# Introduction to Technical Programming

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

7.1.

C/C++ Keyboard Input

Content:

* cin object
* Validation concepts

Learning Outcomes:

Students should be able to

7.1.1 Explain the process that happens when the computer encounters the scanf or cin statement

7.1.2 Use the C++ cin to read user input (numeric) and store the result in a variable

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

7.1.4 Define the term validation

7.1.5 Explain why user input validation is important.

7.2. C/C++ Errors

Content:

* Syntax errors
* Logic Errors

Learning Outcomes:

Students should be able to

7.2.1 Define the term syntax

7.2.2 Define the term syntax error

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

7.2.4 Explain the relationship between syntax and compile time errors

7.2.5 Define the term typographical error

7.2.6 Explain when a typographical error is a syntax error

7.2.7 Define the term logic error

7.2.8 List and explain three common side effects of logic errors

7.3.

Math library

Content:

* Errors in code
* Math Library
* Preprogramed functions

Learning Outcomes:

Students should be able to

7.3.1 Math Library

7.3.1.1 Explain the concept of a preprogramed function

7.3.1.2 Explain the need for preprogramed math functions

7.3.1.3 Explain the purpose of the math.h header file

7.3.1.4 Write C/C++ code that includes the header file math.h

7.3.2 C++ Errors

7.3.2.1 Define the term syntax error

7.3.2.2 Identify and correct syntax errors in C++ code

7.3.2.3 Define the term logic error

7.3.2.4 List and explain three common side effects of logic errors

7.3.3 Write C++ code that makes use of the following common mathematical functions

7.3.3.1 abs(n)

7.3.3.2 pow(n,n2)

7.3.3.3 sqrt(n)

7.3.3.4 sin(n)

7.3.3.5 cos(n,n2)

7.3.3.6 tan(n)

7.3.4 List the mathematical functions covered and the return type they will produce

7.3.5 Write C/C++ code containing complex mathematical expressions and math functions

7.4. Validation

Content:

* Validating Numeric input
* Validation concepts

Learning Outcomes:

Students should be able to

7.4.1 Define the term validation

7.4.2 Explain why user input validation is important

7.4.3 Write C++ code that will validate numeric input ensuring the input is within a pre- defined range

7.4.4 Write C++ code that will validate character input ensuring the expected input is one of 3 (max) characters

7.5. Symbolic Constants

Content:

* Constants

Learning Outcomes:

Students should be able to

7.5.1 Define the concept of a constant value

7.5.2 List common constants found in everyday life

7.5.3 Define the term magic numbers as used by programmers

7.5.4 Explain the advantage of using a constant for magic numbers when programming

7.5.5 Define the term symbolic name

7.5.6 Write C++ code that uses the const declaration qualifier to mark a value as constant

# Topic 8: Selection control structure

8.1 Relational expressions

Content:

* Relational operators

Learning Outcomes:

Students should be able to

8.1.1 Define the term relational expression

8.1.2 Explain the anatomy of a simple relational expression

8.1.3 List and explain the different C++ relational operators

8.1.4 Identify valid relational expressions

8.1.5 Identify invalid relational expressions

8.1.6 Explain the numeric result that is generated from a- relational expression

8.1.7 Write C++ code that will save the result of relation expression that contains both or a mix of numeric variables and literals in an int variable

8.1.8 Write C++ code that will save the result of relation expression that contains both or a mix of numeric variables and literals in a Boolean variable

8.1.9 Explain how the integer result produced by a relation expression relates to bool true/false

8.1.10 Write C++ code that will save the result of relation expression that contains both or a mix of character variables and literals in an int and bool variable

8.2 Selection Statement

Content:

If statement

Else statement

Learning Outcomes:

Students should be able to

8.2.1 Explain the purpose of the if statement in C++

8.2.2 Explain the purpose of the else statement in C++

8.2.3 Determine the program flow when an if statement is encountered

8.2.4 Identify or correct the general form for an if-else statement

8.2.5 Write C++ code that will use relational expressions in if-else statements (Max 2 two nested levels only)

8.2.6 Write C++ code that will use relational expressions in if-else statements with compounded content (Max 2 two nested levels only)

8.2.7 Explore the concept of block scope when writing if-else statements with compounded content

8.3 Logical Operators

Content:

Using logical operators

Learning Outcomes:

Students should be able to

8.3.1 Define the term logic operator

8.3.2 List the different logic operators

8.3.3 Explain how each logic operator will influence the result of an expression

8.3.4 Write C++ code that will save the result of relation expression that contains both or a mix of numeric variables, character variables and literals and a single logical operator in an int variable

8.3.5 Write C++ code that will save the result of relation expression that contains both or a mix of numeric variables, character variables and literals and a single logical operator in a bool variable

8.3.6 Write C++ code that will save the result of relation expression that contains both or a mix of numeric variables, character variables and literals and multiple (Max 3) logical operator in a bool variable

8.3.7 Write C++ code that will save the result of relation expression that contains both or a mix of numeric variables, character variables and literals and multiple (Max 3) logical operator in an int variable

8.4 Selection Statements

Learning Outcomes:

Students should be able to

Content:

* Nested If statement
* Else statement
* Logical operator based if statement

8.4.1 Define the term nested if statement

8.4.2 Determine the application flow when a nested if statement is encountered

8.4.3 Identify or correct the general form for a nested if-else statement

8.4.4 Write C++ code that will use relational expressions in nested if-else statements (Max 3 levels)

8.4.5 Write C++ code that will use relational expressions in nested if-else statements with compounded content (Max 3 levels)

8.4.6 Write C++ code that will use relational expression containing logic operators in nested if-else statements (Max 3 levels, Max 3 logic operators per level)

8.4.7 Write C++ code that will use relational expressions containing logic operator in nested if-else statements with compounded content (Max 3 levels, Max 3 logic

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

## C++ Keyboard Input

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

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

7.1. C++ Keyboard Input

**Introduction**

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

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

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

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

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

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

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

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

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

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

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

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

**Syntax of cin**

cin>>variable

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

**EXAMPLE 7.1**

#include <iostream>

using namespace std;

int main()

{

string f\_name="";

int age=0;

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

cin>>f\_name;

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

cin>>age;

return 0;

}

Output

Please enter your firstname

Tshepo

Please enter your age

19

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

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

**Syntax**

scanf (const char\* format, ….)

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

Non-whitespace characters except % consume one identical character from the input stream. If the next character in the stream is not equal, it can cause the function to fail.

**Example 7.2**

#include <cstdio>

#include <iostream>

using namespace std;

int main()

{

char f\_name [20];

int age=0;

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

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

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

scanf("%d", &age);

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

cout << "Age = " << age;

return 0;

}

**OUTPUT**

Please enter your firstname

Angie

Please enter your age

18

Firstname = Angie

Age = 18

**scanf vs cin**

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

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

Reading numeric input is easy with cin.

Syntax

cin>>variable\_name

Let us take for example, accepting age. We will have to declare a variable to store the age e.g:

int int\_age=0;

To accept the age, we would then code it as below:

//declaring age

int int\_age;

//Text to display before accepting age

cout << "Please eneter age" << endl;

//accepting input

cin>>int\_age;

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

**EXAMPLE 7.3**

using namespace std;

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

int main()

{

//declaring multiple variables

int num1, num2,answer=0;

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

//accepting multiple variables

cin>>num1>>num2;

//displaying the values from the two respective variables

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

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

return 0;

}

Output

**Please enter the two numbers**

**98 65**

**The firs number is 98**

**The second number is 65**

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

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

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

**EXAMPLE 7.4**

#include <iostream>

using namespace std;

int main() {

char grade = 'A';

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

return 0;

}

Output

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

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

char grade = 'Ann-Mary';

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

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

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

### 7.1.4 Define the term validation

**VOCABULARY**

The term 'validation' refers to examining or verifying data before it is incorporated into a computer system. Through this process, data is protected from being compromised or corrupted during input.

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

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

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

**Types of data validation**

* **Format validation**- makes sure the information is submitted in the proper format. Such as MM-DD-YYYY for date.
* **Data type validation**-Makes sure that data is captured in correct data type defined.
* **Range validation**- Ensures the values fall within a [range](https://www.computerhope.com/jargon/r/range.htm) limit eg age for a person is at most is 122 at the time of writing this book.

# FORMATIVE ASSESSMENT 7.1 INDIVIDUAL TASK

7.1.1 Consider the following code snippet below:

int n; float x = 3.8;

n = int(x);

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

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

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

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

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

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

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

**[Total =13 marks]**

## 7.2. C++ Errors

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

**List of different types of errors**

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

### 7.2.1 Define the term syntax

**VOCABULARY**

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

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

* Communication
* Code integration
* Consistency
* Clarity

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

### 7.2.2 Define the term syntax error

**VOCABULARY**

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

An example of a syntax error would be incorrect spelling of reserved words as shows below:

integer my\_age;

When you compile the program, you will get the following error message in the output window:

error: 'integer' does not name a type|

With a syntax error, the program will not compile.

Another example would be to switch the variable and the data type for instance:

age int ;

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

The code presented in example 7.5 illustrates some syntax errors.

**EXAMPLE 7.5**

// C++ program to illustrate syntax error

#include <iostream>

using namespace std;

int main()

{

int x = 10;

integer y = 15; //integer is not valid

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

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

}

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

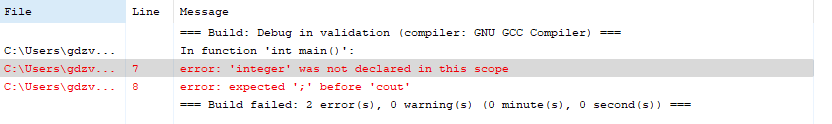
****

Figure 7. 1: Syntax errors

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

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

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

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

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

num1+num2=answer;

**Syntax errors**

The above snippet will generate the following error:

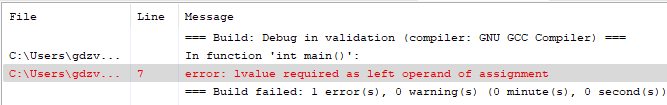


Figure 7. 2:Semantic error

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

### 7.2.5 Define the term typographical error

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

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

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

### 7.2.7 Define the term logic error

VOCABULARY

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

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

**EXAMPLE 7.6**

// C++ program to illustrate syntax error

#include <iostream>

using namespace std;

int main()

{

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

//error in calculating average

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

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

}

**Output**

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

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

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

=15+26+19+4

Average =64 This answer is wrong

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

The program should be entered as follows:

**EXAMPLE 7.7**

// C++ program to illustrate syntax error

#include <iostream>

using namespace std;

int main()

{

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

float average;

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

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

}

Output

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

**NOTE**

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

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

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

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

**EXAMPLE 7.8**

#include <iostream>

using namespace std;

int main()

{

int a = 1;

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

return 0;

}

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

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

Let us consider the following example.

**EXAMPLE 7.9**

int add(int y, int x)

{

return x + y;

}

int main()

{

int x{ 7 };

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

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

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

return 0;

}

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

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

**Define the term runtime error**

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

**EXAMPLE 7.10**

#include <iostream>

#include <cmath>

using namespace std;

int main()

{

int x = 52;

int y = 0;

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

return 0;

}

**OUTPUT**

**52 divided by 0 is**

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

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

Other causes of runtime errors include:

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

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

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

**EXAMPLE 7.11**

#include <iostream>

using namespace std;

int arr[5]; //declaring an array

int main()

{

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

cout<<ans<<endl;

return 0;

}

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

Here is an example:

#include <iostream>

using namespace std;

int main() {

int arr[1000000000];

return 0;

}

After running the program, you get the following error:

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

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

**Linker Error**

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

**EXAMPLE 7.12**

// C++ program to illustrate linker error

#include <iostream>

#include <bits/stdc++.h>

using namespace std;

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

{

int number\_1;

cin>>number\_1;

cout << " "<< number\_1;

}

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



Figure 7. 4:Linker Error

# FORMATIVE ASSESSMENT 7.2 INDIVIDUAL TASK

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

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

7.2.3 Consider the code below:

#include <iostream>

#include <cmath>

using namespace std;

int main()

{

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

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

int area=0;

area=length \* width;

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

return 0;

}

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

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

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

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

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

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

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

**[Total =16 marks]**

## 7.3. Math library

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

### 7.3.1 Math Library

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

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

**Syntax**

max(x,y)

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

Syntax

min(x,y)

**EXAMPLE 7.13**

#include <iostream>

using namespace std;

int main()

{

cout<<"The largest number between "<<11<<" and " << 12<< " is "<<max(11,12)<<"\n";

cout<<"The smallest number between "<<11<<" and " << 12<< " is "<<min(11,12)<<"\n";

return 0;

}

**OUTPUT**

**The largest number between 11 and 12 is 12**

**The smallest number between 11 and 12 is 11**

In the above example, we used min and max function without calling cmath or math.h header file.

