Chapter 5
Decision Making and Branching Statement

This Chapter Includes:

Statement:

- A statement (also called an instruction) is the smallest element of a programming language. A statement or instruction is used to inform the computer to perform an action when a program is executed.
- A statement can set a value of a variable, can alter the value of a variable, it can accept the input, it can manipulate the data and display the data.
- The expressions such as i++ or sum = 0 or scanf ("%d", &n) become statements when the are terminated by a semicolon as shown below :

```
i++; /* statement 1 */
sum = 0; /* statement 2 */
 scanf("%d",& n); /* statement 3 */
a=10;          /* statement 4 */
```

Note: Each statement in C must be terminated by a semicolon (denoted by ;)

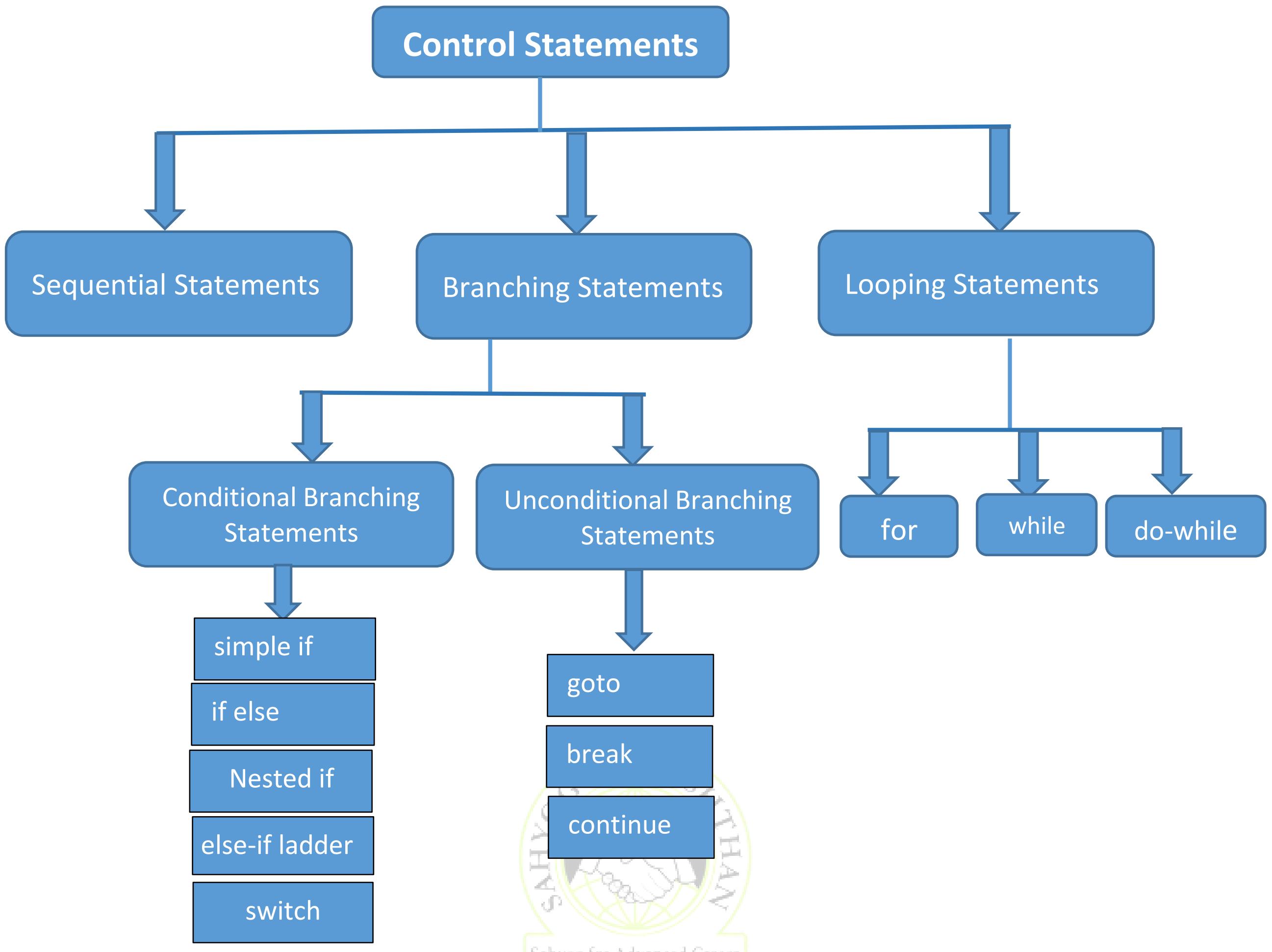
Compound Statement:

- The set of statements enclosed within a pair of curly braces such as '{' and '}' is considered as a compound statement.
  For example, the following compound statement computes and displays the addition of 2 numbers.

```
{
        a = 10; ' /* statement 1 */
        b = 5 ; /* statement 2 */
        c=a+b; /* statement 3 */
         printf ("%d\n", c); /* statement 4 */
    }
```

Control Statements:

- The order in which the statements are executed is called control flow. The statements that are used to control the flow of execution of the program are called control statements.
- Based on the order in which the statements are executed, the various Control statements are classified as shown below:



Sequential Statements:

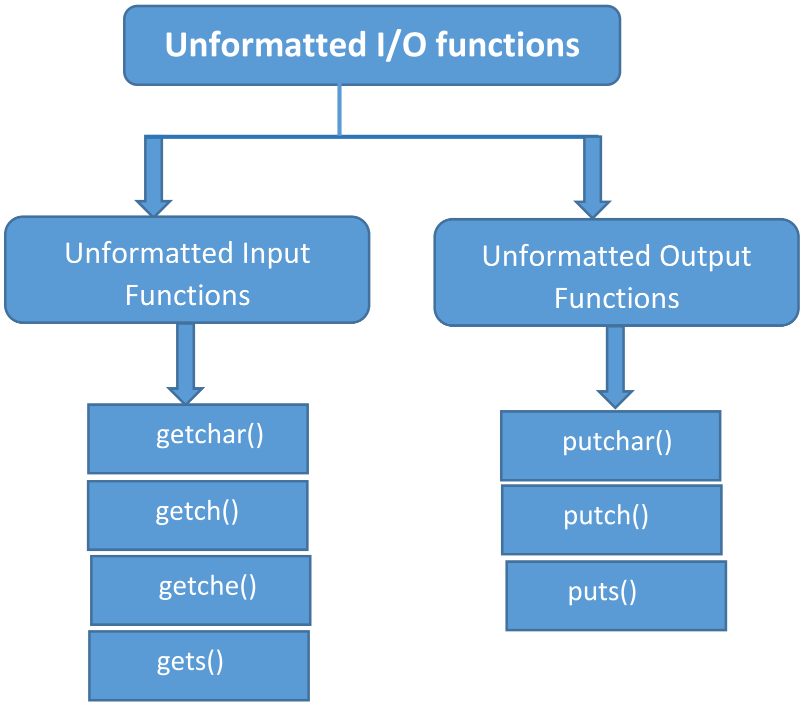
- The programmer writes a sequence of statements to do a specific activity. All these statements in a program are executed in the order in which they appear in the program.
- These programming statements that are executed sequentially, that is one after the other, are called sequential control statements. Here, no separate statements are required to make these statements executed in sequence.

```
printf("Hello"); //Statement 1
ch='A';          //Statement 2
printf("%c",ch); //Statement 3
a=10;            //Statement 4
```

Here, statement-1 is executed first ,followed by statement-2 followed by statement-3 and so on. In short, we say that execution of a program is normally sequential (one after the other).

Unformatted I/O functions:

- It is very easy to use unformatted input output function, but in unformatted we can only read and display characters not integer or floating values.
- unformatted input function are getch(),getchar(),getche(),gets() and unformatted output function is putch(),putchar(),puts().


