

#### Module-03

**Managing Input and Output operations: Reading & writing a character, formatted input and output. Decision Making and Branching: Decision making with if statement, simple if statement, the if else statement, nesting of if ... else statements, the else if ladder, the switch statement, the?: operator, the goto statement. Decision making and looping: The while statement, the do statement, for statement, exit, break, jumps in loops.**

### Managing Input and output operations

Each program revolves round some data. Of course, data are the reasons for the existence of programs. Programs require some inputs to be given; they will then process the inputs to produce the expected outputs.

Accepting inputs, processing the inputs to produce the expected outputs and presenting the outputs to the outside world are the three fundamental functions of programs. Accepting the required inputs from input devices and displaying the produced results on output devices are referred to as **Input-Output operations**.

Each program that uses a standard input/output functions must contain the header file **stdio.h** (standard input output header file). We need to include it as part of our source program file with the help of preprocessor directive #include as: #include<stdio.h>

I/O operations are helpful for a program to interact with users. C stdlib is the standard C library for input-output operations. Two essential streams play their role when dealing with input-output operations in C.

These are:

1. Standard Input (stdin)
  2. Standard Output (stdout)
- Standard input or stdin is used for taking input from devices such as the keyboard as a data stream.
  - Standard output or stdout is used to give output to a device such as a monitor. For I/O functionality, programmers must include the stdio header file within the program.
  - One of the essential operations performed in a C language program is to provide input values to the program and output the data produced by the program to a standard output device.
  - We can assign values to variable through assignment statements such as x = 5 a = 0; and so on.

### PROBLEM SOLVING USING C PROGRAMMING (BCA103)

---

- Another method is to use the Input then scanf which can be used to read data from a keyboard.
- For outputting results, we have used extensively the function printf which sends results out to a terminal.
- There exist several functions in 'C' language that can carry out input output operations.
- These functions are collectively known as standard Input/Output Library.
- Each program that uses standard input / output function must contain the statement.

# include < stdio.h > at the beginning.

#### **Example:**

```
#include <stdio.h>
```

```
void main()
```

```
{
```

```
int a, b, c;
```

```
printf("Please enter any two numbers: \n");
```

```
scanf("%d %d", &a, &b);
```

```
c = a + b;
```

```
printf("The addition of two number is: %d", c);
```

```
}
```

#### **Output**

Please enter any two numbers: 12 3

The addition of two number is:15

In the above program, the scanf() function takes input from the user, and the printf() function displays the output result on the screen.

### PROBLEM SOLVING USING C PROGRAMMING (BCA103)

---

#### Reading A Character

- The easiest and simplest of all I/O operations are taking a character as input by reading that character from standard input (keyboard).
- `getchar()` function can be used to read a single character. This function is an alternative to `scanf()` function.
- Reading a single character can be done by using the function `getchar`.

#### Syntax:

```
variable_name = getchar();
```

Example: `char name;`

```
name=getchar();
```

The **`getc()`** function is used to read a character from a file whereas **`fgetc()`** is used to read a line of text from the file. The syntax for the **`getc()`** function is:

```
int getc(FILE* fp);
```

The **`getc()`** function reads and returns a single character at a time from a file. It either returns the character or an **end of file (EOF)** character. EOF is a character which is returned when there are no more characters to be read from the file or when the process encounters an error.

The syntax for the **`fgetc()`** function is:

```
char[] fgetc(char line [], int maxlen,FILE* fp);
```

where the function reads and returns a single line (up to 'maxlen' characters) from the input file.

#### Reading a character from terminal

```
#include<stdio.h>
```

```
void main()
```

```
{
```

```
char ans;
```

```
printf("Would you like to know my name? \n");
```

```
printf("Type Y for yes and N for no");
```

### PROBLEM SOLVING USING C PROGRAMMING (BCA103)

---

```
ans=getchar();

if(ans == 'Y' || ans == 'y')

printf("\n\n My name is India \n");

else

printf("\n\n You are good for nothing \n");

}
```

#### OUTPUT

Would you like to know my name?

Type Y for yes and N for no:Y

My name is India

Would you like to know my name?

Type Y for yes and N for no:n

You are good for nothing

#### Writing A Character

- Similar to getchar(), there is another function that is used to write characters, but one at a time.

##### Syntax:

```
putchar(var_name);
```

- Similarly, there is another function putc() which is used for sending a single character to the standard output.

##### Syntax:

```
int putc(int c, FILE * stream);
```

The **putc()** function is used to write a character to a file whereas the **fputs()** function is used to write a line of text into a file.

