# **Name: MUHAMMAD JUNAID**

# **ID: 16027**

# **SEMSETER: SUMMER**

# **SUBJECT: PROGRAMMING**

# **QUESTION NO 1**

# What is the purpose of if statement? Discuss its two different forms with examples.

ANSWER

# IF STATEMENT

It is one of the powerful conditional statement. If statement is responsible for modifying the flow of execution of a program. If statement is always used with a condition. The condition is evaluated first before executing any statement inside the body of If. The syntax for if statement is as follows:

If (condition)

instruction;

The condition evaluates to either true or false. True is always a non-zero value, and false is a value that contains zero. Instructions can be a single instruction or a code block enclosed by curly braces { }.

Following program illustrates the use of if construct in 'C' programming:

#include<stdio.h>

int main ()

{

int num1=1;

int num2=2;

if(num1<num2) //test-condition

{

printf("num1 is smaller than num2");

}

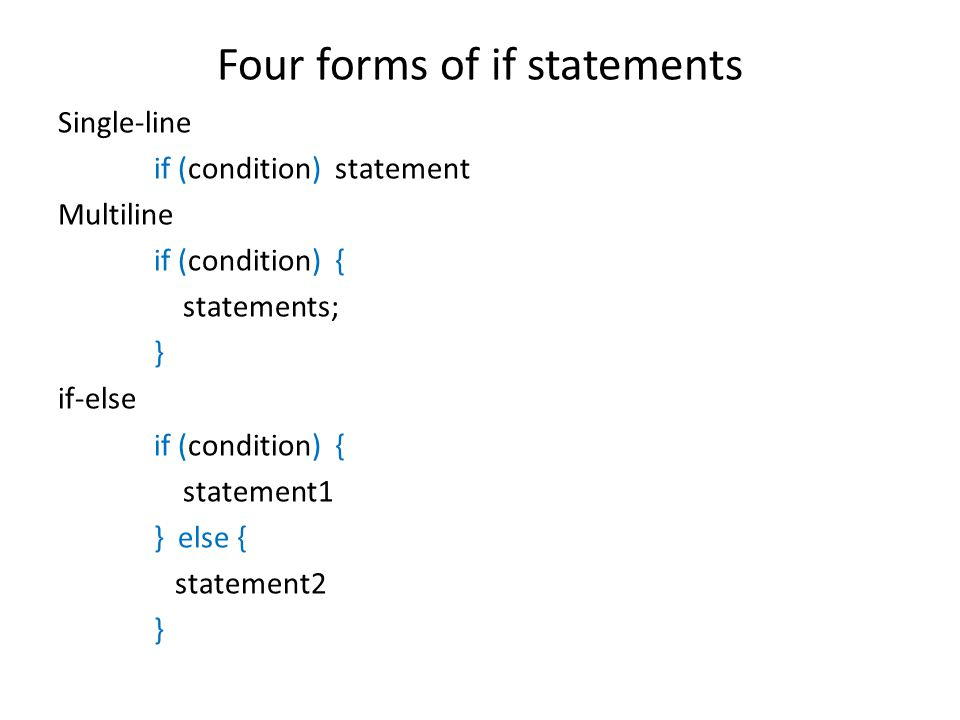
return 0;

}

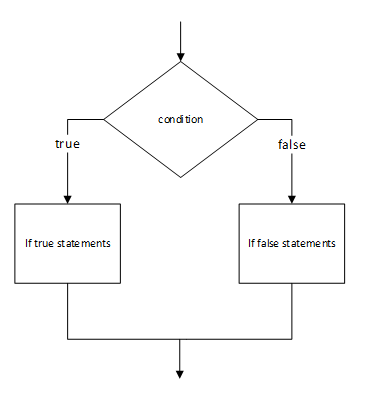
Output:

num1 is smaller than num2

# **FORMS OF IF STATEMENT**



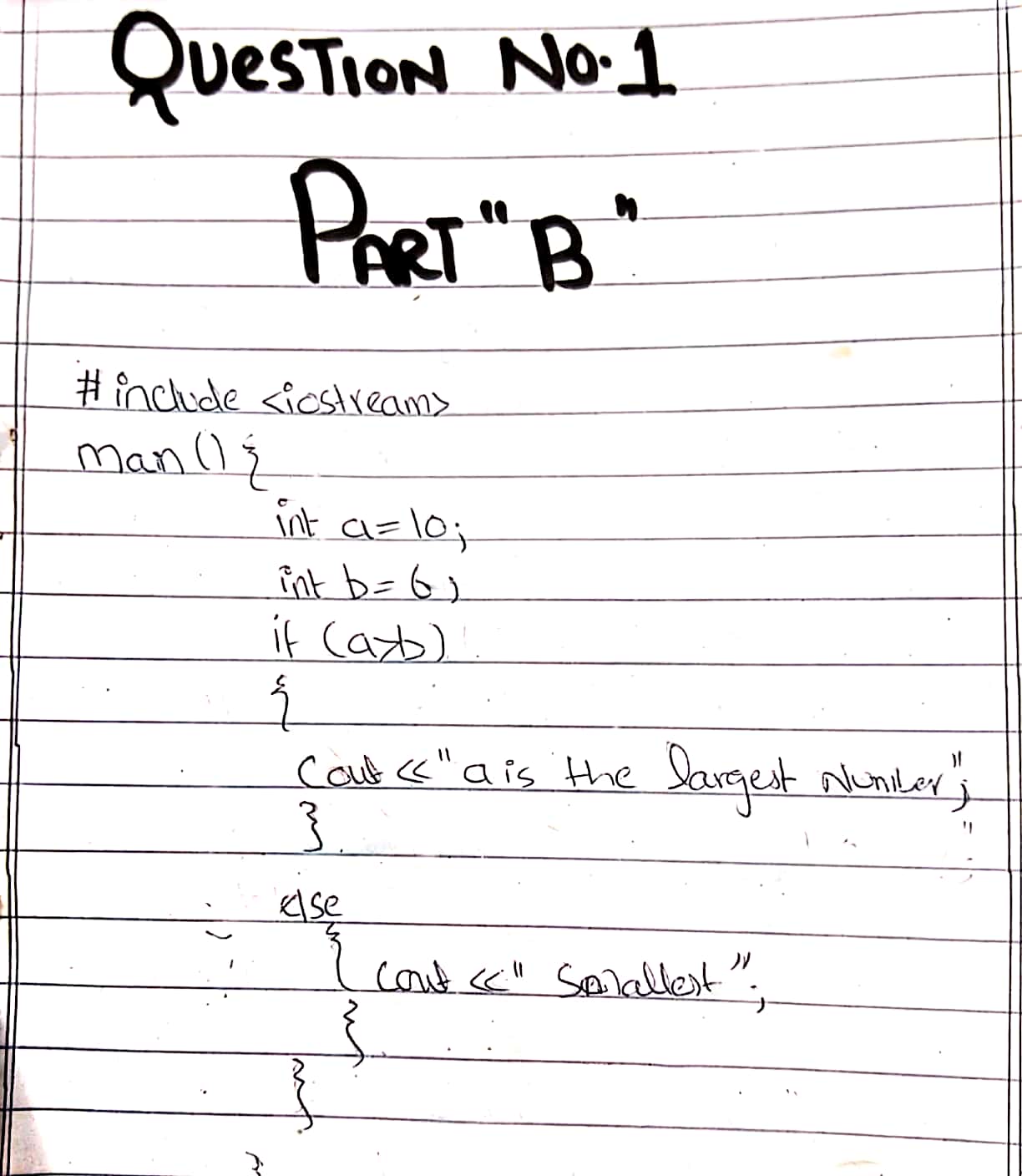
# **FLOWCHART OF IF STATEMENT**



# **QUESTION NO 1**

# **PART B:**

# Write a C++ program to read two numbers from keyboard and then find the LARGEST number of them.



# **QUESTION NO 2**

# What are the Logical Operators? Explain them

ANSWER:

# LOGICAL OPERATORS

Logical operators are used to combine two or more conditions/ constraints or to complement the evalution of the original condition in consideration the result of the operation of a logical operator is a bolean value either true or false

# && (LOGICAL AND)

Used to combine two conditions if both conditions are true

If(gender==1 && AGE=65)

Senior ++;