Unformatted I/O functions
Unformatted Input Functions
Unformatted Output Functions
getchar()
getch()
getche()
gets()
putchar()
putch()
puts()

a. getchar():

- getchar() is a Unformatted input function which is used to input a character from keyboard at run time and only 1 character can be entered at a time.
- In getchar() we **can not** use any format specifiers.
- getchar() is only used for taking characters.

Syntax:
Variable=getchar();

Example:
char ch;
ch=getchar();

b. putchar():

- putchar() is a Unformatted output function which is used to display a character on output screen.

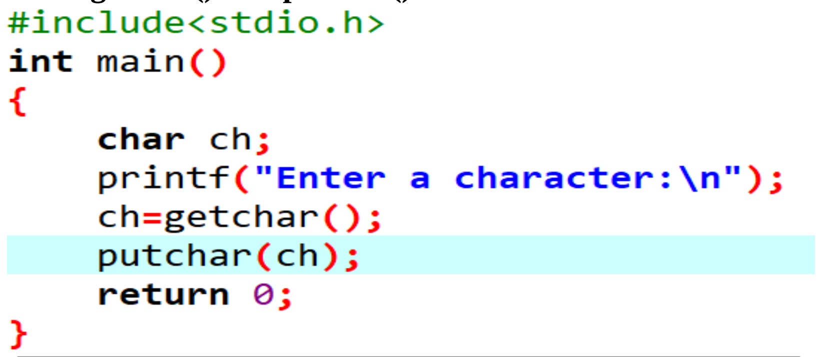
Syntax:
putchar(variable/value);

Example 1:
char ch='A';
putchar(ch);

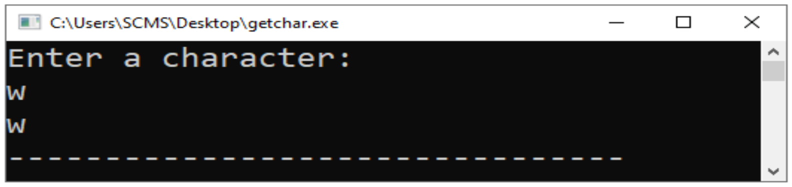
Example 2:
char ch;
ch=getchar();
putchar(ch);

Example 3:
putchar('a');

Program for getchar() and putchar():



```
#include<stdio.h>
int main()
{
	char ch;
	printf("Enter a character:\n");
	ch=getchar();
	putchar(ch);
	return 0;
}
```



```
C:\Users\SCMS\Desktop\getchar.exe
Enter a character:
w
w
--------------------------------
```

c. getch(),getche():

- The functions getch() and getche() are used to read a character from the keyboard similar to getchar().
- While using getchar() a return key pressed terminates the reading of a character. In case of getch() and getche() it is not required. Just a character entered itself terminates reading.
- When getch() is used the character entered is not displayed or echoed on the screen
- whereas when getche() is used the character entered is echoed or displayed on the screen.

The syntax of both the functions is as follows :

ch = getch(); /* typed character will not be displayed on screen */
ch = getche(); /* typed character will be displayed on the screen */

- These functions read a character from the keyboard and copy it into memory area which is identified by the variable ch.
- No arguments are required for these functions. These functions do not wait for the user to press "Enter or Return" key.

Note: The typed character will not be displayed on the screen if we use getch() function. The typed character will be echoed (displayed) on the screen if we use getche() function.

d. putch():

- putch() is a Unformatted output function
- This function displays a character stored in the memory using a variable on the monitor. The variable should be passed as parameter to the function.

Syntax:
putch(character/variable);

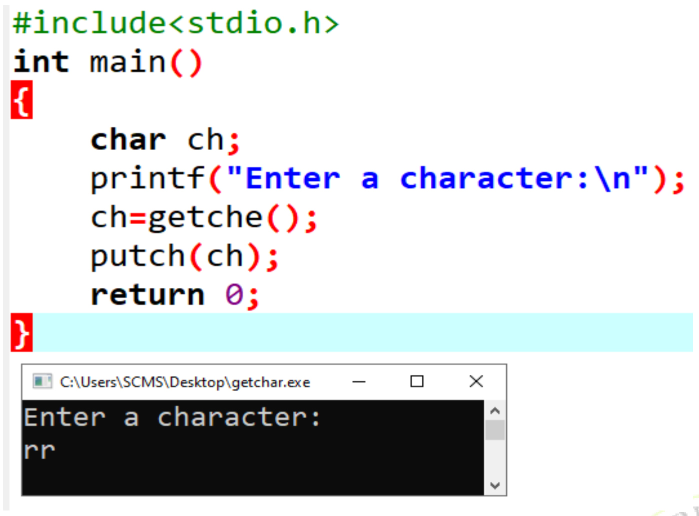
Example 1:
char ch='A';
putch(ch);

