

Module-2

Table of contents

Topics	Page No
1.1 Input and Output Functions	2
2.1 Conditional Branching and Loops: Introduction	10
2.2 Decision making Statements/Branching	10
2.3 Looping	29
2.4 Unconditional Branch Statements (Jump statements)	46
Web Resources	50
Video Resources	51
Question Bank	52
University Questions	53

Chapter 1

Managing Input and Output Operations

1.1 Input and Output Functions

C-programming Language provides several input and output functions to perform input/output operations. **stdio.h** is the header file that contains standard I/O functions namely:

- `scanf()`
- `printf()`
- `getchar()`
- `putchar()`
- `gets()`
- `puts()`

1.1.1 Unformatted Input/Output Functions

- Unformatted I/O functions are those which do not specify data types and ways in which it should be read or printed.

Reading a Character(Input)

- `getchar()` function - `getchar()` function is used to get/read a character from keyboard input.
- **Syntax:** `ch=getchar();`

Where `ch` is a character type variable.

- In a C program, we can use `getchar` function as shown below

`char grade;`

`grade = getchar();`

The `getchar()` function will accept the character entered from keyboard and stores in variable `grade`.

Writing a Character(Output)

- `putchar()` function - `putchar()` is function used to write a character on standard output/screen.
- **Syntax:** `putchar(ch);`

where `ch` is a character or character type variable.

- In a C program, we can use `putchar()` function as shown below.

`putchar('A');` OR `char grade = 'A';`

`putchar(grade);`

Example Program

//Program to accept a character from keyboard and display the same.

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

```
void main()
```

```
{
```

```
    char c;
```

```
    c=getchar( ); //reading a character
```

```
    putchar(c); //displaying the character stored in variable c
```

```
}
```

Sample output:

A

A

Reading a String(Input)

- **gets()** - gets() function is used to read a string(set of characters) from keyboard.
- **Syntax:** gets(str);

Where str is a string type variable.

- In a C program, we can use gets() function as shown below

```
char str[10];
```

```
gets( str);
```

Writing a String(Output)

- **puts()** - puts() is function used to write a string (set of characters) on standard output/screen.

Syntax: puts(ch);

where ch is a string or string type variable.

- In a C program, we can use puts() function as shown below.

```
putchar("hello");
```

OR

```
char str[ ] = "hello";
```

```
puts(str);
```

Example Program

//Program to accept a string from keyboard and display the same.

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

```
void main()
```

```
{
```

```
    char str[10];
```

```
    gets(str); //reading the string
```

```
    puts(str); //displaying the string stored in variable str
```

```
}
```

Sample output:

hello

hello

1.1.2 Formatted Input**scanf()**

- The **scanf()** standard library function is used to read one or more values from the standard input (keyboard) and assign them to specified variables
- **Syntax:**

scanf("ControlString",address_of_variables);

Guidelines for scanf():

- A **scanf()** always contains a control string or format string in quotation marks.
- Each value to be printed needs a conversion specification like %d to hold its place in the control string.
- There should be a comma between the control string and the addresses of variables.

Reading Integer Numbers

- The field specification for reading an integer number is:

%wd

- Where % indicated conversion specification follows,
 - w is the field width of the integer to be read and
 - d known as data type character which indicated integer is to be read.

Example:

```
scanf("%2d %5d", &num1,&num2);
```

- If 50 12345 are read then *num1* contains value 50 and *num2* contains 12345.
- If 12345 50 are read then *num1* contains 12, *num2* contains 345 and 50 is given as input for next `scanf()` statement.
- To avoid this problem default %d can be used like,

```
scanf("%d%d", &num1,&num2);
```

Note: `scanf()` treats whitespaces and newline character as delimiter i.e., end of the input.

- An input field may be skipped by specifying * in the place of field width. For example

```
scanf("%d%*d%d",&num1, &num2);
```

- if 123 456 789 are read then it skips middle number and assigns 123 to *num1* and 789 to *num2*.

Reading Real (floating point) numbers

- `scanf()` reads real numbers using simple specification %f and %e for exponential representation.

Example:

```
scanf("%f%e",&distance1,&distance2);
```

For the above example 3.44 and 3.0e3 can be given as input numbers.

Reading Character and string

- Specification for reading character is %wc and for string %ws

Example:

```
scanf("%c",&grade);
```

- `scanf("%s",name);` // '&' is not required for reading a string

Some versions of `scanf()` supports following conversion specification.

- %[character] –allows only characters specified inside the [] as an input.
- %[^character] –character specified inside [] are not allowed as an input.

Example:

```
scanf(" %[0-9]",phoneno); //allows only uppercase letter as an input.
```

```
scanf("%[^0-9]",name); //digits are not allowed in an input.
```

Reading mixed data types

- One advantage of `scanf()` is, it allows to read 'n' number of inputs of any data type using single `scanf()` function.

Example:

`scanf("%d%c%f%s",&id,&grade,&mark,name);`

The above `scanf()` function reads 4 inputs of type **int**, **char**, **float** and **string** respectively.

Note: The order and number of the inputs should match with number of format specifier.

Table 2.2 Commonly used `scanf()` format specifier

<code>%c</code>	Read a single character
<code>%d</code>	Read a decimal integer
<code>%f</code>	Read a floating point value
<code>%e</code>	Read a floating point value
<code>%g</code>	Read a floating point value
<code>%h</code>	Read a short integer
<code>%i</code>	Read a decimal, hexadecimal or octal integer
<code>%o</code>	Read an octal integer
<code>%s</code>	Read a string
<code>%u</code>	Read an unsigned decimal integer
<code>%x</code>	Read a hexadecimal integer
<code>%[.]</code>	Read a string of words

1.1.3. Formatted Output**`printf()`**

- The **`printf`** (print formatted) standard library function is used to print the values of expressions on standard output (i.e, display) in a specifies format.
- **Syntax:**

`printf("ControlString",list_of_variables or expressions);`

Guidelines for `printf()`:

- A `printf` always contains a control string or format string in quotation marks.
- The control string may or may not be followed by some variables or expressions whose value we want printed.
- Each value to be printed needs a conversion specification like `%d` to hold its place in the control string.
- There should be a comma (,), between the control string and the list of variables.

- The symbol `\n`(called newline character) in the control string tell the machine to skip to a new line. This part of the control string affects the appearance of the output but is not displayed as a part of it.
- If there are variables or expressions to be printed, commas are used to separate them from the control string and each other.

Ex: `printf("%d%d%d",num1,num2,sum);`

Printing Integer Numbers

- The format specification for printing integer number is

`%wd`

- Where % indicated conversion specification follows,
 - `w` is the field width of the integer to be printed and
 - `d` known as data type character which indicated integer is to be printed.

Example

Format	Output
<code>printf("%d", 9876)</code>	9 8 7 6
<code>printf("%6d", 9876)</code>	9 8 7 6
<code>printf("%2d", 9876)</code>	9 8 7 6
<code>printf("%06d" 9876)</code>	9 8 7 6
<code>printf("%06d" 9876)</code>	0 0 9 8 7 6

Printing of Real Numbers

- The format specification for printing integer number is

`%w.pf` OR `%w.pe`

- Where % indicated conversion specification follows,
 - `w` is the minimum number of position that are to be used to print values before decimal point,
 - `p` indicates number of decimal places to be displayed after decimal point and `f` or `e` known as data type character which indicated floating point or exponential format is to be printed.

