch;

cout<<ispunct(ch)<<endl;

return 0;

}

**OUTPUT**

**Enter the character**

**?**

**16**

**Enter the character**

**D**

**0**

If the character is not a punctuation, the above program returns zero; for example, D returned 0. If ch is a punctuation character, a value other than zero (i.e., true) is returned. Our program returned 16 for ? which is a non-zero value to indicate the entered character is a punctuation mark.

Problem: Write a program in C++ to allow users to enter gender and greet them accordingly. If a user enter ‘m’ or ‘M’ the program must print:

Hello sir. How are you?

If the users enters ‘f’ or ‘F’ the program must print

Hello madam. How are you?.

If the users enter ‘O’ or ‘o’ the program must print.

Hello. How are you?

See the solution in Example 7.29.

**EXAMPLE 7.29**

//Checking gender entries

#include <iostream>

using namespace std;

int main ()

{

char gender;

cout<<"Please select your gender from the following [M/F/O] or [m/f/o]"<<endl;

cin>>gender;

gender=toupper(gender);

if (gender=='M'){

cout<<"Hello sir. How are you?"<<endl;

}

else if (gender=='F'){

cout<<"Hello madam. How are you?"<<endl;

}

else if (gender=='O'){

cout<<"Hello. How are you?"<<endl;

}else

{

cout<<"Incorrect entry"<< endl;

}

return 0;

}

**OUTPUT**

**Please select your gender from the following [M/F/O] or [m/f/o]**

**M**

**Hello sir. How are you?**

**Please select your gender from the following [M/F/O] or [m/f/o]**

**f**

**Hello madam. How are you?**

**Please select your gender from the following [M/F/O] or [m/f/o]**

**/**

**Incorrect entry**

In the given example we also use the function toupper().

toupper()-checks whether character is alphabetic & converts to upper case

tolower()checks whether character is alphabetic & converts to lower case

The only challenge we have with our program above is that it accepts non alphabetic characters. We can modify the program a bit by adding a while loop as follows:

**EXAMPLE 7.29**

/\* isalpha and toupper example \*/

#include <iostream>

using namespace std;

int main ()

{

char gender;

cout<<"Please select your gender from the following [M/F/O] or [m/f/o]"<<endl;

cin>>gender;

gender=toupper(gender);

while (isalpha(gender)){

if (gender=='M'){

cout<<"Hello sir. How are you?"<<endl;

}

else if (gender=='F'){

cout<<"Hello madam. How are you?"<<endl;

}

else if (gender=='O'){

cout<<"Hello. How are you?"<<endl;

}else

{

cout<<"Incorrect entry"<< endl;

}

return 0;

}

cout<<"You did not enter an alphabetic character"<< endl;

}

**OUTPUT**

**Please select your gender from the following [M/F/O] or [m/f/o]**

**8**

**You did not enter an alphabetic character**

NOTE

When we were validating integers, we just used cin.fail(). In this instance it will not work if a user enters a number because is will be interpreted as an ASCII character.

# FORMATIVE ASSESSMENT 7.4 INDIVIDUAL TASK

7.4.1 Define the term validation as applied in C++ programming. (2)

7.4.2 Explain why user input validation is important. (2)

7.4.3 List three types of data validation. (3)

7.4.4 In your own words, explain what you understand by the term range check as applied in C++ programming. (3)

7.4.5 Write a C++ program to allow a user to enter a test mark. The program should report an error if the mark is outside the range of 0-100. If the entry is not a number the program must report an error and tell user to re-enter a valid mark. If the mark entered is valid and greater than 49, the program must display a message which says:

Congratulations. “You passed the test”

else the program display:

“Unfortunately. You failed the test” (12)

**[Total =22 marks]**

## 7.5. Symbolic Constants

### 7.5.1 Define the concept of a constant value

Constants refer to fixed values that the program may not alter, and they are called literals. Unlike variables, constants never change in value. You must initialize a constant when it is created. C++ has two types of constants: literal and symbolic.

A literal constant is a value typed directly into your program wherever it is needed. For example, consider the following statement:

int width =7;

This statement assigns the integer variable width the value 7. The 7 in the statement is a literal constant. You can't assign a value to 7, and its value can't be changed. Another example would be the values true and false, which are stored in bool variables, also are literal constants.

A symbolic constant is a name that is given to a constant value. Constant variables are one type of symbolic constant, as a variable has a name (its identifier) and a constant value. The const keyword precedes the type, name, and initialization.

**Syntax**

*const data\_type name =value;*

Example const double PI=3.142;

Once declared, the value of PI can not be assigned a new value. Example 7.29 is a program to calculate area of a circle.

**EXAMPLE 7.29**

#include <cmath>

#include <iostream>

using namespace std;

int main() {

//declaring pi using const keyword

const double PI =3.142;

double area=0, radius=0;

cout<<"Please enter the radius of the circle "<<endl;

cin>>radius;

area=PI\*pow(radius, 2);

cout<<"The area of a circle whose radius is " << radius<< " is " <<area<<"cm^2"<<endl;

return 0;}

**OUTPUT**

**Please enter the radius of the circle**

**5**

**The area of a circle whose radius is 5 is 78.55cm^2**

So, for argument sake, let us try to add the following line in our code to change the value of PI.

PI=5.321;

When we try to compile the program we get an error saying”

|error: assignment of read-only variable 'PI'|

Well-named symbolic constants also make a program more understandable. Constants often are fully capitalized by programmers to make them distinct from variables. This is not required by C++, but the capitalization of a constant must be consistent because the language is case sensitive.

There's another way to define constants that dates to early versions of the C language, the precursor of C++. The preprocessor directive #define can create a constant by specifying its name and value, separated by spaces:

**Syntax**

*#define name value*

Example

#define PI 3.142

Take note that there is no semi-colon, as this is inserted as part of the header files. In addition, the constant does not have a type such as int or char. The #define directive enables a simple text substitution that replaces every instance of PI in the code with 3.142. The compiler sees only the result. Because these constants lack a type, the compiler cannot ensure that the constant has a proper value.

We are going to modify our earlier code on area and use the #define preprocessor to declare our constant PI.

**EXAMPLE 7.30**

#include <cmath>

#define PI 3.142

#include <iostream>

using namespace std;

int main() {

//declaring pi using const keyword

double area=0, radius=0;

cout<<"Please enter the radius of the circle "<<endl;

cin>>radius;

area=PI\*pow(radius, 2);

cout<<"The area of a circle whose radius is " << radius<< " is " <<area<<"cm^2"<<endl;

return 0;

}

**OUTPUT**

**Please enter the radius of the circle**

**5**

**The area of a circle whose radius is 5 is 78.55cm^2**

If we try to reassign PI with a new value, the compiler reports a different error this time:

|error: lvalue required as left operand of assignment|

The <cmath> header of C++ also includes several mathematical constants that can be used in mathematical and quantitative code. These can be used by adding #define directive and specify a macro “\_USE\_MATH\_DEFINES”. This macro is to be added to the program before we include the <cmath> library.

**EXAMPLE 7.31**

#define \_USE\_MATH\_DEFINES

#include <cmath>

#include <iostream>

using namespace std;

int main() {

cout<<"The value of PI is "<<M\_PI<<endl;

return 0;

}

**OUTPUT**

**The value of PI is 3.14159**

For more pre-defined constants, refer to the following ling:

<https://learn.microsoft.com/en-us/cpp/c-runtime-library/math-constants?view=msvc-170>

### 7.5.2 List common constants found in everyday life

The following are examples of constants in our everyday life:

* distance from earth to sun
* Boiling point of water
* Freezing point of water

Discussion

What other constants do you know? Share your answers with your teacher.

### 7.5.3 Define the term magic numbers as used by programmers

A magic number is a numeric literal that is used in the code without any explanation of its meaning. The use of magic numbers makes programs less readable and hence more difficult to maintain and update. Here is an example:

*Example:*

double salary = 500 \* workedhours;

// what is the meaning of 500?

It is better to define symbolic names, so called **constants**, and use these instead of numeric literals. This can be rewritten as:

// definition of a constant SALARY\_PER\_HOUR

final double SALARY\_PER\_HOUR = 500;

// calculating total salary

final double salary = SALARY\_PER\_HOUR \* workedhours

It is now clear that rather than using the value 500 in the program itself. In addition, we added the word final which is mainly used to restrict class inheritance. At this level, we are not going to talk much about inheritance, a feature of the object-oriented paradigm. Here is a link with more clarity on object-oriented programming.

### 7.5.4 Explain the advantage of using a constant for magic numbers when programming

Two key advantages for using constants for magic numbers

* ***Readability*** of the program: an identifier of a constant with a significant name is much more readable than a magic number (e.g., SALARY\_PER\_HOUR is self-explanatory compared to just the value 500
* ***Modifiability of the program***: If the value has to be changed, its only done in the declaration and will be changed throughout the program.

### 7.5.5 Define the term symbolic name

Symbolic names can be used in C++ for various data items used by a programmer in his program. A symbolic name is a known identifier. The identifier is a sequence of characters taken from the C++ character set.

### 7.5.6 Write C/C++ code that uses the const declaration qualifier to mark a value as constant

Write a C++ program that can calculate the price of a product after 15% VAT. For this program, a user needs to enter two inputs, which are price per unit and quantity of a product. The program calculates the price of the product using the following formula:

Total\_price= Price\_per\_unit x Quantity\_of\_product

The program must add the VAT chargeable on the product which is the Total\_price +15%

After calculating the calculating the total price and tax, the program would display the total price without tax, tax\_amount and the total price with tax amount added.

**EXAMPLE 7.32**

#include <iostream>

#include <cmath>

using namespace std;

int main()

{

//declaring variables

double total\_price=0, price\_per\_item=0, quantity\_of\_products=0;

//declaring a constant value for tax

const float vat=0.15;

double vat\_amount=0, grand\_total=0;

//Accepting input

cout<<"Enter the unit price of the item"<<endl;

cin>>price\_per\_item;

cout<<"Enter the number of items bought"<<endl;

cin>>quantity\_of\_products;

//calculating total price

total\_price=price\_per\_item\* quantity\_of\_products;

cout<<"The total price of " <<quantity\_of\_products <<" items with a unit price of "<<price\_per\_item<<" is "<<total\_price<<endl;

//calculating tax amount

vat\_amount=total\_price\*vat;

cout<<"The tax amount for " <<total\_price <<" is R"<<vat\_amount<<endl;

grand\_total=total\_price+vat\_amount;

//displaying the grand total

cout<<"The grand total price is R"<<grand\_total<<endl;

return 0;

}

**OUTPUT**

**Enter the unit price of the item**

**10**

**Enter the number of items bought**

**10**

**The total price of 10 items with a unit price of 10 is 100**

**The tax amount for 100 is R15**

**The grand total price is R115**

# FORMATIVE ASSESSMENT 7.5 INDIVIDUAL TASK

7.5.1 Define the term constant as used in C++ programming. (2)

7.5.2 List two ways to declare constants in C++. (2)

7.5.3 Write code to declare a const to hold distance of 257. (2)

7.5.4 Define the term magic number as used in programming. (2)

7.5.5 Explain the TWO advantages of using a constant for magic numbers when programming. (4)

7.5.6 Consider the code snippet below:

const int LENGTH = 21;

char message[LENGTH];

cout << "Enter a sentence on the line below." << endl;

cin >> message;

cout << message << endl;

Suppose that in response to the prompt, the user types the following line and presses Enter: We are enjoying code.

What will the output of the code fragment look like and give a reason for your answer? (3)

7.5.7 Write a C++ program to calculate the volume of a sphere whose radius is entered through the keyboard by the user. Ensure that the input is validate for entry.

Volume= **4/3 πr3**  (10)

**[Total =25 marks]**

# SUMMATIVE ACTIVITY 7.6 INDIVIDUAL TASK

7.6.1 Which of the following is known as the insertion operator in C++?

1. [<<](javascript:void(0);)
2. [>>](javascript:void(0);)
3. [~](javascript:void(0);)
4. [^^](javascript:void(0);)

7.6.2 Which of the following is known as exertion operator in C++?

1. [<<](javascript:void(0);)
2. [>>](javascript:void(0);)
3. [~](javascript:void(0);)
4. [^^](javascript:void(0);)

7.6.3 Define the term runtime error. (2)

7.6.4 What is the correct output of given code snippets?

a) 10

b) Compile -time error

c) Runtime error

d) Linker error

7.6.5 Consider the following code

#include <iostream>

#include <iomanip>

using namespace std;

int main()

{

const double PI = 3.14159265;

cout<< PI <<endl;

}

Modify the code to print the following output: 3.141593

7.6.6 Write a program in C++ to find the area of any triangle using Heron's Formula. Heron’s formula is used to calculate area of a triangle when you are given the length of the three sides. The formula is as follows:

A = √s(s−a)(s−b)(s−c)

NB: The program should display an error message if the sides are not valid entries and if the area can not be calculated i.e., nan. For example

Find the area of any triangle using Heron's Formula :

----------------------------------------------------------

Input the length of 1st side of the triangle :

9

Input the length of 2nd side of the triangle :

100

Input the length of 3rd side of the triangle :

2

valid The sides of the triangle are not valid :

(15)

7.6.7 Define the term typographical error as applied in programming. (2)

**[Total=26marks]**

# References

Oualline, S., 2003. *Practical C++ programming*. " O'Reilly Media, Inc.".

<https://www.programiz.com/cpp-programming/library-function/cmath>

<https://www.w3schools.com/cpp/cpp_math.asp>

# Topic 8: Selection control structure

After you have completed this module, you should be able to :

* Define the term relational expression
* Explain the anatomy of a simple relational expression
* List and explain the different C++ relational operators
* Identify valid relational expressions
* Identify invalid relational expressions
* Explain the numeric result that is generated from a- relational expression
* Write C++ code that will save the result of relation expression that contains both or a mix of numeric variables and literals in an int variable
* Write C++ code that will save the result of relation expression that contains both or a mix of numeric variables and literals in a Boolean variable
* Explain how the integer result produced by a relation expression relates to bool true/false
* Write C++ code that will save the result of relation expression that contains both or a mix of character variables and literals in an int and bool variable
* Students should be able to
* Explain the purpose of the if statement in C++
* Explain the purpose of the else statement in C++
* Determine the program flow when an if statement is encountered
* Identify or correct the general form for an if-else statement
* Write C++ code that will use relational expressions in if-else statements (Max 2 two nested levels only)
* Write C++ code that will use relational expressions in if-else statements with compounded content (Max 2 two nested levels only)
* Explore the concept of block scope when writing if-else statements with compounded content
* Define the term logic operator
* List the different logic operators
* Explain how each logic operator will influence the result of an expression
* Write C++ code that will save the result of relation expression that contains both or a mix of numeric variables, character variables and literals and a single logical operator in an int variable
* Write C++ code that will save the result of relation expression that contains both or a mix of numeric variables, character variables and literals and a single logical operator in a bool variable
* Write C++ code that will save the result of relation expression that contains both or a mix of numeric variables, character variables and literals and multiple (Max 3) logical operator in a bool variable
* Write C++ code that will save the result of relation expression that contains both or a mix of numeric variables, character variables and literals and multiple (Max 3) logical operator in an int variable
* Define the term nested if statement
* Determine the application flow when a nested if statement is encountered
* Identify or correct the general form for a nested if-else statement
* Write C++ code that will use relational expressions in nested if-else statements (Max 3 levels)
* Write C++ code that will use relational expressions in nested if-else statements with compounded content (Max 3 levels)
* Write C++ code that will use relational expression containing logic operators in nested if-else statements (Max 3 levels, Max 3 logic operators per level)
* Write C++ code that will use relational expressions containing logic operator in nested if-else statements with compounded content (Max 3 levels, Max 3 logic
* operators per level)

# 8.1 Relational expressions

**Introduction**

One of the core functions is to evaluate relational expressions. We will use logic operators, relational operators, and selection statements to accomplish this. Selection is a programming construct that deals with the execution of statements based on conditions. This is what we will discuss in this topic.

### 8.1.1 Define the term relational expression

A relational expression indicates the condition that the system evaluates. The defined operation and the values of the operands or expressions that are compared at execution time determine the outcome of the evaluation of the relational expression in every case.

### 8.1.2 Explain the anatomy of a simple relational expression

In many applications, it is necessary to test the values of variables and then execute various statements based on the outcome. This option allows you to choose between different actions to take. Using the relational operators is the easiest and most typical technique to create such an expression. If the relation is true, it returns 1 whereas if the relation is false, it returns 0. Here is a code snippet of the relational expression.

int num1=7, num2=8;

cout << (num1<num2)<< endl;

When the code is compiled, the output will be a 1 to indicate true

### 8.1.3 List and explain the different C++ relational operators

We discussed and listed the relational operators in topic 6. For the purposes of reinforcement, we are going to list them again below:

= = Is equal to 8==10 gives false

!= not equal to 8!=10 gives true

> Greater than 8>10 gives false

< less than 8<10 gives true

>= Greater than or equal to 8>=10 gives false

<= Less than or equal 8<=10 gives true

Whenever we want to do conditional statements, we will have to make use of boolean or relational operators. Here is an example of equality(==) to compare strings

### 8.1.4 Identify valid relational expressions

The following are examples of valid relational expressions:

* age==19
* flag==done
* 2.0<3.0
* TAX<=0.15
* Marital\_status==’M’
* BMI>=24.9

### 8.1.5 Identify invalid relational expressions

The following are examples of invalid relational expressions in C++

* width=<50 Incorrect symbol used. The symbol should be <=
* 2.0>>3.0 Invalid relational operator
* flag = = done Spcaes not allowed
* age=>45 Invalid symbol used
* true <= false This will give an error as true and false are boolean types and we cannot use <= operator for comparing the boolean values.
* (height=1.8) Incorrect symbol. Used assignment operator (=) instead of equality operator (==)

### 8.1.6 Explain the numeric result that is generated from a- relational expression

When working with relational expressions, the outcome is a boolean value. The expression returns a 1 or a 0. A 1 represents true while a 0 represents false. The boolean integer can be converted into an alphabetic one, as we discussed earlier in topic 6. In such a case, we implement boolalpha as follows:

int num1=7, num2=8;

cout <<boolalpha<<(num1<num2)<< endl;

The code snippet above will return true. If, for any reason, we want to show the integer values, we can use the noboolalpha function.

### 8.1.7 Write C++ code that will save the result of relation expression that contains both or a mix of numeric variables and literals in an int variable

The program below compares a person’s age with the legal\_Age of majority, which is set to 18 and declared as a constant. The relational expression is evaluated, with the result stored in an int data type.

**EXAMPLE 8.1**

//program to check if a person's age is over the legal\_age of majority

#include <iostream>

using namespace std;

int main()

{

//declaring variables

double age=17.5;

//declaring constant

const int legal\_Age=18;

//evaluating a relational expression and storing answer in an integer

int result=(age>legal\_Age);

//displaying output

cout<<result<<endl;

return 0;

}

OUTPUT

0

We can always typecast the output and prefix it with a boolalpha statement so that the output can read as true for 1 or false for 0. The output statement is shown below:

cout<<boolalpha<<(bool)result<<endl;

8.1.8 Write C++ code that will save the result of relation expression that contains both or a mix of numeric variables and literals in a Boolean variable

In  example 8.1, our program stored the result as an int data type. This is not a problem because relational expressions are evaluated to yield a numerical result of 1 or 0. The idea is to check whether the expression results in true or false. However, C++ produces values of 1 or 0. In this program, we will use the same example but store the result in a bool data type, as shown in example 8.2.