**NOTE**

User-defined functions are functions that are not pre-defined but are created by a programmer.

#### 7.3.1.2 Explain the need for preprogramed math functions

Preprogramed functions are quite helpful to the programmer as they save a lot of time is software development. Instead of the programmer recoding the mathematical expressions to do a simple complex calculation like power, square root and average minimum and maximum, to mention a few, the programmer can only call this function from a defined class. In C++, the programmer has to include the cmath library in order to be able to do these calculations. Let us look at the following example to calculate the square root of a number: C++ uses the keyword sqrt when calling the pre-programmed function for square root.

**EXAMPLE 7.14**

#include <iostream>

#include <cmath> //math.h has been depreciated in C++

using namespace std;

int main()

{

double number=0;

cout<<"Please enter the number to calculate the square root"<<"\n";

cin>>number;

cout<<"The square root of "<<number<< " is " <<sqrt(number)<<"\n";

return 0;

}

**Output**

**Please enter the number to calculate the square root**

**81**

**The square root of 81 is 9**

We could have written a longer mathematical expression to find the square root but we simply called sqrt.

#### 7.3.1.3 Explain the purpose of the math.h header file

The math.h header defines various mathematical functions and one macro. All the functions available in this library take double as an argument and return double as the result. math. h is the deprecated C header. cmath is the C++ header. The difference is that cmath puts all the names in the std namespace. For the purposes of this tutorial, we are going to stick to cmath.

#### 7.3.1.4 Write C++ code that includes the header file cmath

We are going to make use of cmath header file and apply pre-programmed functions to solve some mathematical expressions.

We are going to calculate volume of a cube.

Volume of a cube is the total cubic units occupied by it, in a three-dimensional space. A cube is a 3d-shape, that has six faces, twelve edges and eight vertices. Hence, the volume of a cube is the space enclosed by its six faces. Therefore, the volume of cube is equal to product of its length, width and height. It is measured in cubic units. Figure 7.3 shows how a cube looks like.

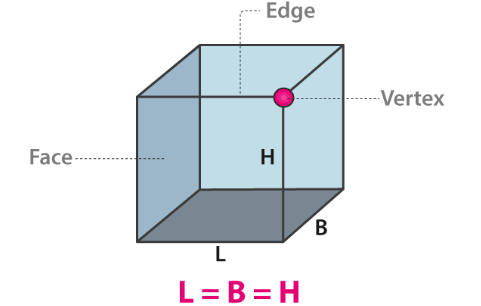


Figure 7. 3: A cuboid [https://byjus.com/volume-of-cuboid-calculator/]

**Volume of a Cube = Length × Width × Height**

Volume = a × a × a

Volume = a3

**Write a program to accept one side of the cube and calculate its volume.**

**EXAMPLE 7.15**

Solution

#include <iostream>

#include <cmath>

//program to calculate volume of a cube.

using namespace std;

int main()

{

double side=0, volume=0;

//Accepting input

cout<<"Please enter size of one side of the cube"<<"\n";

cin>>side;

if (cin.fail()){

cout<<"Enter a valid size of the side"<<"\n";

}

else{

//processing

volume = pow(side, 3);

cout<<"The volume of a cube whose side is "<<side<< "cm is " <<volume<<"cm3"<<"\n";

return 0;

}

}

**Sample Output**

**Please enter size of one side of the cube**

**9**

**The volume of a cube whose side is 9cm is 729cm3**

**Please enter size of one side of the cube**

**bbbbbb**

**Enter a valid size of the side**

In the example, we implemented cmath header file which allowed us to use pow function.

**TASK 7.1**

Write a C++ program using inbuilt function for cube root to **f**ind the length of the edges of the cube, if its volume is equal to 125 cm3. Use cin.fail( ) to trap invalid entry for volume.

**TASK 7.2** :**Write a C++ Program to raise any number X to power N.**

### 7.3.2 Write C++ code that makes use of the following common mathematical functions

So, we have covered some mathematical functions found in the cmath library. Let us explain more functions.

7.3.2.1 abs(n)- The abs() function in C++ returns the absolute value of the argument. Mathematically, abs(num) = |num|.

**EXAMPLE 7.16**

#include <iostream>

#include <cmath>

using namespace std;

int main() {

// get absolute value of -8.5

cout << abs(-8.5);

return 0;

}

// Output: 8.5

abs(x) Parameters

The abs() function takes the following parameter:

x - a floating point number whose absolute value is returned. It can be of the following types:

* + double
  + float
  + long double

#### 7.3.2.2 pow(x,y)

#### The pow() method returns the result of raising the first argument by the second argument's power.

pow(x, y) = xy

**Syntax**

pow(double x, double y)

pow( ) Parameters

The pow() function takes two parameters:

x - the base value

y - exponent of the base

The pow( ) function returns: 1.0 if exponent is zero and 0.0 if base is zero

**EXAMPLE 7.17**

#include <iostream>

#include <cmath>

using namespace std;

int main() {

cout << pow(8,2);

return 0;

}

**Output**

64

#### 7.3.2.3 sqrt(x)

#### The sqrt() function in C++ returns the square root of a number. Mathematically, sqrt(x) = √x

**EXAMPLE 7.18**

#include <iostream>

#include <cmath>

using namespace std;

int main() {

cout << "Square root of 100 = ";

// print the square root of 100

cout << sqrt(100);

return 0;

}

Output

10

#### 7.3.2.4 sin(x)

The sine of an angle (argument) provided in radians is returned by the sin() function in C++. The sin() function takes a single mandatory argument in radians.

**EXAMPLE 7.19**

#include <iostream>

#include <cmath>

using namespace std;

int main()

{

double x = 0.475965, result;

result = sin(x);

cout << "sin(x) = " << result << endl;

return 0;

}

OUTPUT

sin(x) = 0.458196

#### 7.3.2.5 cos(x,y)

When given an angle (argument) in radians, the cos() function in C++ gives the angle's cosine.

**EXAMPLE 7.20**

#include <iostream>

#include <cmath>

using namespace std;

int main()

{

double x = 0.8, result;

result = cos(x);

cout << "cos(x) = " << result << endl;

return 0;

}

**OUTPUT**

**cos(x) = 0.877583**

As can be seen from the example, the cos() function takes one parameter and returns the value in the range of **[-1, 1]**. The returned value is either in double, float, or long double.

#### 7.3.2.6 tan(n)

The tan() function in C++ returns the tangent of an angle (argument) given in radians. The only required argument for the tan() method is in radians (can be positive, negative, or 0). The tan() function returns the value in the range of **[-∞, ∞]**.

**EXAMPLE 7.21**

#include <iostream>

#include <cmath>

using namespace std;

int main()

{

long double x = 0.89947, result;

result = tan(x);

cout << "tan(x) = " << result << endl;

return 0;

}

OUTPUT

tan(x) = 1.25879

### 7.3.4 List the mathematical functions covered and the return type they will produce

Table 7.1 summarises the mathematical functions covered in this section.

Table 7. 1::Mathematical functions

|  |  |  |  |
| --- | --- | --- | --- |
| **Function** | **Description** | **Example** | **Return Type** |
| abs(x) | Returns the absolute value of x | abs(-5.5)=5 | int |
| pow(x) | returns the value of x to the power of y (xy): | pow(6,3)=216 | int |
| sqrt(x) | Returns the value of square root of x | sqrt(81)=9 | int |
| ceil(x) | rounds x number upwards to its nearest integer | ceil(5.1)=6 | int |
| floor(x) | rounds x number downwards to its nearest integer, and returns the result | floor(5.1)=5 | int |
| sin(x) | Returns the sine of x (x is in radians) | sin(0.956421)= 0.817134 | double |
| cos(x) | Returns the cosine of x | cos(0.7)= 0.764842 | double |
| tan(n) | Returns the tangent of an angle | tan(0.54754)= 0.609726 | double |
| cbrt(x) | returns the cube root of the given argument | cbrt(125)=5 | int |
| remainder() | computes the floating-point remainder of numerator/denominator (rounded to nearest). | remainder(5/2.13) =0.74 | float |

NOTE

The return type depends largely on the data type of result and the input. For example:

cout << cbrt(512.6) << endl;

output

8.00312

If we declare our output as int then the output that will be displayed is of int data type.

int result=cbrt(512.6);

cout << result << endl;

output

8

### 7.3.5 Write C/C++ code containing complex mathematical expressions and math functions

In this section, we are going to use one of the common problems which require complex mathematical expressions. We are going to calculate compound interest.

**Question: Write a C++ Program to Calculate Compound Interest**

So, what is compound interest?

It is the addition of interest to the principal sum of a loan or deposit, or in other words, interest on interest. It is the result of reinvesting interest, rather than paying it out, so that interest in the next period is then earned on the principal sum plus previously accumulated interest.

It may be contrasted with simple interest, where interest is not added to the principal, so there is no compounding.

**Annual compound interest formula**

The formula for annual compound interest, including principal sum, is:

A = P (1 + r/n) (nt)

Where:

*A = the future value of the investment/loan, including interest  
P = the principal investment amount (the initial deposit or loan amount)  
r = the annual interest rate (decimal)  
n = the number of times that interest is compounded per year  
t = the number of years the money is invested or borrowed for*

**NOTE**

Note that this formula gives you the future value of an investment or loan, which is compound interest plus the principal. Should you wish to calculate the compound interest only, you need this:

*Total compounded interest = P (1 + r/n) (nt)– P*

This program will read principal, rate and time in years and then print compound interest on entered principal for given time period.

**EXAMPLE 7.22**

**Solution**

/\* C++ Program to Calculate Compound Interest \*/

#include<iostream>

#include<cmath>

using namespace std;

int main()

{

//declaring variables

float principal,rate,time,comp\_int;

//Accepting input

cout<<"Enter Principle (Amount) in Rands:: ";

cin>>principal;

cout<<"\nEnter Rate of Interest :: ";

cin>>rate;

cout<<"\nEnter Time Period in years:: ";

cin>>time;

//processing

comp\_int = principal\*pow((1+rate/100),time);

//output

cout<<"\nThe total compound interest is = R"<< comp\_int<<"\n";

return 0;

}

SAMPLE OUTPUT

Enter Principle (Amount) in Rands:: 285000

Enter Rate of Interest :: 12.75

Enter Time Period in years:: 4

The total compound interest is = R460586

# FORMATIVE ASSESSMENT 7.3 INDIVIDUAL TASK

7.3.1 What is the difference between the cmath and math,h header files. (4)

7.3.2 Complete the table below with pre-programmed functions

|  |  |  |
| --- | --- | --- |
| **Function** | **Description** | **Example** |
| abs(x) |  |  |
| pow(x) |  |  |
| sqrt(x) |  |  |
| ceil(x) |  |  |
| floor(x) |  |  |
| cbrt(x) |  |  |
| remainder() |  |  |

(14)

7.3.3 Write a C++ code to solve the following expression:

(a+b)^2

Accept a and b from the keyboard.

Use a=2.4, b=4.5

Display the output. (10)

7.3.4 What is the value of answer in the following C++ program?

int x=4;

int answer=pow(x,2);

cout<<answer<<endl;

(2)

**[Total =30 marks]**

## 7.4. Validation

### 7.4.1 Define the term validation

### 7.4.2 Explain why user input validation is important

Please refer to Section 7.14 and 7.1.5

### 7.4.3 Write C/C++ code that will validate numeric input ensuring the input is within a pre- defined range

In section 7.1.5, we listed the different types of validation as:

* Range check validation
* Form validation
* Data type validation.

In this section, we are going to implement data type validation and range validation.

**Range validation check**

Let’s assume we want to run a program that allows users to capture their age. We do know that age should be greater than 0 and less than 123. We can implement a while loop. We have not covered loops in detail, but using your knowledge of loops covered when working with Arduino should be sufficient. Example 7.23 illustrates range validation check.