Example:*Format*

```
printf("%7.4f",y)
printf("%7.2f",y)
printf("%-7.2f",y)
printf("%f",y)
printf("%10.2e",y)
printf("%11.4e",-y)
printf("%-10.2e",y)
printf("%e",y)
```

Output

9	8	.	7	6	5	4			
		9	8	.	7	7			
9	8	.	7	7					
9	8	.	7	6	5	4			
		9	.	8	8	e	+	0	1
-	9	.	8	7	6	5	e	+	0
9	.	8	8	e	+	0	1		
9	.	8	7	6	5	4	0	e	+

Printing a single Character

- A single character can be displayed in a desired position using the format
%wc
- % indicated conversion specification follows,
 - w is the field width of the character (default value for w is 1) to be printed and
 - c known as data type character which indicated character is to be printed.

Example:

char grade = 'A'

printf("%c",grade); //Prints A on the console

Printing of strings

- The format specification for printing string is similar to that of real numbers. It is of the form
%w.ps
- % indicated conversion specification follows,
 - w is the field width for display and p instructs that only first p character of the string are to be displayed.
 - s is the format specifier for string data type variable to be printed.

Example:

Specification	Output																			
	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0
%s	N	E	W			D	E	L	H	I		1	1	0	0	0	1			
%20s						N	E	W			D	E	L	H	I		1	1	0	0
%20.10s												N	E	W		D	E	L	H	I
%.5s	N	E	W		D															
%-20.10s	N	E	W		D	E	L	H	I											
%5s	N	E	W		D	E	L	H	I		1	1	0	0	0	1				

Mixed data output

- It is permitted to mix data types in one `printf()` statement like `scanf()`. For example
`printf("%d%f%c%s",id,mark,grade,name);`
- The above statement prints int, float, char and string respectively.

Note: The order and number of format specifier must match with the number of variable list.

Table 2.2: Comparison of various Input/Output functions

getch()	getchar()
Unbuffered input function	Buffered input function
It does not display the entered character on the screen.	Displays the entered character on the screen.
printf()	scanf()
Used to display the output	Used to accept the input data.
Uses variables, constants and expressions as argument	Uses pointers to variables as argument.
printf()	putchar()
Formatted output function	Unformatted output function
Displays multiple line output	Displays single character output.
scanf()	getchar()
Formatted input function	Unformatted input function
Can accept multiple characters at a time	Can accept single character at a time.

Chapter 2

Conditional Branching and Loops

2.1. Introduction

- Any C expression which ends with a semicolon is called C-statements. An expression such as `x=0` or `i++` or `printf(...)` becomes a statement when it is followed by a semicolon, as in
 - `x=0;`
 - `i++;`
 - `printf(...);`
- In C a semicolon is a **statement terminator**.
- Braces { and } are used to group declarations and statements together into a compound statement, or block, so that they are syntactically equivalent to a single statement.
- C language supports decision making or branching with the following statements
 - if statement
 - switch statement
 - Conditional operator statement
 - goto statement
- These statements are also known as control statements.
- A loop (iterative) statement allows us to execute a statement or group of statements multiple times.

2.2 Decision making Statements/Branching

Conditional decision making Statements are used to make appropriate decision by transferring control flow from one place to another place, so as **to execute** a set of instructions, *if some condition is met* or, **to skip** the execution of the statements *if the condition is not met*.

The different decision making or selection statements supported by C language are:

- if statement
- if else statement
- nested if else
- cascaded if else (else if ladder)
- switch

Simple if Statement

The simple if statement is used to express single decision.

Syntax:

```
if (test expression)
{
    Statement;
}
```

Flowchart:

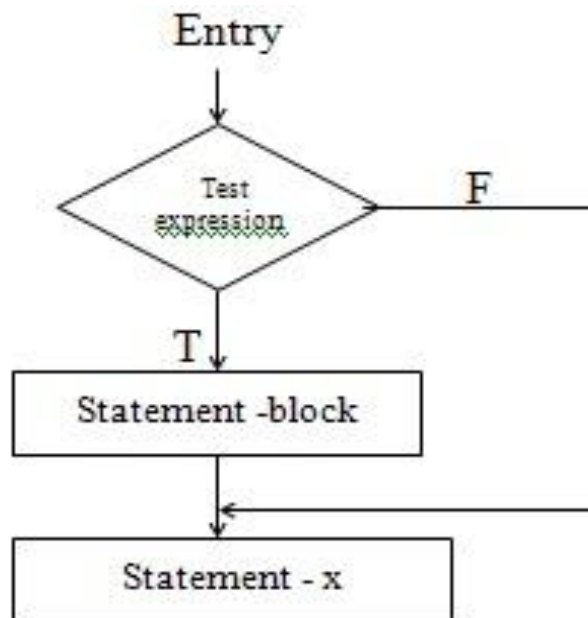


Fig: 2.1: Flowchart for simple if statement

The expression is evaluated, if it is true (that is, if expression has a non-zero value), **statement-block** is executed. If the expression evaluates to false then statement-x is executed.

The if-else statement

The if-else statement is used to express decisions. It is an extension of if statement. It is used to execute any one set of two set of statements at a time. If condition is true it executes one set of statements otherwise it executes another set of statements.

Syntax:

```
if (test expression)
{
    statement1;
}
else
{
    statement2;
}
```

Where the **else** part is optional. The expression is evaluated; if it is true (that is, if expression has a non-zero value), **statement₁** is executed. If it is false (expression is zero) and if there is an else part, **statement₂** is executed instead.

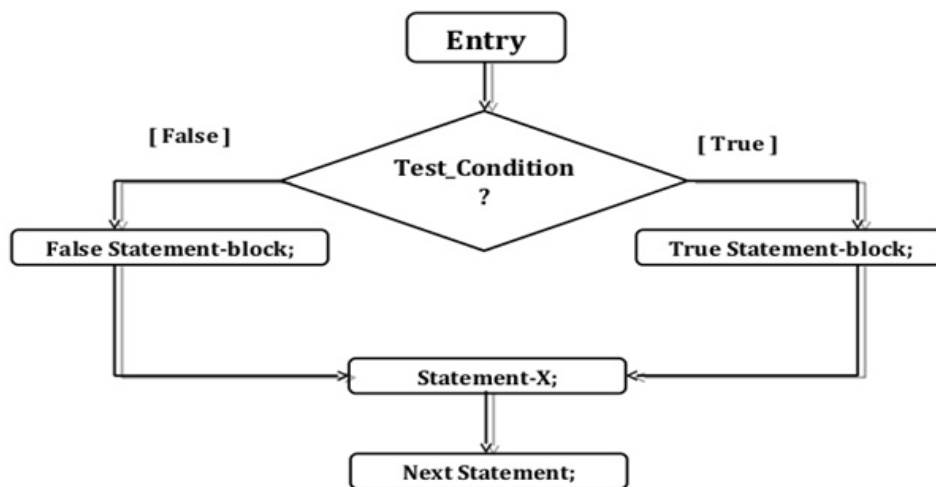
Flowchart:

Fig:2.2: Flowchart for if-else statement

Examples:**/*Program to Check whether a given number is even or odd*/**

```
#include<stdio.h>
#include<conio.h> //this header file is used only in windows based compilers. In gcc
                  //compiler this is not needed

void main()
{
    int n;
    printf("Enter an integer:\n");
    scanf("%d",&n);
    if(n%2==0)
    {
        printf("%d is an Even number\n",n);
    }
    else
    {
        printf("%d is an Odd number\n",n);
    }
    getch(); // this is used only if conio.h header file is used in the program.
}
```