##### The syntax for putc is:

```
int putc(char c,FILE* fp);
```

### PROBLEM SOLVING USING C PROGRAMMING (BCA103)

---

where c is the char you wish to write into the file and fp is the file pointer to that file.

**The syntax for fputs is:**

```
int fputs(char line , FILE* fp);
```

where 'line' is the text you wish to write in the file and fp is the file pointer.

**Example:**

```
#include<stdio.h>
```

```
void main()
```

```
{
```

```
char result = 'P';
```

```
putchar(result);
```

```
putchar('\n');
```

```
}
```

**Example: Check weather a alphabet is upper or lower.**

```
#include<stdio.h>
```

```
#include <conio.h>
```

```
main()
```

```
{
```

```
char alphabet;
```

```
printf("Enter an alphabet");
```

```
putchar('\n');
```

```
alphabet = getchar();
```

```
if(islower(alphabet)) putchar(toupper(alphabet));
```

```
else
```

```
putchar(tolower(alphabet));
```

```
}
```

### PROBLEM SOLVING USING C PROGRAMMING (BCA103)

---

#### OUTPUT

Enter An alphabet a A

Enter An alphabet q Q

Enter An alphabet z Z

#### Formatted I/O Functions

Formatted I/O functions are used to take various inputs from the user and display multiple outputs to the user. These types of I/O functions can help to display the output to the user in different formats using the format specifiers. These I/O supports all data types like int, float, char, and many more.

These functions are called formatted I/O functions because we can use format specifiers in these functions and hence, we can format these functions according to our needs.

Ex: 15.57 123 John

The line contains three pieces of data, arranged in a particular form. Such data has to be read conforming to the format of its appearance. For example, the first part of the data should be read into a variable float, the second into int, and the third part into char. This is possible in C using the scanf function.

In C, formatted input/output statements are used to display data to the screen (printf) or accept input from the user (scanf). These functions are part of the standard input/output library, so you need to include <stdio.h> at the beginning of your program.

#### Formatted Output: printf

The printf function is used to display text and variables. It uses format specifiers to determine how the output should be formatted.

##### Syntax:

```
printf("control string", arg1, arg2, ..., argn);
```

**or**

```
printf("format string", variables);
```

##### Example:

### PROBLEM SOLVING USING C PROGRAMMING (BCA103)

---

```
#include <stdio.h>

int main() {

int num = 10;

float pi = 3.14159;

char letter = 'A';

char name[] = "John";

printf("Integer: %d\n", num);

printf("Float: %.2f\n", pi); // Print float with 2 decimal places

printf("Character: %c\n", letter);

printf("String: %s\n", name);

return 0;

}
```

#### **Output:**

Integer: 10

Float: 3.14

Character: A

String: John

#### **Formatted Input: scanf**

The scanf function is used to accept input from the user. It also uses format specifiers to determine the type of input.

It refers to input data that has been arranged in a specific format. This is possible in C using scanf() function. We have already encountered this and are familiar with this function.

#### **Syntax:**

```
scanf("control string", arg1, arg2, ..., argn);
```

The control string specifies the field format in which the data is to be entered and the arguments arg1, arg2, ..., argn specify the address of locations where data is stored. Control string and arguments are separated by commas.

### PROBLEM SOLVING USING C PROGRAMMING (BCA103)

---

**Syntax:**

```
scanf("format string", &variables);
```

**Example:**

```
#include <stdio.h>

int main()
{
    int age;

    float height;

    printf("Enter your age: ");

    scanf("%d", &age);

    printf("Enter your height (in meters): ");

    scanf("%f", &height);

    printf("You are %d years old and %.2f meters tall.\n", age, height);

    return 0;
}
```

**Output**

Enter your age: 25

Enter your height (in meters): 1.75

You are 25 years old and 1.75 meters tall.



### PROBLEM SOLVING USING C PROGRAMMING (BCA103)

---

#### List of some format specifiers-

S NO.	Format Specifier	Type	Description
1	%d	int/signed int	used for I/O signed integer value
2	%c	char	Used for I/O character value
3	%f	float	Used for I/O decimal floating-point value
4	%s	string	Used for I/O string/group of characters
5	%ld	long int	Used for I/O long signed integer value
6	%u	unsigned int	Used for I/O unsigned integer value
7	%i	unsigned int	used for the I/O integer value
8	%lf	double	Used for I/O fractional or floating data
9	%n	prints	prints nothing

### CONDITIONAL BRANCHING STATEMENTS OR DECISION MAKING

The conditional branching statements help to jump from one part of the program to another depending on whether a particular condition is satisfied or not.