**EXAMPLE 8.2**

//program to check if a person's age is over the legal\_age of majority

#include <iostream>

using namespace std;

int main(){

//declaring variables

double age=17.5;

//declaring constant

const int legal\_Age=18;

//evaluating a relational expression and storing answer in an integer

bool result=(age>legal\_Age);

//displaying output

cout<<boolalpha<<result<<endl;

return 0;

}

### 8.1.9 How the integer result produced by a relation expression relates to bool

When evaluating a relational expression, the outcome is boolean. However, C++ displays 1 for true and 0 for false. Using the boolalpha function can result in a 1 or 0 being displayed as true or false. This has been demonstrated in examples 8.1 and 8.2.

### 8.1.10 C++ code that will save the result of relation expression that contains both or a mix of character variables and literals in an int and bool variable

This section will show you how to store the expression's result in int and bool variables. The program used to explain the concept compares a user's age to the legal aid majority and also verifies citizenship status. If the user's age is greater than or equal to the legal age of majority and their citizenship status is 'C,' the program should print that they are eligible to vote; otherwise, it should display that they are not. Example 8.3 demonstrates the given scenario.

**EXAMPLE 8.3**

//program to check if a person's age is over the legal\_age

//and citizen status checked

#include <iostream>

using namespace std;

int main(){

char cit\_status;

bool isCitizen, isNotCitizen;

cout<<"enter citizen status [C=citizen or N=Non-citizen]"<<endl;

cin >>cit\_status;

isCitizen= cit\_status=='c' || cit\_status=='C';

isNotCitizen=cit\_status=='n' || cit\_status=='N';

//declaring variables

double age=27.5;

//declaring constant

const int legal\_Age=18;

//evaluating a relational expression and storing answer in an integer

int result=(age>legal\_Age);

if (result==isCitizen){

cout<<"You are eligible to vote"<<endl;

}

else if (result==isNotCitizen){

cout<<"You are NOT eligible to vote"<<endl;

}

else{

cout<<"Check cit\_status entry"<<endl;

}

return 0;

}

O**UTPUT**

**enter citizen status [C=citizen or N=Non-citizen]**

**c**

**You are eligible to vote**

**enter citizen status [C=citizen or N=Non-citizen]**

**N**

**You are NOT eligible to vote**

It is important to note that the first expression for age has the output stored in an int variable called result. The output of the citizenship status is stored in bool variables called isCitizen and isNotCitizen. The final expression is evaluated, with result being an int type and isCitizen/isNotCitizen being a bool type.**FORMATIVE ASSESSMENT 8.1 INDIVIDUAL TASK**

8.1.1 Define the term relational expression. (2)

8.1.2 List any FIVE examples of relational operators. (5)

8.1.3 Give THREE examples of invalid relational operators (3)

8.1.4 In C++, both ! and != are relational operators. [True /False] (1)

8.1.5 Evaluate the expression (7 + 8 <= 15) (2)

8.1.6 Evaluate the expression ('a' >'A') (2)

8.1.7 In C++, all relational operators are evaluated before logical operators [True/False] (2)

8.1.8 The result of a relational expression can not be assigned to an int type. [True/False]

8.1.9 What is the output of the following code?

if (6 > 8)

{cout << " \*\* " << endl ; }

else if (9 == 4) {

cout << "\*\*\*" << endl;}

else {

cout << "\*" << endl;

}

(2)

8.1.10 Consider the code below which is supposed to print PASS if y is equal to x.

#include <iostream>

using namespace std;

int main()

{

int x = 7;

int y = 10;

if(y = x) {

cout<< "PASS"<<endl;

}

return 0;

}

When compiled the program is giving pass even though its incorrect. Correct the program. (2)

8.1.11 Rewrite the following program to print either alphabetic answer True or False depending on the expression. Avoid use of if statement. (4)

**[Total 27 marks]**

# 8.2 Selection Statement

**Introduction**

Programming is more than just calculations and expressions. It is also necessary to include decision and control statements, which specify when statements should be executed. While we briefly discussed control statements in Topic 3, most of the code programs we've worked with so far are linear constructs, meaning that statements were executed sequentially or in the order in which they were declared. In this topic, you will learn in detail how to change the control flow of a program through branching statements. By implementing branching statements, you can determine whether one section of code will be executed or not based on a conditional clause. The term "selection statement" is also used to refer to branching statements, conditional statements, or decision-making statements in C++. We use selection statements to select parts of a program to be executed if a certain condition is met. Selection statements can be executed using:

1. if statements
2. if …. else ….
3. if...else if...else ……. (Multiple if statements)
4. if……{if ….{}} (Nested if statements)
5. switch statement

These statements will be discussed in the subsequent sections.

### 8.2.1 Explain the purpose of the if statement in C++.

An "if" statement evaluates a specific condition; if the condition is true, action is taken; otherwise, action is not taken. Action can be thought of as a statement or set of statements. Here is an example. Create a program that asks students for their arrival time at college. They receive house points if they arrive before the starting time. If the condition is met, house points are awarded; otherwise, no action is taken if the arrival time is later than the starting time.

Figure 8.1 depicts the given scenario using a flowchart.

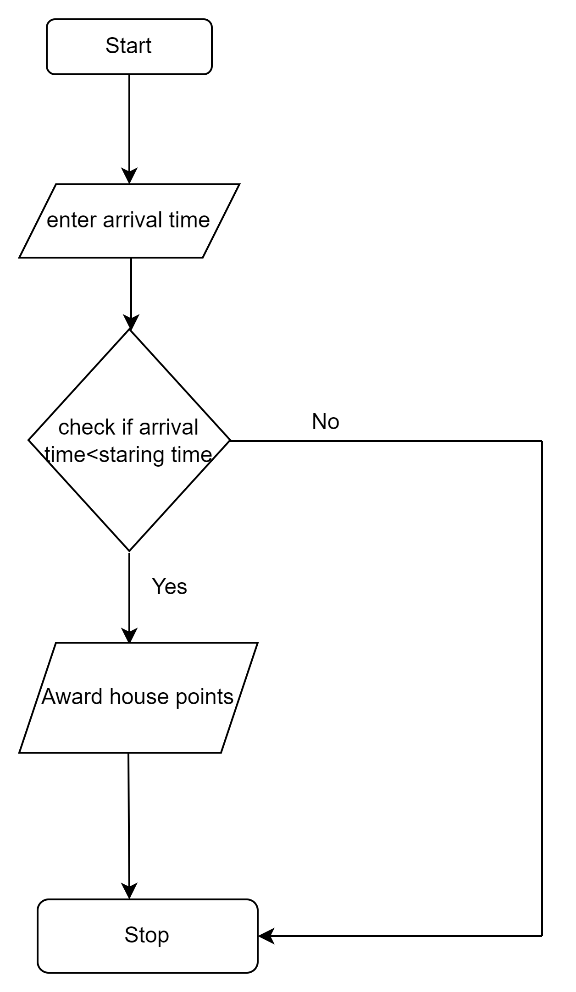


Figure 8. 1:Simple if statement flowchart

The if statement's syntax, or general form, is as follows:

if(expression)

{

statement(s);

}

The if statement evaluates the condition inside the parentheses ( ). Here, a statement may be a single statement, a block of statements, or nothing (in the case of an empty statement). The expression must be enclosed in braces {}.

* If the condition evaluates to true, i.e., a nonzero value, the code inside the body {…..} of if is executed. Otherwise
* If the condition evaluates to false, the code inside the body {….} of if is skipped.

Let us consider the following case where we need to apply the if statement.

**EXAMPLE 8.4**

**Write a program to accept an integer number and display if it’s a positive number.**

**Solution**

/\*Program to print positive number entered by the user

If the user enters a negative number, it is skipped \*/

#include <iostream>

using namespace std;

int main() {

int number=0;

//accepting a number

cout << "Enter an integer: "<<endl;

cin >> number;

//condition to check if the number is positive

if (number > 0) {

//statement to be displayed if condition is met

cout << number << " :is a positive number" << endl;

}

//optional

cout << "This statement is always executed.";

return 0;

}

**OUTPUT**

**Enter an integer:**

**63**

**63 :is a positive number**

**This statement is always executed since its outside the if block.**

**Enter an integer:**

**-365**

**This statement is always executed since its outside the if block**

Notice on the program, when we entered 63, the statement inside the braces was printed out. However, when we entered a negative value of -365, nothing was printed except the statement which is outside the if block. This is because we never instructed our program what to print if the condition was not met. Another example of an if statement is shown in Example 8.5.

**EXAMPLE 8.5**

Write a program to enter a number and print if it is even number.

#include <iostream>

using namespace std;

int main()

{

int num=0;

cout<<"Enter a number of your choice"<<endl;

cin>>num;

if(num%2==0){

cout<<num<< " is an even number." <<endl;

}

//optional

cout << "This statement is always executed since its outside the if block";

return 0;

}

**OUTPUT**

**Enter a number of your choice**

**77**

**This statement is always executed since its outside the if block**

**Enter a number of your choice**

**88**

**88 is an even number.**

**This statement is always executed since its outside the if block**

### 8.2.2 Explain the purpose of the else statement in C++

We discussed the simple if statement in section 8.2.1, which had one outcome based on the condition. If the condition is not satisfied, we note that the simple if statement code would just exit the if block. Often, it is desirable for a program to take one branch if the condition is true and another if it is false. This can be achieved by adding an else block to the simple if statement.

**Syntax**

 if(*boolean-expression* )  
       *statement-1*;  
   else  
       *statement-2*;

So we are going to modify the program to print even numbers. We want to say if the condition is not satisfied, the program prints "num" is a negative number." Figure 8.2 depicts the if... else... logic in the form of a flowchart.

A picture containing diagram

Description automatically generated

Figure 8. 2:if ...else....

So let us do the C++ code to check if a number is positive or negative.

**EXAMPLE 8.6**

//Program to print positive or negative number entered by the user

#include <iostream>

using namespace std;

int main() {

int number=0;//accepting a number

cout << "Enter an integer: "<<endl;

cin >> number; //condition to check if the number is positive

if (number > 0) {

//statement to be displayed if condition is met

cout << number << " :is a positive number" << endl; }

else{

cout << number << " :is a negative number" << endl; }

cout << "This statement is always executed.";

return 0;}

**OUTPUT**

**Enter an integer:**

**-78**

**-78 :is a negative number**

**This statement is always executed.**

**Enter an integer:**

**1235**

**1235 :is a positive number**

### 8.2.3 Determine the program flow when an if statement is encountered

As we can see from the examples in Section 8.2.2, selection statements are executed based on the fulfilment of conditions. The path that the program takes depends on the condition that has been met. If the condition is met, the statements that come after the "if" () {} statements are executed. If not met, the program will execute the statements which come after the else statements.

### 8.2.4 The general form for an if-else statement

The if-else is statement is an extended version of If. The general form of if-else is as follows:

if (test-expression)

{

True block of statements

}

else

{

False block of statements

}

Statements;

If the test-expression value is true in this sort of construct, the true block of statements will be performed. The false block of statements will be executed if the test-value expression is false. Nonetheless, upon execution, control will be immediately passed to the statements outside the If block.

**TASK 8.1:**

Write a C++ program to calculate the total cost of a product purchased in an electrical shop given the following conditions:

A shop will give discount of 10% if the cost of purchased quantity is more than 1000.  
The program must allow the user to enter the quantity of items.  
Suppose, one unit will cost R100-00.  
Display whether the customer qualifies for discount and print total cost for user.

### 8.2.5 Using relational expressions in if-else statements (Max 2 two nested levels only)

In some circumstances, using if...else is insufficient because another condition might exist that does not match any of the criteria stated in the if or else block. Let's examine the example to determine whether a number is positive or negative. The program will run the statements in the else block if we do not validate the input type during data entry. Here's an illustration in example 8.7.

**EXAMPLE 8.7**

//Program to print positive or negative number entered by the user

#include <iostream>

using namespace std;

int main() {

int number=0;

//accepting a number

cout << "Enter an integer: "<<endl;

cin >> number;

//condition to check if the number is positive

if (number > 0) {

//statement to be displayed if condition is met

cout << number << " :is a positive number" << endl;

}

else{

cout << number << " :is a negative number" << endl;

}

//optional

cout << "This statement is always executed.";

return 0;

}

**OUTPUT**

**Enter an integer:**

**kdfdsvgsda**

**0 :is a negative number**

**This statement is always executed.**

From the above example, we can see that the output is incorrect. The program used the statements in the else block. To solve the problem, we use if … else if… else block.

**Syntax**

if(condition1){

//code to be executed if condition1 is true

}else if(condition2){

//code to be executed if condition2 is true

}

else if(condition3){

//code to be executed if condition3 is true

}

...

else{

//code to be executed if all the conditions are false

}

To correct the program for checking positive or negative number using the if else if else block, the code will look as follows:

**EXAMPLE 8.8**

//Program to print positive or negative number entered by the user

#include <iostream>

using namespace std;

int main() {

int number=0;

//accepting a number

cout << "Enter an integer: "<<endl;

cin >> number;

//condition to check if the number is positive

if (number > 0) {

//statement to be displayed if condition is met

cout << number << " :is a positive number" << endl;

}

else if (number<0){

cout << number << " :is a negative number" << endl;

}

else{

cout << number << " :is an invalid entry for the program" << endl;

}

//optional

cout << "This statement is always executed.";

return 0;

}

**OUTPUT**

**Enter an integer:**

**uiryiu**

**0 :is an invalid entry for the program**

**This statement is always executed.**

**NOTE**

This rule applies whenever an if statement is followed by one or more else if statements; the final else if should be followed by an else statement. The requirement for a final else statement is defensive programming.

The else clause should either perform the necessary action or include a reasonable justification for why no action was performed.

**TASK 8.2**

Write a program in C++ to check triangle is an isosceles, equilateral, or scalene using if-else statement. The program must allow the user to enter the size of the three sides.

### 8.2.6 Using relational expressions in if-else statements with compounded content (Max 2 two nested levels only)

We have  been using mathematical expressions to explain if statements. Other expressions can be used if the expression evaluates to true or false. In this section, we are going to use characters using if-else if..... statements. A vowel is a sound such as the ones represented in writing by the letters 'a', 'e', 'i', ' o', and 'u', which you pronounce with your mouth open. A consonant is a speech sound that is articulated with complete or partial closure of the vocal tract. All non-vowel letters are consonants.

**EXAMPLE 8.9**

Write a program in C++ to Check if the input character is a vowel or a Consonant and display appropriate message. If the input is not an alphabetic character the program should also report and error.

SOLUTION

//Program to check whether input is a vowel or consonant

#include <iostream>

using namespace std;

int main() {

char chars;

bool isLowercaseVowel=false, isUppercaseVowel=false;

cout << "Enter an alphabet: "<<endl;

cin >> chars;

// values for lowercase vowels stored in a boolean variable

isLowercaseVowel = (chars == 'a' || chars == 'e' || chars == 'i' || chars == 'o' || chars == 'u');

// values for uppercase vowels stored in a boolean variable

isUppercaseVowel = (chars == 'A' || chars == 'E' || chars == 'I' || chars == 'O' || chars == 'U');

// show error message if c is not an alphabet

if (!isalpha(chars)){

cout<<"Error! The input is a non-alphabetic character."<<endl;

}

else if (isLowercaseVowel || isUppercaseVowel)

{

cout << chars << " is a vowel."<<endl;

}

else

{

cout << chars << " is a consonant."<<endl;

}

return 0;

}

**OUTPUT**

**Enter an alphabet:**

**A**

**A is a vowel.**

**Enter an alphabet:**

**g**

**g is a consonant.**

**Enter an alphabet:**

**5**

**Error! The input is a non-alphabetic character.**

In the above program, we made use of two boolean variables which will evaluate to true if the input is an uppercase or lowercase vowel. We also used the isalpha() function to test for alphabetic character input. If all these conditions are not met, then the input character is a consonant.

**Task 8.3**

Write a C++ program which displays the number of days in a month captured by the user. The user enters an integer to represent the month. January is 1—December is 12.

### 8.2.7 Explore the concept of block scope when writing if-else statements with compounded content

**VOCABULARY**

Block scope means a variable was defined within a block of code, such as a for loop or an if statement.

A block, or block statement (also known as a compound statement), is a collection of zero or more statements that the compiler treats as if it were a single statement. Blocks start with a symbol and terminate with a symbol, with the statements to be executed in between. Blocks can be used wherever a single statement is permitted.

When using if...else statements without braces for the statements, the true and false statements can only be one each; otherwise, if there are more than two, all of them will be performed. Below is an illustration of how to determine a person's eligibility to vote based on the age they entered.

**EXAMPLE 8.10**

#include <iostream>

using namespace std;

int main()

{

cout << "Enter your age (in years): ";

int age=0;

cin >> age;

if (age >=18)

cout << "You are old enough to vote"<<endl;

else

cout << "You are not old enough to be able to vote"<<endl;

cout << "Tough luck!! Try next time"<<endl;

return 0;

}

**OUTPUT**

**Enter your age (in years): 18**

**You are old enough to vote**

**Tough luck!! Try next time**

**Enter your age (in years): 12**

**You are not old enough to be able to vote**

**Tough luck!! Try next time**

Looking at the output of the above program, the code is not working as expected. The line which says "Tough luck!! Try next time" is printed in all cases whether the person is 18 and above or below. In the same way if we move it to the if block, the program will report a compilation error:

|error: 'else' without a previous 'if'|

To deal with these issues, statements which we want to be executed at the same time, we group them in the same block using braces. The code segment can be re-written as :

if (age >=18){

cout << "You are old enough to vote"<<endl;

}

else{

cout << "You are not old enough to be able to vote"<<endl;

cout << "Tough luck!! Try next time"<<endl;

}

Because blocks are regarded as a single sentence, this now works correctly:

Similarly, variables declared within a block are only available within that block. When the code snippet below is compiled, it will produce errors.

int age=0;

cin >> age;

if (age >=18)

{

int a=9;

cout<<a;

cout<<b;

}

else

{

cout<<a;

int b=5;

}

What are the errors. Discuss this with your teacher. Correct the code so that it can compile.

It is even possible to put blocks inside of blocks as shown in example 8.11.

**EXAMPLE 8.11**

#include <iostream>

using namespace std;

int main()

{ // block 1, nesting level 1

cout << "Enter an integer: ";

int value {};

cin >> value;

if (value > 0)

{ // block 2, nesting level 2

if ((value % 2) == 0)

{ // block 3, nesting level 3

cout << value << " is positive and even\n";

}

else

{ // block 4, also nesting level 3

cout << value << " is positive and odd\n";

}

}

return 0;

}

**OUTPUT**

**Enter an integer: 21**

**21 is positive and odd**

**Task 8.4**

A school has following rules for grading system:  
a. Below 25 - F  
b. 25 to 45 - E  
c. 45 to 50 - D  
d. 50 to 60 - C  
e. 60 to 80 - B  
f. Above 80 - A  
Write a C++ program to ask a user to enter marks and print the corresponding grade using if… else if…else statements. Make sure the data is validated.

# FORMATIVE ASSESSMENT 8.2 INDIVIDUAL TASK

8.2.1 Write the correct syntax of a simple IF statement in C++. (3)

8.2.2 Define the term selection/conditional statements. (2)

8.2.3 Explain the purpose of else statement in a selection statement. (2)

8.2.4 Write the general syntax of an if…else statement. (4)

8.2.5 Write a program which accepts three sides of a triangle. The program should first determine whether the sides constitute a valid triangle else print out an error message. If the sides are valid, determine what type of a triangle is it [scalene, isosceles, or equilateral triangle] (15)

8.2.6 Using if else statement, write a C++ program to to find maximum of three numbers. The program must allow user to enter the three numbers. Make sure the input is validated against incorrect type (10)

8.2.7 Write a C++ program to enter two Boolean numbers then, print phrase "A And B" if A and B equal to 1, or print phrase "A Or B" if A equal to 1 and B equal to 0. (10)

**[Total =46 marks]**

# 8.3 Logical Operators

Content:

Using logical operators

Earlier in topic 6 section 6.2, operators were covered extensively. We are now going to reinforce them and see how we implement them with selection statements.