Sample Output

Enter an integer

4

4 is an Even number

/*Program to find larger of two numbers*/

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

void main()
{
    int a,b,big;
    printf("Enter two numbers:\n");
    scanf("%d%d",&a,&b);
```

```
    big=a;
    if(b>big)
    {
        big=b;
    }
    printf("The biggest number is %d\n",big);
    getch();
}
```

Sample Output:

Enter two numbers:

4

2

The biggest number is 4

/*Program to determine whether a person is eligible to vote*/

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

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

```
void main()
```

```
{
```

```
    int age;
```

```
    printf("Enter the age:\n");
```

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

```
    if(age >= 18)
```

```
    {
```

```
        printf("\nYou are eligible to vote");
```

```
    }
```

```
    getch();
```

```
}
```

Sample Output:

Enter the age:

22

You are eligible to vote

/*Program to enter a character and then determine whether it is a vowel or not*/

```
#include<stdio.h>

void main()
{
    char ch;
    printf("Enter any character:\n");
    scanf("%c",&ch);
    if(ch=="a" || ch=="e" || ch=="i" || ch=="o" || ch=="u" || ch=="A" || ch=="E" ||ch=="I"
        || ch=="O" || ch=="U" )
        printf("\n%c is a Vowel",ch);
    else
        printf("\n%c is a not a Vowel",ch);
}
```

Sample Output

Enter any character:

A

A is a Vowel

/*Program to find whether a given year is a leap year or not*/

```
#include<stdio.h>

void main()
{
    int year;
    printf("Enter a year:\n");
    scanf("%d",&year);
    if((year%4==0) && ((year%100!=0) || (year%400==0)))
        printf("\nLeap Year");
    else
        printf("\n Not a leap year");
}
```

Sample Output:

Enter a year:

2004

Leap Year

Nesting of if..else/ Nested if..else statements

- When a series of decisions are involved we may have to use more than one if else statement in nested form. The nested if else statements are multidecision making statements which consist of if else control statement within another if or else control statement.
- An if-else statement inside another if statement is known as Nested if. In the given syntax if test condition-1 is true then test condition-2 is checked else statement-3 will be executed. If testcondition-2 is true then statement-1 is executed else statement-2 is executed.

Syntax:

```
if(test condition-1)
{
    if(test condition-2)
    {
        Statement – 1;
    }
    else
    {
        Statement-2;
    }
}
else
{
    Statement-3;
}
Statement –x;
```

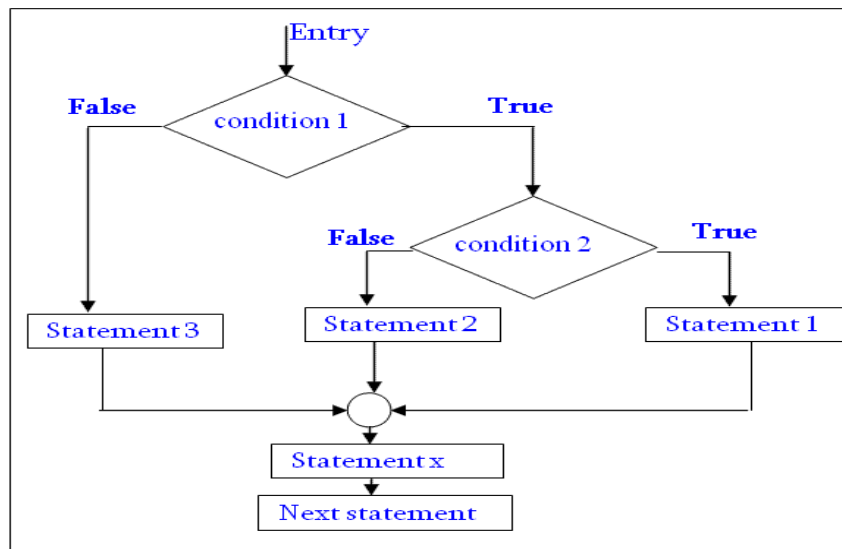
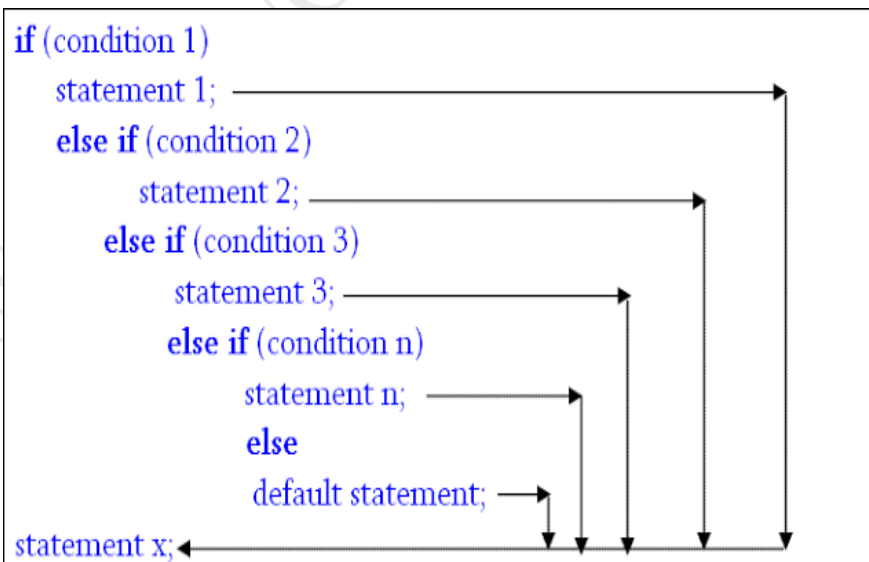
Flowchart:

Fig:2.3: Flowchart for Nested if-else statement

Cascaded if else/ else if ladder

Cascaded *if else* is a multipath decision statements which consist of chain of *if else* wherein the nesting take place only in else block. It is a special case of *nested if else* and is also known as *else if ladder*.

Syntax:

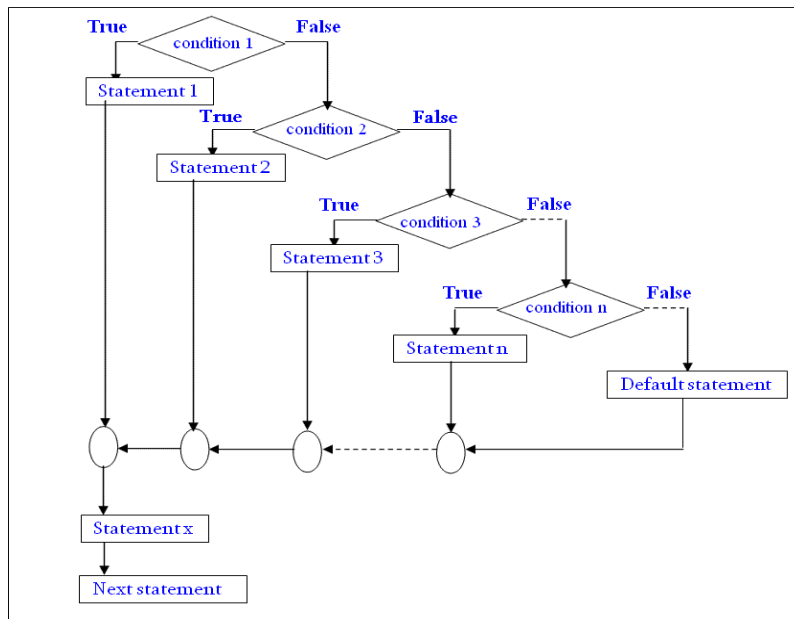
Flowchart:

Fig 2.4: Flowchart for else-if ladder

The else if ladder is used when there is need for testing multiple conditions. The expressions are evaluated in order; if any condition is true, the statement associated with it is executed, and this terminates the whole chain. The code for each statement is either a single statement, or a group in braces. The last else part handles the “none of the above” or default case where none of the other conditions is satisfied.

Examples

/*Program to test whether a number entered is positive, negative, or equal to zero*/

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

