NAME ABDULLAH TAHIR

ID 13789

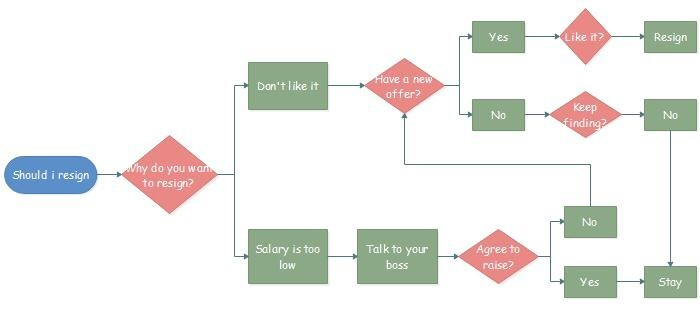
SUBJECT VISUAL PROGRMUING

DATA 23/09/2020

Q1(a)

Ans

Decision making statements help you to make decision based on certain conditions. These conditions are specified by a set of decision making statements having Boolean expressions which are evaluated to a Boolean value true or false



1(b)

Ans

[11:04 am, 23/09/2020] Inuv abdulah tahi Cs: #include <stdio.h>

int main()

{

char gender;

printf("Enter gender (M/m or F/f): ");

scanf("%c",&gender);

switch(gender)

{

case 'M':

case 'm':

printf("Male.");

break;

case 'F':

case 'f':

printf("Female.");

break;

default:

printf("Unspecified Gender.");

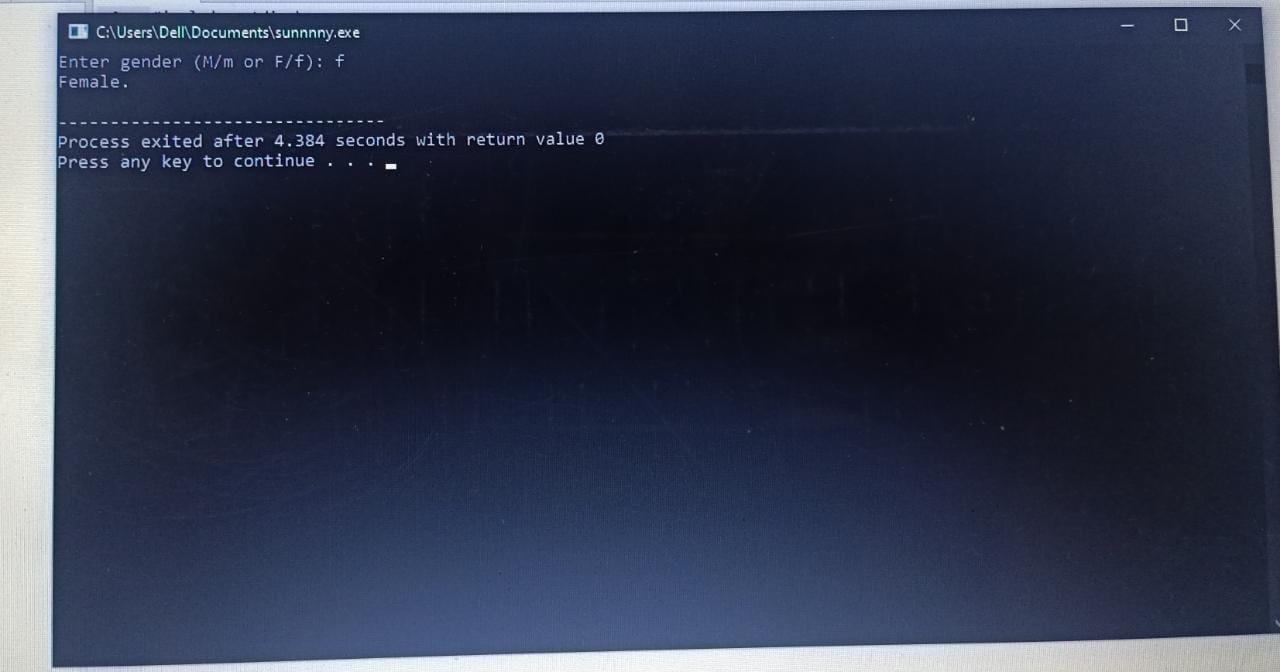
}

printf("\n");

return 0;