Learning Outcomes:

Students should be able to

### 8.3.1 Define the term logic operator

Logical operators are used to check whether an expression is true or false. To determine whether an expression is true or false, logical operators are utilized. Logical operators are frequently used when writing test expressions that govern program execution. These expressions are also known as boolean expressions since they produce a boolean value or answer when evaluated, and the result is stored in a flag variable.

### 8.3.2 List the different logic operators

 There are three common logical operators that give a Boolean value by manipulating other Boolean operand(s) in C++ are:

1. && Logical AND
2. || Logical OR
3. ! Logical NOT

Other languages such as C#, Java, JavaScript and Swift do use the exact symbols for the same operators.

### 8.3.3 Explain how each logic operator will influence the result of an expression

Table 8. 1: Logical Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Meaning** | **Example** |
| && | Logical AND. Returns true only if all the operands are true. | (1&&0) returns 0 (false) |
| || | Logical OR. True if at least one of the operands is true. | (1||0) returns 1 (false) |
| ! | Logical NOT. True only if the operand is false | !(1) returns 0 |

\* Remember, we discussed earlier in topic 7 that we can prefix the expression with the function boolalpha if we want to return the actual words like true or false.

Because the logical operators have a limited domain (input) and range (output), we can easily enumerate all conceivable combinations of inputs and outputs. A truth table is commonly used to summarize all potential combinations.

**Logical AND**

Table 8. 2: Logical AND truth table

|  |  |  |
| --- | --- | --- |
| **x** | **y** | **x&&y** |
| false | false | false |
| false | true | False |
| true | false | False |
| true | true | true |

Table 8.2 shows the evaluation of expressions when using a logical AND operator.

Here are some examples of expressions with logical operators. Observe how each influences the result of the expression.

num1 =25

num2 =37

(num1>10) && (num2 >30) evaluates to true because:

25 is greater than 10 so returns a 1

37 is greater than 30 so return a 1

1 and 1 is the same as (true && true) which results to true.

**Logical OR**

Here is a truth table to represent the logical OR expression.

Table 8. 3: Logical OR truth table

|  |  |  |
| --- | --- | --- |
| **x** | **y** | **x||y** |
| false | false | false |
| false | true | true |
| true | false | true |
| true | true | true |

Table 8.3 shows the evaluation of expressions when using a logical OR operator.

We will utilize logical operators to evaluate the following expressions using the same values of num1 and num2 as 25 and 37, respectively:

(num1>10) || (num2<30) evaluates to true because:

25 <10 returns 1(true), 37<30 returns 0 (false),

1OR 0 evaluates to 1.

Here is another example:

(num1<10) || (num2<30) evaluates to 0 because

Expression 1 returns 0, expression 2 returns 0, 0 OR 0 returns 0.

**Logical NOT**

Table 8. 4: Logical NOT truth table

|  |  |
| --- | --- |
| **x** | **!x** |
| false | true |
| true | false |

As seen in table 8.4, logical NOT returns true if the expression's value is false and false if the expression's value is true..

!(1) returns 0 (false)

!(0) returns (true)

Along the same lines, the following statement evaluates to false

cout << !('b') << endl;

This is because the char value is converted to its ASCII value. So not any number which is not 1 is 0.

cout << !(89) << endl; will evaluate to 0.

Combining logical AND and logical OR operators in the same statement is often unavoidable, but it is a risky domain. Many programmers assume that logical AND and logical OR have the same precedence as addition/subtraction and multiplication/division. Nevertheless, logical AND has a higher precedence than logical OR, so logical AND operators will be assessed ahead of logical OR operators (unless they have been parenthesized).

### 8.3.4 Write C++ code that will save the result of relation expression that contains both or a mix of numeric variables, character variables and literals and a single logical operator in an int variable

We can evaluate relational expressions with numeric values and characters. What will happen is that the program will convert the char to its ASCII value and then evaluate the expression.  For example:

(65>'#') will evaluate to 1(true). This is so because the value of # is 35 on ASCII table

(65>'H') will evaluate to false because H is 72 on ASCII table. So 67>72 is false.

We can add a binary literal as follows:

(65>'H')&& 0b11; This would evaluate to 0 (false) simply because:

65>72 is 0(false), 0b11 is a binary literal representing 3 and we already mentioned that a number not 0 is a 1 when using logical expression.

So ultimately, 0 && 1 is 0 which is false.

We can make use of an int variable to then store the result from the expression as shown below:

int result;

result=(65>'#');

cout << result << endl;

The output of the above code snippet will be 1. There is a downside to this. The program still returns a 1 when you use boolalpha to convert the output to boolean. However, if we had declared the result as bool, it would have worked as we prefer. We will discuss the boolean result in section 8.3.5. Example 8.12 code demonstrates using int variables to store result of an expression with characters.

**EXAMPLE 8.12**

Write a program to check whether a character is a digit or not. Store the result in an int data type.

**SOLUTION**

//C++ program to check for decimal digit characters using conditional operator

#include <iostream>

using namespace std;

#include <stdio.h>

int main() {

char character;

int is\_Digit;

cout<<"Enter a Character"<<endl;

cin>>character;

/\* Check, If input character is digit \*/

is\_Digit = ((character >= '0') && (character <= '9'))? 1 : 0;

if(is\_Digit == 1)

cout<<character<< " is a digit"<<endl;

else

cout<<character<< " is NOT a digit"<<endl;

return 0;

}

### 8.3.5 Write C++ code that will save the result of relation expression that contains both or a mix of numeric variables, character variables and literals and a single logical operator in a bool variable

As previously stated, the result of a logical expression is of the boolean type. The outcome is either true or false, making it easy to store and manipulate if boolean data type variables are used to store the outcome. Let us look at the previous example that we had in section 8.3.4, but this time we are going to store the output in a bool variable:

bool result;

result=(65>'#');

cout << result << endl;

If we run the code snippet above, we get our output as 1. Now, we can change it to its alphabetic representation which is true by using boolalpha. The code will look as follows:

**EXAMPLE 8.13**

#include <iostream>

using namespace std;

int main()

{

bool result;

result=(65>'#');

cout <<boolalpha<< result << endl;

return 0;

}

OUPUT

true

Remember, in the previous example where we stored the result in an int type, we mentioned that we will not be able to see the alphabetic equivalent when we use boolalpha.

### 8.3.6 Saving the result of relation expression that contains both or a mix of numeric variables, character variables and literals and multiple

In the following program, we save the range of vowels in a bool variable. The user's characters will be compared to the vowel range mentioned. Take notice that we stored uppercase A, B, and U as ASCII number equivalents and then typecasted the numbers to create char data values. To check whether the character entered is a vowel (both lowercase and uppercase), we then put the two bool variables through a logical OR operator. Example 8.14 presents the code:

**EXAMPLE 8.14**

//C++ program to check if character is a vowel or not

#include <iostream>

using namespace std;

int main()

{

char chars;

cout<<char(101)<<endl;

bool isLowercaseVowel=false, isUppercaseVowel=false;

cout << "Enter an alphabet: "<<endl;

cin >> chars;

// values for lowercase vowels stored in a boolean variable

isLowercaseVowel = (chars == 'a' || chars == char(101) || chars == char(105) || chars == 'o' || chars == 'u');

// values for uppercase vowels stored in a boolean variable

isUppercaseVowel = (chars == char(65) || chars == char(69) || chars == 'I' || chars == 'O' || chars == char(85));

if (isLowercaseVowel || isUppercaseVowel){

cout << chars<<" is a vowel "<< endl;

}

else{

cout << chars<<" is a not vowel "<< endl;

}

return 0;

}

**OUTPUT**

**Enter an alphabet:**

**A**

**A is a vowel**

**Enter an alphabet:**

**K**

**K is a not vowel**

### 8.3.7 Write C++ code that will save the result of relation expression that contains both or a mix of numeric variables, character variables and literals and multiple

Remember, only one logical operator can be used to combine two relations. However, multiple relations can be combined into a complex logical expression. Regardless of the number of relations and logical operators used to build a logical expression, the result is either true, false, or indeterminate because of missing values.

**Short circuit evaluation**

For the logical AND to yield true, both operands must evaluate to true. If the first operand evaluates to false, the second operand will return false regardless of whether it evaluates to true or false. In this case, the logical AND operator will immediately return false, without even examining the second operand! This is referred to as short circuit evaluation.

Similarly, if the first operand of a **logical OR** condition is true, the entire OR condition must evaluate to true, and the second operand is ignored.

Consider the following program which uses three logical operators.

**EXAMPLE 8.15**

#include <iostream>

using namespace std;

int main(){

int a=5;

int b=3;

char c='A';

int d=0b01;

int answer=((a>3)&&(a>10) || (5<!(d)) && ((int)c<b) );

cout<<answer<<endl;

return 0; }

Output

0

# FORMATIVE ASSESSMENT 8.3 INDIVIDUAL TASK

8.3.1 Suppose P and Q are logical expressions. The logical expression P && Q is true if both P and Q are true. [True/False] (1)

8.3.2 What is a logical operator. (2)

8.3.3 List THREE examples of logic operators. (3)

8.3.4 What do you understand by the term flag variable. (2)

8.3.5 Suppose that score is a variable of type double. Write the C++ code snippet that increases the score by 5 marks if score is between 80 and 90. (3)

8.3.6 Write a C++ statement that prints true if x is an odd number and positive. (3)

**[Total =14 Marks]**

# 8.4 Selection Statements

When we have complex assertions to examine, one of them must meet a specified condition before the second condition is evaluated. In such situations, the nested if statement is used..

### 8.4.1 Define the term nested if statement

**VOCABULARY**

Nested if in C++ is using more than one if statements in the same scope. The if statement is a decision-making statement that allows taking decisions based upon the condition specified.

### 8.4.2 Determine the application flow when a nested if statement is encountered

 When there are multiple conditions that are interdependent, a nested if statement can be used. The nested if statement checks multiple conditions sequentially. A nested if statement is formed by using one if statement inside another. The inner if statement will only execute when it’s outer if statement is true. If the first condition is true, we go into the next if condition and the subsequent condition is checked until we get a false condition, and the checking stops. In C++ there is no limit to the levels of nesting for if or if else statement.

### 8.4.3 Identify or correct the general form for a nested if-else statement

Figure 8.3 illustrates the logic path for nested if statements. In the diagram, the first if condition will be checked first, and if it is false, the program will exit the first if block and go to the next set of statements after the first if block. If the first if condition is true, then the program will go into the body of the first if. In the body of the first if condition, we just added a statement that can be printed out and then have a second if condition. This condition will now be checked, and then the block will be executed. If the second if a condition is false, the second if block is skipped, and the program proceeds to the statements following the second if block. Similarly, when we have more than two nested if statements, the program will go inside as long as the consecutive if statements are true.

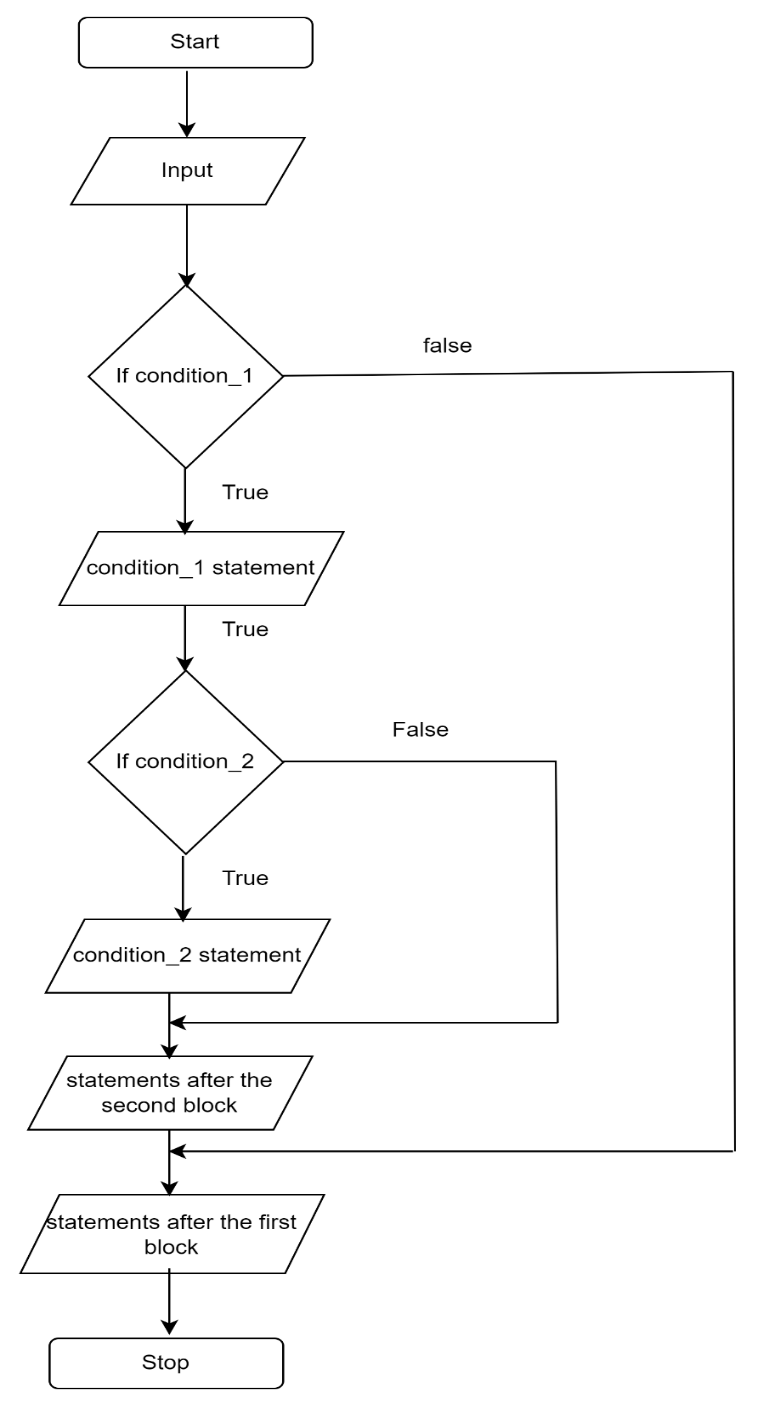


Figure 8. 3: Nested If statement logic path

**Syntax of Nested if statement**

// Outer if conditionif (condition 1) {   **// inner if condition**  
   if (condition 2) {       // Block of Code and Statements  
   }  
   **// inner else condition**  
   else {       // Block of Code and Statements  
   }  
}  
**// Outer else statement**  
else {   **// inner if condition**  
   if (condition 3) {  
       // Block of Code and Statements  
   }  
   **// inner else condition**  
   else {  
       // Block of Code and Statements  
   }  
}

**Let us start with a simple program**

We want to write a program to enter two marks. If the first mark is greater than 50 and you check if the second mark is greater than 50 to calculate the average and print "you qualify for an award".

**EXAMPLE 8.16**

#include<iostream>

using namespace std;

int main()

{

// Declare and assign values to variables

int first\_mark= 0;

int second\_mark = 0;

cout<<"Enter the first mark" <<endl;

cin>>first\_mark;

cout<<"Enter the second mark" <<endl;

cin>>second\_mark;

if (first\_mark >=50) { // first if condition : variable first\_mark is checked

cout << " value of firstmark is: " << first\_mark << endl;

if ( second\_mark>=50) { // second if condition : variable second\_mark is checked

cout << " value of second mark is: " << second\_mark << endl;

int average= (first\_mark+second\_mark)/2;

cout<<"You got an average mark of "<< average<< "You have qualified for an award"<<endl;

}

}

return 0;

}

**OUTPUT**

**Enter the first mark**

**89**

**Enter the second mark**

**79**

**value of first mark is: 89**

**value of second mark is: 79**

**You got an average mark of 84You have qualified for an award**

**Enter the first mark**

**75**

**Enter the second mark**

**23**

**value of firstmark is: 75**

**You failed your second test**

**Enter the first mark**

**23**

**Enter the second mark**

**24**

In the preceding example, the average is calculated only if the first mark is greater than 50 and the second mark is greater. Nothing is printed if the first mark is less than 50, as in the previous entry. We may extend the code by adding otherwise statements to specify what occurs if the first mark is less than 50, as seen below.

**EXAMPLE 8.17**

#include<iostream>

using namespace std;

int main()

{

// Declare and assign values to variables

int first\_mark= 0;

int second\_mark = 0;

cout<<"Enter the first mark" <<endl;

cin>>first\_mark;

cout<<"Enter the second mark" <<endl;

cin>>second\_mark;

if (first\_mark >=50) { // first if condition : variable first\_mark is checked

cout << " value of firstmark is: " << first\_mark << endl;

if ( second\_mark>=50) { // second if condition : variable second\_mark is checked

cout << " value of second mark is: " << second\_mark << endl;

int average= (first\_mark+second\_mark)/2;

cout<<"You got an average mark of "<< average<< "You have qualified for an award"<<endl;

}

cout << "You failed your second test" << endl;

}

else{

cout << "You first mark is below 50%" << endl;

}

return 0;

}

**OUTPUT**

**Enter the first mark**

**12**

**Enter the second mark**

**12**

**You first mark is below 50%**

### 8.4.4 Write C++ code that will use relational expressions in nested if-else statements

We'll employ relational expressions in nested if-else statements next. We're going to change the program that checks for vowels and consonants.

**Example 8.18**

Here we are reading a character from the user and validating that it is valid alphabet or not, if the character if valid alphabet then we are validating it is vowel or not and printing the appropriate message for the input character.

//EXAMPLE of Nested if else

//Read a character a check whether it is VOWEL or CONSONANT

#include<iostream>

using namespace std;

int main()

{

char ch;

//reading a character

cout<<"Enter an alphabet: ";

cin>>ch;

//condiion to check character is alphabet or not

if( (ch>='A' && ch<='Z') || (ch>='a' && ch<='z'))

{

//conditions to check character is VOWEL or not

if( ch=='A' || ch=='a' || ch=='E' || ch=='e' || ch=='I' || ch=='i' || ch=='O' || ch=='o' || ch=='U' || ch=='u')

cout<<"\""<<ch<<"\" is a VOWEL"<<endl;

else

cout<<"\""<<ch<<"\" is a CONSONANT"<<endl;

}

else

{

cout<<"\""<<ch<<"\" is not an alphabet\n";

}

return 0;

}

In example 8.18, we used a boolean variable to check lowercase and uppercase something that has been handled with the use of logical operators to accommodate both cases.

### 8.4.5 Write C++ code that will use relational expressions in nested if-else statements with compounded content

A university has a policy for sponsoring researchers to attend conferences to present their papers. For a researcher to qualify for funding for international conferences, he or she must have accumulated 0.75 or more research points in the previous year. The employment status is checked if the condition- is met. If a researcher is permanently employed, he/she qualifies for funding to travel to an international conference. If employment status is contract, the researcher qualifies for international travel funding if the duration of the contract is equal to or more than 3 years; otherwise, he or she qualifies for local travel. Otherwise, the researcher qualifies ONLY for funding to attend local conferences. The solution of this scenario is given in example 8.19.