```
void main()
```

```
{
```

```
    int num;
```

```
    printf("Enter a number:\n");
```

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

```
    if(num==0)
```

```
    {
```

```
        printf("\nThe value is equal to zero");
```

```
    }
```

```
    else if(num>0)
```

```
    {
```

```
        printf("\nThe number is positive");
```

```
    }  
    else  
    {  
        printf("\nThe number is negative");  
    }  
}
```

Sample Output:

Enter a number: 3

The number is positive

/*Program to display the examination result */

```
#include<stdio.h>  
void main()  
{  
    int marks;  
    printf("\nEnter the marks obtained:");  
    scanf("%d",&marks);  
    if(marks >= 75)  
    {  
        printf("\nDISTINCTION");  
    }  
    else if(marks >= 60 && marks < 75)  
    {  
        printf("\n FIRST CLASS");  
    }  
    else if(marks >= 50 && marks < 60)  
    {  
        printf("\n SECOND CLASS");  
    }  
    else if(marks >= 40 && marks < 50)  
    {  
        printf("\nPASS");  
    }  
}
```

```
    }  
    else  
    {  
        printf("\nFAIL");  
    }  
}
```

Sample Output:

Enter the marks obtained:

55

SECOND CLASS

/*Program to input three numbers and then find largest of them using && operator */

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

```
void main()
```

```
{
```

```
    int num1,num2,num3;
```

```
    printf("Enter the first number:\n");
```

```
    scanf("%d",&num1);
```

```
    printf("Enter the second number:\n");
```

```
    scanf("%d",&num2);
```

```
    printf("Enter the third number:\n");
```

```
    scanf("%d",&num3);
```

```
    if(num1 > num2 && num1 > num3)
```

```
    {
```

```
        printf("%d is the largest number",num1);
```

```
    }
```

```
    else if(num2 > num1 && num2 > num3)
```

```
    {
```

```
        printf("%d is the largest number",num2);
```

```
    }
```

```
    else
```

```
        printf("%d is the largest number",num3);
```

```
}
```

Sample Output:

Enter the first number:

4

Enter the second number:

2

Enter the third number:

5

5 is the largest number

/*Program to input three numbers and then find largest of them using ? operator */

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

```
void main()
```

```
{
```

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

```
    printf("Enter three number:\n");
```

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

```
    big= a>b ? (a > c ? a : c) : (b > c ? b : c);
```

```
    printf("%d is the largest number",big);
```

```
}
```

Sample Output:

Enter the three numbers

4

5

6

6 is the largest number

Program to evaluate Quadratic equation

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

```
#include<math.h>
```

```
main()
```

```
{
```

```
    float a, b, c ;
```

```
float disc,root1, root2, real, imag;

printf(" Enter values for co-effiecient a, b and c:");

scanf("%f%f%f", &a,&b,&c);

if((a == 0) && (b == 0))

{

    printf(" Invalid co-effiecents\n");

}

else if(a == 0)

{

    printf(" Linear equation\n");

    root1 = -c/b;

    printf(" Root = %f\n", root1);

}

else

{

    disc = b*b -4*a*c;

    if(disc == 0)

    {

        printf(" The roots are real and equal\n");

        root1 = root2 = -b/(2*a);

        printf(" Root1 = %f\n Root2 = %f\n", root1, root2);

    }

    else if(disc > 0)

    {
```

```
printf(" The roots are real and distinct\n");

root1 = (-b+sqrt(disc))/(2*a);

root2 = (-b-sqrt(disc))/(2*a);

printf(" Root1 = %f\n Root2 = %f\n",root1, root2);

}

else

{

printf(" The roots are imaginary\n");

real = -b/(2*a);

imag = sqrt(fabs(disc))/(2*a);

printf(" Root1 = %f+i%f\n", real, imag);

printf(" Root2 = %f-i%f\n", real, imag);

}

}

}
```

Sample Ouput1:

```
Enter values for co-effiecient a, b and c:2 4 5
The roots are imaginary
Root1 = -1.000000+i1.224745
Root2 = -1.000000-i1.224745
```

Sample Output2:

```
Enter values for co-effiecient a, b and c:0 5 3
Linear equation
Root = -0.600000
```

Sample Output3:

```
Enter values for co-effiecient a, b and c:0 0 5
Invalid co-effiecents
```

The switch Statement

The **switch statement** is a **multiway decision** making that tests whether an expression matches one of a number of constant integer values and branches accordingly. It is a multiway decision making control statement used to make a selection between many alternatives. It is also known as switch case break and default statement.

Syntax:

```
switch(expression)
{
    case const-expr1: block-1
        break;
    case const-expr2: block-2
        break;
    .
    .
    case const-exprn: block-n
        break;
    default: default block
        break;
}
statement x;
```

- Each case is labelled by one or more integer-valued constants or constant expressions. If a case matches the expression value, execution starts at that case. All case expressions must be unique.
- The case labelled **default** is executed if none of the other cases are satisfied. A **default** is optional; if it is not there and if none of the cases match, no action at all takes place. Cases and the default clause can occur in any order.
- The break statement causes an immediate exit from the switch. Because cases serve just as labels, after the code for one case is done, execution falls through to the next unless a break statement is used at the end of each case.

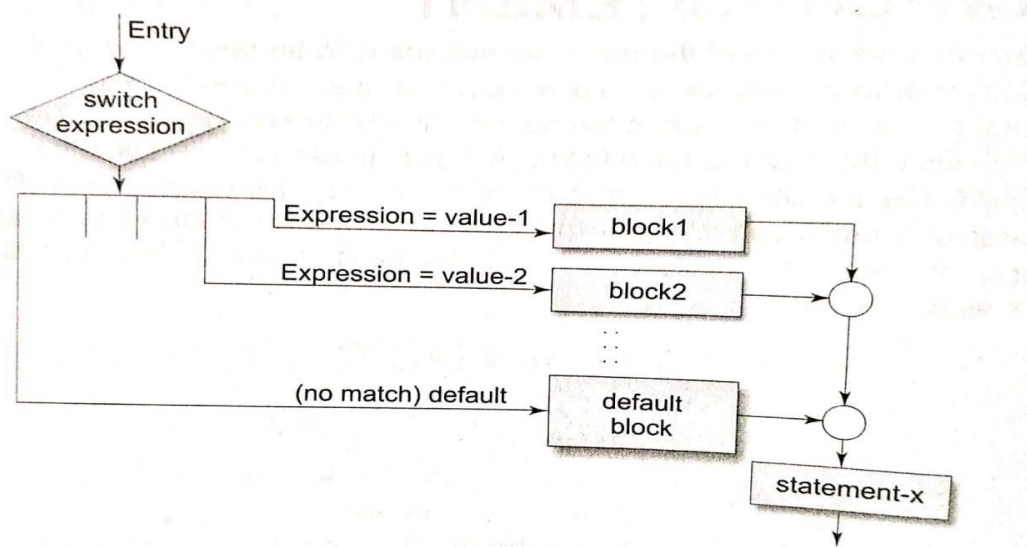
Flowchart:

Fig:2.5:Flowchart for switch statement

Examples

/*Program to determine whether an entered character is a vowel or not using switch case*/

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