}

Out put

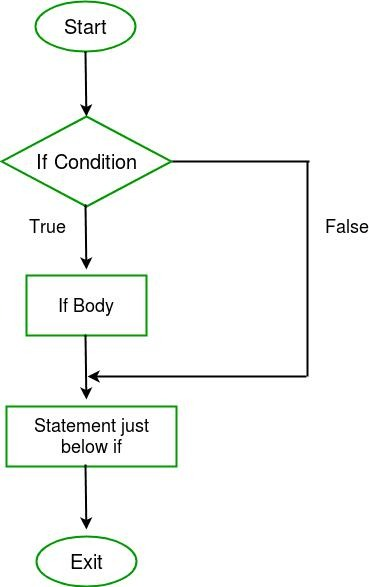


2(**a)**

Ans

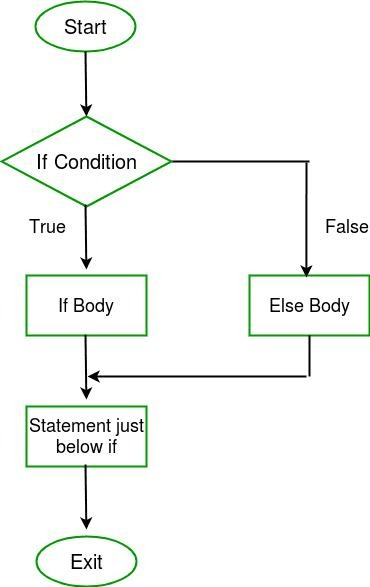
**IF**

The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won’t. But what if we want to do something else if the condition is false. Here comes the C else statement. We can use the else statement with if statement to execute a block of code when the condition is false.



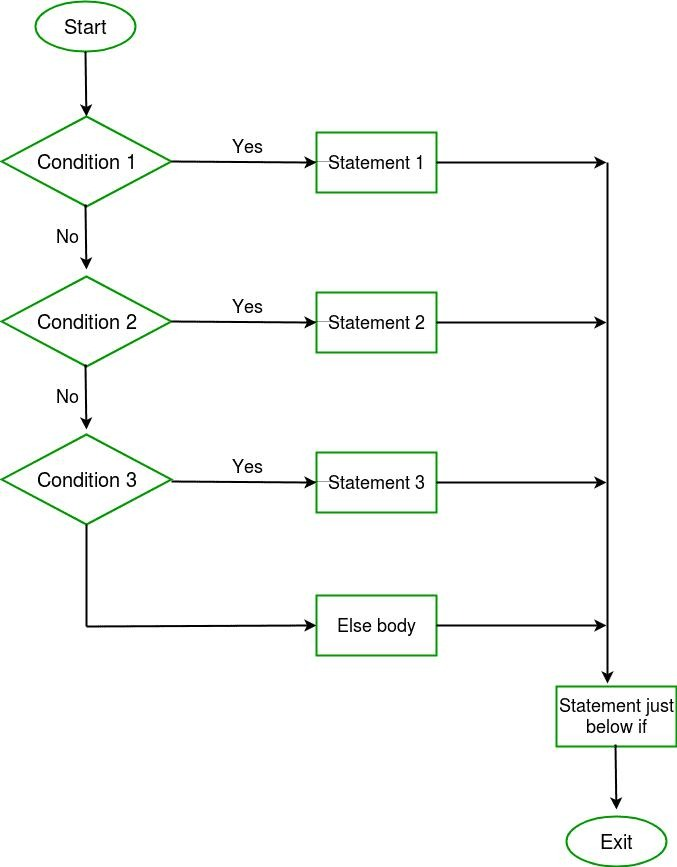
**IF ELSE**

**The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won’t. But what if we want to do something else if the condition is false. Here comes the C else statement. We can use the else statement with if statement to execute a block of code when the condition is false.**



**IF ELSE IF**

**Here, a user can decide among multiple options. The C if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the C else-if ladder is bypassed. If none of the conditions are true, then the final else statement will be executed**