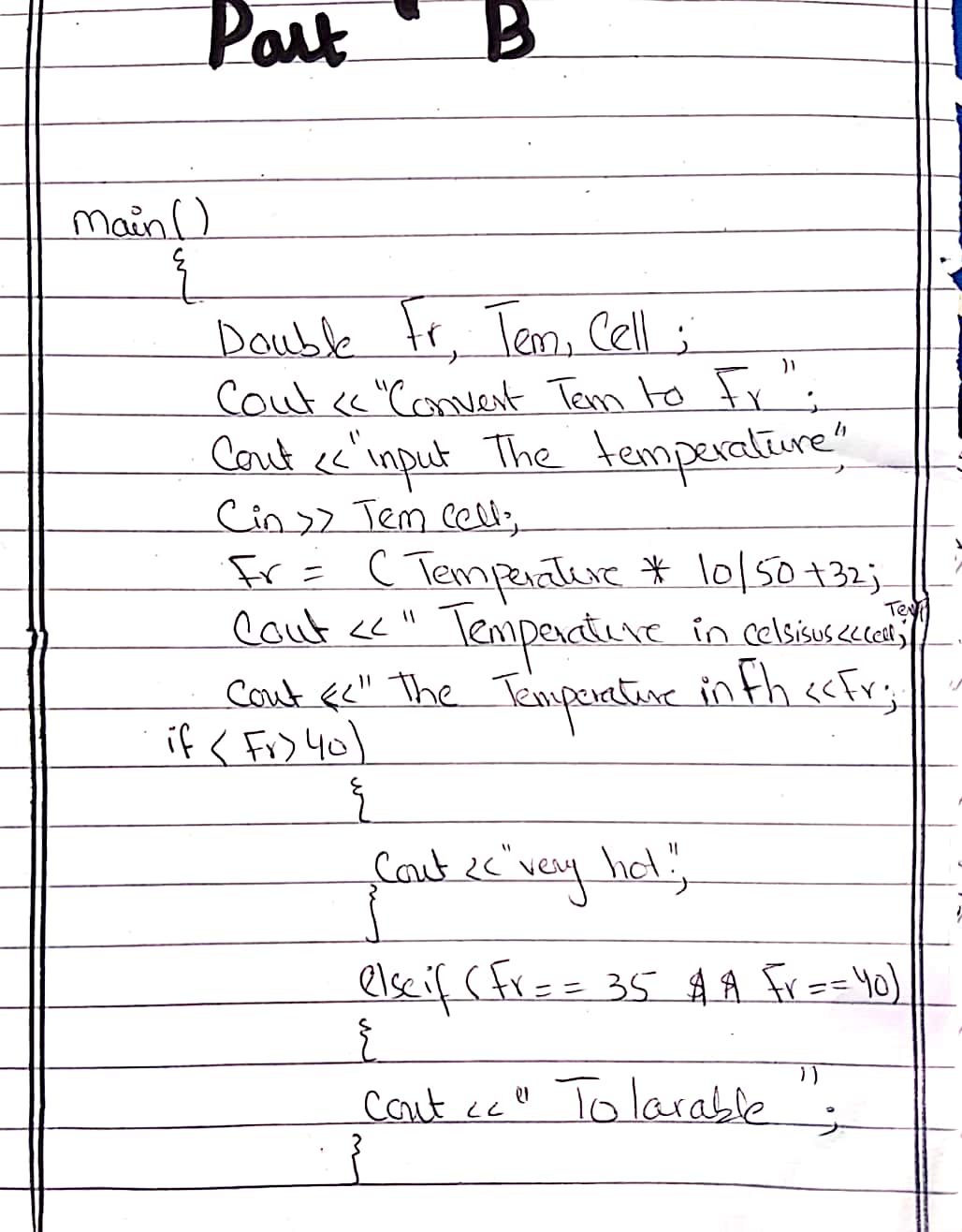
# ll (logical or)