```
void main()
```

```
{
```

```
    char ch;
```

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

```
    scanf("%c",&ch);
```

```
    switch(ch)
```

```
    {
```

```
        case 'A':
```

```
        case 'a':
```

```
            printf("\n %c is a VOWEL",ch);
```

```
            break;
```

```
        case 'E':
```

```
        case 'e':
```

```
            printf("\n %c is a VOWEL",ch);
```

```
            break;
```

```
        case 'I':
```

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

Sample Output:

Enter any character:

A

A is a VOWEL

/*Program to enter a number from 1 to 7 and display the corresponding day of the week using switch

case statement*/

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

```
void main()
```

```
{
```

```
    int day;
```

```
    printf("\nEnter any number from 1 to 7:");
```

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

```
    switch(day)
```

```
    {
```

```
        case 1: printf("\n SUNDAY");
```

```
            break;
```

```
        case 2: printf("\n MONDAY");
```

```
            break;
```

```
        case 3: printf("\n TUESDAY");
                break;
        case 4: printf("\n WEDNESDAY");
                break;
        case 5: printf("\n THURSDAY");
                break;
        case 6: printf("\n FRIDAY");
                break;
        case 7: printf("\n SATURDAY");
                break;
    }
}
```

Sample Output:

Enter any number from 1 to 7:

1

SUNDAY

/*Program that accepts a number from 1 to 10. Print whether the number is even or odd using a switch case construct*/

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

```
void main()
```

```
{
```

```
    int num,rem;
```

```
    printf("\n Enter any number (1 to 10):\n");
```

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

```
    rem = num%2;
```

```
    switch(rem)
```

```
    {
```

```
        case 0:
```

```
            printf("\n EVEN");
```

```
            break;
```

```
        case 1:
```

```

        printf("\n ODD");
        break;
    }
}

```

Sample Output:

Enter any number (1 to 10):

2

EVEN

Character Functions

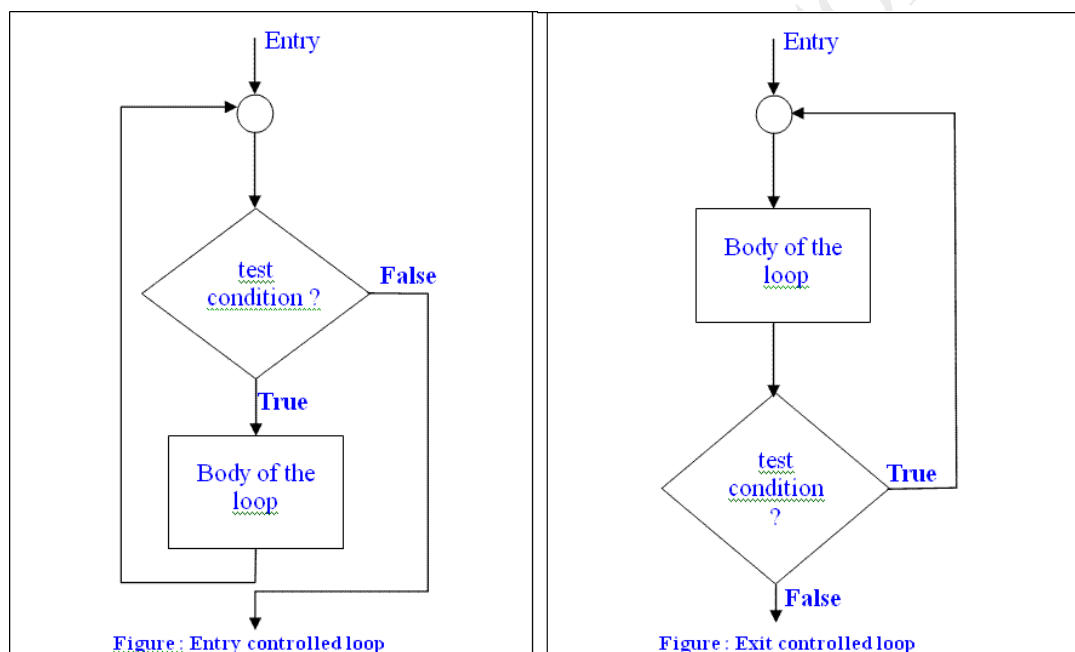
In C-programming built-in character function reside in ctype.h header file. In order to invoke these built-in character functions, ctype.h header file must be included in the program.

Table 2.1: built-in character functions

Function	Description	Example
isspace()	This function is used to check whether a given character is white space or not.	isspace(' ') = True isspace('a') = False
isascii()	This function is used to test if the parameter is between 0 to 127.	isascii(97) = True isascii(200) = False
tolower()	If the character is an uppercase character(A to Z), then it is converted to lower case.	Tolower('A') = A
toupper()	If the character is an lower case character(a to z), then it is converted to upper case(A to Z).	toupper('a') = A
toascii()	This function is used to convert the character to its equivalent ascii value.	toascii('A') = 65 toascii('a') = 97
Function	Description	Example
isalnum()	This function is used to check whether a character is an alphanumeric(i.e A to Z or a to z) or (0 to 9).	isalnum('a') = True (True means non zero value). isalnum('&') = False
isalpha()	This function is used to test whether the given character is alphabetic character or not.	isalpha('a') = True(1) isalpha('5') = False(0)
isdigit()	This function is used to check whether or not it is a digit(0 to 9).	isdigit('9') = True isdigit('a') = False
islower()	This function is used to check whether the given character is lower case or not(a to z).	islower('a') = True islower('A') = False
isupper()	This function is used to check whether the given character is uppercase letter or not(A to Z).	isupper('a') = False isupper('A') = True

2.3 Looping

- A loop (iterative) statement allows us to execute a statement or group of statements multiple times.
- Looping construct may be either entry controlled or exit controlled.
- In the **entry controlled loop** the control conditions are tested before the start of the loop execution. If the conditions are not satisfied, then the body of the loop will not be executed. It is also known as *pretest loop* or *top test loop*.
- In the exit controlled loop the test is performed at the end of the body of the loop and therefore the body is executed unconditionally for the first time. It is also known as *post test loop*. Here the body of the loop will get executed at least once before testing.
- Following diagram shows the flowchart for entry controlled and exit controlled.



The looping includes the following four steps:

- Initialization of a condition variable
- Testing for a specified value of the condition variable for execution of the loop.
- Execution of the statements in the loop
- Updating (Incrementing or Decrementing) the condition variable.

Types of Loops Supported in C

C programming language provides the following types of loops to handle looping requirements.

- **while loop**
- **do-while loop**
- **for loop**

while loop

It is an entry controlled loop statement. In case of while loop the initialization, testing the condition and incrementation or updation is done in separate statements. First initialization of the loop counter is performed. Next the condition is checked. If the condition evaluates to be true then the control enters the body of the loop and executes the statements in the body.

Syntax:

while(test condition)

{

body of the loop

}

- In the above syntax for the *while* loop, the expression is evaluated. If it is non-zero, body of the loop is executed and expression is re-evaluated. This cycle continues until expression becomes zero, at which point execution resumes after *while* loop.

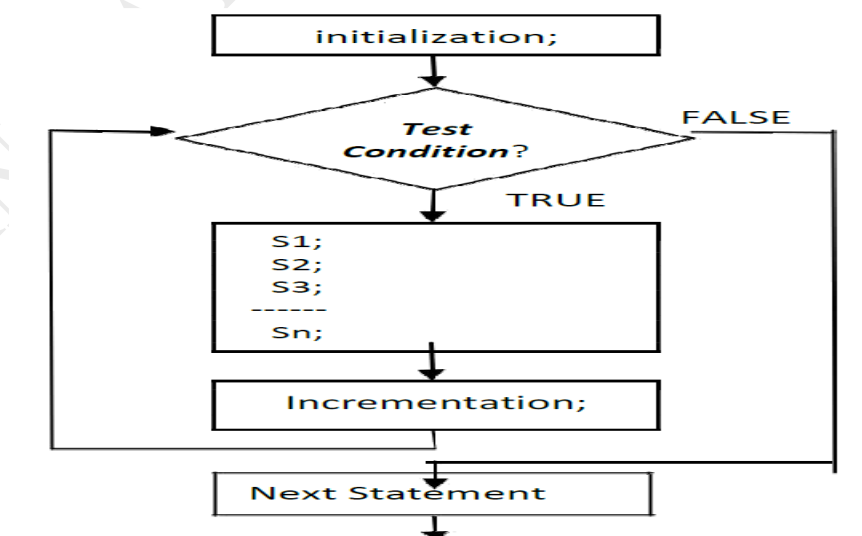


Fig:2.5: Flowchart for while loop

Example:**/*Program to calculate the sum of the first 10 numbers */**