**EXAMPLE 7.23**

#include <iostream>

using namespace std;

int main() {

//variable declaration

int age=0;

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

cin>>age;

//range check

while(age < 0 || age > 123) {

cout << "Invalid age, re-enter: ";

cin >> age;

}

return 0;

}

**Output**

**Please enter your age**

**-23**

**Invalid age, re-enter: -23**

**Invalid age, re-enter: 45**

Looking at the code above, you will notice that if the age entered is less than 0 and greater than 123 the system will prompt the user to capture the age again. So, we can say we are sorted with the range check but another problem arises when a user captures a string. You for the variable age. You notice the program continues. Here is how we can resolve this problem.

**cin Functions**

**cin.fail()** - This function returns true when an input failure occurs. In this case it would be an input that is not an integer. If the cin fails, then the input buffer is kept in an error state.

The code in example 7.24 shows the use of cin.fail().

**EXAMPLE 7.24**

#include <iostream>

using namespace std;

int main()

{

int age;

cout << "Enter number :" << endl;

cin >> age;

if(cin.fail())

{

cout << "Not a number " << endl;

}

else{

cout << "is a number " << endl;

}

return 0;

}

**Output**

**Enter number :**

**A**

**Not a number**

**Enter number :**

**45**

**is a number**

**So, we can modify our earlier program on age range as shown in example 7.25:**

**EXAMPLE 7.25**

#include <iostream>

using namespace std;

int main()

{

int age;

cout << "Enter number :" << endl;

cin >> age;

if(cin.fail())

{

cout << "Not a number " << endl;

}

else{

cout << "is a number " << endl;

//range check

while(age < 0 || age > 123) {

cout << "Invalid age, re-enter: ";

cin >> age;

}

}

return 0;

}

**cin.clear()** - This is used to clear the error state of the buffer so that further processing of input can take place. This ensures that the input does not lead to an infinite loop of error message display.

**cin.ignore()** - This function is used to ignore the rest of the line after the first instance of error that has occurred, and it skips to or moves to the next line.

Example 7.26 shows the implementation of cin.clear(), cin.ignore()

**EXAMPLE 7.26**

#include <iostream>

using namespace std;

int main()

{

int age;

cout<<"Enter the number "<<endl;

while(!(cin >> age)) {

cin.clear();

cin.ignore(1000, '\n');

cout << "The data entry failed to meet the requirements"<<endl;

}

return 0;

}

Output

Enter the number

fgg

The data entry failed to meet the requirements

What is important is that if we capture a value like “12abc”, the program will discard “abc” and extract the number. However, if the user captures the value as abc12, an error message will be displayed.

7.4.4 Write C++ code that will validate character input ensuring the expected input is one of 3 (max) characters

The char data type is used to store a single character. The character must be surrounded by single quotes, like 'A' or 'c':

**isalpha()**

We can start by implementing the isalpha() function. The isalpha() function in C++ checks if the given character is an alphabet or not.  Here is an example to validate if a character entered through the keyboard is an alphabet.

**EXAMPLE 7.27**

/\* isalpha example \*/

#include <iostream>

using namespace std;

int main ()

{

char chars, chars2;

cout<<"Enter alphabet"<<endl;

cin>>chars;

//checking for alphabetic character

if (isalpha(chars)){

cout<<chars<< "is an alphabetic character"<<endl;

}

else {

cout<<chars<< "is not an alphabetic character"<<endl;

}

//displaying the ASCII equivalent of the entered value

cout<<int(chars)<< " is the ASCII equivalent of "<<chars<<endl;

return 0;

}

**OUTPUT**

**Enter alphabet**

**M**

**Mis an alphabetic character**

**77 is the ASCII equivalent of M**

In the above example, we also casted the char entered using int to check its ASCII equivalent. This is quite important if you want specific character you can specify their ASCII values as well.

**ispunct()-** Checks if character is a punctuation character. See example 7.28.

**EXAMPLE 7.28**

/\* ispucnt example \*/

#include <iostream>

using namespace std;

int main ()

{

char ch;

cout<<"Enter the character"<<endl;

cin>>ch;

cout<<ispunct(ch)<<endl;

return 0;

}

**OUTPUT**

**Enter the character**

**?**

**16**

**Enter the character**

**D**

**0**

If the character is not a punctuation, the above program returns zero; for example, D returned 0. If ch is a punctuation character, a value other than zero (i.e., true) is returned. Our program returned 16 for ? which is a non-zero value to indicate the entered character is a punctuation mark.

Problem: Write a program in C++ to allow users to enter gender and greet them accordingly. If a user enter ‘m’ or ‘M’ the program must print:

Hello sir. How are you?

If the users enters ‘f’ or ‘F’ the program must print

Hello madam. How are you?.

If the users enter ‘O’ or ‘o’ the program must print.

Hello. How are you?

See the solution in Example 7.29.

**EXAMPLE 7.29**

//Checking gender entries

#include <iostream>

using namespace std;

int main ()

{

char gender;

cout<<"Please select your gender from the following [M/F/O] or [m/f/o]"<<endl;

cin>>gender;

gender=toupper(gender);

if (gender=='M'){

cout<<"Hello sir. How are you?"<<endl;

}

else if (gender=='F'){

cout<<"Hello madam. How are you?"<<endl;

}

else if (gender=='O'){

cout<<"Hello. How are you?"<<endl;

}else

{

cout<<"Incorrect entry"<< endl;

}

return 0;

}

**OUTPUT**

**Please select your gender from the following [M/F/O] or [m/f/o]**

**M**

**Hello sir. How are you?**

**Please select your gender from the following [M/F/O] or [m/f/o]**

**f**

**Hello madam. How are you?**

**Please select your gender from the following [M/F/O] or [m/f/o]**

**/**

**Incorrect entry**

In the given example we also use the function toupper().

toupper()-checks whether character is alphabetic & converts to upper case

tolower()checks whether character is alphabetic & converts to lower case

The only challenge we have with our program above is that it accepts non alphabetic characters. We can modify the program a bit by adding a while loop as follows:

**EXAMPLE 7.29**

/\* isalpha and toupper example \*/

#include <iostream>

using namespace std;

int main ()

{

char gender;

cout<<"Please select your gender from the following [M/F/O] or [m/f/o]"<<endl;

cin>>gender;

gender=toupper(gender);

while (isalpha(gender)){

if (gender=='M'){

cout<<"Hello sir. How are you?"<<endl;

}

else if (gender=='F'){

cout<<"Hello madam. How are you?"<<endl;

}

else if (gender=='O'){

cout<<"Hello. How are you?"<<endl;

}else

{

cout<<"Incorrect entry"<< endl;

}

return 0;

}

cout<<"You did not enter an alphabetic character"<< endl;

}

**OUTPUT**

**Please select your gender from the following [M/F/O] or [m/f/o]**

**8**

**You did not enter an alphabetic character**

**NOTE**

When we were validating integers, we just used cin.fail(). In this instance it will not work if a user enters a number because is will be interpreted as an ASCII character.

# FORMATIVE ASSESSMENT 7.4 INDIVIDUAL TASK

7.4.1 Define the term validation as applied in C++ programming. (2)

7.4.2 Explain why user input validation is important. (2)

7.4.3 List three types of data validation. (3)

7.4.4 In your own words, explain what you understand by the term range check as applied in C++ programming. (3)

7.4.5 Write a C++ program to allow a user to enter a test mark. The program should report an error if the mark is outside the range of 0-100. If the entry is not a number the program must report an error and tell user to re-enter a valid mark. If the mark entered is valid and greater than 49, the program must display a message which says:

Congratulations. “You passed the test”

else the program display:

“Unfortunately. You failed the test” (12)

**[Total =22 marks]**

## 7.5. Symbolic Constants

### 7.5.1 Define the concept of a constant value

Constants refer to fixed values that the program may not alter, and they are called literals. Unlike variables, constants never change in value. You must initialize a constant when it is created. C++ has two types of constants: literal and symbolic.

A literal constant is a value typed directly into your program wherever it is needed. For example, consider the following statement:

int width =7;

This statement assigns the integer variable width the value 7. The 7 in the statement is a literal constant. You can't assign a value to 7, and its value can't be changed. Another example would be the values true and false, which are stored in bool variables, also are literal constants.

A symbolic constant is a name that is given to a constant value. Constant variables are one type of symbolic constant, as a variable has a name (its identifier) and a constant value. The const keyword precedes the type, name, and initialization.

**Syntax**

*const data\_type name =value;*

Example const double PI=3.142;

Once declared, the value of PI can not be assigned a new value. Example 7.29 is a program to calculate area of a circle.

**EXAMPLE 7.29**

#include <cmath>

#include <iostream>

using namespace std;

int main() {

//declaring pi using const keyword

const double PI =3.142;

double area=0, radius=0;

cout<<"Please enter the radius of the circle "<<endl;

cin>>radius;

area=PI\*pow(radius, 2);

cout<<"The area of a circle whose radius is " << radius<< " is " <<area<<"cm^2"<<endl;

return 0;}

**OUTPUT**

**Please enter the radius of the circle**

**5**

**The area of a circle whose radius is 5 is 78.55cm^2**

So, for argument sake, let us try to add the following line in our code to change the value of PI.

PI=5.321;

When we try to compile the program we get an error saying”

|error: assignment of read-only variable 'PI'|

Well-named symbolic constants also make a program more understandable. Constants often are fully capitalized by programmers to make them distinct from variables. This is not required by C++, but the capitalization of a constant must be consistent because the language is case sensitive.

There's another way to define constants that dates to early versions of the C language, the precursor of C++. The preprocessor directive #define can create a constant by specifying its name and value, separated by spaces:

**Syntax**

*#define name value*

Example

#define PI 3.142

Take note that there is no semi-colon, as this is inserted as part of the header files. In addition, the constant does not have a type such as int or char. The #define directive enables a simple text substitution that replaces every instance of PI in the code with 3.142. The compiler sees only the result. Because these constants lack a type, the compiler cannot ensure that the constant has a proper value.

We are going to modify our earlier code on area and use the #define preprocessor to declare our constant PI.

**EXAMPLE 7.30**

#include <cmath>

#define PI 3.142

#include <iostream>

using namespace std;

int main() {

//declaring pi using const keyword

double area=0, radius=0;

cout<<"Please enter the radius of the circle "<<endl;

cin>>radius;

area=PI\*pow(radius, 2);

cout<<"The area of a circle whose radius is " << radius<< " is " <<area<<"cm^2"<<endl;

return 0;

}

**OUTPUT**

**Please enter the radius of the circle**

**5**

**The area of a circle whose radius is 5 is 78.55cm^2**

If we try to reassign PI with a new value, the compiler reports a different error this time:

|error: lvalue required as left operand of assignment|

The <cmath> header of C++ also includes several mathematical constants that can be used in mathematical and quantitative code. These can be used by adding #define directive and specify a macro “\_USE\_MATH\_DEFINES”. This macro is to be added to the program before we include the <cmath> library.

**EXAMPLE 7.31**

#define \_USE\_MATH\_DEFINES

#include <cmath>

#include <iostream>

using namespace std;

int main() {

cout<<"The value of PI is "<<M\_PI<<endl;

return 0;

}

**OUTPUT**

**The value of PI is 3.14159**

For more pre-defined constants, refer to the following ling:

<https://learn.microsoft.com/en-us/cpp/c-runtime-library/math-constants?view=msvc-170>

### 7.5.2 List common constants found in everyday life

The following are examples of constants in our everyday life:

* distance from earth to sun
* Boiling point of water
* Freezing point of water

Discussion

What other constants do you know? Share your answers with your teacher.

### 7.5.3 Define the term magic numbers as used by programmers

A magic number is a numeric literal that is used in the code without any explanation of its meaning. The use of magic numbers makes programs less readable and hence more difficult to maintain and update. Here is an example:

*Example:*

double salary = 500 \* workedhours;

// what is the meaning of 500?

It is better to define symbolic names, so called **constants**, and use these instead of numeric literals. This can be rewritten as:

// definition of a constant SALARY\_PER\_HOUR

final double SALARY\_PER\_HOUR = 500;

// calculating total salary

final double salary = SALARY\_PER\_HOUR \* workedhours

It is now clear that rather than using the value 500 in the program itself. In addition, we added the word final which is mainly used to restrict class inheritance. At this level, we are not going to talk much about inheritance, a feature of the object-oriented paradigm. Here is a link with more clarity on object-oriented programming.

### 7.5.4 Explain the advantage of using a constant for magic numbers when programming

Two key advantages for using constants for magic numbers

* ***Readability*** of the program: an identifier of a constant with a significant name is much more readable than a magic number (e.g., SALARY\_PER\_HOUR is self-explanatory compared to just the value 500
* ***Modifiability of the program***: If the value has to be changed, its only done in the declaration and will be changed throughout the program.