These decision-making statements include:

- if statement
- if-else statement
- Nested if else statement
- Else if ladder
- switch statement
- conditional operator
- goto statement

#### **If statement:**

The if statement is used to control the flow of execution of statements. It is basically a two-way decision statement and is used in conjunction with an expression.

**if(test expression)**

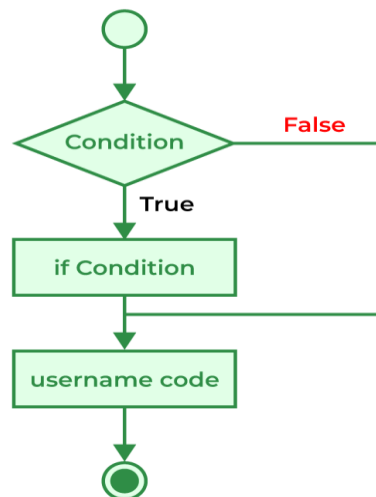
it allows the computer to evaluate the expression first and then depending on whether the value of the expression is true or false, it transfers the control to a particular statement.

#### **Syntax:**

```
if(expression)
{
//code to be executed
}
```

The 'statement-block' may be a single statement or a group of statements. If the test expression is true, the statement-block will be executed; otherwise, the statement-block will be skipped and the execution will jump to the statement-x.

#### **Flowchart of if statements**



**Example program: Check Whether the Number is Even or Odd.**

```
#include<stdio.h>

void main()
{
    int num, rem;

    printf("enter a number\n");

    scanf("%d",&num);

    rem=num%2;

    if(rem==0)

    printf("%d is even",num);

    if(rem!=0)

    printf("%d is odd",num);

    getch();

}
```

#### **Output**

Enter a number: 4

4 is even number

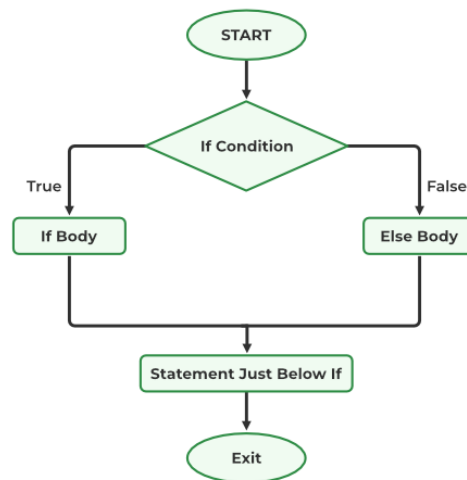
## **2. The if...else statement:**

The if-else statement is a decision-making statement that is used to decide whether the part of the code will be executed or not based on the **specified condition (test expression)**. If the given condition is true, then the code inside the if block is executed, otherwise the code inside the else block is executed.

**Syntax:**

```
if(test expression)
{
    true-block statements
}

else
{
    False-block statements
}
```

**Flowchart:**

If the test exp. is true, then the true block statements, immediately following the if statements are executed otherwise, the false-block statements are executed. In either case, either true-block or false-block will be executed, not both.

**example program: Write a program to find whether the given number is even or odd.**

```
#include<stdio.h>
```

```
void main()
{
    int num, rem;
    printf("enter a number\n");
    scanf("%d", &num);
    rem = num % 2;

    if (rem == 0)
        printf("%d is even", num);
    else
        printf("%d is odd", num);
    getch();
}
```

#### **Output**

Enter any number: 11

11 is an odd number

### **3. The nested if- else statement:**

When a series of decisions are involved, we may have to use more than one if-else statement in nested form as shown below.

#### **Syntax: –**

```
if (test condition1)
{
    if(test condition2)
    {
        Statement1;
    }
}
```

```
else
```

```
{
```

```
    Statement2
```

```
}
```

```
else
```