```
#include<stdio.h>

void main()
{
    int i=0,sum=0;
    while(i<=10)
    {
        sum=sum+i;
        i=i+1;
    }
    printf("\nSum - %d",sum);
}
```

Output:

55

/*Program to calculate the sum of the first 10 numbers */

```
#include<stdio.h>

void main()
{
    int n,m,sum=0;
    printf("\nEnter the value of m: ");
    scanf("%d",&m);
    printf("\nEnter the value of n: ");
    scanf("%d",&n);
    while(m<=n)
    {
        sum=sum+m;
        m=m+1;
    }
    printf("\nSum = %d",sum);
}
```

/*Program to print the reverse of a number */

```
#include<stdio.h>

void main()
{
    int num,temp;
    printf("Enter the number:\n");
    scanf("%d",&num);
    printf("The reversed number is:\n");
    while(num!=0)
    {
        temp=num%10;
        printf("%d",temp);
        num=num/10;
    }
}
```

Sample Output:

Enter the number:

567

The reversed number is:

765

/*Program to calculate GCD of two numbers */

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

void main()
{
    int num1,num2,temp;
    int dividend,divisor,remainder;
    printf("\nEnter the first number:");
    scanf("%d",&num1);
    printf("\nEnter the second number:");
    scanf("%d",&num2);
```



```
if(num1>num2)
{
    dividend=num1;
    divisor=num2;
}
else
{
    dividend=num2;
    divisor=num1;
}
while(divisor)
{
    remainder=dividend%divisor;
    dividend=divisor;
    divisor=remainder;
}
printf("\n GCD of %d and %d is = %d",num1,num2,dividend);
getch();
}
```

Sample Output

Enter the first number:

2

Enter the second number:

4

GCD of 2 and 4 is = 2

do-while loop

The do-while loop is an exit controlled loop.

Syntax:

```
do
{
    body of the loop
} while(test condition);
```

- In the do-while loop, the body of the loop is executed, then the expression is evaluated. If it is true, body of the loop is evaluated again, and so on. When the expression becomes false, the loop terminates.

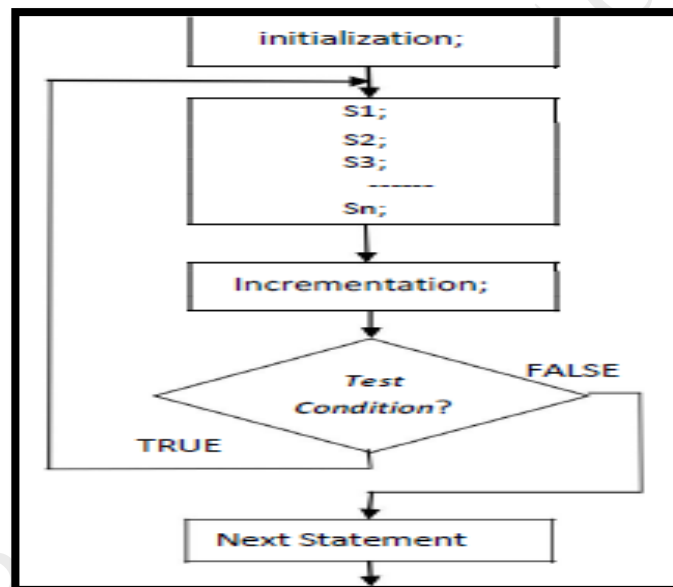


Fig: 2.6: Flowchart for do-while loop

Examples:

/*Program to print n numbers using do while loop*/

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

```
void main()
```

```
{
```

```
    int n,i;
```

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

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

```
i=1;
do
{
    print("%d",i);
    i++;
}while(i<=n);
}
```

Sample Output:

Enter the value of n:

4

1 2 3 4

/*Program to print the average of first n natural numbers using do-while loop*/

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

```
void main()
```

```
{
    int n,i,sum=0;
    float avg=0.0;
    printf("Enter the value of n:\n");
    scanf("%d",&n);
    i=1;
    do
    {
        sum=sum+i;
        i++;
    }while(i<=n);
    avg=sum/n;
    printf("\nThe sum of first n natural numbers = %d\n",sum);
    printf("\nThe average of first n natural numbers = %f\n",avg);
}
```

Sample output:

Enter the value of n

4

The sum of first n natural numbers = 10

The average of first n natural numbers = 2.5

for loop

- It is an entry control loop

Syntax:

```
for ( init; condition; increment )  
{  
    statements;  
}
```

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 the 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.

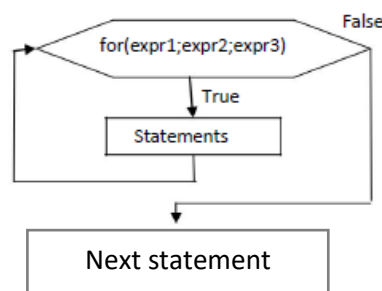


Fig: 2.7: Flowchart of for loop

Examples:**/*Program to print numbers from m to n using for loops*/**

#include<stdio.h>

void main()

```
{
    int m,n,i;
    printf("Enter the value of m:\n");
    scanf("%d",&m);
    printf("Enter the value of n:\n");
    scanf("%d",&n);
    for(i=m;i<=n;i++)
    {
        printf("\t%d",i);
    }
}
```

Sample Output:

Enter the value of m

2

Enter the value of n

4

2 3 4

/*Program to print all the numbers from m to n, thereby classifying them as even or odd*/

#include<stdio.h>

void main()

```
{
    int m,n,i;
    printf("Enter the value of m:\n");
    scanf("%d",&m);
    printf("Enter the value of n:\n");
    scanf("%d",&n);
    for(i=m;i<=n;i++)
```

```
{
    if(i%2==0)
        printf("%d is even\n",i);
    else
        printf("%d is odd\n",i);
}
```

Sample Output:

Enter the value of m

2

Enter the value of n

4

2 is even

3 is odd

4 is even

/*Program to print the average of first n natural numbers using for loop*/

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

```
void main()
```

```
{
```

```
    int n,i,sum=0;
```

```
    float avg=0.0;
```

```
    printf("Enter the value of m:\n");
```

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

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

```
    {
```

```
        sum=sum+i;
```

```
    }
```

```
    avg=sum/n;
```

```
    printf("\nThe sum of first n natural numbers = %d\n",sum);
```

```
    printf("\nThe average of first n natural numbers = %f\n",avg);
```

```
}
```

Sample Output:

Enter the value of n

4

The sum of first n natural numbers = 10

The average of first n natural numbers = 2.5

Nested Loops

- C allows its users to have nested loops, i.e, loops that can be placed inside other loops. Although this feature will work with any loop such as *while*, *do-while*, and *for*, but it is most commonly used with the *for* loop, because this is easiest to control.
- In C, loops can be nested to any desired level. However, the loops should be properly indented in order to enable the reader to easily determine which statements are contained within each for statement.