**EXAMPLE 8.19**

// C++ Program to Nested-if conditions

//check qualification for funding to attend conferences

#include <iostream>

using namespace std;

int main(){

double research\_units=0;

char emp\_status;

int contact\_duration;

//Entering input

cout<<"Enter research units accumulated in the previous year"<<endl;

cin>>research\_units;

cout<<"Enter your work status [c=contract, p=permanent]"<<endl;

cin>>emp\_status;

if (research\_units>=0.75) {

if (emp\_status =='p' || emp\_status =='P') {

if (true) {

cout << "Congratulations!! You qualify for international travel funding" << endl;

}

else {

cout << " You ONLY qualify for local travel funding" << endl;

}

}

else if (emp\_status=='c' || emp\_status=='C'){

cout<<"Enter your contract duration in years [0-9]"<<endl;

cin>>contact\_duration;

if (contact\_duration>=3){

cout << "Congratulations!! You qualify for international travel funding" << endl;

}

else{

cout << " You ONLY qualify for local travel funding" << endl;

}

}

else{

cout << "Invalid employment status" << endl;

}

}

else{

cout << "Unfortunately you ONLY qualify for local travel funding" << endl;

}

return 0;}

**OUTPUT**

**Enter research units accumulated in the previous year**

**.9**

**Enter your work status [c=contract, p=permanent]**

**c**

**Enter your contract duration in years [0-9]**

**2**

**You ONLY qualify for local travel funding**

**Enter research units accumulated in the previous year**

**1.2**

**Enter your work status [c=contract, p=permanent]**

**P**

**Congratulations!! You qualify for international travel funding**

So, the program can now assist the university in deciding when to assist researchers with international conference funding or local conference funding.

Task: The program above is not validated for input entries. If you enter for example characters for number of research\_units, the program still prints that

"Unfortunately, you ONLY qualify for local travel funding"

Validate the two entries: research units and emp\_status for correct data type.

### 8.4.6 Using relational expression containing logic operators in nested if-else statements

A European country has embarked on a massive recruitment of its youths into a national youth service program. The requirements are that you should be at least 18 years old but less than or equal to 45. If age is outside the range, the program will display a message:

“Your age is outside the acceptable range"

Further to this, you should be a male and your BMI must be between 18.5 and 24.9. If your BMI is outside the range, you will be referred to a physician. However, for female candidates, they are given a preference to skip the national service and go straight for an internship.

**EXAMPLE 8.20**

#include <iostream>

using namespace std;

int main(){

//declaring variables

int age=0;

double BMI=0;

char gender;

//accepting input

cout<<"Enter age"<<endl;

cin>>age;

cout<<"Enter your gender"<<endl;

cin>>gender;

cout<<"Enter your BMI"<<endl;

cin>>BMI;

//validation of data type input

if (cin.fail())

{

cout<<"entry error"<<endl;

cin.clear();

cin.ignore(1000, '\n');

return 0;

}

//Nested if statements to check conditions

if (age >=18 && age<=45) {

if (gender=='M' || gender =='m') {

if ( BMI <=24.9 && BMI >=18.5) {

cout << "You must go for a national service" << endl;

}

else{ cout << "We are going to be referred to a physician" << endl;}

}

else if (gender=='F' || gender =='f'){

cout << "You are allowed to skip national service" << endl;

}

else{

cout << "You entered invalid gender" << endl;

}

}

else {

cout << " Your age is outside the acceptable range"<<endl;

}

return 0;

}

**OUTPUT**

**Enter age**

**11**

**Enter your gender**

**m**

**Enter your BMI**

**20**

**Your age is outside the acceptable range**

**Enter age**

**20**

**Enter your gender**

**M**

**Enter your BMI**

**11**

**We are going to be referred to a physician**

### 8.4.7 Write C++ code that will use relational expressions containing logic operator in nested if-else statements with compounded content (Max 3 levels, Max 3 logic

### operators per level)

At a school, the SRC wants to conduct some elections to choose its representatives for the academic year. There have been some cheating allegations in the past few years. They have asked the programming class to develop a system to assist in this regard. Only students in grades 8 through 11 will be able to vote. The grade must be authenticated first.

They put up a condition each grade as follows:

Table 8. 5::SRC Voting Venues

|  |  |  |  |
| --- | --- | --- | --- |
| **Grade** | **Identifiable Grouping** | **Voting Room** | **Accepted Age Groups** |
| Grade 8 | Male | Room 5 | 14 to less than 15 |
| Female | Room 6 | 14 to less than 15 |
| Grade 9 | Male | Room 7 | 15 to less than 16 |
| Female | Room 8 | 15 to less than 16 |
| Grade 10 | Male | Room 9 | 16 to less than 17 |
| Female | Room 10 | 16 to less than 17 |
| Grade 11 | Male | Room 3 | 17 to less than 18 |
| Female | Room 4 | 17 to less than 18 |

Write a C++ program to assist the campaign managers in directing the voters.

**EXAMPLE 8.21**

//Voter verification and venue allocation system

#include<iostream>

using namespace std;

int main()

{

char gender;

int grade;

double age;

cout<<"\nEnter Your grade : "<<endl;

cin>>grade;

if(grade==8)

{

cout<<"\nEnter Your Gender : ";

cin>>gender;

cout<<"\nEnter Your age in years : ";

cin>>age;

if (cin.fail()){

cout<<"Entry not recognised"<<endl;

return 0;

}

if((gender=='M' || gender=='m') && (age>=14 || age<15))

{

cout<<"\nGo To Room-5";

}

else if((gender=='F' || gender=='f') && (age>=14 || age<15))

{

cout<<"\nGo To Room-6";

}

else

{

cout<<"\n Invalid range of input. Please try again";

}

}

else if(grade==9)

{

cout<<"\nEnter Your Gender : ";

cin>>gender;

cout<<"\nEnter Your age in years : ";

cin>>age;

if (cin.fail()){

cout<<"Entry not recognised"<<endl;

return 0;

}

if((gender=='M' || gender=='m') && (age>=15 && age<16))

{

cout<<"\nGo To Room-7";

}

else if((gender=='F' || gender=='f')&& (age>=15 && age<16))

{

cout<<"\nGo To Room-8";

}

else

{

cout<<"\n Invalid range of input. Please try again";

}

}

else if(grade==10)

{

cout<<"\nEnter Your Gender : ";

cin>>gender;

cout<<"\nEnter Your age in years : ";

cin>>age;

if (cin.fail()){

cout<<"Entry not recognised"<<endl;

return 0;

}

if((gender=='M' || gender=='m') && (age>=16 && age<17))

{

cout<<"\nGo To Room-9";

}

else if((gender=='F' || gender=='f') && (age>=16 && age<17))

{

cout<<"\nGo To Room-10";

}

else

{

cout<<"\n Invalid range of input. Please try again";

}

}

else if(grade==11)

{

cout<<"\nEnter Your Gender : ";

cin>>gender;

cout<<"\nEnter Your age in years : ";

cin>>age;

if (cin.fail()){

cout<<"Entry not recognised"<<endl;

return 0;

}

if((gender=='M' || gender=='m') && (age>=17 && age>=18))

{

cout<<"\nGo To Room-3";

}

else if((gender=='F' || gender=='f') && (age>=17 && age<18))

{

cout<<"\nGo To Room-4";

}

else

{

cout<<"\n Invalid range of input. Please try again";

}

}

else{

cout<<"Voter verification failed. Please consult the polling officer"<<endl;

}

return 0;

}

**TASK 8.5**

Write a C++ program to check that the entered character is small, capital or a special character. The program must allow the user to enter any character and then display the ASCII value and the type of character it is.

**Hint:**

**Small letters**

Ascii\_value>=97 && Ascii\_value<=122

**Capital Letter**

 Ascii\_value>=65 && Ascii\_value<=90

**Special Characters**

Ascii\_value>=0 && Ascii\_value>=47 || Ascii\_value>=54 && Ascii\_value<=64 || Ascii\_value>=91 && Ascii\_value<=96 || Ascii\_value>=123 && Ascii\_value<=127

So far we have demonstrated how to implement select/conditional statements using if, if …else… and nested if statements. There is also Switch Case.

A switch statement checks a variable for equality against a set of values. Each value is referred to as a case, and the variable is tested for each case.

**Syntax for Switch Case**

// switch-case

switch ( selector ) {

case value-1:

block-1; break;

case value-2:

block-2; break;

case value-3:

block-3; break;

......

case value-n:

block-n; break;

default:

default-block;

}

Example: Write a program in C++ to do arithmetic operations based on the operator selected.

**EXAMPLE 8.21**

//Using switch for arithmetic operators

#include <iostream>

using namespace std;

int main(){

char oper; int num1, num2, result;

cout<<"Enter first number"<<endl;

cin>>num1;

cout<<"Enter second number"<<endl;

cin>>num2;

cout<<"Enter the operator [+, -, / \*]"<<endl;

cin>>oper;

if (cin.fail()){

cout<<"Error in input"<<endl;

}

switch (oper) {

case '+':

result = num1 + num2;

break;

case '-':

result = num1 - num2;

break;

case '\*':

result = num1 \* num2;

break;

case '/':

result = num1 / num2;

break;

default:

cout << "Unknown operator" << endl;}

cout<<num1<< oper<<num2<< " = "<<result<<endl;

return 0; }

**OUTPUT**

**Enter first number**

**45**

**Enter second number**

**33**

**Enter the operator [+, -, / \*]**

**+**

**45+33=78**

# FORMATIVE ASSESSMENT 8.4 INDIVIDUAL TASK

8.4.1 Define the term nested if statement. (2)

8.4.2 Using a flowchart, show the general form of a nested if statement. (5)

8.4.3 What will be the output of the code below:

#include <iostream>

using namespace std;

int main (){

int x = 35;

int y = 45;

int z;

if (x > y){

z = x + y;

}

else

{

z =y-x;

}

cout << x << " " << y << " " << z << endl;

return 0;

}

(2)

8.4.4 In nested if statement which statement evaluated first ?

Explain your answer. (3)

8.4.5 How many levels of nested if statement is allowed in C++. (1)

**[Total =13 Marks]**

# SUMMATIVE ASSESSMENT 8.4 INDIVIDUAL TASK

8.5.1 Explain how the integer result produced by a relation expression relates to bool true/false (2)

8.5.2 Write a C++ program to check whether the given angles constitute a valid/not valid triangle. (10)

8.5.3 What will be the output of the following code in C++. Explain your answer. (3)

int result='a'<'A';

8.5.4 Write the syntax for nested if..else if statements. (4)

8.5.5 Business require a program to determine whether they have made a profit or loss from the sales that they do on a day-to-day basis. Write a C++ program to enter cost price and selling price and find profit or loss. Consider the following scenario:

If Selling Price > Cost Price  
Profit = Selling Price - Cost Price

If Selling Price < Cost Price  
Loss = Cost Price - Selling Price

If Selling Price = Cost Price  
No Profit .. No Loss

(10)

8.5.6 The nested conditional statement shown below has been written by an inexperienced C++ programmer. The behaviour of the statement is not correctly represented by the formatting.

if (n < 10)

if (n > 0)

cout << "The number is positive." << endl;

else

cout << "The number is \_\_\_\_\_\_\_\_\_\_\_\_\_\_." << endl;

return 0;

* 1. What is the output of the statement if the variable n has the value 7 ?

If n has the value 15 ?

If n has the value -3 ? (6)

8.5.7 Answer the questions below concerning the following fragment of code.

int n;

cout << "Enter an integer: ";

cin>> n;

if (n < 10)

cout << "less than 10" << endl;

else if (n > 5)

cout << "greater than 5" << endl;

else

cout << "not interesting" << endl;

1. What will be the output of the fragment above if the interactive user enters the integer value 0 ? (2)
2. What will be the output of the fragment above if the interactive user enters the integer value 15 ? (2)
3. What will be the output of the fragment above if the interactive user enters the integer value 7 ? (2)
4. What values for n will cause the output of the fragment above to be "not interesting"? (2)

**[Total =43 Marks]**

# References

Bjarne, S., 2013. The C++ Programming Language.-4th

Grimes, R., 2017. *Beginning C++ Programming*. Packt Publishing Ltd.

Oualline, S., 2003. *Practical C++ programming*. " O'Reilly Media, Inc.".

<https://www.programiz.com/cpp-programming>

<https://www.javatpoint.com/cpp-tutorial>

<https://www.geeksforgeeks.org/cpp-tutorial/>

# Topic 9: Repetition control structure

After you have completed this module, you should be able to :

* Define the term pre-test loop and entrance-controlled loop
* Define the term fixed-count loop
* Define the term variable condition loop
* Explain the purpose of the while statement in C/C++
* Determine the application flow when a while statement is encountered
* Identify or correct the general form for a while statement
* Write C/C++ code that will use relational expressions in while statement. (Max 2 levels only)
* Write C/C++ code that will use relational expressions in while statements with compounded content (Max 2 two nested levels only)
* Explore the concept of block scope when writing while statements with compounded content
* Explain the difference between fixed-count loop and variable condition loop
* Write C/C++ code that will use relational expressions in while statements with interactive input (Max 2 two nested levels only)
* Explain what a break statement is used for
* Explain what the continue statement is used for
* Write C/C++ code that will use relational expressions in while statements making use of the break statement
* Write C/C++ code that will use relational expressions in while statements making use of the continue statement
* Write C/C++ code that will use relational expression containing logic operators in while (Max 3 level, Max 3 logic operators per level)
* Write C/C++ code that will use relational expressions containing logic operator in while loop with compounded content (Max 3 level, Max32 logic operators per level)
* Define the term post-test loop and exit controlled loop
* Determine the application flow when a do while statement is encountered
* Identify or correct the general form for a do while statement
* Define the term sentinel
* Explain where and for what reason a sentinel is used
* Write C/C++ code that will use relational expression containing logic operators in a do while loop (Max31 level, Max 3 logic operators per level)
* Write C/C++ code that will use relational expressions containing logic operator in a do while loop with compounded content (Max 3 level, Max 3 logic operators per level)
* Write C/C++ code that will use relational expression containing logic operators in a sentinel controlled do while loop (Max 3 level, Max 3 logic operators per level)
* Write C/C++ code that will use relational expressions containing logic operator in a sentinel controlled do while loop with compounded content (Max 3 level, Max 3 logic operators per level)
* Define the term variable condition loop
* Explain how the, for statement differs from the while
* Explain how the, for statement is similar to the while
* Identify or correct the general form for a for statement
* Explain the purpose of each part in the for-loop initialisation
* Explain what effect an empty semicolon will have on the for loop
* Explain the term infinite loop
* Write C++ code to create an infinite for loop.
* Use the break statement to end an infinite for loop based on the loop reaching a condition.
* Determine the application flow when for statement is encountered.
* Write C++ code that will use a for loop with a predetermined number of loops.
* Write C++ code that will use a for loop with a non-sequential counter variable.
* Define the term nested loop
* Define the terms inner and outer loops
* Write C++ code that will nest identical type loops (Max 3 levels)
* Write C++ code that will nest different type loops (Max 3 levels)
* Determine the application flow when nested loop statement is encountered
* Explain what a break statement is used for
* Explain what the continue statement is used for.
* Implement a break or continue statement as required as part of a code solution.
* Plan, Design, create and code a C++ solution incorporating various structures and nested strictest to solve a given problem.

# 9.1 While Repetition control

So far, we have discussed sequential and selection structures in programming. In this chapter, we are going to focus on repletion or looping. The other term used in place of these two is iteration. This construct is used when a programmer needs to have a specific action repeated several times. Instead of writing several statements, the programmer can instruct the code to change values as long as the action is the same. Loops are useful because they save time, reduce errors, and improve code readability. In this chapter, we are going to focus on while…loop, do…while loop and for…loop. We are also going to look at statements that can alter the loop structure, and this includes the break and continue statements.

### 9.1.1 Define the term pre-test loop and entrance-controlled loop

A pre-test loop is a loop in which the condition is checked first before executing any statement inside the body of the loop. Another term used is entry-controlled loop. For anything inside the loop to happen, it is controlled at the entry point of the loop structure. The condition in a loop must evaluate to a Boolean value i.e., True or False. If the condition is true, then the statements inside the loop will be executed. If condition turns false, the statements inside the body of the loop will never be executed. We are going to illustrate a pretest loop using a flowchart as shown in Figure 9.1.

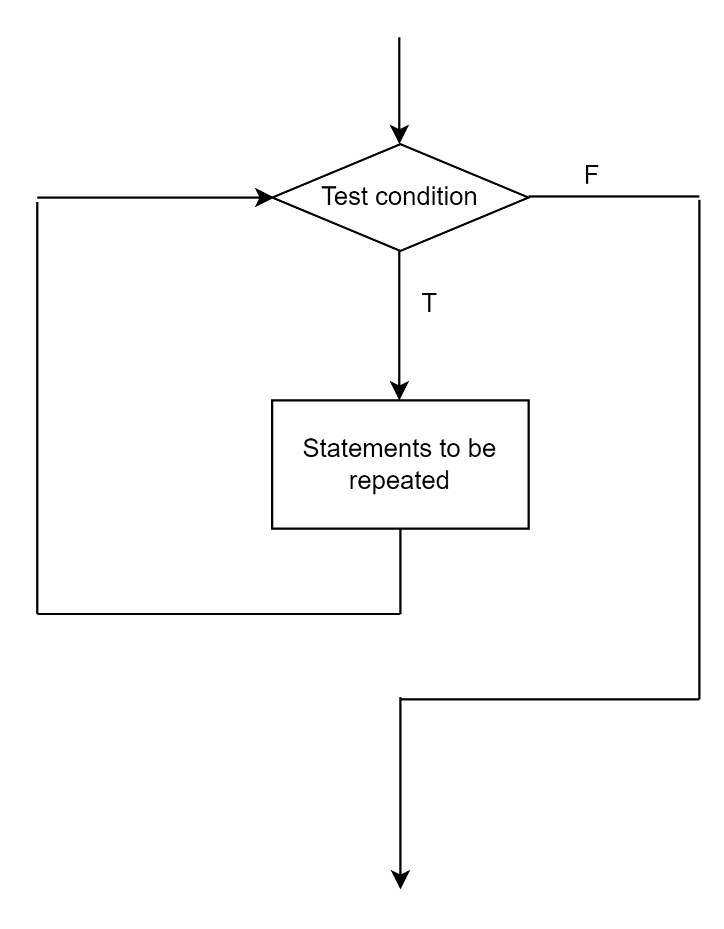


Figure 9. 1: Pretest Loop

As shown in Figure 9.1, the condition is checked first. If the condition is true, statements will be executed until the condition is false. When the condition becomes false, the program exits the loop structure. If the condition is not met from the start, none of the loop body statements will be executed. The loop body will be skipped by the program. An example of a pretest loop in C++ is the while …loop and a for..loop structure.

### 9.1.2 Define the term fixed-count loop

A fixed-count loop is a loop that allows a statement to be repeated a "fixed" number of times as specified by the counter when the loop is entered. The starting value and the ending value is always to be explicit. The counter becomes the control Each time the iteration (cycle) is completed, the number that appears in the counter is incremented by a specified value. The increment is done to avoid a continuous or infinite loop. The most common loops under this category are the for…loops.

### 9.1.3 Define the term variable condition loop

**Vocabulary**

Variable condition loops allow loop body statements to be repeated an indefinite number of times. The repetition continues until a control condition is no longer fulfilled, i.e., when the boolean condition turns to false. Prior to the start of the loop, the condition must be initialized and changed (incremented or decremented) on each cycle. The common examples of loops which fall into this category are the while…loop and the do…. while loop.