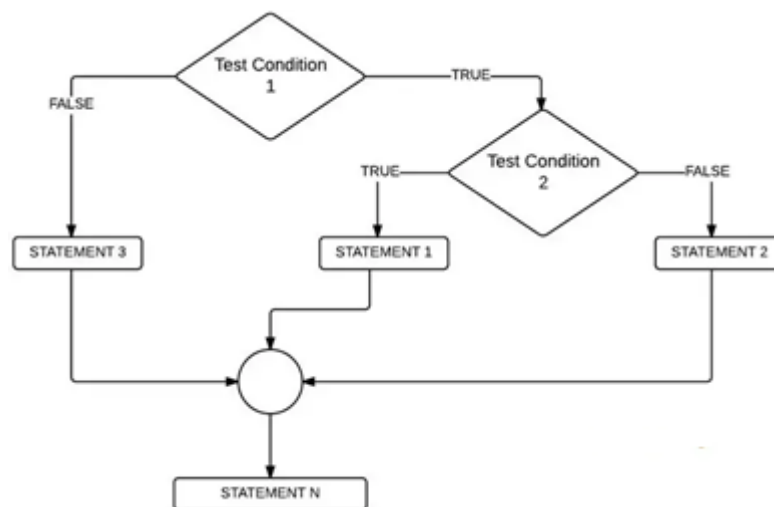
```
{
```

```
    Statement3;
```

```
}
```

```
Statement-x;
```

If the condition1 is false, statement 3 will be executed otherwise it continues to perform the 2<sup>nd</sup> test. If the condition2 is true, the statement1 will evaluated & then the control is transferred to the statement-x.



**Example : Write a program to find largest of three numbers.**

```
#include<stdio.h>
```

```
void main()
```

```
{
```

```
    int a,b,c;
```

### PROBLEM SOLVING USING C PROGRAMMING (BCA103)

---

```
printf("Enter 3 numbers: \n");  
scanf("%d%d%d",&a,&b,&c);  
if(a>=b)  
{  
    if(a>=c)  
        printf("%d is largest",a);  
    else  
        printf("%d is largest",c);  
}  
else if(b>=c)  
    printf("%d is largest",b);  
else  
    printf("%d is largest",c);  
}
```

#### **Output**

Enter three number: 4 6 10

10 is largest

#### **5.The else if ladder:**

There is another way of putting if's together when multipath decisions are involved. A multipath decision is a chain of if's in which the statement associated with each else is an if. The general form is:

#### **Syntax:**

```
if (condition1)  
{  
    Statement-1;  
}
```

```
else if(condition2)
{
statement-2;
}

else if(condition3)
{
statement-3

}

else if(condition-n)
{
statement-n
}

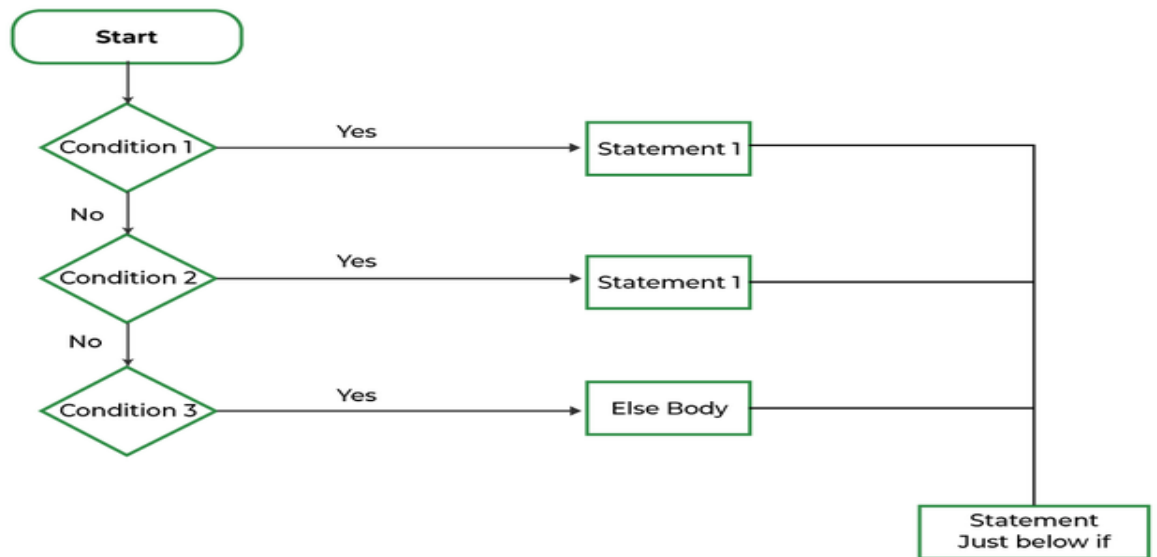
else
{
Statements;
}
```

This construction is known as the else if ladder.

The conditions are evaluated from the top, downward as soon as a true condition is found, the statement associated with it is executed & the control is transferred to the statement-x (skipping the rest of the ladder).