Example 2:

char ch;
ch=getche();
putchar(ch);

Example 3:
putch('d');

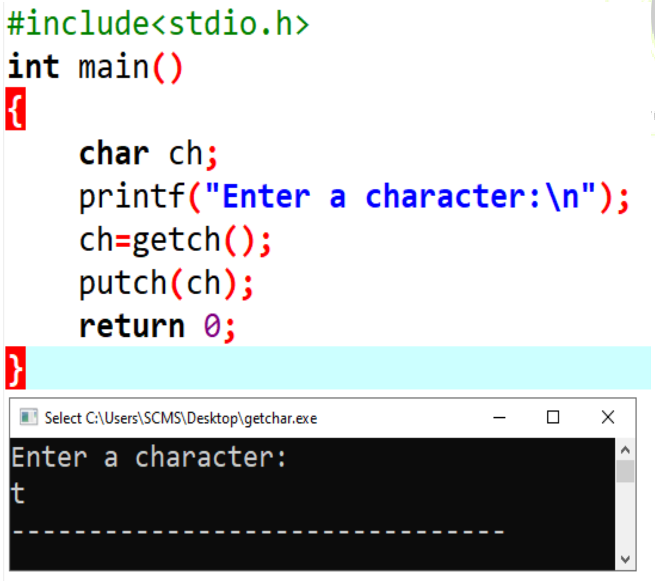
Program for getche() and putch().



```
#include<stdio.h>
int main()
{
    char ch;
    printf("Enter a character:\n");
    ch=getche();
    putch(ch);
    return 0;
}
```

```
Enter a character:
rr
```

Program for getch() and putch().



```
#include<stdio.h>
int main()
{
    char ch;
    printf("Enter a character:\n");
    ch=getch();
    putch(ch);
    return 0;
}
```

```
Enter a character:
t
--------------------------------
```

e. **gets() :**
gets() is used to read a set of characters(string) from the keyboard .

Example:
char str[10];
gets(str); /* reads a set of characters into memory area str */

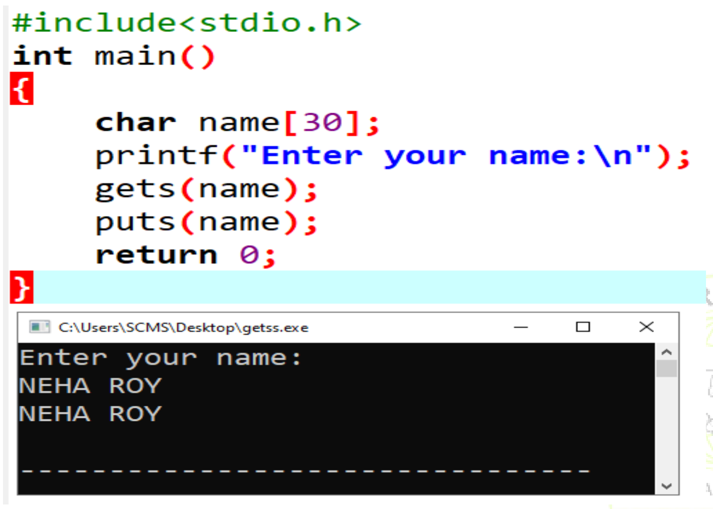
f. **puts():**
puts() function is used to display string on output screen

Example 1:
char s1[10]=”sahyog”;
puts(s1);

Example 2:
char s1[10];
gets(s1);
puts(s1);

Example 3:
puts(“good morning”);

Program for gets() and puts().