### 7.5.5 Define the term symbolic name

Symbolic names can be used in C++ for various data items used by a programmer in his program. A symbolic name is a known identifier. The identifier is a sequence of characters taken from the C++ character set.

### 7.5.6 Write C/C++ code that uses the const declaration qualifier to mark a value as constant

Write a C++ program that can calculate the price of a product after 15% VAT. For this program, a user needs to enter two inputs, which are price per unit and quantity of a product. The program calculates the price of the product using the following formula:

Total\_price= Price\_per\_unit x Quantity\_of\_product

The program must add the VAT chargeable on the product which is the Total\_price +15%

After calculating the calculating the total price and tax, the program would display the total price without tax, tax\_amount and the total price with tax amount added.

**EXAMPLE 7.32**

#include <iostream>

#include <cmath>

using namespace std;

int main()

{

//declaring variables

double total\_price=0, price\_per\_item=0, quantity\_of\_products=0;

//declaring a constant value for tax

const float vat=0.15;

double vat\_amount=0, grand\_total=0;

//Accepting input

cout<<"Enter the unit price of the item"<<endl;

cin>>price\_per\_item;

cout<<"Enter the number of items bought"<<endl;

cin>>quantity\_of\_products;

//calculating total price

total\_price=price\_per\_item\* quantity\_of\_products;

cout<<"The total price of " <<quantity\_of\_products <<" items with a unit price of "<<price\_per\_item<<" is "<<total\_price<<endl;

//calculating tax amount

vat\_amount=total\_price\*vat;

cout<<"The tax amount for " <<total\_price <<" is R"<<vat\_amount<<endl;

grand\_total=total\_price+vat\_amount;

//displaying the grand total

cout<<"The grand total price is R"<<grand\_total<<endl;

return 0;

}

**OUTPUT**

**Enter the unit price of the item**

**10**

**Enter the number of items bought**

**10**

**The total price of 10 items with a unit price of 10 is 100**

**The tax amount for 100 is R15**

**The grand total price is R115**

# FORMATIVE ASSESSMENT 7.5 INDIVIDUAL TASK

7.5.1 Define the term constant as used in C++ programming. (2)

7.5.2 List two ways to declare constants in C++. (2)

7.5.3 Write code to declare a const to hold distance of 257. (2)

7.5.4 Define the term magic number as used in programming. (2)

7.5.5 Explain the TWO advantages of using a constant for magic numbers when programming. (4)

7.5.6 Consider the code snippet below:

const int LENGTH = 21;

char message[LENGTH];

cout << "Enter a sentence on the line below." << endl;

cin >> message;

cout << message << endl;

Suppose that in response to the prompt, the user types the following line and presses Enter: We are enjoying code.

What will the output of the code fragment look like and give a reason for your answer? (3)

7.5.7 Write a C++ program to calculate the volume of a sphere whose radius is entered through the keyboard by the user. Ensure that the input is validate for entry.

Volume= **4/3 πr3**  (10)

**[Total =25 marks]**

# SUMMATIVE ACTIVITY 7.6 INDIVIDUAL TASK

7.6.1 Which of the following is known as the insertion operator in C++?

1. [<<](javascript:void(0);)
2. [>>](javascript:void(0);)
3. [~](javascript:void(0);)
4. [^^](javascript:void(0);)

7.6.2 Which of the following is known as exertion operator in C++?

1. [<<](javascript:void(0);)
2. [>>](javascript:void(0);)
3. [~](javascript:void(0);)
4. [^^](javascript:void(0);)

7.6.3 Define the term runtime error. (2)

7.6.4 What is the correct output of given code snippets?

a) 10

b) Compile -time error

c) Runtime error

d) Linker error

7.6.5 Consider the following code

#include <iostream>

#include <iomanip>

using namespace std;

int main()

{

const double PI = 3.14159265;

cout<< PI <<endl;

}

Modify the code to print the following output: 3.141593

7.6.6 Write a program in C++ to find the area of any triangle using Heron's Formula. Heron’s formula is used to calculate area of a triangle when you are given the length of the three sides. The formula is as follows:

A = √s(s−a)(s−b)(s−c)

NB: The program should display an error message if the sides are not valid entries and if the area can not be calculated i.e., nan. For example

Find the area of any triangle using Heron's Formula :

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

Input the length of 1st side of the triangle :

9

Input the length of 2nd side of the triangle :

100

Input the length of 3rd side of the triangle :

2

valid The sides of the triangle are not valid :

(15)

7.6.7 Define the term typographical error as applied in programming. (2)

**[Total=26marks]**

# References

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

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

<https://www.w3schools.com/cpp/cpp_math.asp>

# Topic 8: Selection control structure

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

* Define the term relational expression
* Explain the anatomy of a simple relational expression
* List and explain the different C++ relational operators
* Identify valid relational expressions
* Identify invalid relational expressions
* Explain the numeric result that is generated from a- relational expression
* Write C++ code that will save the result of relation expression that contains both or a mix of numeric variables and literals in an int variable
* Write C++ code that will save the result of relation expression that contains both or a mix of numeric variables and literals in a Boolean variable
* Explain how the integer result produced by a relation expression relates to bool true/false
* Write C++ code that will save the result of relation expression that contains both or a mix of character variables and literals in an int and bool variable
* Students should be able to
* Explain the purpose of the if statement in C++
* Explain the purpose of the else statement in C++
* Determine the program flow when an if statement is encountered
* Identify or correct the general form for an if-else statement
* Write C++ code that will use relational expressions in if-else statements (Max 2 two nested levels only)
* Write C++ code that will use relational expressions in if-else statements with compounded content (Max 2 two nested levels only)
* Explore the concept of block scope when writing if-else statements with compounded content
* Define the term logic operator
* List the different logic operators
* Explain how each logic operator will influence the result of an expression
* Write C++ code that will save the result of relation expression that contains both or a mix of numeric variables, character variables and literals and a single logical operator in an int variable
* Write C++ code that will save the result of relation expression that contains both or a mix of numeric variables, character variables and literals and a single logical operator in a bool variable
* Write C++ code that will save the result of relation expression that contains both or a mix of numeric variables, character variables and literals and multiple (Max 3) logical operator in a bool variable
* Write C++ code that will save the result of relation expression that contains both or a mix of numeric variables, character variables and literals and multiple (Max 3) logical operator in an int variable
* Define the term nested if statement
* Determine the application flow when a nested if statement is encountered
* Identify or correct the general form for a nested if-else statement
* Write C++ code that will use relational expressions in nested if-else statements (Max 3 levels)
* Write C++ code that will use relational expressions in nested if-else statements with compounded content (Max 3 levels)
* Write C++ code that will use relational expression containing logic operators in nested if-else statements (Max 3 levels, Max 3 logic operators per level)
* Write C++ code that will use relational expressions containing logic operator in nested if-else statements with compounded content (Max 3 levels, Max 3 logic
* operators per level)

# 8.1 Relational expressions

**Introduction**

One of the core functions is to evaluate relational expressions. We will use logic operators, relational operators, and selection statements to accomplish this. Selection is a programming construct that deals with the execution of statements based on conditions. This is what we will discuss in this topic.

### 8.1.1 Define the term relational expression

A relational expression indicates the condition that the system evaluates. The defined operation and the values of the operands or expressions that are compared at execution time determine the outcome of the evaluation of the relational expression in every case.

### 8.1.2 Explain the anatomy of a simple relational expression

In many applications, it is necessary to test the values of variables and then execute various statements based on the outcome. This option allows you to choose between different actions to take. Using the relational operators is the easiest and most typical technique to create such an expression. If the relation is true, it returns 1 whereas if the relation is false, it returns 0. Here is a code snippet of the relational expression.

int num1=7, num2=8;

cout << (num1<num2)<< endl;

When the code is compiled, the output will be a 1 to indicate true

### 8.1.3 List and explain the different C++ relational operators

We discussed and listed the relational operators in topic 6. For the purposes of reinforcement, we are going to list them again below:

= = Is equal to 8==10 gives false

!= not equal to 8!=10 gives true

> Greater than 8>10 gives false

< less than 8<10 gives true

>= Greater than or equal to 8>=10 gives false

<= Less than or equal 8<=10 gives true

Whenever we want to do conditional statements, we will have to make use of boolean or relational operators. Here is an example of equality(==) to compare strings

### 8.1.4 Identify valid relational expressions

The following are examples of valid relational expressions:

* age==19
* flag==done
* 2.0<3.0
* TAX<=0.15
* Marital\_status==’M’
* BMI>=24.9

### 8.1.5 Identify invalid relational expressions

The following are examples of invalid relational expressions in C++

* width=<50 Incorrect symbol used. The symbol should be <=
* 2.0>>3.0 Invalid relational operator
* flag = = done Spcaes not allowed
* age=>45 Invalid symbol used
* true <= false This will give an error as true and false are boolean types and we cannot use <= operator for comparing the boolean values.
* (height=1.8) Incorrect symbol. Used assignment operator (=) instead of equality operator (==)

### 8.1.6 Explain the numeric result that is generated from a- relational expression

When working with relational expressions, the outcome is a boolean value. The expression returns a 1 or a 0. A 1 represents true while a 0 represents false. The boolean integer can be converted into an alphabetic one, as we discussed earlier in topic 6. In such a case, we implement boolalpha as follows:

int num1=7, num2=8;

cout <<boolalpha<<(num1<num2)<< endl;

The code snippet above will return true. If, for any reason, we want to show the integer values, we can use the noboolalpha function.

### 8.1.7 Write C++ code that will save the result of relation expression that contains both or a mix of numeric variables and literals in an int variable

The program below compares a person’s age with the legal\_Age of majority, which is set to 18 and declared as a constant. The relational expression is evaluated, with the result stored in an int data type.

**EXAMPLE 8.1**

//program to check if a person's age is over the legal\_age of majority

#include <iostream>

using namespace std;

int main()

{

//declaring variables

double age=17.5;

//declaring constant

const int legal\_Age=18;

//evaluating a relational expression and storing answer in an integer

int result=(age>legal\_Age);

//displaying output

cout<<result<<endl;

return 0;

}

OUTPUT

0

We can always typecast the output and prefix it with a boolalpha statement so that the output can read as true for 1 or false for 0. The output statement is shown below:

cout<<boolalpha<<(bool)result<<endl;

8.1.8 Write C++ code that will save the result of relation expression that contains both or a mix of numeric variables and literals in a Boolean variable

In  example 8.1, our program stored the result as an int data type. This is not a problem because relational expressions are evaluated to yield a numerical result of 1 or 0. The idea is to check whether the expression results in true or false. However, C++ produces values of 1 or 0. In this program, we will use the same example but store the result in a bool data type, as shown in example 8.2.

**EXAMPLE 8.2**

//program to check if a person's age is over the legal\_age of majority

#include <iostream>

using namespace std;

int main(){

//declaring variables

double age=17.5;

//declaring constant

const int legal\_Age=18;

//evaluating a relational expression and storing answer in an integer

bool result=(age>legal\_Age);

//displaying output

cout<<boolalpha<<result<<endl;

return 0;

}

### 8.1.9 How the integer result produced by a relation expression relates to bool

When evaluating a relational expression, the outcome is boolean. However, C++ displays 1 for true and 0 for false. Using the boolalpha function can result in a 1 or 0 being displayed as true or false. This has been demonstrated in examples 8.1 and 8.2.

### 8.1.10 C++ code that will save the result of relation expression that contains both or a mix of character variables and literals in an int and bool variable

This section will show you how to store the expression's result in int and bool variables. The program used to explain the concept compares a user's age to the legal aid majority and also verifies citizenship status. If the user's age is greater than or equal to the legal age of majority and their citizenship status is 'C,' the program should print that they are eligible to vote; otherwise, it should display that they are not. Example 8.3 demonstrates the given scenario.