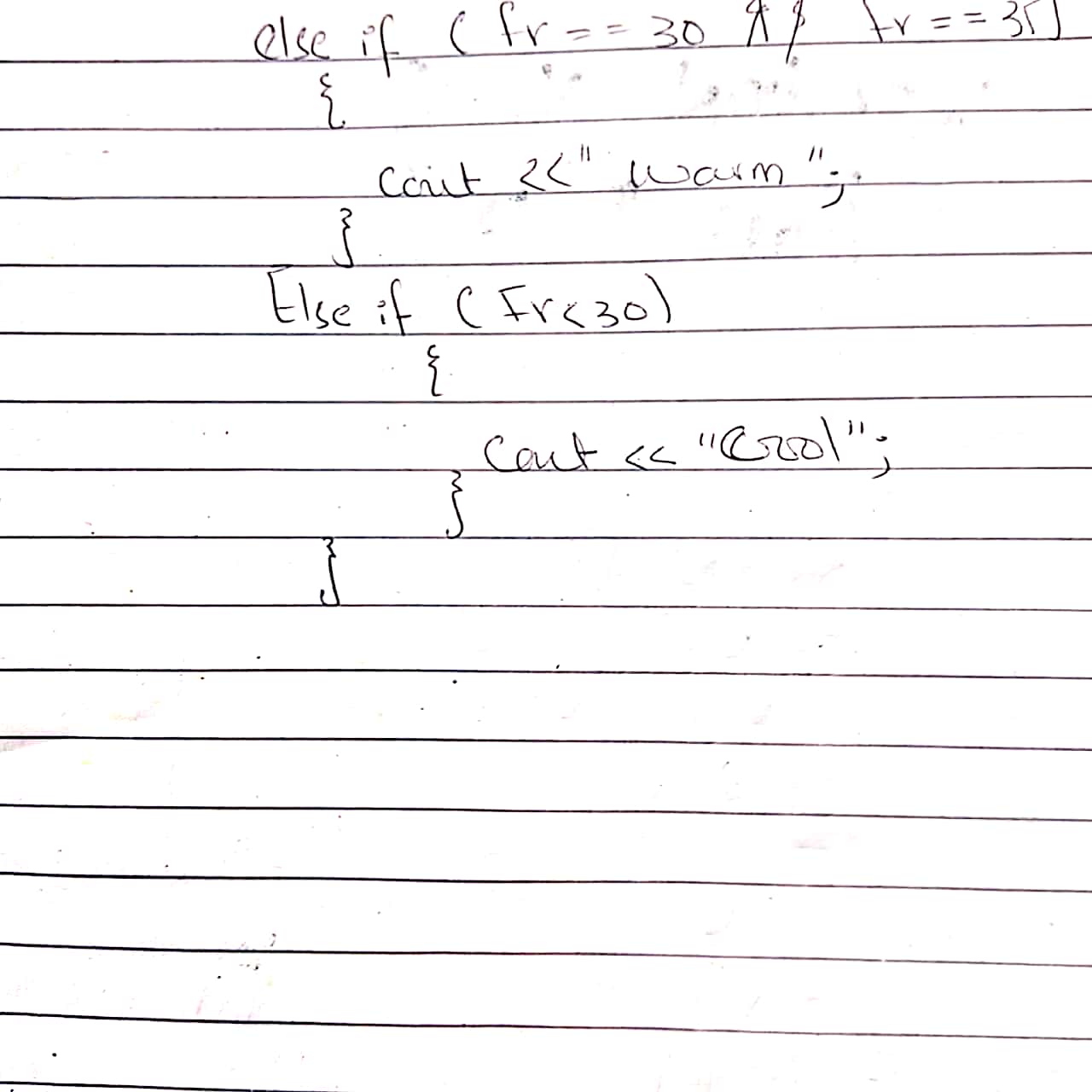
true if either of condition is true

if (semester Avg>=90 ll Final exam>=90)

cout<<”student grade is A”;

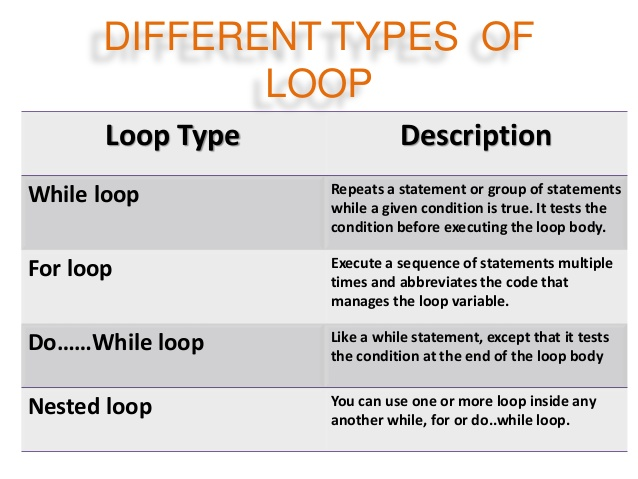
# question 2 part b





# QUESTION 3

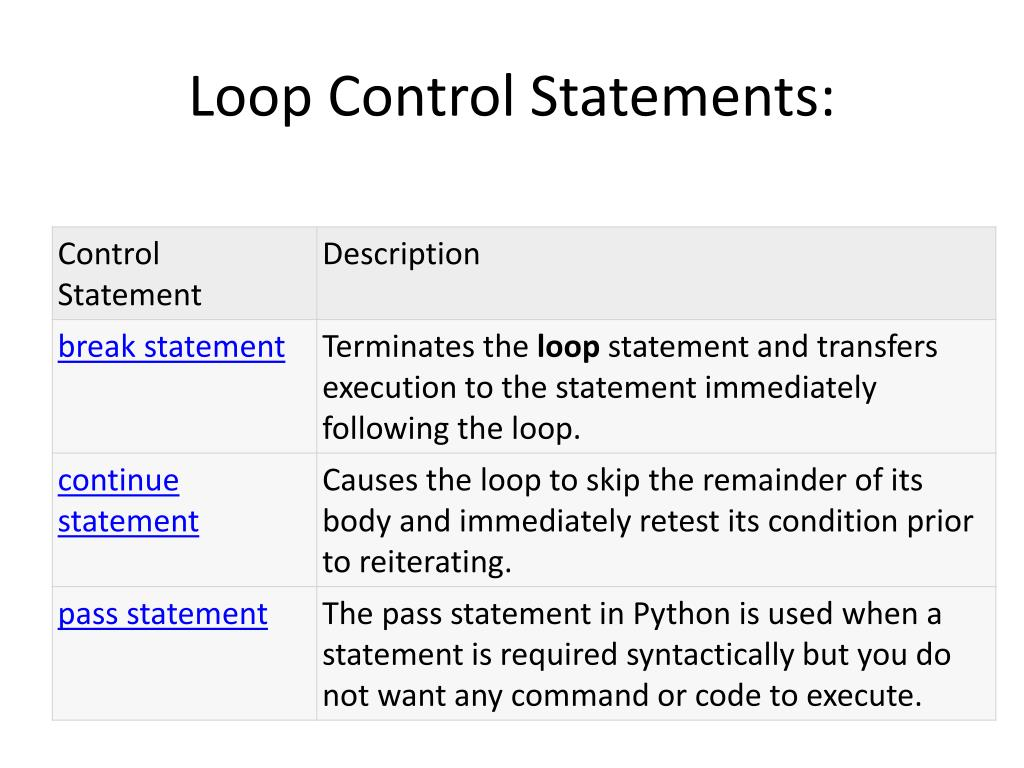
# What does ***Looping*** mean? Explain different loops in C++.



# Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

C++ supports the following control statements.



## **The Infinite Loop**

A loop becomes infinite loop if a condition never becomes false. The **for** loop is traditionally used for this purpose. Since none of the three expressions that form the ‘for’ loop are required, you can make an endless loop by leaving the conditional expression empty.

#include <iostream>

using namespace std;

int main () {

for( ; ; ) {

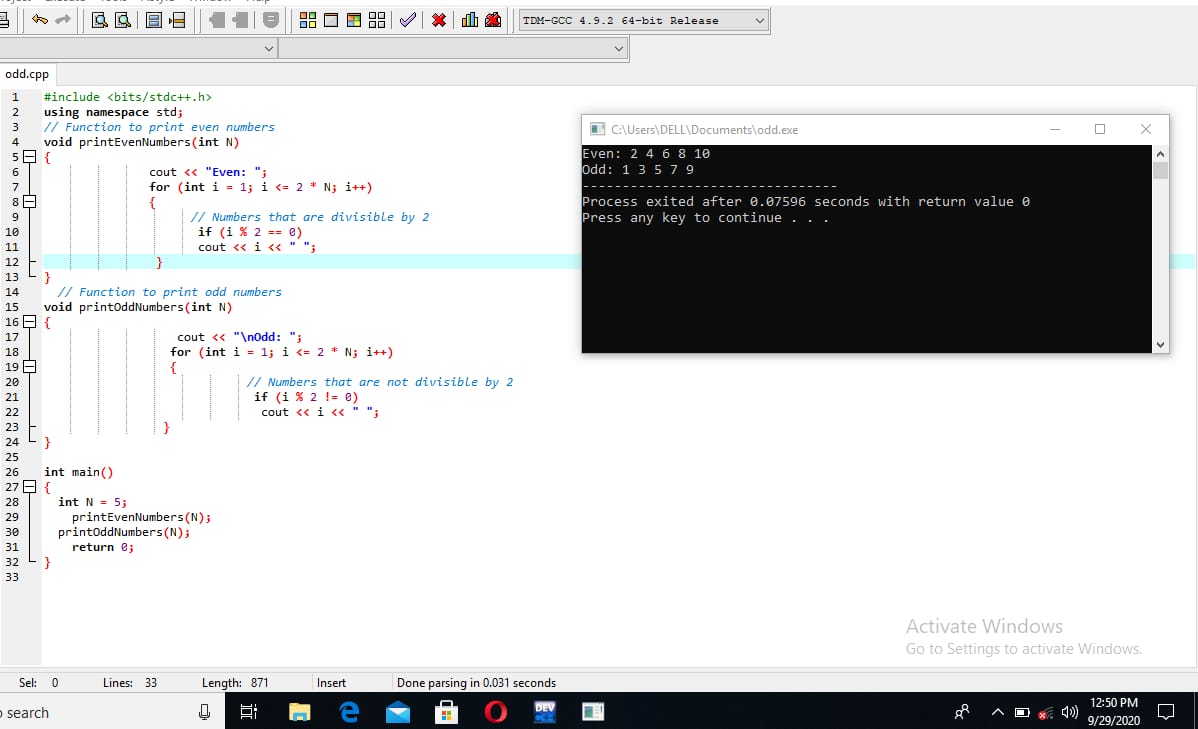
printf("This loop will run forever.\n");

When the conditional expression is absent, it is assumed to be true. You may have an initialization and increment expression, but C++ programmers more commonly use the ‘for (;;)’ construct to signify an infinite loop