### 9.1.4 Purpose of the while statement in C++

While loops are used when we do not know the exact number of loop iterations ahead of time. The execution of the loop is terminated based on the test condition. The while statement can be used to loop over a statement until a specified expression returns false. As we discussed in section 9.1, a programmer may want the code to perform specific actions if a certain condition is met. Assume we want to keep printing a user's first and last name as long as the counter is less than 10. We don't know how many times this will be done in this case, but if the counter is less than 10, it will be repeated. We'll also need to decide where to begin. If we use sequential statements, it will take us longer. Also, keep in mind that these are conditional, so sequential will not work. In while loops, the checking of condition before execution of statements is of paramount importance.

### 9.1.5 Determine the application flow when a while statement is encountered

Here is a flow diagram of a while loop.

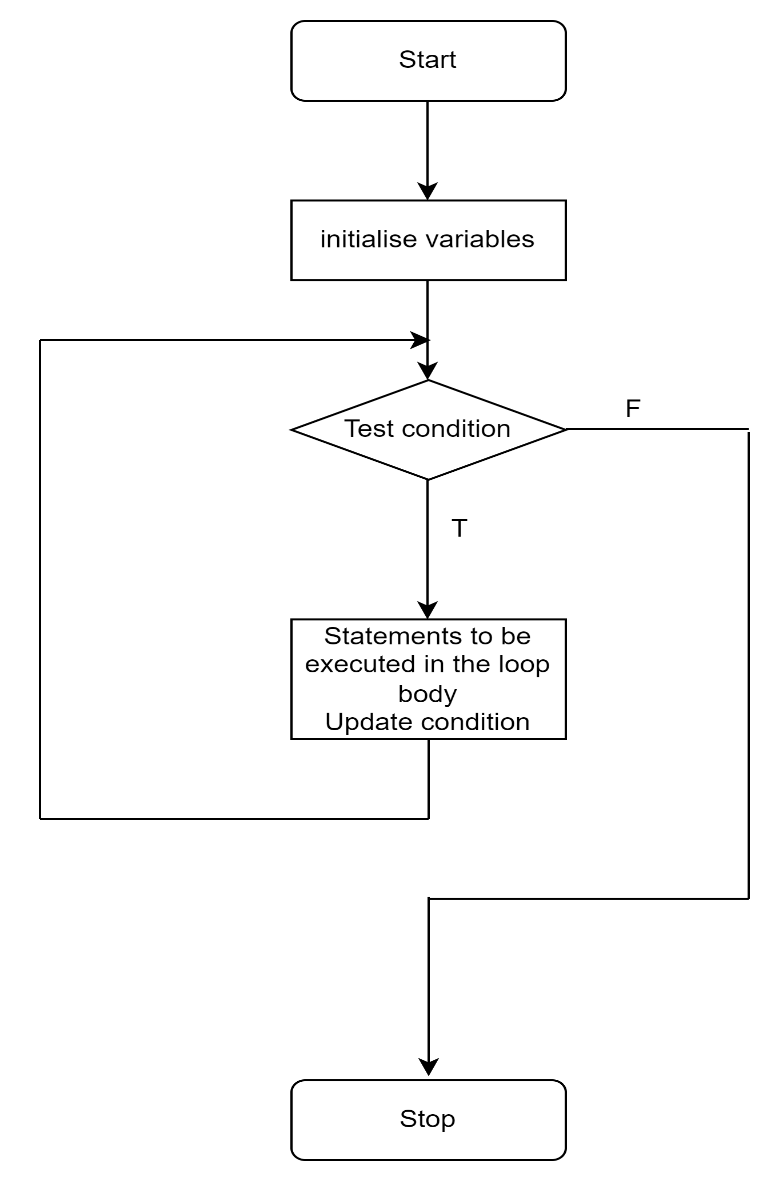


Figure 9. 2: While loop structure

Figure 9.2 illustrates what happens when a while loop is encountered. When a program is running and then meets a while loop, a condition is checked first. If the test condition is false, the program skips the loop and executes any statement(s) after the loop. If the condition is true, then the statements inside the loop will be executed. The counter variable is incremented or decremented so that every time the loop runs, the condition is checked against a new value. Alternatively, a value can be entered into the program that makes the loop condition false when it tries to iterate again.

### 9.1.6 Identify or correct the general form for a while statement

The syntax of a while statement in C++ is as follows:

while (condition) {  
*// statements to be executed*

*//update condition*  
}

**Parts of a while loop**

Condition-the test expression is evaluated first to see if the condition turns to true or false.

If the condition evaluates to true, then the statements to be executed are performed. This can be one or more statements.

Update condition- The update condition is used to increment/decrement the loop variable. In some text this is called counter variable. If the condition evaluates to false in the beginning, the loop body is skipped and if there any statements after the loop, they are then executed.

The code in Example 9.1 illustrates a while loop. We want to write a program to print full name of a user as long as the counter variable is age is less than 10 years.

**EXAMPLE 9.1**

#include <iostream>

using namespace std;

int main()

{

//declaring variables

string firstname="Mike";

string surname="Pienaar";

// counter variable age intialised to 1

int age=1;

//test expression to check if age is not greater than 10

while (age<10)//test condition

{

//loop body and expression(s) to be executed

cout <<firstname + " " + surname + " " + "age is "<<age<< endl;

//updating the condition by incrementing test variable

age=age +1;

}

return 0;

cout <<"We are now outside the loop"<< endl;

}

OUTPUT

Mike Pienaar age is 1

Mike Pienaar age is 2

Mike Pienaar age is 3

Mike Pienaar age is 4

Mike Pienaar age is 5

Mike Pienaar age is 6

Mike Pienaar age is 7

Mike Pienaar age is 8

Mike Pienaar age is 9

We are now outside the loop

Let us analyse the code above:

The variable age is initialised to 1. We do not know the number of times this has to be printed but our condition has been set as: age<10. As long as the value of age is less than 10, the fullname “Mike Pienar age is *age”* will be printed. After execution of the statements in the body we added a statement to update our condition and this has been coded as:

age=age +1;

Take note that age is the variable that is being incremented since its used in our condition.

Let us assume, we had declared age and initialised it to 10 or any other number greater- the loop body statements would never be executed and full name and age will never be printed.

The statement:

We are now outside the loop

is only printed once because it is outside the loop. This statement would be printed even if the condition had turned false.

NOTE

If the condition of loop never goes to false then loop iterations will never end. This type of loop are called infinite loop.

Let us do another example. This time we want to print all numbers divisible by 3 starting from 30 going down. At the start we do not know how many numbers will be printed. We want to print numbers as long as they are greater than 0.

**EXAMPLE 9.2**

#include <iostream>

using namespace std;

int main()

{

//initialise our control variable

int i=30;

// add the test condition

while(i>0) {

//numbers divisible by 3. We use mod symbol

if(i%3==0) {

cout<<"value of i is "<<i<<endl;

}

//updating our counter variable.

i--;

}

cout<<"We are now outside the loop"<<endl;

return 0;

}

OUTPUT

value of i is 30

value of i is 27

value of i is 24

value of i is 21

value of i is 18

value of i is 15

value of i is 12

value of i is 9

value of i is 6

value of i is 3

We are now outside the loop

Take note that in Example 9.2, we used a while loop. The loop body had some conditional statements to check if a number is divisible by 3. In addition, the control variable is updated by decrementing it.

**Task 9.1**

Write a C++ program to calculate factorial of a number requested by the user using a while… loop.

HINT: Factorial n! = n \* (n-1) \* (n-2) \* (n-3) … \*.

**SOLUTION**

#include <iostream>

using namespace std;

int main() {

int n;

cout<<"Please enter the number to be calculated"<<endl;

cin>>n;

int factorial = 1;

int i = 1;

if (cin.fail()){

cout<<"The value entered in not a number"<<endl;

}

else{

while (i<=n ) {

factorial = factorial\* i;

i++;

}

cout << "Factorial of " << n<< " is "<<factorial;

}

return 0;

}

**OUTPUT**

**Please enter the number to be calculated**

**5**

**Factorial of 5 is 120**

**TASK 9.2**

One of the common programs implemented in most programming courses is coding a solution to display Fibonacci numbers. The Fibonacci series is simply a series of numbers found by adding the two preceding (previous) numbers. The Fibonacci sequence's first two values are 0 and 1.

**QUESTION**

Write a program to generate Fibonacci numbers up to a number entered by the user. For instance is a user 78 the output must be as follows:

Enter a positive number : 78

The following are the Fibonacci numbers between 0 and 78

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55,

**SOLUTION**

#include <iostream>

using namespace std;

int main() {

//declaring and intialising variables

int first\_num = 0;

int second\_num = 1;

int next\_num = 0;

int user\_limit;

cout << "Enter a positive number : ";

cin >> user\_limit;

if (cin.fail() || user\_limit<0){

cout<<"Your input is invalid"<<endl;

goto ineligible;

}

// output

cout << "The following are the Fibonacci numbers between 0 and " <<user\_limit<< endl;

cout<<first\_num << ", " << second\_num << ", ";

//calculating the third value

next\_num = first\_num + second\_num;

//looping to generate the Fibonacci numbers

while(next\_num <= user\_limit) {

cout << next\_num << ", ";

first\_num = second\_num;

second\_num = next\_num;

next\_num = first\_num + second\_num;

}

return 0;

ineligible:

return 0;

}

**OUTPUT**

**Enter a positive number : 150**

**The following are the Fibonacci numbers between 0 and 150**

**0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,**

### 9.1.7 Write C++ code that will use relational expressions in while statement. (Max 2 levels only)

There are cases when we need to have more than one condition to be evaluated before any iteration occurs. C++ allows more than one condition when using while loops. With multiple conditions, while loops can be extended beyond their basic capabilities. With logical operators like && (and) and || (or), you can chain together multiple conditions. Essentially, we will be creating an equation that returns True or False based on whether the loop executes. For the && operator to return True, both conditions must be True. A true result is returned by the || operator as long as at least one condition is True. Let us look at the following code below where logical operators have been used to chain the different condition:

**EXAMPLE 9.3**

#include <iostream>

using namespace std;

int main()

{

int num1 = 0;

int num2 = 0;

while (num1 < 10 && num2 < 10) {

num1 = num1 + 1;

num2 = num2 + 2;

cout << "Value of num1: " << num1 << " ";

cout << "Value of num2: " << num2 << endl;

}

return 0;

}

**OUTPUT**

**Value of num1: 1 Value of num2: 2**

**Value of num1: 2 Value of num2: 4**

**Value of num1: 3 Value of num2: 6**

**Value of num1: 4 Value of num2: 8**

**Value of num1: 5 Value of num2: 10**

The code in Example 9.3 uses a while loop with multiple conditions chained together using a logical AND operator. If we are to use a logical OR to chain the conditions, we will have a different output as follows:

**OUTPUT**

**Value of num1: 1 Value of num2: 2**

**Value of num1: 2 Value of num2: 4**

**Value of num1: 3 Value of num2: 6**

**Value of num1: 4 Value of num2: 8**

**Value of num1: 5 Value of num2: 10**

**Value of num1: 6 Value of num2: 12**

**Value of num1: 7 Value of num2: 14**

**Value of num1: 8 Value of num2: 16**

**Value of num1: 9 Value of num2: 18**

**Value of num1: 10 Value of num2: 20**

Now, if we modify the statement a bit and have 2 level while loops the output will be different as shown below. The inner loop runs for every iteration of the outer loop, up to the termination point. For example, in the example given below, “num2” will run up to 10 for every “num1” and thus have a time complexity O(n²).

#include <iostream>

using namespace std;

int main()

{

int num1 = 0;

int num2 = 0;

while (num1 < 10) {

while (num2 < 10){

cout << "Value of num1: " << num1 << " Value of num2: " << num2 << endl;

num2 = num2 + 2;

}

num1 = num1 + 1;

}

return 0;

}

**OUTPUT**

**Value of num1: 0 Value of num2: 0**

**Value of num1: 0 Value of num2: 1**

**Value of num1: 0 Value of num2: 2**

**Value of num1: 0 Value of num2: 3**

**Value of num1: 0 Value of num2: 4**

**Value of num1: 0 Value of num2: 5**

**Value of num1: 0 Value of num2: 6**

**Value of num1: 0 Value of num2: 7**

**Value of num1: 0 Value of num2: 8**

**Value of num1: 0 Value of num2: 9**

### 9.1.8 C++ code that will use relational expressions in while statements with compounded content

**A nested loop is a loop statement nested within another loop statement. It is for this reason that nested loops are also referred to as "loop inside loops".** An inner loop is executed first, satisfying all the conditions within the loop, followed by an outer loop set of conditions.

**Below is the syntax of a nested loop structure.**

while(condition) {

while(condition) {

// statement of inside loop

}

// statement of outer loop

}

**We are going to represent a nested loop using a flow diagrams as illustrated in Figure 9.**

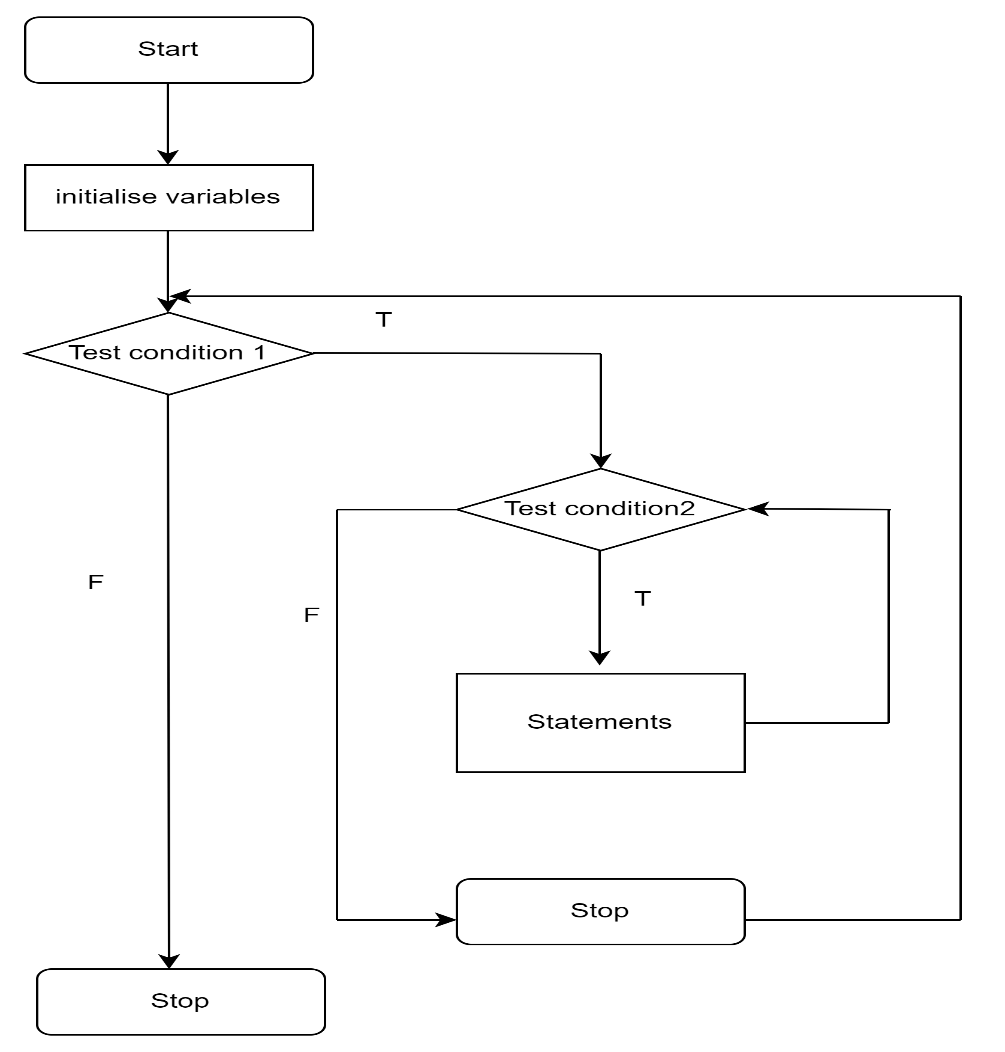
****

Figure 9. 3:Nested While Loop flow diagram

**NOTE**

It is also not required that the loop be placed within a loop of the same genre. It can be any loop, which means it can be placed inside any other loop.

Let us use a nested loop to print a 5 x 4 matrix table (5 rows and 4 columns) of characters. In C++, the sequence of uppercase letters or the sequence of lowercase letters can be compared as whole numbers, as shown in the nested while loop below:

**EXAMPLE 9.4**

#include <iostream>

using namespace std;

int main()

{

int i = 0; //initialisation of outer loop

char j = 'A'; //initialisation of inner loop

//outer loop

while (i < 5)

{

//inner loop

while (j < 'E')

{

cout << j << ' ';

j++;

};

j = 'A';

cout << endl;

i++;

};

return 0;

}

**OUTPUT**

**A B C D**

**A B C D**

**A B C D**

**A B C D**

**A B C D**

It can help to visualize a nested loop in two dimensions(i.e., rows and columns). Both loops have an effect on each cell. The inner loop is in charge of the columns, while the outer loop is in charge of the rows. The entire inner loop has been placed just after the outer loop's opening brace. Each character is printed by only one statement, which must be in the inner loop. This is due to the fact that loops only access one cell per combined iteration. The outer loop will run five times starting from 0 to 4 and this is controlled by the condition in the outer loop:

while (i < 5)

No further characters are printed if the while condition for the inner loop returns false. It is very important to understand how nested loops function.

Let us work on another example of a nested while loop.

We want to write a C++ program to print the output of a triangular pattern of numbers as shown below using nested loops.

1

12

123

1234

12345

123456

1234567

12345678

123456789

12345678910

**EXAMPLE 9.5**

#include <iostream>

using namespace std;

int main()

{

int i=1,j;

//outer loop

while (i <= 10)

{

j=1;

//inner loop

while (j <= i )

{

cout <<j ;

//incrementing the counter variable for the second loop

j++;

}

cout << endl;

//incrementing the counter variable for the outer loop

i++;

}

return 0;

}

The pattern is printed using a nested while loop in this program. The outermost loop executes 10 times, and for each loop, the innermost loop executes i times, which is 1 at first, meaning only "1" is printed, then 2 numbers printing "1 2" and so on until the loop executes 10 times, printing "1 2 3 4 5 6 7 8 9 10". The given number pattern is printed in this manner.

**TASK 9.3**

**Write a C++ program to print a timestable- from 1 to 10 using nested while loops. The output should be as follows.**

1 2 3 4 5 6 7 8 9 10

2 4 6 8 10 12 14 16 18 20

3 6 9 12 15 18 21 24 27 30

4 8 12 16 20 24 28 32 36 40

5 10 15 20 25 30 35 40 45 50

6 12 18 24 30 36 42 48 54 60

7 14 21 28 35 42 49 56 63 70

8 16 24 32 40 48 56 64 72 80

9 18 27 36 45 54 63 72 81 90

10 20 30 40 50 60 70 80 90 100

NOTE: Pay particular attention to the way how the output is displayed and aligned correctly.

**SOLUTION**

#include <iostream>

#include <iomanip>

using namespace std;

int main()

{

int i = 1;

//Outer loop

while (i <=10) {

int j = 1;

cout <<"" << endl;

//Inner loop

while ( j <= 10) {

cout << i \* j <<setw(5);

j++;

}

cout <<setw(5)<< endl;

i++;

}

return 0;

}

### 9.1.9 Explore the concept of block scope when writing while statements with compounded content

A block scope is a space between two curly braces that can be used to separate loops, if conditions, or switch statements. Variables defined in a code block are only visible within that block. Another type of scoping is the global scope. We explained these when we were working on variables in chapter 3.