**EXAMPLE 8.3**

//program to check if a person's age is over the legal\_age

//and citizen status checked

#include <iostream>

using namespace std;

int main(){

char cit\_status;

bool isCitizen, isNotCitizen;

cout<<"enter citizen status [C=citizen or N=Non-citizen]"<<endl;

cin >>cit\_status;

isCitizen= cit\_status=='c' || cit\_status=='C';

isNotCitizen=cit\_status=='n' || cit\_status=='N';

//declaring variables

double age=27.5;

//declaring constant

const int legal\_Age=18;

//evaluating a relational expression and storing answer in an integer

int result=(age>legal\_Age);

if (result==isCitizen){

cout<<"You are eligible to vote"<<endl;

}

else if (result==isNotCitizen){

cout<<"You are NOT eligible to vote"<<endl;

}

else{

cout<<"Check cit\_status entry"<<endl;

}

return 0;

}

O**UTPUT**

**enter citizen status [C=citizen or N=Non-citizen]**

**c**

**You are eligible to vote**

**enter citizen status [C=citizen or N=Non-citizen]**

**N**

**You are NOT eligible to vote**

It is important to note that the first expression for age has the output stored in an int variable called result. The output of the citizenship status is stored in bool variables called isCitizen and isNotCitizen. The final expression is evaluated, with result being an int type and isCitizen/isNotCitizen being a bool type.**FORMATIVE ASSESSMENT 8.1 INDIVIDUAL TASK**

8.1.1 Define the term relational expression. (2)

8.1.2 List any FIVE examples of relational operators. (5)

8.1.3 Give THREE examples of invalid relational operators (3)

8.1.4 In C++, both ! and != are relational operators. [True /False] (1)

8.1.5 Evaluate the expression (7 + 8 <= 15) (2)

8.1.6 Evaluate the expression ('a' >'A') (2)

8.1.7 In C++, all relational operators are evaluated before logical operators [True/False] (2)

8.1.8 The result of a relational expression can not be assigned to an int type. [True/False]

8.1.9 What is the output of the following code?

if (6 > 8)

{cout << " \*\* " << endl ; }

else if (9 == 4) {

cout << "\*\*\*" << endl;}

else {

cout << "\*" << endl;

}

(2)

8.1.10 Consider the code below which is supposed to print PASS if y is equal to x.

#include <iostream>

using namespace std;

int main()

{

int x = 7;

int y = 10;

if(y = x) {

cout<< "PASS"<<endl;

}

return 0;

}

When compiled the program is giving pass even though its incorrect. Correct the program. (2)

8.1.11 Rewrite the following program to print either alphabetic answer True or False depending on the expression. Avoid use of if statement. (4)

**[Total 27 marks]**

# 8.2 Selection Statement

**Introduction**

Programming is more than just calculations and expressions. It is also necessary to include decision and control statements, which specify when statements should be executed. While we briefly discussed control statements in Topic 3, most of the code programs we've worked with so far are linear constructs, meaning that statements were executed sequentially or in the order in which they were declared. In this topic, you will learn in detail how to change the control flow of a program through branching statements. By implementing branching statements, you can determine whether one section of code will be executed or not based on a conditional clause. The term "selection statement" is also used to refer to branching statements, conditional statements, or decision-making statements in C++. We use selection statements to select parts of a program to be executed if a certain condition is met. Selection statements can be executed using:

1. if statements
2. if …. else ….
3. if...else if...else ……. (Multiple if statements)
4. if……{if ….{}} (Nested if statements)
5. switch statement

These statements will be discussed in the subsequent sections.

### 8.2.1 Explain the purpose of the if statement in C++.

An "if" statement evaluates a specific condition; if the condition is true, action is taken; otherwise, action is not taken. Action can be thought of as a statement or set of statements. Here is an example. Create a program that asks students for their arrival time at college. They receive house points if they arrive before the starting time. If the condition is met, house points are awarded; otherwise, no action is taken if the arrival time is later than the starting time.

Figure 8.1 depicts the given scenario using a flowchart.

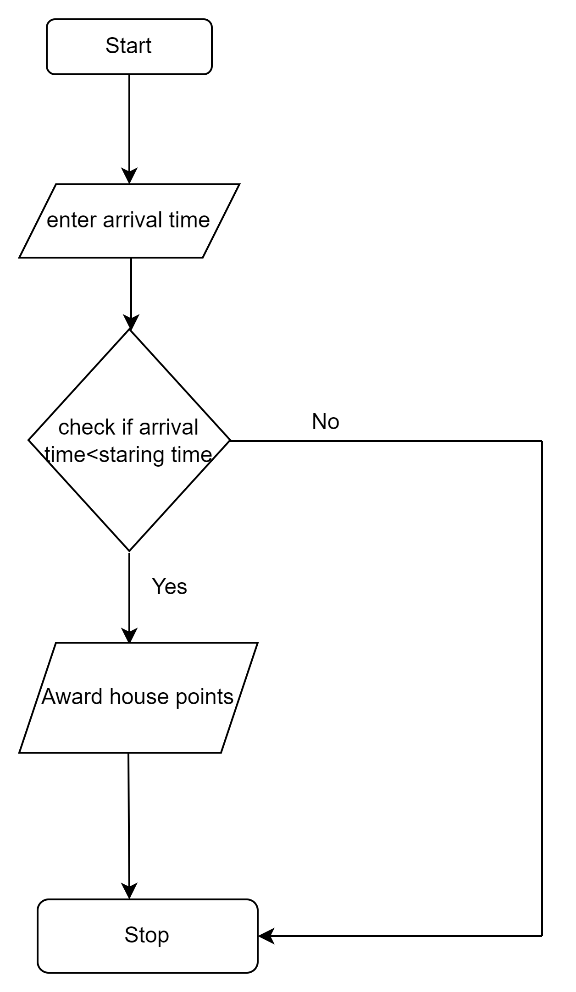


Figure 8. 1:Simple if statement flowchart

The if statement's syntax, or general form, is as follows:

if(expression)

{

statement(s);

}

The if statement evaluates the condition inside the parentheses ( ). Here, a statement may be a single statement, a block of statements, or nothing (in the case of an empty statement). The expression must be enclosed in braces {}.

* If the condition evaluates to true, i.e., a nonzero value, the code inside the body {…..} of if is executed. Otherwise
* If the condition evaluates to false, the code inside the body {….} of if is skipped.

Let us consider the following case where we need to apply the if statement.

**EXAMPLE 8.4**

**Write a program to accept an integer number and display if it’s a positive number.**

**Solution**

/\*Program to print positive number entered by the user

If the user enters a negative number, it is skipped \*/

#include <iostream>

using namespace std;

int main() {

int number=0;

//accepting a number

cout << "Enter an integer: "<<endl;

cin >> number;

//condition to check if the number is positive

if (number > 0) {

//statement to be displayed if condition is met

cout << number << " :is a positive number" << endl;

}

//optional

cout << "This statement is always executed.";

return 0;

}

**OUTPUT**

**Enter an integer:**

**63**

**63 :is a positive number**

**This statement is always executed since its outside the if block.**

**Enter an integer:**

**-365**

**This statement is always executed since its outside the if block**

Notice on the program, when we entered 63, the statement inside the braces was printed out. However, when we entered a negative value of -365, nothing was printed except the statement which is outside the if block. This is because we never instructed our program what to print if the condition was not met. Another example of an if statement is shown in Example 8.5.

**EXAMPLE 8.5**

Write a program to enter a number and print if it is even number.

#include <iostream>

using namespace std;

int main()

{

int num=0;

cout<<"Enter a number of your choice"<<endl;

cin>>num;

if(num%2==0){

cout<<num<< " is an even number." <<endl;

}

//optional

cout << "This statement is always executed since its outside the if block";

return 0;

}

**OUTPUT**

**Enter a number of your choice**

**77**

**This statement is always executed since its outside the if block**

**Enter a number of your choice**

**88**

**88 is an even number.**

**This statement is always executed since its outside the if block**

### 8.2.2 Explain the purpose of the else statement in C++

We discussed the simple if statement in section 8.2.1, which had one outcome based on the condition. If the condition is not satisfied, we note that the simple if statement code would just exit the if block. Often, it is desirable for a program to take one branch if the condition is true and another if it is false. This can be achieved by adding an else block to the simple if statement.

**Syntax**

 if(*boolean-expression* )  
       *statement-1*;  
   else  
       *statement-2*;

So we are going to modify the program to print even numbers. We want to say if the condition is not satisfied, the program prints "num" is a negative number." Figure 8.2 depicts the if... else... logic in the form of a flowchart.

A picture containing diagram

Description automatically generated

Figure 8. 2:if ...else....

So let us do the C++ code to check if a number is positive or negative.

**EXAMPLE 8.6**

//Program to print positive or negative number entered by the user

#include <iostream>

using namespace std;

int main() {

int number=0;//accepting a number

cout << "Enter an integer: "<<endl;

cin >> number; //condition to check if the number is positive

if (number > 0) {

//statement to be displayed if condition is met

cout << number << " :is a positive number" << endl; }

else{

cout << number << " :is a negative number" << endl; }

cout << "This statement is always executed.";

return 0;}

**OUTPUT**

**Enter an integer:**

**-78**

**-78 :is a negative number**

**This statement is always executed.**

**Enter an integer:**

**1235**

**1235 :is a positive number**

### 8.2.3 Determine the program flow when an if statement is encountered

As we can see from the examples in Section 8.2.2, selection statements are executed based on the fulfilment of conditions. The path that the program takes depends on the condition that has been met. If the condition is met, the statements that come after the "if" () {} statements are executed. If not met, the program will execute the statements which come after the else statements.

### 8.2.4 The general form for an if-else statement

The if-else is statement is an extended version of If. The general form of if-else is as follows:

if (test-expression)

{

True block of statements

}

else

{

False block of statements

}

Statements;

If the test-expression value is true in this sort of construct, the true block of statements will be performed. The false block of statements will be executed if the test-value expression is false. Nonetheless, upon execution, control will be immediately passed to the statements outside the If block.

**TASK 8.1:**

Write a C++ program to calculate the total cost of a product purchased in an electrical shop given the following conditions:

A shop will give discount of 10% if the cost of purchased quantity is more than 1000.  
The program must allow the user to enter the quantity of items.  
Suppose, one unit will cost R100-00.  
Display whether the customer qualifies for discount and print total cost for user.

### 8.2.5 Using relational expressions in if-else statements (Max 2 two nested levels only)

In some circumstances, using if...else is insufficient because another condition might exist that does not match any of the criteria stated in the if or else block. Let's examine the example to determine whether a number is positive or negative. The program will run the statements in the else block if we do not validate the input type during data entry. Here's an illustration in example 8.7.

**EXAMPLE 8.7**

//Program to print positive or negative number entered by the user

#include <iostream>

using namespace std;

int main() {

int number=0;

//accepting a number

cout << "Enter an integer: "<<endl;

cin >> number;

//condition to check if the number is positive

if (number > 0) {

//statement to be displayed if condition is met

cout << number << " :is a positive number" << endl;

}

else{

cout << number << " :is a negative number" << endl;

}

//optional

cout << "This statement is always executed.";

return 0;

}

**OUTPUT**

**Enter an integer:**

**kdfdsvgsda**

**0 :is a negative number**

**This statement is always executed.**

From the above example, we can see that the output is incorrect. The program used the statements in the else block. To solve the problem, we use if … else if… else block.

**Syntax**

if(condition1){

//code to be executed if condition1 is true

}else if(condition2){

//code to be executed if condition2 is true

}

else if(condition3){

//code to be executed if condition3 is true

}

...

else{

//code to be executed if all the conditions are false

}

To correct the program for checking positive or negative number using the if else if else block, the code will look as follows:

**EXAMPLE 8.8**

//Program to print positive or negative number entered by the user

#include <iostream>

using namespace std;

int main() {

int number=0;

//accepting a number

cout << "Enter an integer: "<<endl;

cin >> number;

//condition to check if the number is positive

if (number > 0) {

//statement to be displayed if condition is met

cout << number << " :is a positive number" << endl;

}

else if (number<0){

cout << number << " :is a negative number" << endl;

}

else{

cout << number << " :is an invalid entry for the program" << endl;

}

//optional

cout << "This statement is always executed.";

return 0;

}

**OUTPUT**

**Enter an integer:**

**uiryiu**

**0 :is an invalid entry for the program**

**This statement is always executed.**

**NOTE**

This rule applies whenever an if statement is followed by one or more else if statements; the final else if should be followed by an else statement. The requirement for a final else statement is defensive programming.

The else clause should either perform the necessary action or include a reasonable justification for why no action was performed.

**TASK 8.2**

Write a program in C++ to check triangle is an isosceles, equilateral, or scalene using if-else statement. The program must allow the user to enter the size of the three sides.

### 8.2.6 Using relational expressions in if-else statements with compounded content (Max 2 two nested levels only)

We have  been using mathematical expressions to explain if statements. Other expressions can be used if the expression evaluates to true or false. In this section, we are going to use characters using if-else if..... statements. A vowel is a sound such as the ones represented in writing by the letters 'a', 'e', 'i', ' o', and 'u', which you pronounce with your mouth open. A consonant is a speech sound that is articulated with complete or partial closure of the vocal tract. All non-vowel letters are consonants.