# **QUESTION NO .3**

# **PART B**

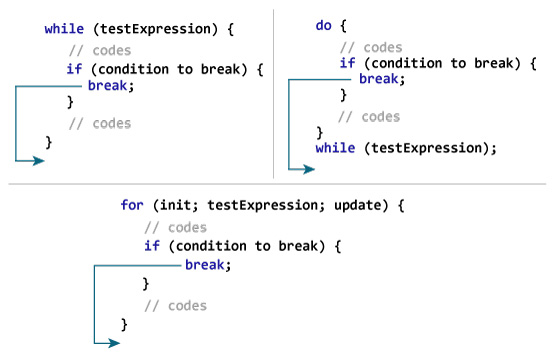
Write a C++ program to read a number from keyboard and then determine whether it is Even or Odd number

ANSWER: 

# **QUESTION NO 4**

# What is the purpose of using break and continue statements?

# **How break statement works?**



### **How continue statement works?**



The one-token statements continue and break may be used within loops to alter control flow; continue causes the next iteration of the loop to run immediately, whereas break terminates the loop and causes execution to resume after the loop. Both control structures must appear in loops. Both break and continue scope to the most deeply nested loop, but pass through non-loop statements.

Although these control statements may seem undesirable because of their goto-like behaviour, their judicious use can greatly improve readability by reducing the level of nesting or eliminating bookkeeping inside loops.

### Break Statements

When a break statement is executed, the most deeply nested loop currently being executed is ended and execution picks up with the next statement after the loop. For example, consider the following program:

while (1) {

if (n < 0) break;

foo(n);

n = n - 1;

}

The while~(1) loop is a “forever” loop, because 1 is the true value, so the test always succeeds. Within the loop, if the value of n is less than 0, the loop terminates, otherwise it executes foo(n) and then decrements n. The statement above does exactly the same thing as

while (n >= 0) {

foo(n);

n = n - 1;

}

This case is simply illustrative of the behavoiur; it is not a case where a break simplifies the loop.

### Continue Statements

The continue statement ends the current operation of the loop and returns to the condition at the top of the loop. Such loops are typically used to exclude some values from calculations. For example, we could use the following loop to sum the positive values in the array x,

real sum;

sum = 0;

for (n in 1:size(x)) {

if (x[n] <= 0) continue;

sum += x[n];

}

When the continue statement is executed, control jumps back to the conditional part of the loop. With while and for loops, this causes control to return to the conditional of the loop. With for loops, this advances the loop variable, so the the above program will not go into an infinite loop when faced with an x[n] less than zero. Thus the above program could be rewritten with deeper nesting by reversing the conditional,

real sum;

sum = 0;