A global scope variable can be found in any other part of a C++ program, from a function definition to a block of code. A global variable is one that is defined outside of the main function. The last scoping is local scoping. As a reminder, Any variable that is defined within a function definition is local to that function. The code in Example 9.6 illustrates block scoping, global and local scoping.

**EXAMPLE 9.6**

#include <iostream>

using namespace std;

// global scope

int num1 = 100;

int main()

{cout << "Value of num1 in a global scope is: " << num1 << endl;

// local scope

int num1 = 20;

cout << "Value of num1 in a local scope is: " << num1 << endl;

while (num1<350)

{

//block scope

num1 = 350;

cout << "Value of num1 in a block scope is: " << num1 << endl;

}

return 0;

}

**OUTPUT**

**Value of num1 in a global scope is: 100**

**Value of num1 in a local scope is: 20**

**Value of num1 in a block scope is: 350**

### 9.1.10 Explain the difference between fixed-count loop and variable condition loop

When a fixed-count loop is used, the test condition keeps track of how many times it has been repeated, whereas in variable-condition loops, the test condition is dependent on the changing value of variables. As discussed already, a fixed-count loop is used in circumstances where the programmer knows the number of times the program must iterate, whereas a variable condition loop is dependent on the test expression, which allows it to run if the condition is true and stops when the condition turns false.

### 9.1.11 C++ code that will use relational expressions in while statements with interactive input

In some cases we need to use while statements with interactive input to control the number of times our program will be executed. Foe example, we want to write a C++ program that will produce the following printout.

1

12

123

1234

12345

123456

1234567

The user must enter the number of times the iteration must run. This will then be used in the condition.

Here is an example:

**EXAMPLE 9.7**

#include <iostream>

using namespace std;

int main()

{

int i = 1, number, inner;

cout << "Please enter number of line you want printed ";

cin >> number;

while (i <= number)

{

cout << "\n";

inner = 1;

while (inner <= i)

{

cout << inner;

inner = inner + 1;

}

i = i + 1;

}

return 0;

}

**OUTPUT**

Please enter number of line you want printed 4

1

12

123

1234

**TASK**

**Write a C++ program using a while loop to produce the following output:**

54321

4321

321

21

1

**SOLUTION**

#include <iostream>

using namespace std;

int main()

{

int i, number, inner;

cout << "Please enter number of line you want printed ";

cin >> number;

i = number;

while (i >= 1)

{

cout << "\n";

inner = i;

while (inner >= 1)

{

cout << inner;

inner = inner - 1;

}

i = i - 1;

}

return 0;

}

### 9.1.12 Explain what a break statement is used for

Looping statements are terminated with a break command. After the break command, control is transferred to the next command outside the loop. Statements in the loop after the break statement do not execute. There are times when we want the program to exit the loop if it encounters a specific value or expression. Below is the syntax of a break statement when used in while loops.

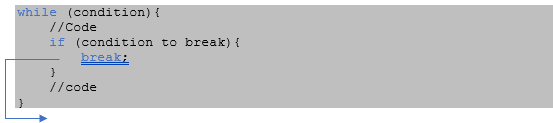


Figure 9. 4: Syntax of a break in while loop

As can be seen in Figure 9.4, once the condition in the if statement to break turns to true, the program will exit the whole loop.

Let’s take an example-we are going to use a while loop to print first integer numbers from 40 to 60. The program must exit when it counters the value 55. In such instances we do not know when the program will encounter the given value. So, we make use of a break statement.

**EXAMPLE 9.8**

#include <iostream>

using namespace std;

// global scope

int num1 = 40;

int main()

{

while (num1 <=60) {

cout << num1 << " ";

num1++;

//introducing a break

if (num1 == 55){

break;

}

}

return 0;

}

**OUTPUT**

**40 41 42 43 44 45 46 47 48 49 50 51 52 53 54**

TASK: Write a program to enter names of countries. If a user types “South Africa” the program should terminate.

### 9.1.13 Explain what the continue statement is used for

There are instances when we may want our code to skip specific values or actions in our program execution. In such instances we can make use of the continue statement, It is possible to skip a portion of a loop in an iteration using continue statement. For instance in Example 9.6, we had our program to exit the loop when it encountered the value 55. However, the loop was prematurely terminated. We can use the continue statement to skip the value and continue executing the loop.

**EXAMPLE 9.9**

#include <iostream>

using namespace std;

// global scope

int num1 = 40;

int main()

{

while (num1 <=60) {

//introducing a continue within an if condition

if (num1 == 55){

cout <<"\nI have met an unwanted value- so will skip" << endl;

num1++;

continue;

}

cout << num1 << " ";

//increment the counter

num1++;

}

return 0;

}

**OUTPUT**

**40 41 42 43 44 45 46 47 48 49 50 51 52 53 54**

**I have met an unwanted value- so will skip.**

**56 57 58 59 60**

As you can see from the output of Example 9.8, the program did not exit the loop when it encountered the value 55; instead, it skipped the value and continued with executing the remaining part of the loop.

### 9.1.14 C++ code that will use relational expressions in while statements making use of the break statement

We explained the purpose of a break statement in section 9.1.12. We also illustrated how to implement the break statement within a while loop. In this section we will work out some more challenging problems using the while loop. So let us implement more challenges on using break statement.

**QUESTION:** Write a program to find the sum and average of positive numbers entered by the user from the keyboard. If a user enters a negative number, the loop must break and display the sum and average of the numbers excluding the negative number.

**EXAMPLE 9.10**

// calculating sum and average of positive numbers entered

#include <iostream>

#include <iomanip>

using namespace std;

int main() {

int number;

int sum = 0, counter=0;

float average=0;

while (true) {

// prompt user for the input

cout << "Enter any number: "<< endl;

cin >> number;

// break condition

if (number < 0) {

break;

}

counter=counter+1;

// add all positive numbers

sum =sum+ number;

}

average=float(sum)/counter;

// display the sum

cout << "The sum is " << sum << endl;

cout.precision(2);

cout << "The average of the numbers is " << fixed<<average << endl;

return 0;

}

**OUTPUT**

**Enter any number:**

**21**

**Enter any number:**

**78**

**Enter any number:**

**56**

**Enter any number:**

**-5**

**The sum is 155**

**The average of the numbers is 51.67**

Let us try another scenario with user input.

**TASK**

Write a C++ program using while loops to calculate salary of an employee whose rate and hours are entered by the user. The program must repeat the process and break when the number of hours entered by the user is equal to 0.

The output should be as follows.

**Input your number of hours:5**

**Input your hourly rate:250.00**

**Your total salary is 1250.00**

**Input your number of hours:0**

**Unfortunately, incorrect entry or threshold not met to get a salary**

**Good bye**

**SOLUTION**

#include <iostream>

#include <cmath>

#include <iomanip>

using namespace std;

int main() {

//declaring variables

double hours, rate;

float salary=0;

while (true){

//accepting input

cout<<"Input your number of hours:";

cin>> hours;

//validating input

if (hours <=1 || cin.fail()){

cout<<"Unfortunately, incorrect entry or threshold not met to get a salary"<<endl;

break;

}

cout<<"Input your hourly rate:";

cin>> rate;

//processing

salary=hours \* rate;

//formatting output

cout.precision(2);

cout<<"Your total salary is R"<<fixed<<salary<<endl;

}

cout<<"Good bye"<<endl;

return 0;

}

**Implementing break in nested while loops**

The break statement only has an effect on the loop it is in. So, if you use a break statement in a nested loop, keep in mind that it only breaks the loop that surrounds it. Example 9.11 shows a nested loop with a break statement.

**EXAMPLE 9.11**

using namespace std;

int main() {

int i = 1;

while (i <= 5) {

int j = 1;

while (j <= 5) {

if(j>i) {

break;

}

cout << "\*";

j++;

}

cout << "\n";

i++;

}

}

OUTPUT

\*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

If we change the condition for the if statement within the inner loop to:

if(j>i)

then, the printout will a series of stars (\*) in one line. This is because in the first cycle, j is not less than i for the row.

### 9.1.15 C++ code that will use relational expressions in while statements making use of the continue statement

Sometimes we want to run our program to perform specified actions while excluding one or two of them. Continue comes in useful in these situations. In this section, we will execute a program that will print multiples of 3 from 30 to 3 while ignoring the value 15.

**EXAMPLE 9.12 Write a program to print multiples of 3 from 30 to 3 but exclude 15.**

//Program to show multiples of 3 from 30 to 3 excluding 15

#include <iostream>

using namespace std;

int i = 30;

int main() {

while (i > 0) {

if (i%3==0){

if (i == 15) {

i--;

continue;

}

cout << i << "\n";

}

i--;

}

return 0;

}

Often, people forget to add a decrement/increment within the condition to be excluded, the ultimate result is that the program will stop instead of continuing as if a break has been implemented.

**TASK**

**Write a program which prints all multiples of 5 from 100. If the value is divisible by3 and 5, skip the number**

**Solution**

//Program to show multiples of 3 from 30 to 3 excluding 15

#include <iostream>

using namespace std;

int i = 100;

int main() {

while (i > 0) {

if (i%5==0){

if (i%5==0 && i%3==0) {

i--;

continue;

}

cout << i << "\n";

}

i--;

}

return 0;

}

**OUTPUT**

**100**

**95**

**85**

**80**

**70**

**65**

**55**

**50**

**40**

**35**

**25**

**20**

**10**

**5**

# FORMATIVE ASSESSMENT 9.1 INDIVIDUAL TASK

9.1.1 Define the term pre-test loop as applied in iteration constructs. (2)

9.1.1 A pre-test loop is a repetition structure where the condition of the loop is upfront in the loop declaration. (2)

9.1.2  If condition of while loop remain true for unlimited iterations, then this type of loop is called\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (1)

1. Finite loop
2. Infinite loop
3. True loop
4. Power loop

9.1.2 B. Infinite Loop (1)

9.1.3 The variable used for iterations in condition of while loop is called. (1)

1. Static Variable
2. Counter Variable
3. Inner Variable
4. Condition Variable

9.1.3 B. Counter Variable (1)

9.1.4 Which of the following structure is implemented using while statement ? (1)

1. Sequence
2. Selection
3. Iterative
4. None of the above

9.1.4 C. Iteration (1)

9.1.5 What is the effect of writing a break statement inside a while loop? (1)

1. It cancels remaining iterations.
2. It skips a particular iteration.
3. The program terminates immediately.
4. Loop counter is reset.

9.1.5 A. It cancels remaining iterations and causes the code to exit the loop but can still any other code outside the loop. (1)

9.1.6 In a group of nested loops, which loop is executed the greatest number of times? (1)

1. The outermost loop
2. The innermost loop
3. All loops are executed the same number of times.
4. cannot be determined without knowing the size of the loops.

9.1.6 B. The innermost loop (1)

9.1.7 A junior programmer wrote the following loop condition in the process of solving a problem:

while( (i < 10) && (i > 24)){

//code

}

What is wrong with the given code. (2)

9.1.7 The condition is always false. So ultimately, no statements will ever be executed in the body of the loop. (2)

9.1.8 Write a C++ program to calculate the power of an integer number using the while loop structure. The base must be raised to a positive integer. Do not use the cmath library for this task.

//Program to calculate power of a integer number

#include <iostream>

using namespace std;

//Declaring variables

int main() {

unsigned int i=1,result =1;

int base, exponent;

cout<<"Please enter the base of a number"<<endl;

cin>>base;

cout<<"Please enter the number which you are raising by (exponent)"<<endl;

cin>>exponent;

//validating input

if(cin.fail() || exponent<1 || base<1){

cout<<"Incorect entry"<<endl;

return 0;

}

else{

//processing

while (i<=exponent){

result= result \* base;

++i;

}

//Displaying the output

cout<<base <<" raised to the power of " << exponent<< " = " <<result<<endl;

}

return 0;

}

**OUTPUT**

**Please enter the base of a number**

**4**

**Please enter the number which you are raising by (exponent)**

**5**

**4 raised to the power of 5 = 1024**

(10)

9.1.9 Consider the following code below and answer the question which follows:

using namespace std;

int main()

{

int i = 1 ;

i = i - 1 ;

while(i)

{

cout<<"its a while loop";

i++ ;

}

return 0;

}

How many times will the statement: "its a while loop". Explain your answer. (2)

9.1.9 The statement will never be printed. This is because the statement is never true because of the following line:

i = i - 1 ;

If the line is removed, the program will generate an infinite loop. (2)

# 9.2. Logic operators in While loop

While loops allow use of logical operators in the testing of condition which then allows execution of statements inside the loop or not.

### 9.2.1 C++ code that will use relational expression containing logic operators in while (Max 3 level, Max 3 logic operators per level)

### 9.2.2 C++ code that will use relational expressions containing logic operator in while loop with compounded content (Max 3 level, Max32 logic operators per level)

# 9.3. Do while loop

We are going to introduce the do... while loop as one of the repetition components. Just like in every other programming language, the premise is the same. Just the syntax is different. Remember how we used an Arduino to develop the do.. loop in chapter 3? The idea is the same, and we'll build on that understanding to use the do...loop to address more challenging issues. This is going to be re-emphasis of concepts.

### 9.3.1 Define the term post-test loop and exit controlled loop

A post-test loop is a type of repetition structure where the test condition is placed at the end of the loop body. A post-test loop executes the statements in the loop body at least once before evaluating the test condition. An example of a post-test loop is the do...while construct. Figure 9.5 illustrates a post-test loop flowchart.

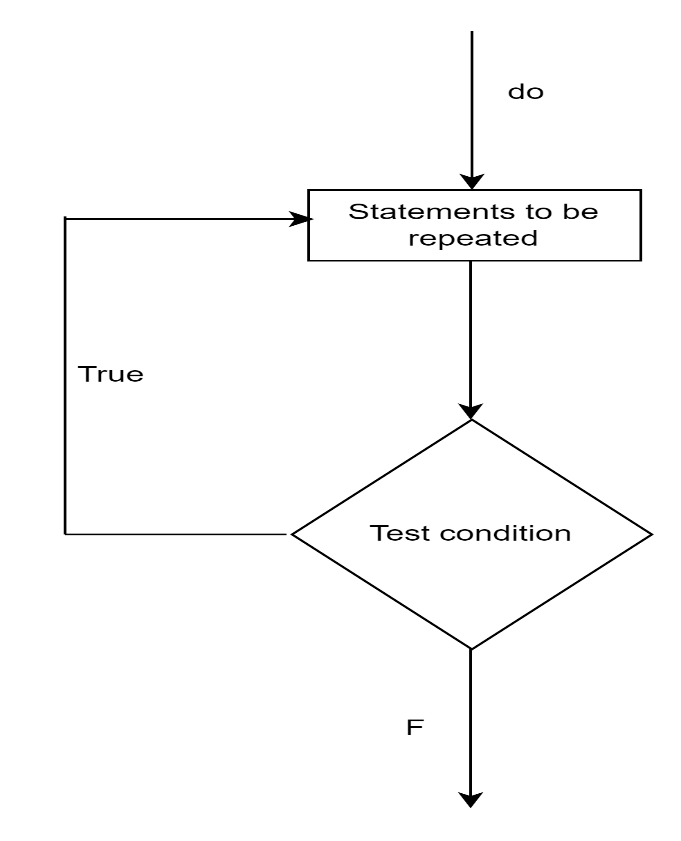


Figure 9. 5: Post-test loop

Another term used to refer to a post-test loop is an exit-controlled loop. Looking at the flowchart in Figure 9.5, the given definition fits well with what is depicted. The condition is only tested at the exit point of the loop. No loop statements are found after the test condition.

### 9.3.2 Determine the application flow when a do while statement is encountered

The conditional expression appears at the end of the loop, as you may have seen in Figure 9.5, so the statement(s) in the loop run once before the condition is tested. If the condition is met, the statement(s) in the loop are once again executed and the flow of control repeats. If the statement proves false after at least one execution, the program exits the loop body, and any statements that follow the loop are then performed.

### 9.3.3 Identify or correct the general form for a do while statement

The syntax of a *do…while* statement in C++ is as follows:

**do**{

//code to be executed

} **while**(condition);

**Parts of a do while loop**

**do** -this indicates the start of the loop.

Statements or code to be executed- These consists of one of more statements that have to repeated a number of times. At first these statements are executed before checking conditions. If the condition is true, the same statements are repeated several times until the condition turns to false.

**Update condition**- Just like in while loop, the update condition is used to increment/decrement the loop variable and is done before the condition is tested.

**while (condition )** this is where the condition is tested. Path of the program is decided here.

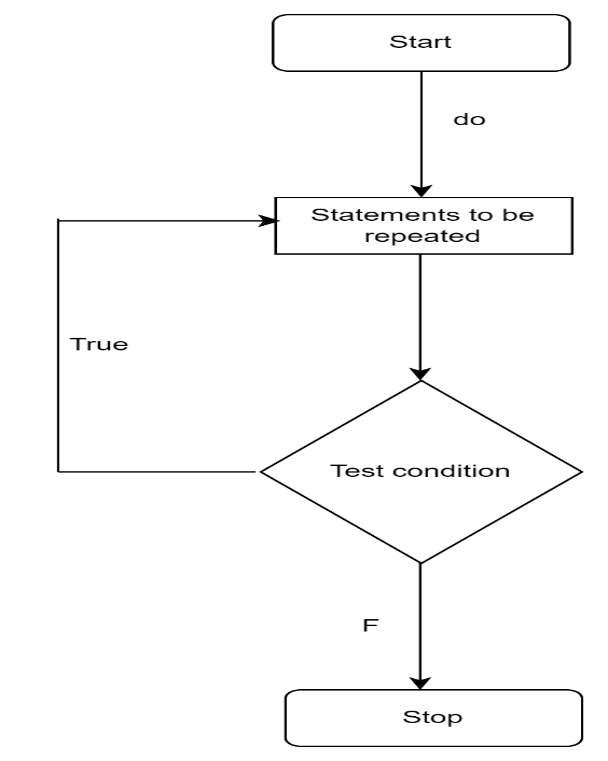


Figure 9. 6: Flowchart to show a do while loop

The code in Example XXX demonstrates a do ..while loop construct to display numbers from 1 to 10.

**EXAMPLE**

#include <iostream>

using namespace std;

int main()

{ //initialisation of counter variable

int i =1;

do{//statements to be executed

cout << i<< endl;

//updating the counter variable

i=i+1;

}//testing the condition

while (i<=10);

//statements outside the loop

cout <<"This is outside the loop"<< endl;

return 0;

}

**OUTPUT**

**1**

**2**

**3**

**4**

**5**

**6**

**7**

**8**

**9**

**10**

**This is outside the loop**

Looking at the code above, the program is run 10 times. If the initial value was set to any value greater than 10, that value will be printed but immediately the program would then exit the loop after running only once. The statement:

**This is outside the loop**

is executed only after the loop condition after the loop is done. So, it is independent of the loop only that the sequence is after the loop.

TASK

Write a program using do loop to print small letters of the alphabet from ‘a’ to ‘z’.

//Program to display alphabet characters from a to z

#include<iostream>

using namespace std;

int main()

{

//declare and initialise first letter

char letter='a';

do

{

cout<<letter<<" ";

letter++;

}

while(letter<='z');

cout<<"\nOut of the loop"<<endl;

return 0;

}

**OUTPUT**

**a b c d e f g h i j k l m n o p q r s t u v w x y z**

**Out of the loop**

TASK

Write a program to find the greatest common divisor of two numbers.

**Greatest Common Divisor:**

The biggest positive integer number that can evenly divide two positive integer numbers without leaving a remainder is known as the GCD.

