# 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.

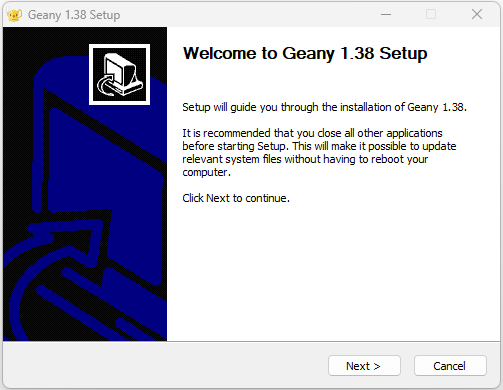


Figure 5. 2: Geany welcome screen

**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.

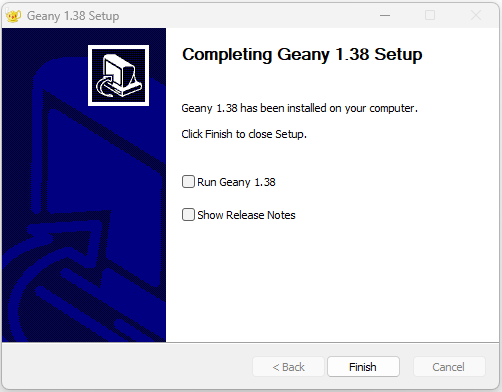


Figure 5. 4: Complete setup

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.

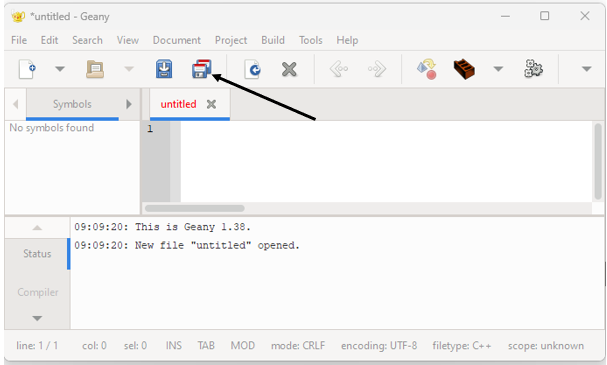


Figure 5. 5:Geany start-up window

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.

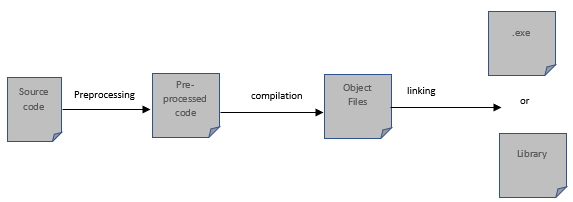


Figure 5. 7: C++ compilation

### 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.

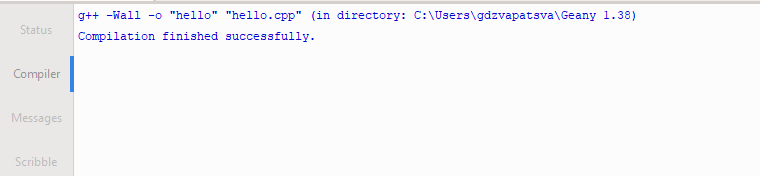


Figure 5. 9:Message window

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

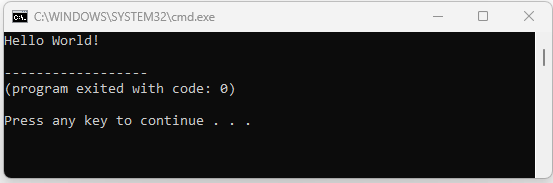


Figure 5. 12:C++ Output

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

{

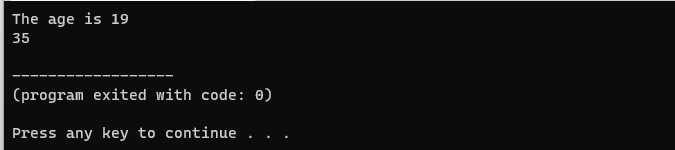
//calling a block of code

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.

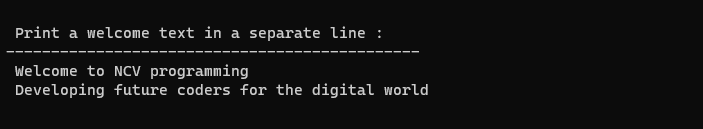
**Importance of comments**

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";

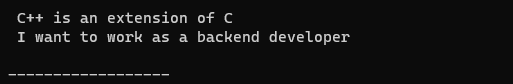
}

}

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++.

When placed at the end of a line, the backslash character (\) continues the line. You must type two backslashes in a row (\\) in order for a backslash to appear as a literal character.

**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 characters are used characters which starts with a backslash and then a symbol or sequence of symbols. They are used to format out for instance inserting a backspace, horizontal tab, newline and many others.

### 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 It is important to note that manipulation functions are specially designed to be used in conjunction with the insertion (<<) and extraction (>>) operators on stream objects. A manipulator changes the formatting parameters of streams and inserts or extracts 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()-**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 shown in 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.

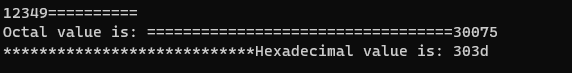


Figure 6. 1: Manipulator output

# 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/>