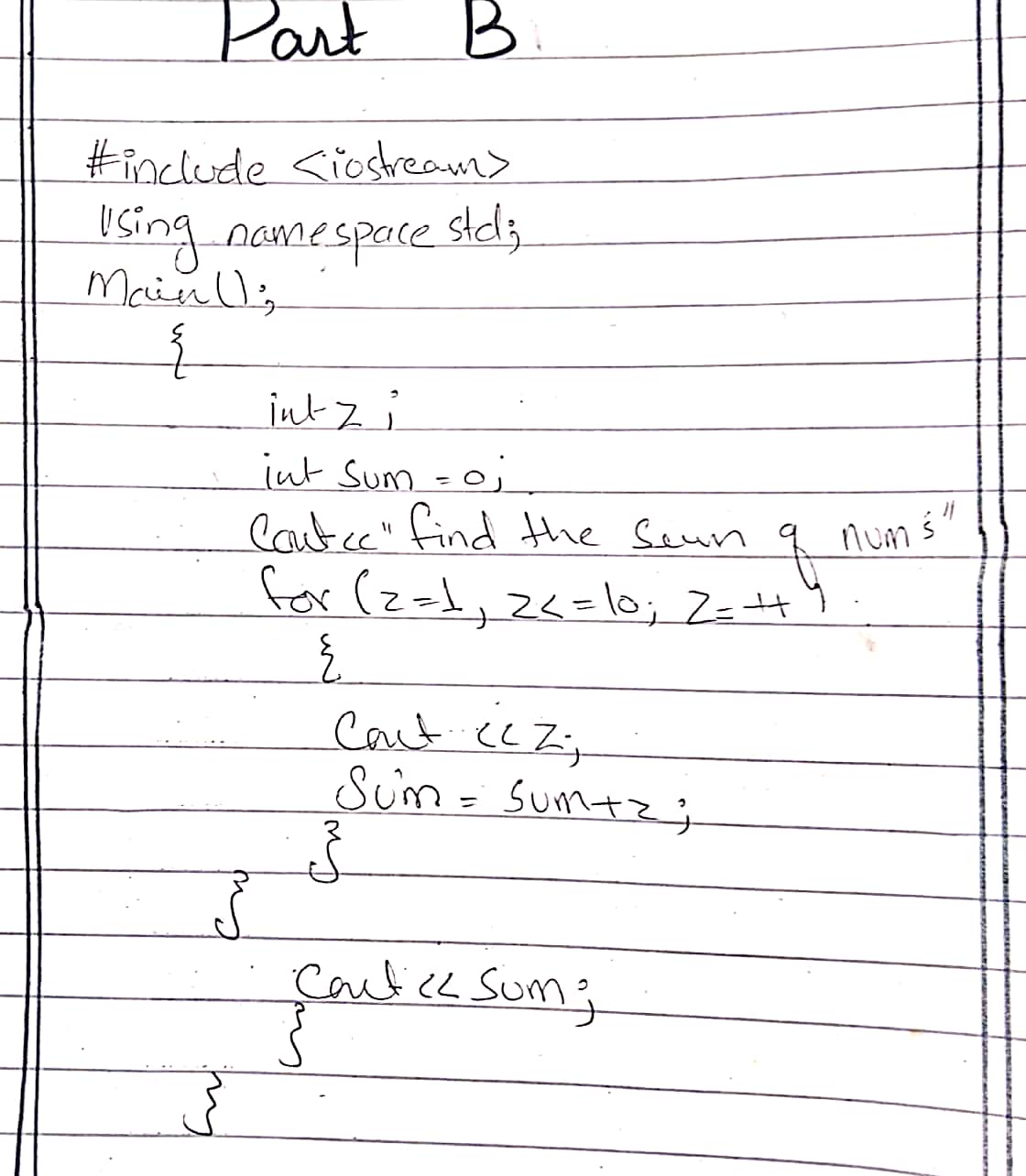
for (n in 1:size(x)) {

if (x[n] > 0)

sum += x[n];

}

While the latter form may seem more readable in this simple case, the former has the main line of execution nested one level less deep. Instead, the conditional at the top finds cases to exclude and doesn’t require the same level of nesting for code that’s not excluded. When there are several such exclusion conditions, the break or continue versions tend to be much easier to read.

QUESTION 4 PART B

# **QUESTION NO. 5**

# Explain the following with proper examples

# a) C++ Character set

# b) Constants

# c) Variables

# d) Keywords

# e) Relational Operators

**constant**

If you want to define a variable whose value cannot be changed, you can use the const keyword. This will create a constant. For example

const double PI = 3.14;

Notice, we have added keyword const.

Here, PI is a symbolic constant; its value cannot be changed.

const double PI = 3.14;

PI = 2.9; //Error

# C++ CHARACTER SET

Character set is a set of valid characters that a language can recognizse. A character represents any letter, digits, or any other sign.

C++ has the following character set :

Letters : A-Z, a-z

Digits : 0-9

Special Symbols : Space + - ∗ ⁄ ^ \ ( ) [ ] { } = != < > . ′ ″ $ , ; : % ! & \_ # <= >= @

White Spaces : Blank space, Horizontal tab (→), Carriage return (↵), Newline, Form feed

Other Characters : C++ can process any of the 256 ASCII characters as data or as literals.

C++ Character Set Example

Following C++ program gives you an idea about using of character set in C++ programming language :

/\* C++ Character Set Example \*/

#include<iostream.h>

#include<conio.h>

void main()

{

clrscr();

char letter, digit, special, white;

cout<<"Enter a Letter : ";

cin>>letter;

cout<<"You entered a letter '"<<letter<<"'"<<"\n";

cout<<"Enter a Digit : ";

cin>>digit;

cout<<"You entered a digit '"<<digit<<"'"<<"\n";

cout<<"Enter a special character : ";

cin>>special;

cout<<"You entered a special character '"<<special<<"'"<<"\n";

cout<<"A horizontal(\t) tab";

getch();

}

# KEYBOARD

These are reserved words

Compiler knows their meaning

Cannot be used as variable name

Cannot be changed

# VARIABLES

variable is a name given to a memory location. It is the basic unit of storage in a program.

The value stored in a variable can be changed during program execution.

A variable is only a name given to a memory location, all the operations done on the variable effects that memory location.

In C++, all the variables must be declared before use.

How to declare variables?