**2(b)**

**Ans**

**#include <stdio.h>**

**void main()**

**{**

**int tmp;**

**printf("Input days temperature : ");**

**scanf("%d",&tmp);**

**if(tmp<0)**

**printf("Freezing weather.\n");**

**else if(tmp<10)**

**printf("Very cold weather.\n");**

**else if(tmp<20)**

**printf("Cold weather.\n");**

**else if(tmp<30)**

**printf("Normal in temp.\n");**

**else if(tmp<40)**

**printf("Its Hot.\n");**

**else**

**printf("Its very hot.\n");**

**}**

**Out put**

**Input days’ temperature: 42**

**It’s very hot.**

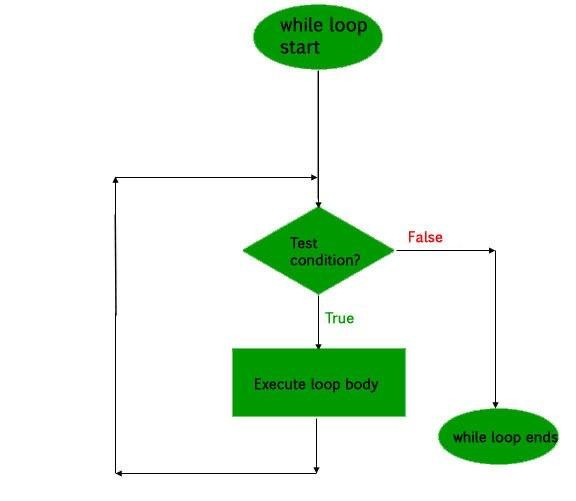
**3(a)**

**Ans**

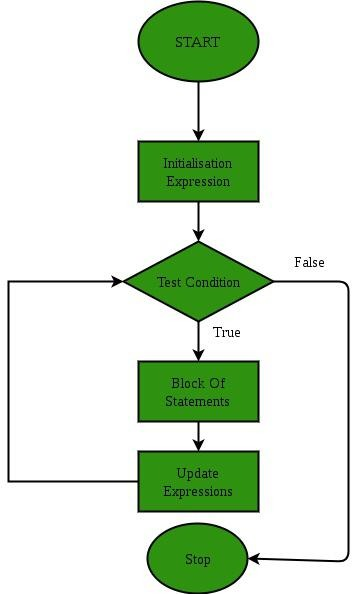
**Loops are very useful when you want to perform a task repeatedly. Loop’s body has set of statements, which gets executed on every iteration until a given condition is met.**

**We have three types of loops in C. The working of these loops are almost similar, however they are being used in different scenarios. You may need to choose the loop based on the requirement**

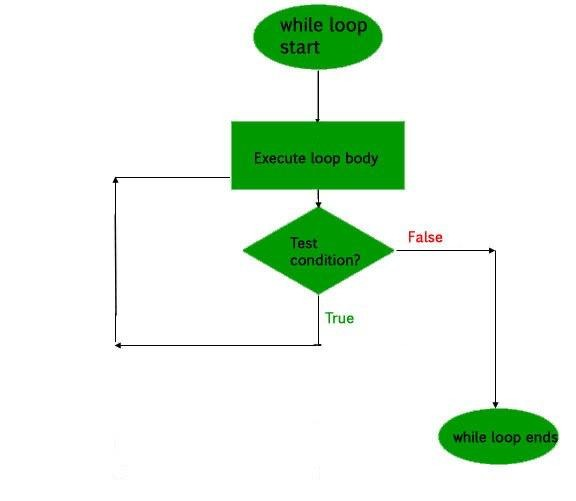
**While loop**



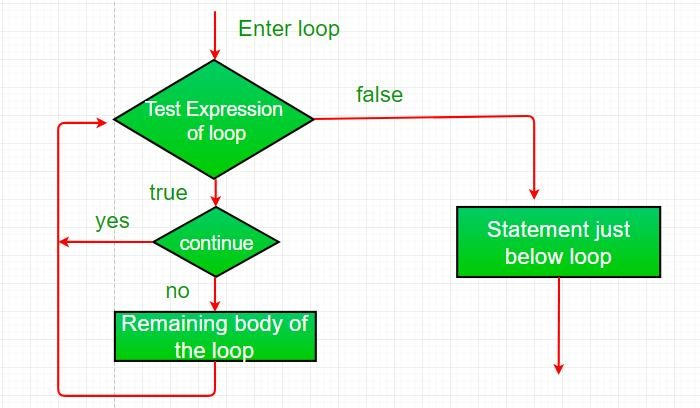
**For loop**



**Do while loop**



**Continues loop**



**3(b)**

**Ans**

**for loop:**

**This is most commonly used loop in C language. The syntax and flow of this loop is simple and easy to learn. However there are few cases when you may prefer any other loop, instead of this.**

**while loop:**

**This is used when you need to execute a block of statements repeatedly until a given condition is met. Read this tutorial to understand the flow of this loop.**

**do-While loop:**

**It is similar to the while loop, the only difference is that it evaluates the test condition after execution of the statements enclosed in the loop body.**

**break statement: It is used with various loops (for, while and do-While) and switch case statements. When a break statement is encountered inside a loop, the control comes out of the loop. When it gets encountered in switch-case, the control comes out of the switch case and continue execution with the statement following switch-case body.**

**continue statement:**

**Continue statement is used inside loops. Whenever it is encountered inside a loop, control directly jumps to the beginning of the loop for next iteration, skipping the execution of statements inside loop’s body for the current iteration.**

**goto statement:**

**When goto statement is encountered in a C program, the control jumps to the mentioned label. It is rarely used as it makes the program complex and confusing.**

**for loop:**

**This is most commonly used loop in C language. The syntax and flow of this loop is simple and easy to learn. However there are few cases when you may prefer any other**