Solution

#include <iostream>

using namespace std;

int main()

{

int num1, num2, gcd;

cout << "\n\n Find the Greatest Common Divisor of two numbers:\n";

cout << "-----------------------------------------------------\n";

cout << " Input the first number: ";

cin >> num1;

cout << " Input the second number: ";

cin >> num2;

int i=1;

do{

if (num1 % i == 0 && num2 % i == 0)

{

gcd = i;

}

i++;

}while ( i <= num1 && i <= num2) ;

cout << " The Greatest Common Divisor is: " << gcd << endl;

return 0;

}

**OUTPUT**

**Find the Greatest Common Divisor of two numbers:**

**-----------------------------------------------------**

**Input the first number: 88**

**Input the second number: 12**

**The Greatest Common Divisor is: 4**

### 9.3.4 Define the term sentinel

**VOCABULARY**

A sentinel value is a value or character that is entered to stop the loop from repeating itself.

### 9.3.5 Explain where and for what reason a sentinel is used

We frequently have a sentinel or end-of-file controlled loop when reading data from a file or standard input. The sentinel value is used to indicate when processing or reading more data is complete. Let us use the following example of asking a user to enter marks for tests and then calculating the average: We can code our program in such a way that the user keeps entering marks for tests and when done enters a special value or character to signal the end. This special signal or value is called a **sentinel value.**

#include <iostream>

using namespace std;

int main(void){

//declaring variables

int mark;

float average=0;

int sum=0, counter=0;

//accepting average

cout << "Please enter a test score:";

cin >> mark;

do

{

//

sum=sum + mark;

counter=counter+1;

cout << "Type 101 if to terminate else enter any other test score." << endl;

cout << "Please enter a number:";

cin >> mark;

//stoping value is checked

} while(mark!=101);

//calculating average

average=float(sum)/counter;

//formating output to 2 decimal places

cout.precision(2);

//displaying average

cout << "The average is " <<fixed<<average<<endl;

return 0;

}

**OUTPUT**

**Please enter a test score:56**

**Type 101 if to terminate else enter any other test score.**

**Please enter a number:47**

**Type 101 if to terminate else enter any other test score.**

**Please enter a number:56**

**Type 101 if to terminate else enter any other test score.**

**Please enter a number:101**

**The average is 53.00**

**Another example would be when user is drawing money or doing any transaction at an ATM. After completing the transaction, the ATM will ask the user if he/she wants to do another transaction. If the user selects no, then the processing will be terminated.**

**TASK**

**Write a program to implement a sentinel value using do loop to stop a user from continuing with transaction on the ATM.**

**SOLUTION**

#include <iostream>

using namespace std;

int main()

{

char response;

cout<<"Would you like to do another transaction"<<endl;

cin>>response;

do{

cout<<"Withdrawal"<<endl;

cout<<"Statements"<<endl;

cout<<"Transfers"<<endl;

cout<<"Would you like to do another transaction"<<endl;

cin>>response;

}while(response!= 'N' );

cout << "Thank you for banking with us. enjoy your day" << endl;

return 0;

}

**OUTPUT**

**Would you like to do another transaction**

**y**

**Withdrawal**

**Statements**

**Transfers**

**Would you like to do another transaction**

**Y**

**Withdrawal**

**Statements**

**Transfers**

**Would you like to do another transaction**

**N**

**Thank you for banking with us. enjoy your day**

**break statement in do while loops**

With the break statement, you can end the do-while loop even before the condition turns out to be false. break statements is added to stop when a certain condition is met. The do-while loop in the example that follows outputs integers from 1 to 10. But after that, we add a break expression so that the loop is broken when i equals 4.

**EXAMPLE**

#include <iostream>

using namespace std;

int main() {

int i = 1;

do {

if (i == 4) {

break;

}

cout << i << "\n";

} while (++i <= 10);

return 0;

}

**OUTPUT**

**1**

**2**

**3**

### 9.3.6 C++ code that will use relational expression containing logic operators in a do while loop (Max3 level, Max 3 logic operators per level)

As we have shown below with while loops, do while loop also allows the use of multiple conditions as the test expression. We can use multiple conditions evaluated using relational operators and chained by logical operators. Here is an example:

**EXAMPLE**

Write a program to accept 4 test scores and calculate average of the numbers. Repeat the process as long as the answer is Yes or YES or yes.

//Program to display alphabet characters from a to z

#include<iostream>

using namespace std;

int main()

{

//declare and initialise first letter

int test\_1, test\_2, test\_3, test\_4, sum;

float average=0;

string response;

do

{

cout<<"Please enter first number"<<endl;

cin>>test\_1;

cout<<"Please enter second number"<<endl;

cin>>test\_2;

cout<<"Please enter third number"<<endl;

cin>>test\_3;

cout<<"Please enter fourth number"<<endl;

cin>>test\_4;

if (cin.fail()){

cin.clear();

cin.ignore(1000, '\n');

cout << "The data entry failed to meet the requirements"<<endl;

cout<<"Please enter first number"<<endl;

cin>>test\_1;

cout<<"Please enter second number"<<endl;

cin>>test\_2;

cout<<"Please enter third number"<<endl;

cin>>test\_3;

cout<<"Please enter fourth number"<<endl;

cin>>test\_4;

}

average=(test\_1+test\_2+test\_3 +test\_4)/4;

cout<<"The average of the 4 tests is "<<average<<endl;

cout<<"Do you want to enter another set of marks "<<endl;

cin>>response;

}

while(response== "Yes" || response == "YES" || response== "yes");

cout<<"\nOut of the loop"<<endl;

return 0;

}**OUTPUT**

**Please enter first number**

**5**

**Please enter second number**

**6**

**Please enter third number**

**2**

**Please enter fourth number**

**3**

**The average of the 4 tests is 4**

**Do you want to enter another set of marks**

**no**

**Out of the loop**

### 9.3.7 C++ code that will use relational expressions containing logic operator in a do while loop with compounded content

We are going to implement a multiple level do..while loop with maximum of 3 logic operators. In demonstrating this concept, we are going to code a Floyd’s triangle. Floyd's triangle is a right-angled triangle with natural values that are consecutive. The number in Floyd's triangle begins with 1 in the upper left corner and proceeds to fill the specified rows with numbers. For example, if we create four rows in Floyd's triangle, it produces the following pattern in increasing order:

1

01

101

0101

**EXAMPLE**

#include<iostream>

using namespace std;

int main(){

int counter=1;

int number\_rows,j,p,q;

cout<<"Please enter number of rows : ";

cin>>number\_rows;

do{

if (!(cin.fail()) ){

do

{

if(counter%2==0){

p=1;

q=0;

}

else{

p=0;

q=1;

}

j=1;

//inner loop

do

{

if(j%2==0)

{

cout<<p;

}

else

{

cout<<q;

}

j++;

}

while(j<=counter && j<= number\_rows && counter>0);

counter++;

cout<<endl;

}

while(counter<=number\_rows && counter>=j && counter>0);

counter++;

cout<<"The end of the program"<<endl;

} else{

cout<<"Incorrect entry"<<endl;

}

}while (counter<=number\_rows && j>0 && counter>=j);

}

OUTPUT

Please enter number of rows : 7

**1**

**01**

**101**

**0101**

**10101**

**010101**

**1010101**

**The end of the program**

### 9.3.8 Write C++ code that will use relational expression containing logic operators in a sentinel controlled do while loop (Max 3 level, Max 3 logic operators per level)

In this section, we are going to implement a program to use three relational operators and three logical operators. The program is terminated when a user enters a BMI which classifies an individual as underweight or overweight. Assume an organisation wants to recruit all people whose BMI is classified as normal. The company would then go and capture their detail such as firstname, surname and ID number with the intention of interviewing them. We are going to implement a C++ program using a do…while loop to demonstrate the scenario.

**EXAMPLE**

#include <iostream>

using namespace std;

double BMI, ID;

string name, surname;

int main()

{

cout << "Please enter your BMI\n";

cin >> BMI;

do {

cout << "Please enter your name\n";

cin>> name;

cout << "Please enter your surname\n";

cin>> surname;

cout << "Please enter your ID number\n";

cin>> ID;

if (BMI>=18.5 && BMI<=24.9){

cout << "-----------------------------------------------------\n";

cout<<"Firstname\t"<<name<<endl;

cout<<"Surname \t"<<surname<<endl;

cout.precision(0);

cout<<"ID Number\t"<<fixed<<ID<<endl;

}else{ cin.clear();

cin.ignore(1000, '\n');

cout << "The data entry failed to meet the requirements"<<endl;

}

cout<<"What is the BMI of the next person"<<endl;

cin>>BMI;

}while (!(BMI < 18.5 || (BMI>=25 && BMI<=29.9) || BMI >30));

return 0;

}

**OUTPUT**

**Please enter your BMI**

**22**

**Please enter your name**

**Wellington**

**Please enter your surname**

**Stephens**

**Please enter your ID number**

**3310135929187**

**-----------------------------------------------------**

**Firstname Wellington**

**Surname Stephens**

**ID Number 3310135929187**

**What is the BMI of the next person**

**12**

### 9.3.9 C++ code that will use relational expressions containing logic operator in a sentinel controlled do while loop with compounded content (Max 3 level, Max 3 logic operators per level)

In this program we are going to use compounded do while loop to check sides of a triangle and calculate its area if valid.

Write a program using do while loop in C++ to find the area of any triangle using Heron's Formula. Heron’s formula is used to calculate area of a triangle when you are given the length of the three sides. The formula is as follows:

A = √s(s−a)(s−b)(s−c)

The sides of a triangle must be verified by the program. The program must end if the sides, for example, [100, 5, 20], form an invalid triangle. In the event that the sides are correct, the program must determine whether the triangle is scalene, isosceles, or equilateral before calculating its area. If the response to accepting another set of sides is not Yes, YES, or Yes, the program should end.

**EXAMPLE**

//Program to accept three sides of a triangle and calculate the area

#include<iostream>

#include <cmath>

using namespace std;

int main()

{

//declare and initialise first letter

int s1,s2,s3;

float area=0, s=0;

string response;

do

{

//accepting input

int s1,s2,s3;

cout<<"please enter first side of the triangle"<<endl;

cin>>s1;

cout<<"please enter second side of the triangle"<<endl;

cin>>s2;

cout<<"please enter third side of the triangle"<<endl;

cin>>s3;

//validating input

if (cin.fail()){

cin.clear();

cin.ignore(1000, '\n');

cout << "The data entry failed to meet the requirements"<<endl;

//accepting input

cout<<"please enter first side of the triangle"<<endl;

cin>>s1;

cout<<"please enter second side of the triangle"<<endl;

cin>>s2;

cout<<"please enter third side of the triangle"<<endl;

cin>>s3;

}

//checking if it’s a valid triangle

if ((!(s1+s2>s3)) || (!(s1+s3>s2)) ||(!(s3+s2>s1))){

cout<<"not a valid triangle"<<endl;

}

else{

//evaluating the conditions

if(s1==s2 && s2==s3)

{

cout<<"This is an equilateral triangle"<<endl;

}

else if(s1==s2||s1==s3 || s2==s3)

{

cout<<"This is an isosceles triangle"<<endl;

}

else

{

cout<<"This is a scalene triangle"<<endl;;

}

//calculating area

s = float(s1+s2+s3)/2;

area =float(sqrt(s\*(s-s1)\*(s-s2)\*(s-s3)));

cout.precision(2);

cout<<" The area of the triangle is : "<<fixed << area <<"cm^2"<< endl;

cout << endl;

cout<<"Do you still want to input another set of sides "<<endl;

cin>>response;

}

}

while(response== "Yes" || response == "YES" || response== "yes");

cout<<"\nOut of the loop"<<endl;

return 0;

}

**OUTPUT**

**please enter first side of the triangle**

**6**

**please enter second side of the triangle**

**6**

**please enter third side of the triangle**

**6**

**This is an equilateral triangle**

**The area of the triangle is : 15.59cm^2**

**Do you still want to input another set of sides**

**yes**

**please enter first side of the triangle**

**k**

**please enter second side of the triangle**

**please enter third side of the triangle**

**The data entry failed to meet the requirements**

**please enter first side of the triangle**

**7**

**please enter second side of the triangle**

**7**

**please enter third side of the triangle**

**8**

**This is an isosceles triangle**

**The area of the triangle is : 22.98cm^2**

**TASK**

Write a program using do while loops in C++ to print a print a parallelogram or rhombus star pattern as shown below.

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

\*\*\*\*\*

**SOLUTION**

#include<iostream>

using namespace std;

int main()

{

int i=1,j,k,n;

//outer loop

do

{

j=i;

do

{

cout<<" ";

j++;

}

while(j<=5);

k=1;

//inner loop

do

{

cout<<"\*";

k++;

}

while(k<=5);

i++;

cout<<endl;

}

while(i<=5);

return 0;

}

# FORMATIVE ASSESSMENT INDIVIDUAL TASK

9.3.1 Which of the following is an exit-controlled loop?

1. While loop
2. For loop
3. Do while loop
4. both b & c (1)

9.3.1 C. do…while loop (1)

9.3.2 Which looping process checks the test condition at the end of the loop? (1)

1. for
2. while
3. **do-while**
4. no looping process checks the test condition at the end

9.3.2 C. **do-while (1)**

**9.3.3 Write the correct syntax of a do..loop.**

**9.3.3 do…loop (3)**

**do**{

//code to be executed

} **while**(condition);

**(3)**

**9.3.4 What is the effect of writing a break statement inside a do loop? (2)**

**9.3.4 break statement in a do loop cancels remaining iterations and exits from the loop but any other statements after the loop will be executed. (2)**

**9.3.4 What is the output of the C++ code below? Explain your answer? (3)**

#include <iostream>

using namespace std;

int main()

{

int a=32;

do

{

cout<<a;

a++;

}while(a <= 30);

return 0;

}

**9.3.4 The answer is 32. do { } block is executed even before checking while(condition) at least once. This prints 32. To loop for the second time, while (32 <= 30) fails. So, loop is quit. (3)**

# 9.4. The for loop

In this section we are going to discuss for…loop as another construct implemented in repetition structures. For…loops are used when you know exactly how many times you want to repeat a block of code. Again, the for…loops which we discussed in section 3.2.1.6 of chapter 3 is the same with what we are going to discuss in this section.

### 9.4.1 Define the term variable condition loop

Variable condition loops allow loop body statements to be repeated a number of times as long as the control condition is still fulfilled. Once it turns false, the program exits the loop.

### 9.4.2 Explain how the, for statement differs from the while

First, we need to understand that the for loop and while loop are both iteration constructs. However, the application of the two depends on the situation. For loops are used when the number of iterations is known, whereas while loops are used until a statement in the program is proven wrong before the program exits. We are going to use a table to list more differences between the for loop and the while loop.

Table 9. 1: Comparison of for loop and while loop

|  |  |
| --- | --- |
| **For Loop** | **While loop** |
| Number of iterations known | Used when number of iterations is not known |
| In the absence of a condition, the loop is infinitely repeated. | In the absence of a condition, an error will be displayed. |
| Initialization is not repeated | If the initialization is performed during the checking stage, it is repeated |
| Initialization can take place inside or outside of the loop. | Initialization is always performed outside of the loop. |
|  |  |

### 9.4.3 Explain how the, for statement is similar to the while

One major similarity between a for..loop and while loop in C++ is that they are both entry controlled loop. The condition is tested first for any statements in the loop body to be executed otherwise if condition is not met, the statements in the loop body will never be executed at all.

### 9.4.4 Identify or correct the general form for a for statement

The flowchart representation of a for loop is shown in figure 9.7.

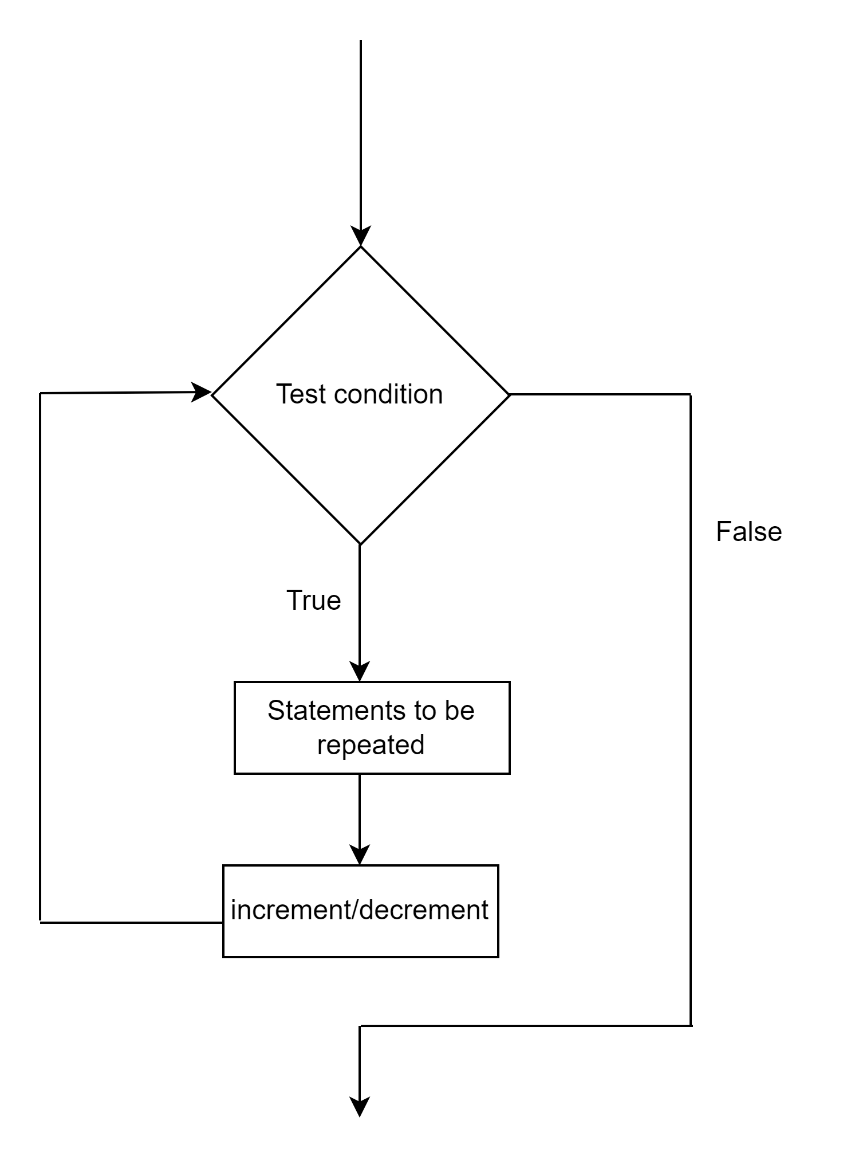
****

Figure 9. 7: For loop structure

The general syntax of the for loop is as follows:

for (initialization; condition; incr/decr){

//code to be executed

}

### 9.4.5 Explain the purpose of each part in the for-loop initialisation

**Parts of a for…loop**

**Initialization Expression in for Loop**- This expression is used to initialise the starting value of the for loop. Example: int i=4; Please take note the initialisation expression ends with a semi-colon not a comma. Take note that the data type is important.

**Test Condition in for Loop**-This expression is the one used to check if the condition is true before continuing to execute statements in the loop body. If the condition is not true, the program won’t execute any statements in the loop body. An example of a test condition is:

i < 10;

**Update Expression in for Loop-** The control variable must be incremented or decremented in order for the loop to advance to the end. Otherwise, the loop will continuously perform the same statement, resulting in a continuous or infinite loop. An update expression might look like this:

i++

Example XXX gives a full sample code of a for loop program to print numbers from 4 to 10.