**EXAMPLE 8.9**

Write a program in C++ to Check if the input character is a vowel or a Consonant and display appropriate message. If the input is not an alphabetic character the program should also report and error.

SOLUTION

//Program to check whether input is a vowel or consonant

#include <iostream>

using namespace std;

int main() {

char chars;

bool isLowercaseVowel=false, isUppercaseVowel=false;

cout << "Enter an alphabet: "<<endl;

cin >> chars;

// values for lowercase vowels stored in a boolean variable

isLowercaseVowel = (chars == 'a' || chars == 'e' || chars == 'i' || chars == 'o' || chars == 'u');

// values for uppercase vowels stored in a boolean variable

isUppercaseVowel = (chars == 'A' || chars == 'E' || chars == 'I' || chars == 'O' || chars == 'U');

// show error message if c is not an alphabet

if (!isalpha(chars)){

cout<<"Error! The input is a non-alphabetic character."<<endl;

}

else if (isLowercaseVowel || isUppercaseVowel)

{

cout << chars << " is a vowel."<<endl;

}

else

{

cout << chars << " is a consonant."<<endl;

}

return 0;

}

**OUTPUT**

**Enter an alphabet:**

**A**

**A is a vowel.**

**Enter an alphabet:**

**g**

**g is a consonant.**

**Enter an alphabet:**

**5**

**Error! The input is a non-alphabetic character.**

In the above program, we made use of two boolean variables which will evaluate to true if the input is an uppercase or lowercase vowel. We also used the isalpha() function to test for alphabetic character input. If all these conditions are not met, then the input character is a consonant.

**Task 8.3**

Write a C++ program which displays the number of days in a month captured by the user. The user enters an integer to represent the month. January is 1—December is 12.

### 8.2.7 Explore the concept of block scope when writing if-else statements with compounded content

**VOCABULARY**

Block scope means a variable was defined within a block of code, such as a for loop or an if statement.

A block, or block statement (also known as a compound statement), is a collection of zero or more statements that the compiler treats as if it were a single statement. Blocks start with a symbol and terminate with a symbol, with the statements to be executed in between. Blocks can be used wherever a single statement is permitted.

When using if...else statements without braces for the statements, the true and false statements can only be one each; otherwise, if there are more than two, all of them will be performed. Below is an illustration of how to determine a person's eligibility to vote based on the age they entered.

**EXAMPLE 8.10**

#include <iostream>

using namespace std;

int main()

{

cout << "Enter your age (in years): ";

int age=0;

cin >> age;

if (age >=18)

cout << "You are old enough to vote"<<endl;

else

cout << "You are not old enough to be able to vote"<<endl;

cout << "Tough luck!! Try next time"<<endl;

return 0;

}

**OUTPUT**

**Enter your age (in years): 18**

**You are old enough to vote**

**Tough luck!! Try next time**

**Enter your age (in years): 12**

**You are not old enough to be able to vote**

**Tough luck!! Try next time**

Looking at the output of the above program, the code is not working as expected. The line which says "Tough luck!! Try next time" is printed in all cases whether the person is 18 and above or below. In the same way if we move it to the if block, the program will report a compilation error:

|error: 'else' without a previous 'if'|

To deal with these issues, statements which we want to be executed at the same time, we group them in the same block using braces. The code segment can be re-written as :

if (age >=18){

cout << "You are old enough to vote"<<endl;

}

else{

cout << "You are not old enough to be able to vote"<<endl;

cout << "Tough luck!! Try next time"<<endl;

}

Because blocks are regarded as a single sentence, this now works correctly:

Similarly, variables declared within a block are only available within that block. When the code snippet below is compiled, it will produce errors.

int age=0;

cin >> age;

if (age >=18)

{

int a=9;

cout<<a;

cout<<b;

}

else

{

cout<<a;

int b=5;

}

What are the errors. Discuss this with your teacher. Correct the code so that it can compile.

It is even possible to put blocks inside of blocks as shown in example 8.11.

**EXAMPLE 8.11**

#include <iostream>

using namespace std;

int main()

{ // block 1, nesting level 1

cout << "Enter an integer: ";

int value {};

cin >> value;

if (value > 0)

{ // block 2, nesting level 2

if ((value % 2) == 0)

{ // block 3, nesting level 3

cout << value << " is positive and even\n";

}

else

{ // block 4, also nesting level 3

cout << value << " is positive and odd\n";

}

}

return 0;

}

**OUTPUT**

**Enter an integer: 21**

**21 is positive and odd**

**Task 8.4**

A school has following rules for grading system:  
a. Below 25 - F  
b. 25 to 45 - E  
c. 45 to 50 - D  
d. 50 to 60 - C  
e. 60 to 80 - B  
f. Above 80 - A  
Write a C++ program to ask a user to enter marks and print the corresponding grade using if… else if…else statements. Make sure the data is validated.

# FORMATIVE ASSESSMENT 8.2 INDIVIDUAL TASK

8.2.1 Write the correct syntax of a simple IF statement in C++. (3)

8.2.2 Define the term selection/conditional statements. (2)

8.2.3 Explain the purpose of else statement in a selection statement. (2)

8.2.4 Write the general syntax of an if…else statement. (4)

8.2.5 Write a program which accepts three sides of a triangle. The program should first determine whether the sides constitute a valid triangle else print out an error message. If the sides are valid, determine what type of a triangle is it [scalene, isosceles, or equilateral triangle] (15)

8.2.6 Using if else statement, write a C++ program to to find maximum of three numbers. The program must allow user to enter the three numbers. Make sure the input is validated against incorrect type (10)

8.2.7 Write a C++ program to enter two Boolean numbers then, print phrase "A And B" if A and B equal to 1, or print phrase "A Or B" if A equal to 1 and B equal to 0. (10)

**[Total =46 marks]**

# 8.3 Logical Operators

Content:

Using logical operators

Earlier in topic 6 section 6.2, operators were covered extensively. We are now going to reinforce them and see how we implement them with selection statements.

Learning Outcomes:

Students should be able to

### 8.3.1 Define the term logic operator

Logical operators are used to check whether an expression is true or false. To determine whether an expression is true or false, logical operators are utilized. Logical operators are frequently used when writing test expressions that govern program execution. These expressions are also known as boolean expressions since they produce a boolean value or answer when evaluated, and the result is stored in a flag variable.

### 8.3.2 List the different logic operators

 There are three common logical operators that give a Boolean value by manipulating other Boolean operand(s) in C++ are:

1. && Logical AND
2. || Logical OR
3. ! Logical NOT

Other languages such as C#, Java, JavaScript and Swift do use the exact symbols for the same operators.

### 8.3.3 Explain how each logic operator will influence the result of an expression

Table 8. 1: Logical Operators

|  |  |  |
| --- | --- | --- |
| **Operator** | **Meaning** | **Example** |
| && | Logical AND. Returns true only if all the operands are true. | (1&&0) returns 0 (false) |
| || | Logical OR. True if at least one of the operands is true. | (1||0) returns 1 (false) |
| ! | Logical NOT. True only if the operand is false | !(1) returns 0 |

\* Remember, we discussed earlier in topic 7 that we can prefix the expression with the function boolalpha if we want to return the actual words like true or false.

Because the logical operators have a limited domain (input) and range (output), we can easily enumerate all conceivable combinations of inputs and outputs. A truth table is commonly used to summarize all potential combinations.

**Logical AND**

Table 8. 2: Logical AND truth table

|  |  |  |
| --- | --- | --- |
| **x** | **y** | **x&&y** |
| false | false | false |
| false | true | False |
| true | false | False |
| true | true | true |

Table 8.2 shows the evaluation of expressions when using a logical AND operator.

Here are some examples of expressions with logical operators. Observe how each influences the result of the expression.

num1 =25

num2 =37

(num1>10) && (num2 >30) evaluates to true because:

25 is greater than 10 so returns a 1

37 is greater than 30 so return a 1

1 and 1 is the same as (true && true) which results to true.

**Logical OR**

Here is a truth table to represent the logical OR expression.

Table 8. 3: Logical OR truth table

|  |  |  |
| --- | --- | --- |
| **x** | **y** | **x||y** |
| false | false | false |
| false | true | true |
| true | false | true |
| true | true | true |

Table 8.3 shows the evaluation of expressions when using a logical OR operator.

We will utilize logical operators to evaluate the following expressions using the same values of num1 and num2 as 25 and 37, respectively:

(num1>10) || (num2<30) evaluates to true because:

25 <10 returns 1(true), 37<30 returns 0 (false),

1OR 0 evaluates to 1.

Here is another example:

(num1<10) || (num2<30) evaluates to 0 because

Expression 1 returns 0, expression 2 returns 0, 0 OR 0 returns 0.

**Logical NOT**

Table 8. 4: Logical NOT truth table

|  |  |
| --- | --- |
| **x** | **!x** |
| false | true |
| true | false |

As seen in table 8.4, logical NOT returns true if the expression's value is false and false if the expression's value is true..

!(1) returns 0 (false)

!(0) returns (true)

Along the same lines, the following statement evaluates to false

cout << !('b') << endl;

This is because the char value is converted to its ASCII value. So not any number which is not 1 is 0.

cout << !(89) << endl; will evaluate to 0.

Combining logical AND and logical OR operators in the same statement is often unavoidable, but it is a risky domain. Many programmers assume that logical AND and logical OR have the same precedence as addition/subtraction and multiplication/division. Nevertheless, logical AND has a higher precedence than logical OR, so logical AND operators will be assessed ahead of logical OR operators (unless they have been parenthesized).

### 8.3.4 Write C++ code that will save the result of relation expression that contains both or a mix of numeric variables, character variables and literals and a single logical operator in an int variable

We can evaluate relational expressions with numeric values and characters. What will happen is that the program will convert the char to its ASCII value and then evaluate the expression.  For example:

(65>'#') will evaluate to 1(true). This is so because the value of # is 35 on ASCII table

(65>'H') will evaluate to false because H is 72 on ASCII table. So 67>72 is false.

We can add a binary literal as follows:

(65>'H')&& 0b11; This would evaluate to 0 (false) simply because:

65>72 is 0(false), 0b11 is a binary literal representing 3 and we already mentioned that a number not 0 is a 1 when using logical expression.

So ultimately, 0 && 1 is 0 which is false.

We can make use of an int variable to then store the result from the expression as shown below:

int result;

result=(65>'#');

cout << result << endl;

The output of the above code snippet will be 1. There is a downside to this. The program still returns a 1 when you use boolalpha to convert the output to boolean. However, if we had declared the result as bool, it would have worked as we prefer. We will discuss the boolean result in section 8.3.5. Example 8.12 code demonstrates using int variables to store result of an expression with characters.

**EXAMPLE 8.12**

Write a program to check whether a character is a digit or not. Store the result in an int data type.

**SOLUTION**

//C++ program to check for decimal digit characters using conditional operator

#include <iostream>

using namespace std;

#include <stdio.h>

int main() {

char character;

int is\_Digit;

cout<<"Enter a Character"<<endl;

cin>>character;

/\* Check, If input character is digit \*/

is\_Digit = ((character >= '0') && (character <= '9'))? 1 : 0;

if(is\_Digit == 1)

cout<<character<< " is a digit"<<endl;

else

cout<<character<< " is NOT a digit"<<endl;

return 0;

}

### 8.3.5 Write C++ code that will save the result of relation expression that contains both or a mix of numeric variables, character variables and literals and a single logical operator in a bool variable

As previously stated, the result of a logical expression is of the boolean type. The outcome is either true or false, making it easy to store and manipulate if boolean data type variables are used to store the outcome. Let us look at the previous example that we had in section 8.3.4, but this time we are going to store the output in a bool variable:

bool result;

result=(65>'#');

cout << result << endl;

If we run the code snippet above, we get our output as 1. Now, we can change it to its alphabetic representation which is true by using boolalpha. The code will look as follows:

**EXAMPLE 8.13**

#include <iostream>

using namespace std;

int main()

{

bool result;

result=(65>'#');

cout <<boolalpha<< result << endl;

return 0;

}

OUPUT

true

Remember, in the previous example where we stored the result in an int type, we mentioned that we will not be able to see the alphabetic equivalent when we use boolalpha.