A typical variable declaration is of the form:

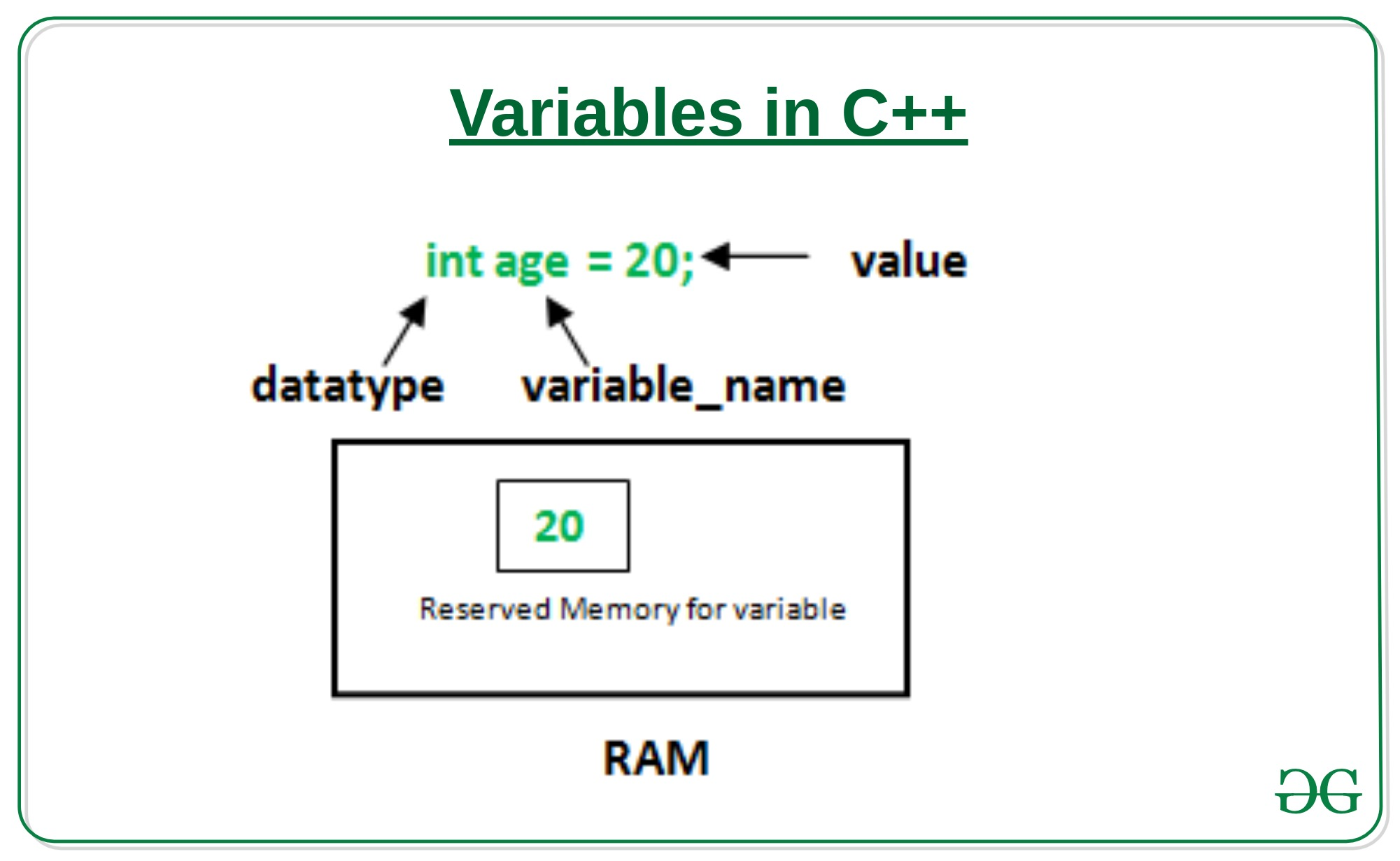
// Declaring a single variable

type variable\_name;

// Declaring multiple variables:

type variable1\_name, variable2\_name, variable3\_name;

A variable name can consist of alphabets (both upper and lower case), numbers and the underscore ‘\_’ character. However, the name must not start with a number.



# **Relational Operators**

[Live Demo](http://tpcg.io/jk7t7c)

#include <iostream>

using namespace std;

main() {

int a = 21;

int b = 10;

int c ;

if( a == b ) {

cout << "Line 1 - a is equal to b" << endl ;

} else {

cout << "Line 1 - a is not equal to b" << endl ;

}

if( a < b ) {

cout << "Line 2 - a is less than b" << endl ;

} else {

cout << "Line 2 - a is not less than b" << endl ;

}

if( a > b ) {

cout << "Line 3 - a is greater than b" << endl ;

} else {

cout << "Line 3 - a is not greater than b" << endl ;

}

/\* Let's change the values of a and b \*/

a = 5;

b = 20;

if( a <= b ) {

cout << "Line 4 - a is either less than \ or equal to b" << endl ;

if( b >= a ) {

cout << "Line 5 - b is either greater than \ or equal to b" << endl ;

}

return 0;

}

When the above code is compiled and executed, it produces the following result −

Line 1 - a is not equal to b

Line 2 - a is not less than b

Line 3 - a is greater than b

Line 4 - a is either less than or equal to b

Line 5 - b is either greater than or equal to b