EXAMPLES**Pascal's Triangle**

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
```

//Program to print Pascal's Triangle for a given no. of rows

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

```
int main()
```

```
{
```

```
    int rows, coef = 1, space, i, j;
```

```
    printf("Enter number of rows: ");
```

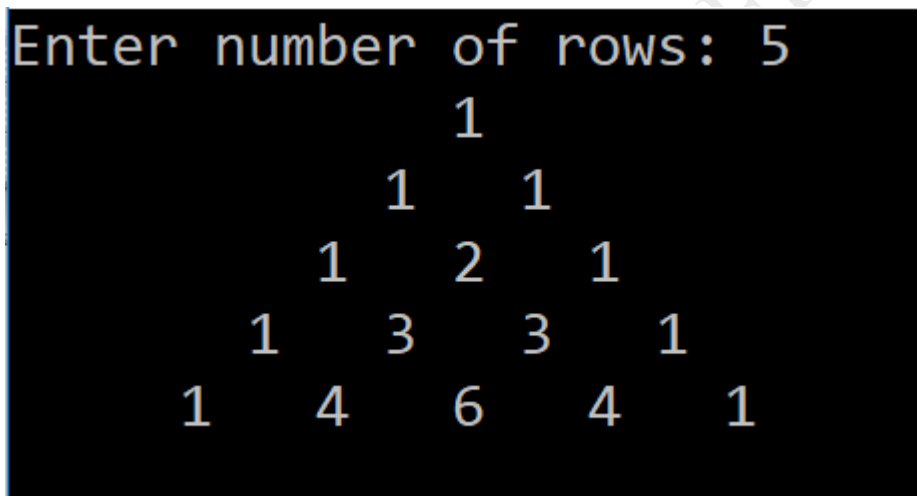
```
    scanf("%d",&rows);
```

```
    for(i=0; i<rows; i++)
```

```
    {
```

```
for(space=1; space <= rows-i; space++)
    printf(" ");
for(j=0; j <= i; j++)
{
    if (j==0 || i==0)
        coef = 1;
    else
        coef = coef*(i-j+1)/j;
    printf("%4d", coef);
}
printf("\n");
}
return 0;
}
```

Output:



```
Enter number of rows: 5
      1
     1 1
    1 2 1
   1 3 3 1
  1 4 6 4 1
```

// Program to calculate Binomial Coefficient

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

```
#define MAX 10
```

```
void main()
```

```
{
```

```
    int m, x, binom;
```



```
printf(" m x");
for (m = 0; m <= 10 ; ++m)
printf("%4d", m);
printf("\n-----\n");
m = 0;
do
{
    printf("%2d ", m);
    x = 0; binom = 1;
    while (x <= m)
    {
        if(m == 0 || x == 0)
            printf("%4d", binom);
        else
        {
            binom = binom * (m - x + 1)/x;
            printf("%4d", binom);
        }
        x = x + 1;
    }
    printf("\n");
    m = m + 1;
}
while (m <= MAX);
printf("-----\n");
}
```

Output:

m x	0	1	2	3	4	5	6	7	8	9	10
0	1										
1	1	1									
2	1	2	1								
3	1	3	3	1							
4	1	4	6	4	1						
5	1	5	10	10	5	1					
6	1	6	15	20	15	6	1				
7	1	7	21	35	35	21	7	1			
8	1	8	28	56	70	56	28	8	1		
9	1	9	36	84	126	126	84	36	9	1	
10	1	10	45	120	210	252	210	120	45	10	1

Write a Program to print the following pattern

```
*
**
***
****
*****
```

//program for generation of above pattern

```
#include<stdio.h>
void main( )
{
    int i,j;
    for(i=1;i<=5;i++)
    {
        printf("\n");
        for(j=1;j<=i;j++)
        {
            printf("*");
```

```
        }  
    }  
}
```

Write a Program to print the pattern

```
1  
12  
123  
1234  
12345
```

//program for generation of given pattern

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

```
void main()
```

```
{  
    int i,j;  
    for(i=1;i<=5;i++)  
    {  
        printf("\n");  
        for(j=1;j<=i;j++)  
        {  
            printf("%d",j);  
        }  
    }  
}
```

Write a Program to print the pattern

```
1  
22  
333  
4444  
55555
```

//program for generation of given pattern

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

```
void main()
```

```
{
```

```
    int i,j;
```

```
    for(i=1;i<=5;i++)
```

```
    {
```

```
        printf("\n");
```

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

```
        {
```

```
            printf("%d",i);
```

```
        }
```

```
    }
```

```
}
```

Write a Program to print the pattern

1

12

123

1234

12345

//program for generation of given pattern

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

```
#define N 5
```

```
void main()
```

```
{
```

```
    int i,j,k;
```

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

```
    {
```

```
        for(k=N;k>=i;k--)
```

```
        printf(" ");
```

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

```

        {
            printf("%d",j);
        }
        printf("\n");
    }

```

```

}

```

Write a Program to print the pattern

0

12

345

6789

//program for generation of given pattern

```

#include<stdio.h>

```

```

void main()

```

```

{

```

```

    int i,j,count=0;

```

```

    for(i=1;i<=5;i++)

```

```

    {

```

```

        printf("\n");

```

```

        for(j=1;j<=i;j++)

```

```

        {

```

```

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

```

```

        }

```

```

    }

```

```

}

```

Write a Program to find sum of the series $1 + 1/2 + 1/3 + \dots + 1/n$

```

#include<stdio.h>

```

```

void main()

```

```

{

```

```

    int n;

```

```

    float sum=0.0,a,i;

```

```
printf("\n Enter the value of n:");
scanf("%d",&n);
for(i=1.0;i<=n;i++)
{
    a=1/i;
    sum=sum+a;
}
printf("\n The sum of series 1/1 + 1/2 + ..... + 1/%d = %f",n,sum);
}
```

Write a program to sum of the series $1/2 + 2/3 + 3/4 + \dots + n/n+1$