### 8.3.6 Saving the result of relation expression that contains both or a mix of numeric variables, character variables and literals and multiple

In the following program, we save the range of vowels in a bool variable. The user's characters will be compared to the vowel range mentioned. Take notice that we stored uppercase A, B, and U as ASCII number equivalents and then typecasted the numbers to create char data values. To check whether the character entered is a vowel (both lowercase and uppercase), we then put the two bool variables through a logical OR operator. Example 8.14 presents the code:

**EXAMPLE 8.14**

//C++ program to check if character is a vowel or not

#include <iostream>

using namespace std;

int main()

{

char chars;

cout<<char(101)<<endl;

bool isLowercaseVowel=false, isUppercaseVowel=false;

cout << "Enter an alphabet: "<<endl;

cin >> chars;

// values for lowercase vowels stored in a boolean variable

isLowercaseVowel = (chars == 'a' || chars == char(101) || chars == char(105) || chars == 'o' || chars == 'u');

// values for uppercase vowels stored in a boolean variable

isUppercaseVowel = (chars == char(65) || chars == char(69) || chars == 'I' || chars == 'O' || chars == char(85));

if (isLowercaseVowel || isUppercaseVowel){

cout << chars<<" is a vowel "<< endl;

}

else{

cout << chars<<" is a not vowel "<< endl;

}

return 0;

}

**OUTPUT**

**Enter an alphabet:**

**A**

**A is a vowel**

**Enter an alphabet:**

**K**

**K is a not vowel**

### 8.3.7 Write C++ code that will save the result of relation expression that contains both or a mix of numeric variables, character variables and literals and multiple

Remember, only one logical operator can be used to combine two relations. However, multiple relations can be combined into a complex logical expression. Regardless of the number of relations and logical operators used to build a logical expression, the result is either true, false, or indeterminate because of missing values.

**Short circuit evaluation**

For the logical AND to yield true, both operands must evaluate to true. If the first operand evaluates to false, the second operand will return false regardless of whether it evaluates to true or false. In this case, the logical AND operator will immediately return false, without even examining the second operand! This is referred to as short circuit evaluation.

Similarly, if the first operand of a **logical OR** condition is true, the entire OR condition must evaluate to true, and the second operand is ignored.

Consider the following program which uses three logical operators.

**EXAMPLE 8.15**

#include <iostream>

using namespace std;

int main(){

int a=5;

int b=3;

char c='A';

int d=0b01;

int answer=((a>3)&&(a>10) || (5<!(d)) && ((int)c<b) );

cout<<answer<<endl;

return 0; }

Output

0

# FORMATIVE ASSESSMENT 8.3 INDIVIDUAL TASK

8.3.1 Suppose P and Q are logical expressions. The logical expression P && Q is true if both P and Q are true. [True/False] (1)

8.3.2 What is a logical operator. (2)

8.3.3 List THREE examples of logic operators. (3)

8.3.4 What do you understand by the term flag variable. (2)

8.3.5 Suppose that score is a variable of type double. Write the C++ code snippet that increases the score by 5 marks if score is between 80 and 90. (3)

8.3.6 Write a C++ statement that prints true if x is an odd number and positive. (3)

**[Total =14 Marks]**

# 8.4 Selection Statements

When we have complex assertions to examine, one of them must meet a specified condition before the second condition is evaluated. In such situations, the nested if statement is used..

### 8.4.1 Define the term nested if statement

**VOCABULARY**

Nested if in C++ is using more than one if statements in the same scope. The if statement is a decision-making statement that allows taking decisions based upon the condition specified.

### 8.4.2 Determine the application flow when a nested if statement is encountered

 When there are multiple conditions that are interdependent, a nested if statement can be used. The nested if statement checks multiple conditions sequentially. A nested if statement is formed by using one if statement inside another. The inner if statement will only execute when it’s outer if statement is true. If the first condition is true, we go into the next if condition and the subsequent condition is checked until we get a false condition, and the checking stops. In C++ there is no limit to the levels of nesting for if or if else statement.

### 8.4.3 Identify or correct the general form for a nested if-else statement

Figure 8.3 illustrates the logic path for nested if statements. In the diagram, the first if condition will be checked first, and if it is false, the program will exit the first if block and go to the next set of statements after the first if block. If the first if condition is true then the program will go into the body of the first if. In the body of the first if condition, we just added a statement that can be printed out and then have a second if condition. This condition will now be checked, and then the block will be executed. If the second if a condition is false, the second if block is skipped, and the program proceeds to the statements following the second if block. Similarly, when we have more than two nested if statements, the program will go inside as long as the consecutive if statements are true.

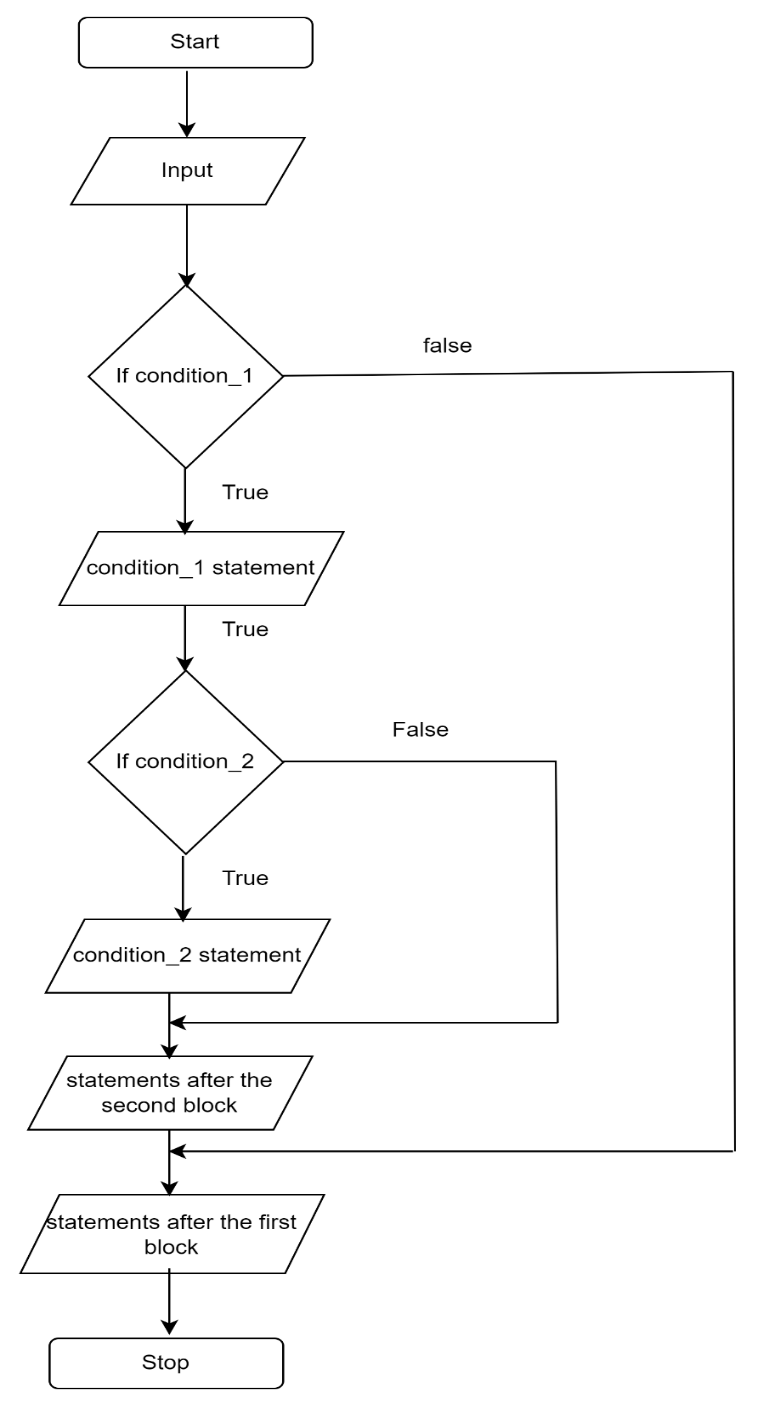


Figure 8. 3: Nested If statement logic path

**Syntax of Nested if statement**

// Outer if conditionif (condition 1) {   **// inner if condition**  
   if (condition 2) {       // Block of Code and Statements  
   }  
   **// inner else condition**  
   else {       // Block of Code and Statements  
   }  
}  
**// Outer else statement**  
else {   **// inner if condition**  
   if (condition 3) {  
       // Block of Code and Statements  
   }  
   **// inner else condition**  
   else {  
       // Block of Code and Statements  
   }  
}

**Let us start with a simple program**

We want to write a program to enter two marks. If the first mark is greater than 50 and you check if the second mark is greater than 50 to calculate the average and print "you qualify for an award".

**EXAMPLE 8.16**

#include<iostream>

using namespace std;

int main()

{

// Declare and assign values to variables

int first\_mark= 0;

int second\_mark = 0;

cout<<"Enter the first mark" <<endl;

cin>>first\_mark;

cout<<"Enter the second mark" <<endl;

cin>>second\_mark;

if (first\_mark >=50) { // first if condition : variable first\_mark is checked

cout << " value of firstmark is: " << first\_mark << endl;

if ( second\_mark>=50) { // second if condition : variable second\_mark is checked

cout << " value of second mark is: " << second\_mark << endl;

int average= (first\_mark+second\_mark)/2;

cout<<"You got an average mark of "<< average<< "You have qualified for an award"<<endl;

}

}

return 0;

}

**OUTPUT**

**Enter the first mark**

**89**

**Enter the second mark**

**79**

**value of first mark is: 89**

**value of second mark is: 79**

**You got an average mark of 84You have qualified for an award**

**Enter the first mark**

**75**

**Enter the second mark**

**23**

**value of firstmark is: 75**

**You failed your second test**

**Enter the first mark**

**23**

**Enter the second mark**

**24**

In the preceding example, the average is calculated only if the first mark is greater than 50 and the second mark is greater. Nothing is printed if the first mark is less than 50, as in the previous entry. We may extend the code by adding otherwise statements to specify what occurs if the first mark is less than 50, as seen below.

**EXAMPLE 8.17**

#include<iostream>

using namespace std;

int main()

{

// Declare and assign values to variables

int first\_mark= 0;

int second\_mark = 0;

cout<<"Enter the first mark" <<endl;

cin>>first\_mark;

cout<<"Enter the second mark" <<endl;

cin>>second\_mark;

if (first\_mark >=50) { // first if condition : variable first\_mark is checked

cout << " value of firstmark is: " << first\_mark << endl;

if ( second\_mark>=50) { // second if condition : variable second\_mark is checked

cout << " value of second mark is: " << second\_mark << endl;

int average= (first\_mark+second\_mark)/2;

cout<<"You got an average mark of "<< average<< "You have qualified for an award"<<endl;

}

cout << "You failed your second test" << endl;

}

else{

cout << "You first mark is below 50%" << endl;

}

return 0;

}

**OUTPUT**

**Enter the first mark**

**12**

**Enter the second mark**

**12**

**You first mark is below 50%**

### 8.4.4 Write C++ code that will use relational expressions in nested if-else statements

We'll employ relational expressions in nested if-else statements next. We're going to change the program that checks for vowels and consonants.

**Example 8.18**

Here we are reading a character from the user and validating that it is valid alphabet or not, if the character if valid alphabet then we are validating it is vowel or not and printing the appropriate message for the input character.

//EXAMPLE of Nested if else

//Read a character a check whether it is VOWEL or CONSONANT

#include<iostream>

using namespace std;

int main()

{

char ch;

//reading a character

cout<<"Enter an alphabet: ";

cin>>ch;

//condiion to check character is alphabet or not

if( (ch>='A' && ch<='Z') || (ch>='a' && ch<='z'))

{

//conditions to check character is VOWEL or not

if( ch=='A' || ch=='a' || ch=='E' || ch=='e' || ch=='I' || ch=='i' || ch=='O' || ch=='o' || ch=='U' || ch=='u')

cout<<"\""<<ch<<"\" is a VOWEL"<<endl;

else

cout<<"\""<<ch<<"\" is a CONSONANT"<<endl;

}

else

{

cout<<"\""<<ch<<"\" is not an alphabet\n";

}

return 0;

}

In example 8.18, we used a boolean variable to check lowercase and uppercase something that has been handled with the use of logical operators to accommodate both cases.