When all the n conditions become false then the final else containing the default statement will be executed.



**Flowchart**

**Example:** Write a program to find whether the given number is less than or greater than.

```
#include<stdio.h>
```

```
void main()
```

```
{
```

```
int n;
```

```
printf("enter the value of n");
```

```
scanf("%d",&n);
```

```
if(n<=10)
```

```
printf("n is less than or equal to 10");
```

```
else if(n<=20)
```

```
printf("n is less than or equal to 20");
```

```
else if(n<=30)
```

```
printf("n is less than or equal to 30");
```

```
else
```

```
printf("n is greater than 30");  
  
getch();  
  
}
```

#### **Output**

Enter the value of n: 30

30 is less than or equal to 30

#### **6. The switch statement: –**

C has a built-in multiway decision statement known as a switch. The switch statement tests the value of a given variable (or expression) against a list of case values & when a match is found, a block of statements announced with that case is executed.

The general form is :

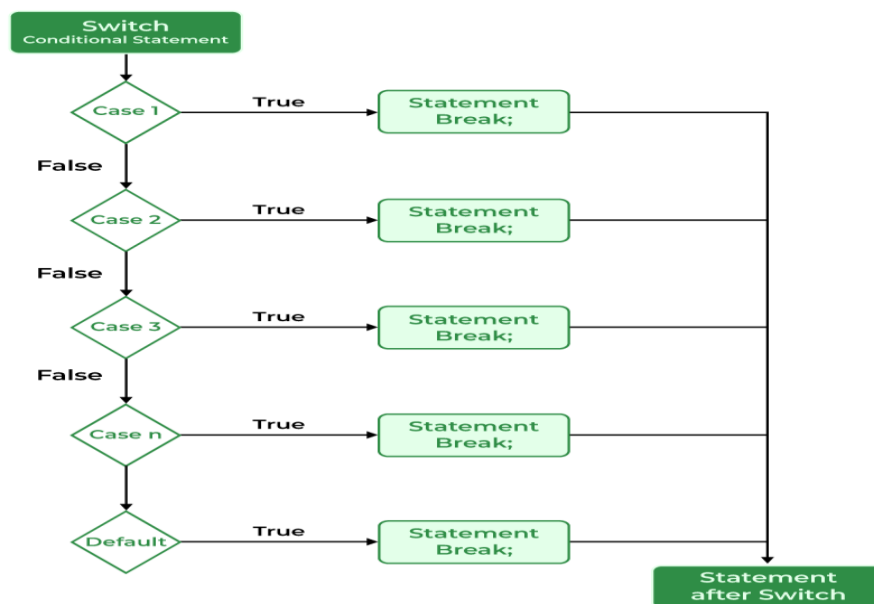
#### **Syntax:**

```
switch(expression)  
{  
    case value-1:  
        block-1;  
        break;  
    case value-2:  
        block-2;  
        break;  
    ----  
    ---  
    default: default-block;  
    break;  
}
```

Statement-x;

The switch case statement compares the value of the variable given in the switch statement with the value of each case statement that follows. When the value of the switch and the case statement matches, the statement block of that particular case is executed. Default is also a case that is executed when the value of the variable does not match with any of the values of the case statement, i.e., the default case is executed when no match is found between the values of switch and case statements and thus there are no statements to be executed. Although the default case is optional, it is always recommended to include it as it handles any unexpected cases. In the syntax of the switch case statement, we have used another keyword break. The break statement must be used at the end of each case because if it is not used, then the case that matched and all the following cases will be executed.

### Flowchart:



#### Rules for switch statement in C language

- The expression is an integer expression or characters.
- value-1, value-2 are constants (or) constants expressions and are known as case labels. Each of these values should be unique with in a switch statement.
- block-1, block-2, are statement list & may contain 0 or more statements. There is no need to put braces around blocks.
- Note that case labels end with a colon(:)
- When the switch is executed, the value of the exp is successfully compared against the values value-1, value-2,
- If a case is found whose value matches with the value of the exp, then the block of statements that follow the case are executed.
- The break statement at the end of each block signals the end of a particular case and causes an exit from the switch statement transforming the control to the statement x following the switch.

The default is an optional case. When present, it will be executed if the value of the exp. does not present no action takes place if all matches fail& the control goes to the statement.