#include <iostream>

using namespace std;

int main()

{

// initialisation expression, test condition & update expression in loop header

for (int i=4; i<=10; i++){

//loop statements to be executed if condition is satisfied

cout<<i<< " ";

}

return 0;

}

**OUTPUT**

**4 5 6 7 8 9 10**

When the condition is met, i=10, the loop will execute once for the last time but increase the value of i to 11. If i is 11, the condition will not hold. The program will bypass the loop and execute any subsequent statements.

EXAMPLE Write a C++ program to generate a 2 times table using a for loop statement. The output of the program must be as follows:

**2 Times Table**

**-----------------------------------------------------**

**1 x 2 = 2**

**2 x 2 = 4**

**3 x 2 = 6**

**4 x 2 = 8**

**5 x 2 = 10**

**6 x 2 = 12**

**7 x 2 = 14**

**8 x 2 = 16**

**9 x 2 = 18**

**10 x 2 = 20**

**Solution**

#include <iostream>

using namespace std;

int main()

{

int answer=0;

cout << "\t\t2 Times Table\n";

cout << "-----------------------------------------------------\n";

// initialisation expression, test condition & update expression in loop header

for (int i=1;i<=10;i++)

{

//processing

answer = i \* 2;

//displaying the solution

cout<<i << " x "<<2<<" = "<< answer<<endl;

}

return 0;

}

Task

Write a C++ program using for loops to calculate the factorial of a number entered by a user. Validate the input to allow positive numbers. If the value entered is not a number or less than 0, the program must give display an error message.

SOLUTION

#include<iostream>

#include<conio.h>

using namespace std;

int main()

{

int i;

int num;

unsigned long long fact=1;

cout<<"Please enter a number: ";

//accepting input

cin>>num;

//validating input

if (cin.fail() or num<0){

cin.clear();

cin.ignore(1000, '\n');

cout << "The data entry failed to meet the requirements"<<endl;

}

else{

//calculating factorial of a number

for(i=1;i<=num;i++)

{

fact=fact\*i;

}

cout<<"Factorial of entered number is: "<<fact;

return 0;

}

}

**TASK**

Fizz Buzz is a simple programming challenge that is commonly posed in software developer job interviews. We are going to do the same program following the given specifications.

Write a C++ program using for loop which will loop from 1 to 20 and print the values. The must replace all occurrences of number divisible by 3 with the word FIZZ and occurrences of numbers divisible by 5 with the word BUZZ and all occurrences of numbers divisible by both 3 and 5 with the word FIZZ BUZZ. The program should then count instances of each.

**NOTE**: For FIZZ BUZZ the program should add 1 to instances of FIZZ and 1 to instances of BUZZ.

**SOLUTION**

#include <iostream>

using namespace std;

int main()

{

// declare variables

int fizz = 3;

int buzz = 5;

int end\_value = 20;

int fizzBuzz = fizz \* buzz;

int counter1=0, counter2=0, counter3=0;

// ^ numbers divisible by 3 and 5 are also divisible by 3 \* 5

// start the loop, continue until the counter (start\_value) reaches the 'end\_value

for (int start\_value = 1; start\_value <= end\_value; ++start\_value)

{

if (start\_value % fizzBuzz == 0) // divisible by 3 and 5

{

cout<<"FIZZ BUZZ!!\n";

counter3=counter3+1;

counter2=counter2+1;

counter1=counter1+1;

}

else if (start\_value % fizz == 0) // divisible by 3

{

cout<<"FIZZ\n";

counter1=counter1+1;

}

else if (start\_value % buzz == 0)// divisible by 5

{

cout<<"BUZZ\n";

counter2=counter2 +1;

}

else // not divisible by 3 or 5

{

cout<<start\_value<<endl;

}

}

//displaying the occurences of each word

cout<<"The number of occurences of the word FIZZ is "<<counter1<<endl;

cout<<"The number of occurences of the word BUZZ is "<<counter2<<endl;

cout<<"The number of occurences of the word FIZZBUZZ is "<<counter3<<endl;

return 0;

}

**OUTPUT**

**1**

**2**

**FIZZ**

**4**

**BUZZ**

**FIZZ**

**7**

**8**

**FIZZ**

**BUZZ**

**11**

**FIZZ**

**13**

**14**

**FIZZ BUZZ!!**

**16**

**17**

**FIZZ**

**19**

**BUZZ**

**The number of occurences of the word FIZZ is 6**

**The number of occurences of the word BUZZ is 4**

**The number of occurences of the word FIZZBUZZ is 1**

### 9.4.6 Explain what effect an empty semicolon will have on the for loop

If an empty semicolon is added in the for loop,   the initial assignment of x is excluded from the loop. For example, consider the code snippet below:

#include <iostream>

using namespace std;

int main()

{

int i;

// initialisation expression, test condition & update expression in loop header

for (; i<=10; i++){

//loop statements to be executed if condition is satisfied

cout<<5<< " ";

}

return 0;

}

If the I is not declared outside the loop, we will get an error but if i is declared, only the initialisation part is skipped. The given snippet will then execute the code as if the initial value of I is 0 resulting in the statements being executed repeatedly 11 times. If semicolons alone are put in the for loop as follows:

#include <iostream>

using namespace std;

int main()

{

int i;

// initialisation expression, test condition & update expression in loop header

for (; ;){

//loop statements to be executed if condition is satisfied

cout<<5<< " ";

}

return 0;

}

The above code would generate an infinite loop since there is no condition to stop the execution of the statements. Only if a break is included then the program will terminate. If a semicolon is added outside the loop declaration expression, it means do nothing. Semicolon is a legitimate statement called null statement \* that means "do nothing". Consider the code below:

#include <iostream>

using namespace std;

int main()

{

int i;

// initialisation expression, test condition & update expression in loop header

for (i=0;i<5;i++);

{

cout<<"hello"<<i;

}

return 0;

}

The above code will print nothing from the loop body because the ; is a null statement and it does just nothing except terminating the loop. Instead, the new scope with the hello statement. We have seen that if we place a semicolon after nothing, you will create a statement. Thus the semicolon by itself is a statement, which is called the empty statement or the null statement.

### 9.4.7 Explain the term infinite loop

**VOCABULARY**

An infinite loop is a loop which runs endlessly. The test condition is always true resulting in statements being executed repeatedly without termination.

### 9.4.8 Write C++ code to create an infinite for loop

Here is an example that will generate an infinite for loop.

#include <iostream>

using namespace std;

int main()

{

int i;

// initialisation expression, test condition & update expression in loop header

for ( int i=0; ;){

//loop statements to be executed if condition is satisfied

cout<<"infinite loop"<<endl;

}

return 0;

}

The test condition of the program above is nothing so the code will run endlessly printing the statement: "infinite loop”.

In some cases, the programmer can code the test condition incorrectly leading to endless looping. For instance, the following code declaration for a for loop will result in an endless loop.

for ( int i=1; i>0; i++)

This is because he value of i is always greater than 0 from the given declaration.

**TASK**

Write your own program in C++ to create an infinite loop using for loop.

### 9.4.9 Use the break statement to end an infinite for loop based on the loop reaching a condition

As in while or do loop, the break statement works the same. When encountered, it causes the program to terminate the loop and execute any statements outside the loop if any. There are times when we want our program to stop when it encounters a certain value after the condition has been satisfied and may be we do not know exactly at what point this would happen. So, we set another condition inside the loop and add a break statement which will force the program to jump out of the loop. Consider the code in the example below.

**EXAMPLE**

The program below is a countdown loop that is supposed to print multiples of 3 from 50 to 1. If the at any given value, the test loop counter variable is exactly equal to 27, the program must break.

//program to print multiples of 3 from 50 down to 1

//if i is equal to 27 the program breaks

#include <iostream>

using namespace std;

int main()

{

//loop from 50 down to 1

for (int i =50; i>0; i--){

//checking for multiples of 3 from 50 down to 1

if (i%3==0){

//when the progam encounters a 28 it breaks

if (i==27){

break;

}

cout<<i<<endl;

}

}

return 0;

}

**OUTPUT**

**48**

**45**

**42**

**39**

**36**

**33**

**30**

### 9.4.10 Determine the application flow when for statement is encountered.

This is what happens when a for loop is encountered.

* If the test condition is not met, the loop will never execute.
* The update expression when executed changes the value of the loop control variable (initialised by the initial expression) which eventually sets the value of the condition to false. The for-loop body executes indefinitely if the loop condition is always true.
* A semicolon at the end of the for-loop statement (just before the body of the loop) will cause the loop to be empty. It will cause a semantic error if done.
* In the for statement, if the loop condition is omitted, it is assumed to be true.
* If the test condition is true, the loop body is executed, and the control incremented or decremented as per code. The loop body statement will continue to be executed for as long as the condition is true. The test condition is checked in each cycle of the loop.

Care is required in getting the for loop to perform the desired action.

### 9.4.11 Write C++ code that will use a for loop with a predetermined number of loops.

With a predetermined number of loops, we know exactly how many times our program will have to iterate. We are going to demonstrate this with compound assignment operators. Our program will loop five times. As predetermined in the loop test condition, the loop will run 5 times.

**EXAMPLE**

#include<iostream>

using namespace std;

int main()

{

int a, b;

cout<<"Enter the value of 'a'"<<endl;

cin>>a;

cout<<"Enter the value of 'b'"<<endl;

cin>>b;

for(int i=0; i<5; i++)

{

if(i==0)

{

a=a+b;

cout<<"The value of 'a' after a=a+b is = "<<a<<endl;

}

else if(i==1)

{

a-=b;

cout<<"The value of 'a' after a=a-b is = "<<a<<endl;

}

else if(i==2)

{

a\*=b;

cout<<"The value of 'a' after a=a\*b is = "<<a<<endl;

}

else if(i==3)

{

a/=b;

cout<<"The value of 'a' after a=a/b is = "<<a<<endl;

}

else

{

a%=b;

cout<<"The value of 'a' after a=a%b is = "<<a<<endl;

}

}

}

**OUTPUT**

**Enter the value of 'a'**

**20**

**Enter the value of 'b'**

**3**

**The value of 'a' after a=a+b is = 23**

**The value of 'a' after a=a-b is = 20**

**The value of 'a' after a=a\*b is = 60**

**The value of 'a' after a=a/b is = 20**

**The value of 'a' after a=a%b is = 2**

If we change the test condition and increase the stop value to say for example 10, the program will output 10 lines of code but one of the lines which is the last in the else statement will be repeated several times. See the printout below:

**Enter the value of 'a'**

**20**

**Enter the value of 'b'**

**3**

**The value of 'a' after a=a+b is = 23**

**The value of 'a' after a=a-b is = 20**

**The value of 'a' after a=a\*b is = 60**

**The value of 'a' after a=a/b is = 20**

**The value of 'a' after a=a%b is = 2**

**The value of 'a' after a=a%b is = 2**

**The value of 'a' after a=a%b is = 2**

**The value of 'a' after a=a%b is = 2**

**The value of 'a' after a=a%b is = 2**

**The value of 'a' after a=a%b is = 2**

### 9.4.12 Write C++ code that will use a for loop with a non-sequential counter variable.

For loop can be used with what is known as range-based values. In this instance, its non-sequential. The loop runs and iterates over a given range. This is common when working with collection of items.

Syntax of range-based loops

for ( declaration : range ) statement;

**EXAMPLE**

#include <iostream>

using namespace std;

int main() {

int num\_array[] = {3, 6, 9, 12, 15};

for (int n : num\_array) {

cout << n << " ";

}

return 0;

}

**OUTPUT**

**3 6 9 12 15**

In the above example, we declared an array called num\_array of integer type holding numbers 3,6,9, 12 and 15. The array has 5. As you can see, our loop declaration is different from the ones we have been doing from 9.44 to 9.4.11.

# FORMATIVE ASSESSMENT INDIVIDUAL TASK

9.4.1 Which of the following depicts the correct syntax of for loop? (1)

1. for(initialization; condition; increment/decrement)
2. for(increment/decrement; initialization; condition)
3. for(initialization, condition, increment/decrement
4. None of These

9.4.1 A. for(initialization; condition; increment/decrement) (1)

9.4.2 Consider the program below and answer the question which follows:

#include <iostream>

using namespace std;

int main()

{

int i=0,x=0;

for(i=1;i<10;i\*=2)

{

x++;

cout<<x;

}

cout<<x;

return 0;

}

What will be the output of the code above?

9.4.2 The output will be: 12344

9.4.3 Briefly explain the main difference between a for loop and a while loop. (4)

9.4.3 For loops are used when the number of iterations is known, whereas while loops are used when the user only knows the condition at which the program must stop/exit the loop. (4)

9.4.4 With examples, explain what effect an empty semicolon will have on the for loop. (2)

9.4.4 if we place a semicolon after nothing, you will create a statement. Thus, the semicolon by itself is a statement, which is called the empty statement or the null statement. For example, the following statement will do nothing:

for(;;);{

cout<<"NCV";

}

However, if we remove the semicolon after the loop declaration, we will generate an endless loop printing the word NCV. (2)

9.4.5 Explain using examples the difference between a break and a continue statement when used in for loops. (6)

9.4.5 A break statement terminates the loop and a continue statement will skip a statement and then carry on with execution of loop statements until the end of the loop as specified by the test condition. Here is an example:

**break statement**

for(int i=0; i<=10; i++){

if (i==4){

break;

}

cout<<i;

}

**OUTPUT**

**0123**

**continue statement**

for(int i=0; i<=10; i++){

if (i==4){

continue;

}

cout<<i;

}

**OUTPUT**

**01235678910**

**NB** when break is used, the program terminates after printing up to 3. When continue is used the program prints 0 to 10 except 4. (6)

9.4.6 Consider the following code which is supposed to print multiples of 3 starting from 50 to 0. The program below has errors.

#include <iostream>

using namespace std;

int main()

{

for (int i=50; i>=0; i++){

if (i/3==0){

cout<<i<<endl;

break;

}

}

return 0;

}

Rewrite the code so that it will print the correct multiples of 3 as follows:

**48 45 42 39 36 33 30 27 24 21 18 15 12 9 6 3**

(5)

9.4.6 Program to print multiples of 3 from 50 down.

#include <iostream>

using namespace std;

int main()

{

for (int i=50; i>0; i--){

if (i%3==0){

cout<<i<< " ";

}

}

return 0;

}

# 9.5. Nested loops

Sometimes, there are situations where we want to display multidimensional data, and this is easily achieved using nested loops. Using nested loop statements, we can iterate several times with more than one condition and return different values depending on the results. Nested loops can be confusing at times but the fundamental aspect that should be understood is the execution flow of inner and outer loops. We will explain what an inner loop and an outer loop are in section 9.5.2.

### 9.5.1 Define the term nested loop

**VOCABULARY**

A nested loop is a loop statement within another loop statement.

### 9.5.2 Define the terms inner and outer loops

An inner loop is a loop statement which is inside another loop statement. The inner loop gets executed first and then outer loop.

**Syntax of a nested while loop**

**while(condition)**

**{**

**while(condition)**

**{**

**// set of statement of inside while loop**

**}**

**//set of statement for outer while loop**

**}**

### 9.5.3 C++ code that will nest identical type loops (Max 3 levels)

### 9.5.4 C++ code that will nest different type loops (Max 3 levels)

### 9.5.5 Determine the application flow when nested loop statement is encountered

When a loop starts, the inner loop's collection of statements is executed first. When the first condition is met, it will return inside and complete the second condition. If that condition becomes false, it tries to fulfil all of the conditions in the outer loop. This is how the flow continues. It is also not necessary for the loop to be placed within another loop of the same type. It can be any loop, which means it can be contained within any other loop. Figure 9.8 illustrates the sequence of statements in a nested loop.

A picture containing diagram

Description automatically generated

Figure 9. 8: Nested loop structure

# 9.6. Program flow and control

Learning Outcomes:

Students should be able to

Content:

* Break and continue statements

### 9.6.1 Explain what a break statement is used for

### 9.6.2 Explain what the continue statement is used for.

### 9.6.3 Implement a break or continue statement as required as part of a code solution.

TASK

Write a C++ program using for loops

SOLUTION

#include<iostream>

using namespace std;

int main()

{

for(int outer=6; outer>=1; outer--)

{

for(int s=1; s<=6-outer; s++)

cout<<" ";

for (int innermost=1; innermost<=outer; innermost++)

{

cout<<"\*";

}

cout<<endl;

}

}

SUMMATIVE ASSIGNMENT INDIVIDUAL WORK

9.8.1 Which loop is guaranteed to execute at-least once?

1. for loop
2. while loop
3. **do-while loop**
4. None

9.8.1 C

9.8.2 What will be the output of the following code?

#include <iostream>  
  
 using namespace std;  
   
 int main() {  
 for (int i = 1; i <= 5; ++i) {  
 cout << i << " ";  
 }  
 return 0;  
 }

1. 1 3 4 5
2. 1 2 3 4 5
3. 5 4 3 2 1
4. 4 5 1 2 3

9.8.2 B. 1 2 3 4 5

9.8.3 Which of the loop is also called counter loop?

1. for  
   (b) do while  
   (c) while  
   (d) All of these

9.8.3 A for

9.8.4 A for loop contains three expression: initialization, condition, and \_\_\_\_\_\_\_.

1. Increment
2. Decrement
3. Increment/Decrement
4. Validate

9.8.4 C

9.8.5 Semicolon is placed at the end of condition in \_\_\_\_\_\_\_\_\_\_\_.

1. for loop
2. while loop
3. do while loop
4. Nested Loop

9.8.5 C do…while loop

9.8.6 Which of the following will not generate an error

1. for(int i = 0: i<=10: i++)
2. for(int i = 0; i<=10, i++)
3. for(int i = 0; i<=10; i++);
4. for(int i = 0, i<=10; i++)

9.8.6 C for(int i = 0; i<=10; i++);

9.8.6 How many times is a do while loop guaranteed to loop?

1. 0
2. Infinitely
3. 1
4. Variable
   * 1. C 1

9.8.7 Write a C++ program using a for…loop to print alphabets from a to z in lowercase.

9.8.7 Solution

/\*\*

\* C++ program to print all alphabets from a to z lowercase

\*/

#include<iostream>

using namespace std;

int main()

{

//declaring variable

char ch;

cout<<"Alphabets from a - z are: \n";

//Loop with starting value, test expression and incrementor

for(ch='a'; ch<='z'; ch++)

{

//printing the results

cout<<" "<<ch;

}

return 0;

}

9.8.8 What is the difference between a while and a do while loop? Explain with examples. (6)

9.8.8 While…loop is a pretest loop and do… loop is a post test loop. The structure is different: (4)

**while (**expression**)**  
**{statements;}**

**do {**statement;}

**while(**condition**);**

(2)

9.8.9 Write a C++ program using while loops to print a number in reverse. The program must allow a user to input numbers (10)

**9.8.9Solution:** Program to print a number in reverse

/\*\*

\* C program to find reverse of a number

\*/

#include <iostream>

using namespace std;

int main()

{

int num, reverse = 0;

/\* Input a number from user \*/

cout<<"Enter any number to find reverse: ";

cin>>num;

/\* Repeat the loop until value of num== 0 \*/

while(num != 0)

{

/\*

\* Increase place value of reverse and

\* add last digit to reverse

\*/

reverse = (reverse \* 10) + (num % 10);

/\* Remove last digit from 'num' \*/

num =num/ 10;

}

cout<<"Reverse number is : "<< reverse<<endl;

return 0;

}

**OUTPUT**

**Enter any number to find reverse: 951**

**Reverse number is : 159**

(10)

# References

<https://byjusexamprep.com/difference-between-for-loop-and-while-loop-i>