**loop, instead of this.**

**Example of For loop**

**#include <stdio.h>**

**int main()**

**{**

**int i;**

**for (i=1; i<=3; i++)**

**{**

**printf("%d\n", i);**

**}**

**return 0;**

**}**

**while loop:**

**This is used when you need to execute a block of statements repeatedly until a given condition is met. Read this tutorial to understand the flow of this loop**

**Example of while loop**

**#include <stdio.h>**

**int main()**

**{**

**int count=1;**

**while (count <= 4)**

**{**

**printf("%d ", count);**

**count++;**

**}**

**return 0;**

**}**

**do-While loop:**

**It is similar to the while loop, the only difference is that it evaluates the test condition after execution of the statements enclosed in the loop body.**

**Example of do while loop**

**#include <stdio.h>**

**int main()**

**{**

**int j=0;**

**do**

**{**

**printf("Value of variable j is: %d\n", j);**

**j++;**

**}while (j<=3);**

**return 0;**

**}**

**break statement:**

**It is used with various loops (for, while and do-While) and switch case statements. When a break statement is encountered inside a loop, the control comes out of the loop. When it gets encountered in switch-case, the control comes out of the switch case and continue execution with the statement following switch-case body**

**#include <stdio.h>**

**int main () {**

**/\* local variable definition \*/**

**int a = 10;**

**/\* while loop execution \*/**

**while( a < 20 ) {**

**printf("value of a: %d\n", a);**

**a++;**

**if( a > 15) {**

**/\* terminate the loop using break statement \*/**

**break;**

**}**

**}**

**return 0;**

**}**

**continue statement:**

**Continue statement is used inside loops. Whenever it is encountered inside a loop, control directly jumps to the beginning of the loop for next iteration, skipping the execution of statements inside loop’s body for the current iteration.**

**Q 4**

**Ans**

**A for loop is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.**

**Syntax**

**The syntax of a for loop in C# is −**

**for ( init; condition; increment ) {**

**statement(s);**

**}**

**Here is the flow of control in a for loop −**

**• The init step is executed first, and only once. This step allows you to declare and initialize any loop control variables. You are not required to put a statement here, as long as a semicolon appears.**

**• Next, the condition is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement just after the for loop.**

**• After the body of the for loop executes, the flow of control jumps back up to the increment statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.**

**• The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again testing for a condition). After the condition becomes false, the for loop terminates.**

**Flow Diagram**

**Example**

**Live Demo**

**using System;**

**namespace Loops {**

**class Program {**

**static void Main(string[] args) {**

**/\* for loop execution \*/**

**for (int a = 10; a < 20; a = a + 1) {**

**Console.WriteLine("value of a: {0}", a);**

**}**

**Console.ReadLine();**

**}**

**}**

**}**

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

**value of a: 10**

**value of a: 11**

**value of a: 12**

**value of a: 13**

**value of a: 14**

**value of a: 15**

**value of a: 16**

**value of a: 17**

**value of a: 18**

**value of a: 19**

**Q5(a)**

**Ans**

**Encapsulation is one of the fundamentals of OOP (object-oriented programming). It refers to the bundling of data with the methods that operate on that data. Encapsulation is used to hide the values or state of a structured data object inside a class, preventing unauthorized parties' direct access to them.**

**In object-oriented computer programming languages, the notion of encapsulation refers to the bundling of data, along with the methods that operate on that data, into a single unit. Many programming languages use encapsulation frequently in the form of classes. A class is a program-code-template that allows developers to create an object that has both variables (data) and behaviors (functions or methods). A class is an example of encapsulation in that it consists of data and methods that have been bundled into a single unit.**

**Encapsulation may also refer to a mechanism of restricting the direct access to some components of an object, such that users cannot access state values for all of the variables of a particular object. Encapsulation can be used to hide both data members and data functions or methods associated with an instantiated class or object**

**5(b)**

**Encapsulation also lead to data abstraction or hiding. As using encapsulation also hides the data. In the above example the data of any of the section like sales, finance or accounts is hidden from any other section.**

**In C++ encapsulation can be implemented using Class and access modifiers**

**#include<iostream>**

**using namespace std;**

**class Encapsulation**

**{**

**private:**

**// data hidden from outside world**

**int x;**

**public:**

**// function to set value of**

**// variable x**

**void set(int a)**

**{**

**x =a;**

**}**

**// function to return value of**

**// variable x**

**int get()**

**{**

**return x;**

**}**

**};**

**// main function**

**int main()**

**{**

**Encapsulation obj;**

**obj.set(5);**

**cout<<obj.get();**

**return 0;**

**}**

**Out put**