**Example: Write a program to determine whether an entered character is a vowel or not.**

```
#include

int main()

{

char ch;

printf("\n Enter any character:");

scanf("%c", &ch);

switch(ch)

{

case 'A':

case 'a':

printf("\n% c is vowel", ch);

break;

case 'E':
```

```
case 'e':  
    printf("\n% c is vowel", ch);  
    break;  
case 'I':  
case 'i':  
    printf("\n% c is vowel", ch);  
    break;  
case 'O':  
case 'o':  
    printf("\n% c is vowel", ch);  
    break;  
case 'U':  
case 'u':  
    printf("\n% c is vowel", ch);  
    break;  
default:  
    printf("%c is not a vowel", ch);  
}  
return 0;  
}
```

#### **Output**

Enter any character: E

E is vowel

#### The ?: operator:–

Conditional op is useful for making 2–way decisions. This op is a combination of ?and :, & takes 3 operands. The general form is :

**Syntax:–** conditional exp?exp–1:exp–2

The conditional exp. is evaluated first. If the result is nonzero, exp–1 is evaluated & its value is returned.

**Ex:–** if(a>b) big=a;

Else

big=b;

can be written as

big=(a>b)?a:b

#### Goto statement: –

- C supports the goto statement to branch unconditionally from one point to another in the program.
- The goto requires a label in order to identify the place where the branch is to be made. A label is any valid variable name & must be followed by a colon.
- The label is placed immediately before the statement where the control is to be transferred. The general forms of goto& label statements are:
- The label can be anywhere in the program either before or after the goto label; statement.
- If the label; is before the statement goto label; a loop will be formed & some statements will be executed repeatedly. Such a jump is known as a **backward jump**.
- On the other hand, if the label: is placed after the goto label; some statements will be skipped & the jump is known as **forward jump**.

- **Example of forward jump:**

- If(age>=18)
  - goto vote;

- else
- Printf("Not Eligible");
- Vote: printf("eligible for vote");
- **Example of backward jump: printing of number from 1 to n**
- i=1;
- Loop: printf("%d\n",i);
- i++;
- If(i<n)
- goto loop;

**Example: Write a program to determine whether an entered number is Eligible for voting or not.**

```
#include<stdio.h>

void main()

{

    int age;

    g: //label name

    printf("you are Eligible\n");

    s: //label name

    printf("you are not Eligible");

    printf("Enter you age:");

    scanf("%d", &age);

    if(age>=18)

        goto g; //goto label g

    else
```

```
goto s; //goto label s  
  
}
```

### Looping

There may be a situation when you need to execute a block of code several number of times.

In general, statements are executed sequentially:

The first statement in a function is executed first, followed by the second, and so on. Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times.

- Loops are structures that control repeated executions of a block of statements.
- Part of the loop that contains the statements to be repeated is called the loop body.
- A one-time execution of a loop body is referred to as an iteration of the loop.
- Each loop contains a loop-continuation-condition, a Boolean expression that controls the execution of the body.
- After each iteration, the loop-continuation-condition is reevaluated. If the condition is true, the execution of the loop body is repeated. If the condition is false, the loop terminates.

### ITERATIVE STATEMENTS

Iterative statements are used to repeat the execution of a list of statements, depending on the value of an integer expression. C language supports three types of iterative statements also known as looping statements.

**They are:**

- while loop
- do-while loop
- for loop



#### 1.While loop

A while loop statement in C programming language repeatedly executes a target statement as long as a given condition is true.

##### Syntax:

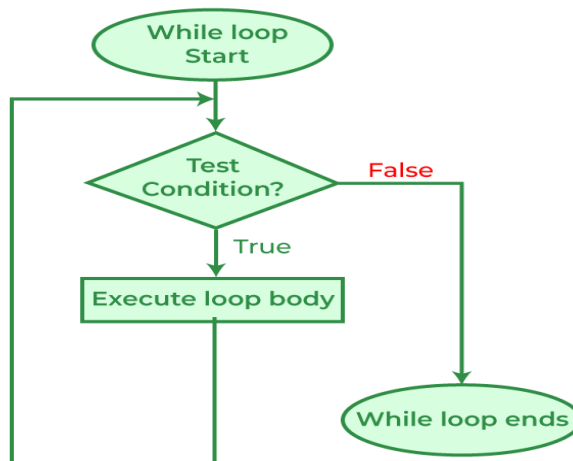
```
while(condition)
```

```
{
```

```
statement(s);
```

```
}
```

##### Flowchart



- Here, statement(s) may be a single statement or a block of statements.
- The condition may be any expression, and true is any nonzero value. □The loop iterates while the condition is true.
- When the condition becomes false, program control passes to the line immediately following the loop.