### 8.4.5 Write C++ code that will use relational expressions in nested if-else statements with compounded content

A university has a policy for sponsoring researchers to attend conferences to present their papers. For a researcher to qualify for funding for international conferences, he or she must have accumulated 0.75 or more research points in the previous year. The employment status is checked if the condition- is met. If a researcher is permanently employed, he/she qualifies for funding to travel to an international conference. If employment status is contract, the researcher qualifies for international travel funding if the duration of the contract is equal to or more than 3 years; otherwise, he or she qualifies for local travel. Otherwise, the researcher qualifies ONLY for funding to attend local conferences. The solution of this scenario is given in example 8.19.

**EXAMPLE 8.19**

// C++ Program to Nested-if conditions

//check qualification for funding to attend conferences

#include <iostream>

using namespace std;

int main(){

double research\_units=0;

char emp\_status;

int contact\_duration;

//Entering input

cout<<"Enter research units accumulated in the previous year"<<endl;

cin>>research\_units;

cout<<"Enter your work status [c=contract, p=permanent]"<<endl;

cin>>emp\_status;

if (research\_units>=0.75) {

if (emp\_status =='p' || emp\_status =='P') {

if (true) {

cout << "Congratulations!! You qualify for international travel funding" << endl;

}

else {

cout << " You ONLY qualify for local travel funding" << endl;

}

}

else if (emp\_status=='c' || emp\_status=='C'){

cout<<"Enter your contract duration in years [0-9]"<<endl;

cin>>contact\_duration;

if (contact\_duration>=3){

cout << "Congratulations!! You qualify for international travel funding" << endl;

}

else{

cout << " You ONLY qualify for local travel funding" << endl;

}

}

else{

cout << "Invalid employment status" << endl;

}

}

else{

cout << "Unfortunately you ONLY qualify for local travel funding" << endl;

}

return 0;}

**OUTPUT**

**Enter research units accumulated in the previous year**

**.9**

**Enter your work status [c=contract, p=permanent]**

**c**

**Enter your contract duration in years [0-9]**

**2**

**You ONLY qualify for local travel funding**

**Enter research units accumulated in the previous year**

**1.2**

**Enter your work status [c=contract, p=permanent]**

**P**

**Congratulations!! You qualify for international travel funding**

So, the program can now assist the university in deciding when to assist researchers with international conference funding or local conference funding.

Task: The program above is not validated for input entries. If you enter for example characters for number of research\_units, the program still prints that

"Unfortunately, you ONLY qualify for local travel funding"

Validate the two entries: research units and emp\_status for correct data type.

### 8.4.6 Using relational expression containing logic operators in nested if-else statements

A European country has embarked on a massive recruitment of its youths into a national youth service program. The requirements are that you should be at least 18 years old but less than or equal to 45. If age is outside the range, the program will display a message:

“Your age is outside the acceptable range"

Further to this, you should be a male and your BMI must be between 18.5 and 24.9. If your BMI is outside the range, you will be referred to a physician. However, for female candidates, they are given a preference to skip the national service and go straight for an internship.

**EXAMPLE 8.20**

#include <iostream>

using namespace std;

int main(){

//declaring variables

int age=0;

double BMI=0;

char gender;

//accepting input

cout<<"Enter age"<<endl;

cin>>age;

cout<<"Enter your gender"<<endl;

cin>>gender;

cout<<"Enter your BMI"<<endl;

cin>>BMI;

//validation of data type input

if (cin.fail())

{

cout<<"entry error"<<endl;

cin.clear();

cin.ignore(1000, '\n');

return 0;

}

//Nested if statements to check conditions

if (age >=18 && age<=45) {

if (gender=='M' || gender =='m') {

if ( BMI <=24.9 && BMI >=18.5) {

cout << "You must go for a national service" << endl;

}

else{ cout << "We are going to be referred to a physician" << endl;}

}

else if (gender=='F' || gender =='f'){

cout << "You are allowed to skip national service" << endl;

}

else{

cout << "You entered invalid gender" << endl;

}

}

else {

cout << " Your age is outside the acceptable range"<<endl;

}

return 0;

}

**OUTPUT**

**Enter age**

**11**

**Enter your gender**

**m**

**Enter your BMI**

**20**

**Your age is outside the acceptable range**

**Enter age**

**20**

**Enter your gender**

**M**

**Enter your BMI**

**11**

**We are going to be referred to a physician**

### 8.4.7 Write C++ code that will use relational expressions containing logic operator in nested if-else statements with compounded content (Max 3 levels, Max 3 logic

### operators per level)

At a school, the SRC wants to conduct some elections to choose its representatives for the academic year. There have been some cheating allegations in the past few years. They have asked the programming class to develop a system to assist in this regard. Only students in grades 8 through 11 will be able to vote. The grade must be authenticated first.

They put up a condition each grade as follows:

Table 8. 5::SRC Voting Venues

|  |  |  |  |
| --- | --- | --- | --- |
| **Grade** | **Identifiable Grouping** | **Voting Room** | **Accepted Age Groups** |
| Grade 8 | Male | Room 5 | 14 to less than 15 |
| Female | Room 6 | 14 to less than 15 |
| Grade 9 | Male | Room 7 | 15 to less than 16 |
| Female | Room 8 | 15 to less than 16 |
| Grade 10 | Male | Room 9 | 16 to less than 17 |
| Female | Room 10 | 16 to less than 17 |
| Grade 11 | Male | Room 3 | 17 to less than 18 |
| Female | Room 4 | 17 to less than 18 |

Write a C++ program to assist the campaign managers in directing the voters.

**EXAMPLE 8.21**

//Voter verification and venue allocation system

#include<iostream>

using namespace std;

int main()

{

char gender;

int grade;

double age;

cout<<"\nEnter Your grade : "<<endl;

cin>>grade;

if(grade==8)

{

cout<<"\nEnter Your Gender : ";

cin>>gender;

cout<<"\nEnter Your age in years : ";

cin>>age;

if (cin.fail()){

cout<<"Entry not recognised"<<endl;

return 0;

}

if((gender=='M' || gender=='m') && (age>=14 || age<15))

{

cout<<"\nGo To Room-5";

}

else if((gender=='F' || gender=='f') && (age>=14 || age<15))

{

cout<<"\nGo To Room-6";

}

else

{

cout<<"\n Invalid range of input. Please try again";

}

}

else if(grade==9)

{

cout<<"\nEnter Your Gender : ";

cin>>gender;

cout<<"\nEnter Your age in years : ";

cin>>age;

if (cin.fail()){

cout<<"Entry not recognised"<<endl;

return 0;

}

if((gender=='M' || gender=='m') && (age>=15 && age<16))

{

cout<<"\nGo To Room-7";

}

else if((gender=='F' || gender=='f')&& (age>=15 && age<16))

{

cout<<"\nGo To Room-8";

}

else

{

cout<<"\n Invalid range of input. Please try again";

}

}

else if(grade==10)

{

cout<<"\nEnter Your Gender : ";

cin>>gender;

cout<<"\nEnter Your age in years : ";

cin>>age;

if (cin.fail()){

cout<<"Entry not recognised"<<endl;

return 0;

}

if((gender=='M' || gender=='m') && (age>=16 && age<17))

{

cout<<"\nGo To Room-9";

}

else if((gender=='F' || gender=='f') && (age>=16 && age<17))

{

cout<<"\nGo To Room-10";

}

else

{

cout<<"\n Invalid range of input. Please try again";

}

}

else if(grade==11)

{

cout<<"\nEnter Your Gender : ";

cin>>gender;

cout<<"\nEnter Your age in years : ";

cin>>age;

if (cin.fail()){

cout<<"Entry not recognised"<<endl;

return 0;

}

if((gender=='M' || gender=='m') && (age>=17 && age>=18))

{

cout<<"\nGo To Room-3";

}

else if((gender=='F' || gender=='f') && (age>=17 && age<18))

{

cout<<"\nGo To Room-4";

}

else

{

cout<<"\n Invalid range of input. Please try again";

}

}

else{

cout<<"Voter verification failed. Please consult the polling officer"<<endl;

}

return 0;

}

**TASK 8.5**

Write a C++ program to check that the entered character is small, capital or a special character. The program must allow the user to enter any character and then display the ASCII value and the type of character it is.

**Hint:**

**Small letters**

Ascii\_value>=97 && Ascii\_value<=122

**Capital Letter**

 Ascii\_value>=65 && Ascii\_value<=90

**Special Characters**

Ascii\_value>=0 && Ascii\_value>=47 || Ascii\_value>=54 && Ascii\_value<=64 || Ascii\_value>=91 && Ascii\_value<=96 || Ascii\_value>=123 && Ascii\_value<=127

So far we have demonstrated how to implement select/conditional statements using if, if …else… and nested if statements. There is also Switch Case.

A switch statement checks a variable for equality against a set of values. Each value is referred to as a case, and the variable is tested for each case.

**Syntax for Switch Case**

// switch-case

switch ( selector ) {

case value-1:

block-1; break;

case value-2:

block-2; break;

case value-3:

block-3; break;

......

case value-n:

block-n; break;

default:

default-block;

}

Example: Write a program in C++ to do arithmetic operations based on the operator selected.

**EXAMPLE 8.21**

//Using switch for arithmetic operators

#include <iostream>

using namespace std;

int main(){

char oper; int num1, num2, result;

cout<<"Enter first number"<<endl;

cin>>num1;

cout<<"Enter second number"<<endl;

cin>>num2;

cout<<"Enter the operator [+, -, / \*]"<<endl;

cin>>oper;

if (cin.fail()){

cout<<"Error in input"<<endl;

}

switch (oper) {

case '+':

result = num1 + num2;

break;

case '-':

result = num1 - num2;

break;

case '\*':

result = num1 \* num2;

break;

case '/':

result = num1 / num2;

break;

default:

cout << "Unknown operator" << endl;}

cout<<num1<< oper<<num2<< " = "<<result<<endl;

return 0; }

**OUTPUT**

**Enter first number**

**45**

**Enter second number**

**33**

**Enter the operator [+, -, / \*]**

**+**

**45+33=78**

# FORMATIVE ASSESSMENT 8.4 INDIVIDUAL TASK

8.4.1 Define the term nested if statement. (2)

8.4.2 Using a flowchart, show the general form of a nested if statement. (5)

8.4.3 What will be the output of the code below:

#include <iostream>

using namespace std;

int main (){

int x = 35;

int y = 45;

int z;

if (x > y){

z = x + y;

}

else

{

z =y-x;

}

cout << x << " " << y << " " << z << endl;

return 0;

}

(2)

8.4.4 In nested if statement which statement evaluated first ?

Explain your answer. (3)

8.4.5 How many levels of nested if statement is allowed in C++. (1)

**[Total =13 Marks]**

# SUMMATIVE ASSESSMENT 8.4 INDIVIDUAL TASK

8.5.1 Explain how the integer result produced by a relation expression relates to bool true/false (2)

8.5.2 Write a C++ program to check whether the given angles constitute a valid/not valid triangle. (10)

8.5.3 What will be the output of the following code in C++. Explain your answer. (3)

int result='a'<'A';

8.5.4 Write the syntax for nested if..else if statements. (4)

8.5.5 Business require a program to determine whether they have made a profit or loss from the sales that they do on a day-to-day basis. Write a C++ program to enter cost price and selling price and find profit or loss. Consider the following scenario:

If Selling Price > Cost Price  
Profit = Selling Price - Cost Price

If Selling Price < Cost Price  
Loss = Cost Price - Selling Price

If Selling Price = Cost Price  
No Profit .. No Loss

(10)

8.5.6 The nested conditional statement shown below has been written by an inexperienced C++ programmer. The behaviour of the statement is not correctly represented by the formatting.

if (n < 10)

if (n > 0)

cout << "The number is positive." << endl;

else

cout << "The number is \_\_\_\_\_\_\_\_\_\_\_\_\_\_." << endl;

return 0;

* 1. What is the output of the statement if the variable n has the value 7 ?

If n has the value 15 ?

If n has the value -3 ? (6)

8.5.7 Answer the questions below concerning the following fragment of code.

int n;

cout << "Enter an integer: ";

cin>> n;

if (n < 10)

cout << "less than 10" << endl;

else if (n > 5)

cout << "greater than 5" << endl;

else

cout << "not interesting" << endl;

1. What will be the output of the fragment above if the interactive user enters the integer value 0 ? (2)
2. What will be the output of the fragment above if the interactive user enters the integer value 15 ? (2)
3. What will be the output of the fragment above if the interactive user enters the integer value 7 ? (2)
4. What values for n will cause the output of the fragment above to be "not interesting"? (2)

**[Total =43 Marks]**

# References

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

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

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

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

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

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