```
#include<stdio.h>
void main()
{
    int n;
    float sum=0.0,a,i;
    printf("\n Enter the value of n:");
    scanf("%d",&n);
    for(i=1.0;i<=n;i++)
    {
        a=i/(i+1);
        sum=sum+a;
    }
    printf("\n The sum of series 1/2 + 2/3 + ..... = %f",sum);
}
```

2.4 Unconditional Branch Statements (Jump statements)

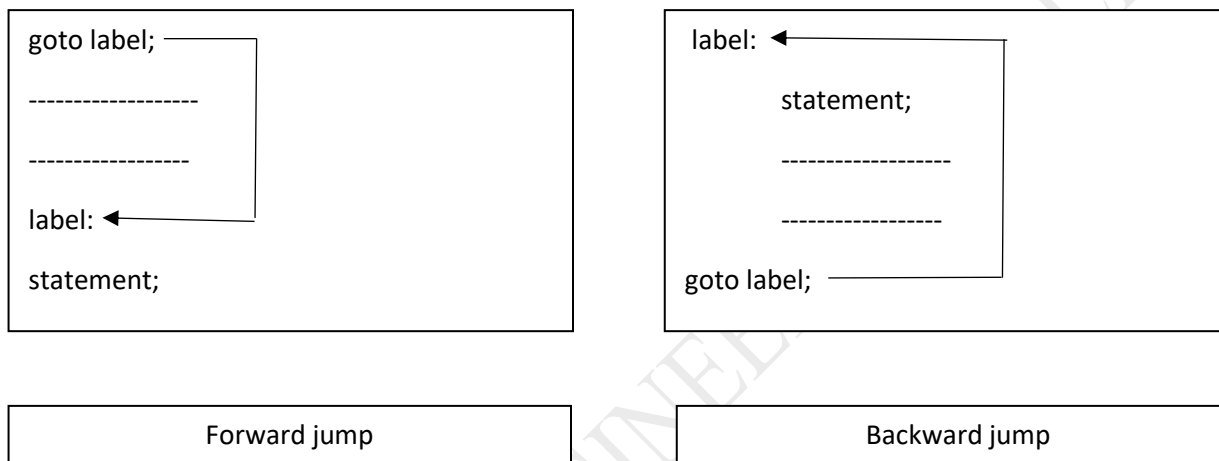
- Unconditional statements transfer control or flow of execution unconditionally from one block to another block of statements. These are also called as jump statements.
 - break statement.
 - Continue statement.
 - goto statement.

- return statement.

goto statement

- The goto statement is a jump statement which is sometimes also referred to as unconditional jump statement. The goto statement can be used to jump from anywhere to anywhere within a function. The **goto** statement is used to transfer control to a specified **label**.
- The 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.

Syntax:



- 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.
- The goto statement is often combined with the if statement to cause a conditional transfer of control.

Example:

```
#include<stdio.h>

main()
{
    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);
}
```

Sample output:

Enter the number. Enter 999 to end:

-2

Enter the number. Enter 999 to end:

3

Enter the number. Enter 999 to end:

999

Sum of the numbers entered by the user is = 3

break Statement

The break statement is used to terminate execution of the nearest en-closing loop in which it appears.

Example:

```
#include<stdio.h>
void main()
{
    int i=0;
    while(i<=10)
    {
        if(i == 5)
            break;
        printf("\n%d",i);
        i=i+1;
    }
}
```



```
}
```

- In the above example the while terminates when the value of variable i becomes 5.
- In switch statement if the break statement is missing then every case from the matched case till the end of the switch, including the default , is executed.

continue Statement

The continue statement is used to skip iterations in a loop, based on certain conditions.

Example:

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

```
void main()
```

```
{
```

```
    int i;
```

```
    for(i=0;i<=10;i++)
```

```
    {
```

```
        if(i==5)
```

```
            continue;
```

```
        printf("\t%d",i);
```

```
    }
```

```
}
```

The output of the above program will be:

0 1 2 3 4 6 7 8 9 10

The loop iteration will be skipped when i==5. Hence value 5 will not be printed.

return Statement

- return statement terminates the execution of function and returns control to the calling function. It can also return a value to the calling function.

Syntax: return expression;

Web Resources

1. Formatted Output: <https://www.w3resource.com/c-programming/c-printf-statement.php>
2. Input/output operations: <https://www.w3resource.com/c-programming-exercises/input-output/index.php>
3. Decision making statements: <https://www.w3resource.com/c-programming-exercises/conditional-statement/index.php>
4. for loop: <https://www.w3resource.com/c-programming/c-for-loop.php>
5. while loop: <https://www.w3resource.com/c-programming/c-while-loop.php>
6. do-while loop: <https://www.w3resource.com/c-programming/c-do-while-loop.php>
7. for loop: <https://www.w3resource.com/c-programming-exercises/for-loop/index.php>

Video Resources

1. Managing Input/Output:

<https://www.youtube.com/watch?v=z8Yp0N57rw&list=PLVDfFatHsysSm84-O26R9WbZVKmD6zakV&index=14>

2. Decision Making and branching:

<https://www.youtube.com/watch?v=pXRaggdNKeQ&list=PLVDfFatHsysSm84-O26R9WbZVKmD6zakV&index=15>

3. Looping:

<https://www.youtube.com/watch?v=NeKFovNjLy8&list=PLVDfFatHsysSm84-O26R9WbZVKmD6zakV&index=16>

4. Character Functions

<https://www.youtube.com/watch?v=cqngjoNZRb4&list=PLVDfFatHsysSm84-O26R9WbZVKmD6zakV&index=17>

5. Example Programs:

<https://www.youtube.com/watch?v=2q64OQMN4bQ&list=PLVDfFatHsysSm84-O26R9WbZVKmD6zakV&index=18>

6. Binomial coefficient

<https://www.youtube.com/watch?v=nVO89GxsVjE&list=PLVDfFatHsysSm84-O26R9WbZVKmD6zakV&index=19>

7. Arithmetic Expressions

<https://www.youtube.com/watch?v=XVuFWzcXPkM&list=PLVDfFatHsysSm84-O26R9WbZVKmD6zakV&index=20>

8. Pascal's triangle

<https://www.youtube.com/watch?v=jnOBUNO7DGY&list=PLVDfFatHsysSm84-O26R9WbZVKmD6zakV&index=21>

9. Quadratic Equation

<https://www.youtube.com/watch?v=uCHdsYAKbq0&list=PLVDfFatHsysSm84-O26R9WbZVKmD6zakV&index=22>

10. <https://www.youtube.com/watch?v=rk2fK2IliiQ>

11. https://www.youtube.com/watch?v=FwpP_MsZWnU

12. <https://www.youtube.com/watch?v=TpcdSWsVhG8>

13. <https://www.youtube.com/watch?v=huMTljgjPrg>

Question Bank

1. Explain the formatted I/O function of C language with syntax.
2. Write the syntax of different looping control constructs and explain their working.
3. Distinguish between the following: i) goto and if ii) break and continue
4. Write the syntax of nested if... else statement and explain its working.
5. Explain the else – if ladder with syntax and example.
6. Differentiate between do...while loop and while loop, with the help of syntax.
7. Explain the switch statement with syntax and example.
8. Write a C program to plot Pascal's triangle.

University Questions

1. Define and write the classification of input and output statements in C. Write a C-program that prints the following output: (6-marks)

Screen →

```
" I am
an"      'Engineering
Student'
```

2. Define branching statements. Explain them with syntax and suitable example. (10 marks)

3. Evaluate:


```
i=1
L:if(i>2)
{
printf("Saturday");
i=i+1;
goto L;
}
printf("Sunday");
```

Explain Your Result briefly. (04 marks)

4. State the drawback of ladder if-else. Explain how do you resolve with suitable example. (08 marks)
5. Write a C program to get the triangle of numbers as a result: (06 marks)

```
1
1 2
1 2 3
1 2 3 4
```

6. Write a C-program to check whether given number is prime or not. (06 marks)
7. With examples how would you describe the formatted input and formatted output statements in C language. (08 marks)
8. How would you explain if-else statement in C-language? Give the relevant example. (06 marks)

9. Write a c- program to display grade based on the marks as follows:

Marks	Grades
0 to 39	F
40 to 49	E
50 to 59	D
60 to 69	C
70 to 79	B
80 to 89	A
90 to 100	O

10. How would you explain switch statement with example? (08 marks)
11. How while loop differs from do..while loop? (06 marks)
12. Write a program to check whether a given integer is palindrome or not. (06 marks)