Here, key point of the while loop is that the loop might not ever run.

### PROBLEM SOLVING USING C PROGRAMMING (BCA103)

---

When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

**Example: Program to find the sum of n natural numbers.**

```
#include <stdio.h>

void main()
{
    int n, sum = 0, i = 1;

    printf("Enter the value of n: ");

    scanf("%d", &n);

    while (i <= n)
    {
        sum = sum + i;

        i++;
    }

    printf("The sum of the first %d natural numbers is: %d\n", n, sum);
}
```

**Output:**

Enter the value of n: 5

The sum of the first 5 natural numbers is 15

## 2.For loop

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.

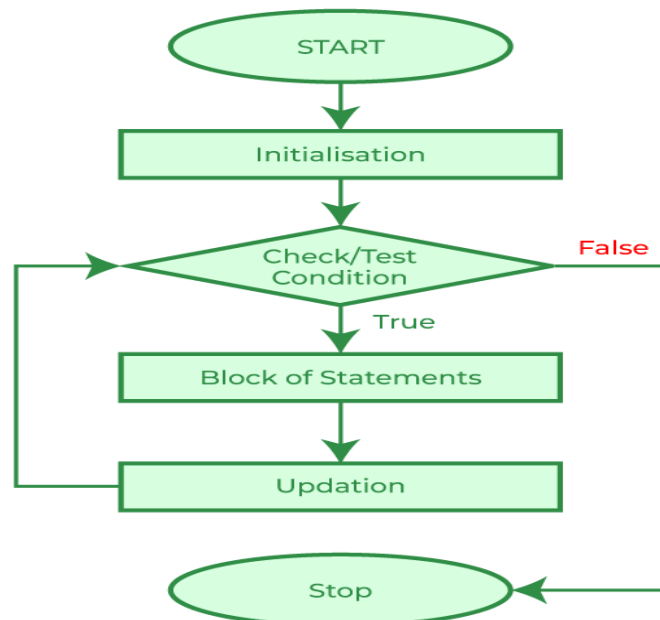
Like the while and do-while loops, the for loop provides a mechanism to repeat a task until a particular condition is true. For loop is usually known as a determinate or definite loop because the programmer knows exactly how many times the loop will repeat. The number

of times the loop has to be executed can be determined mathematically by checking the logic of the loop.

**Syntax:**

for (initialization; test expression; action-after-each-iteration/updation)

```
{  
    Statement(s);  
}
```

**Flowchart**

Here is the flow of control in a for loop:

### PROBLEM SOLVING USING C PROGRAMMING (BCA103)

---

- The initialization 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 condition). After the condition becomes false, the for loop terminates.

**Example: Program to find the sum of first n numbers.**

```
#include <stdio.h>

void main()
{
    int n, i, sum;

    printf("Enter a number: ");

    scanf("%d", &n);

    sum = 0;

    for (i = 1; i <= n; i++)
    {
        sum = sum + i;
    }

    printf("The sum of the first %d natural numbers is: %d\n", n, sum);
}
```

#### **Output**

Enter a number: 5

The sum of the first 5 natural numbers is 15

#### 3.Do while loop

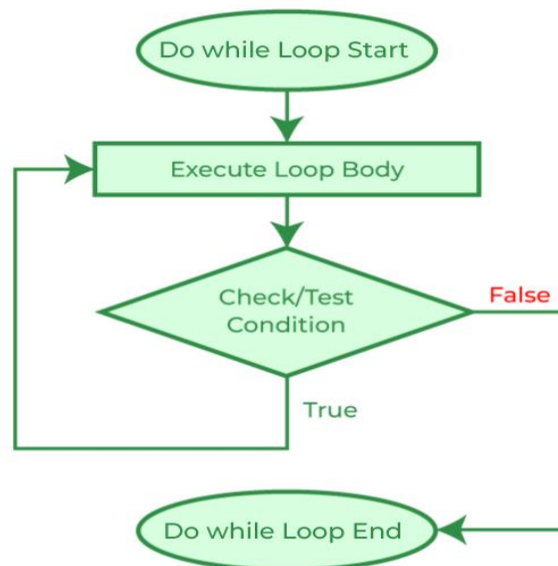
Unlike for and while loops, which test the loop condition at the top of the loop, the do...while loop in C programming language checks its condition at the bottom of the loop.

A do...while loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

#### Syntax:

```
do
{
statement(s);
}
while(condition );
```

#### Flowchart:



- Notice that the conditional expression appears at the end of the loop, so the statement(s) in the loop execute once before the condition is tested.

### PROBLEM SOLVING USING C PROGRAMMING (BCA103)

---

- If the condition is true, the flow of control jumps back up to do, and the statement(s) in the loop execute again. This process repeats until the given condition becomes false.

**Example: sum of n numbers using do while loop.**

```
#include<stdio.h>

void main()

{
intn,i,sum;

printf("enter a number");

scanf("%d",&n);

i=1;

sum=0;

do

{

sum=sum+i;

i++;

}

while(i<=n);

printf("The sum of the first %d natural numbers is: %d\n", n, sum);

}
```

#### **Output**

Enter a number: 5

The sum of the first 5 natural numbers is 15

**Difference between While and do while**

<b>while</b>	<b>do-while</b>
Condition is checked first then statement(s) is executed.	Statement(s) is executed atleast once, thereafter condition is checked.
It might occur statement(s) is executed zero times, If condition is false.	At least once the statement(s) is executed.
No semicolon at the end of while. while(condition)	Semicolon at the end of while. while(condition);
Variable in condition is initialized before the execution of loop.	variable may be initialized before or within the loop.
while loop is entry controlled loop.	do-while loop is exit controlled loop.
while(condition) { statement(s); }	do { statement(s); } while(condition);

**BREAK AND CONTINUE STATEMENTS**

**Break statement** In C, the break statement is used to terminate the execution of the nearest enclosing loop in which it appears. When the compiler encounters a break statement, the control passes to the statement that follows the loop in which the break statement appears. Its syntax is quite simple, just type keyword break followed with a semicolon.

In switch statement if the break statement is missing then every case from the matched case label till the end of the switch, including the default, is executed.

**Example:**

```
#include<stdio.h>
```

### PROBLEM SOLVING USING C PROGRAMMING (BCA103)

---

```
int main()

{

int i = 1;

while(i<=10)

{

if (i==5)

break;


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

i = i + 1;

}

return 0;

}
```

#### **Goto STATEMENT**

The goto statement is used to transfer control to a specified label. However, the label must reside in the same function and can appear only before one statement in the same function. Here, label is an identifier that specifies the place where the branch is to be made. Label can be any valid variable name that is followed by a colon (:). The label is placed immediately before the statement where the control has to be transferred. The label can be placed anywhere in the program either before or after the goto statement. Whenever the goto statement is encountered, the control is immediately transferred to the statements following the label. Therefore, goto statement breaks the normal sequential execution of the program. If the label is placed after the goto statement, then it is called a forward jump and in case it is located before the goto statement, it is said to be a backward jump.

#### **Example**

```
#include <stdio.h>

int main()

{
```



### PROBLEM SOLVING USING C PROGRAMMING (BCA103)

---

```
int num, sum=0;

read:// label for goto statement

printf("\n Enter the number. Enter 999 to end: ");

scanf("%d", &num);

if (num != 999)

{

if(num < 0)

goto read; // jump to label- read


    sum += num;

    goto read; // jump to label- read

}


printf("\n Sum of the numbers entered by the user is = %d", sum);

return 0;

}
```

#### **Exit()**

1. We must include the `stdlib.h` header file while using the `exit ()` function.
2. It is used to terminate the normal execution of the program while encountered the `exit ()` function.
3. The `exit ()` function calls the registered `atexit()` function in the reverse order of their registration.
4. We can use the `exit()` function to flush or clean all open stream data like read or write with unwritten buffered data.
5. It closed all opened files linked with a parent or another function or file and can remove all files created by the `tmpfile` function.
6. The program's behaviour is undefined if the user calls the `exit` function more than one time or calls the `exit` and `quick_exit` function.

7. The exit function is categorized into two parts: exit(0) and exit(1).

#### **Syntax of the exit() function**

**void** exit ( **int** status);

#### **Example:**

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
int main ()
```

```
{
```

```
// declaration of the variables
```

```
int i, num;
```

```
printf ( " Enter the last number: ");
```

```
scanf ( " %d", &num);
```

```
for ( i = 1; i<num; i++)
```

```
{
```

```
// use if statement to check the condition
```

```
if ( i == 6 )
```

```
/* use exit () statement with passing 0 argument to show termination of the program without any error message. */
```

```
exit(0);
```

```
else
```

```
printf ( " \n Number is %d", i);
```

```
}
```

```
return 0;
```

```
}
```

#### **Output**

Enter the last number: 10

Number is 1

Number is 2

Number is 3

Number is 4

Number is 5