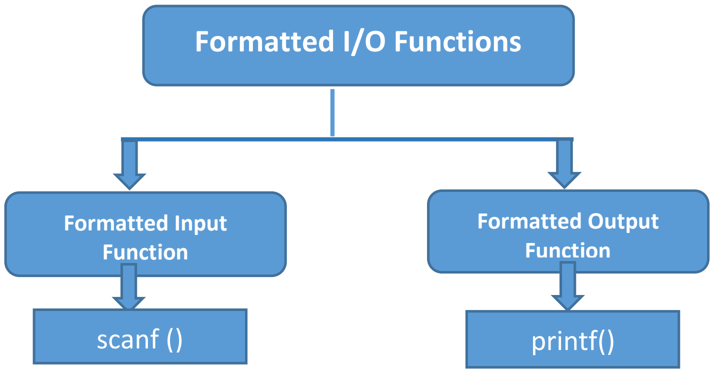
```
#include<stdio.h>
int main()
{
	char name[30];
	printf("Enter your name:\n");
	gets(name);
	puts(name);
	return 0;
}
```

```
C:\Users\SCMS\Desktop\getss.exe
Enter your name:
NEHA ROY
NEHA ROY

--------------------------------
```

Formatted Input/ Output Functions:

- It is very easy to use unformatted input output function, but in unformatted we can only read and display characters not integer or floating values.
- But, using formatted I/O functions, we can read integers, floating point numbers and characters.
- Formatted input function is scanf() and formatted output function is printf().



scanf():

- scanf() is formatted input function which is defined in **stdio.h** header file.
- This function scanf( ) is used to read the data of any type such as int, char, float, double and even series of characters (called string) from the keyboard.
- The function returns an integer value. This integer value is the number of items that have been read successfully using the keyboard. But, we rarely use this return value in our programs.

Syntax for scanf():
scanf("format specifiers", &variable_name);

Where,

- Format specifiers can be %d,%c,%f etc.
- &variable_name represent the variables. Here, the symbol '& ' refers to the address of a variable. In the memory locations identified by these addresses, the respective values are stored.

Example:

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

The following table shows the way the user has to enter the values from the keyboard:

Input statement	What has to entered	How to enter
scanf ("%d%d%d",& a,& b,& c);	3 integer values 1 2 3	1 2 3 Here, a = 1, b = 2 and c = 3
scanf ("%d,%d,%d",& a,& b,& c);	3 integer values 1 2 3	1,2,3 values Here, a = 1, b = 2 and c = 3
scanf ("'%f",& a);	1 floating point value	35.14 Here, a = 35.14
scanf ("%c",& a);	1 character value	n Here, a = n
scanf ("%d/%d/%d",& a,& b,& c);	3 integer values separated by / symbol	8/8/2008 Here, a = 8, b = 8, c = 2008
scanf ("%d-%d-%d",& a,& b,& c);	3 integer values separated by - symbol	8-8-2008 Here, a = 8, b = 8, c = 2008

NOTE : Do not use any back slash characters such as \t,\n etc., in scanf().

For example : consider the statement :
scanf("%d\t", & a);
In the above statement, remove '\t'. Otherwise, you will get wrong output.

Some more ways of reading data/input using scanf() function:

scanf(%3d%3d%3d",&a,&b,&c)	3 integer value	1 2 3 Here a=1, b=2 ,c=3
		222 333 444 Here a=222, b=333 ,c=444
		123456789 Here,a=123,b=456,c=789
		1234567892415 Here,a=123,b=456,c=789 Remaining 4 digits 2415 will be ignored.
		2468 54321 Here a=246, b=8,c=543 And remaining degits 2 1 will be ignored. (Space Matters)
scanf("%4s",&str);	1 string	Sahyog Here, str=sahy **og** will be ignored.
Note: To read string with spaces use gets() function		

Program to get the total no. of inputs taken using scanf().

```
#include<stdio.h>
int main()
{
  int a,b;
  printf("%d",scanf("%d%d",&a,&b));
  return 0;
}
```

Output:

```
2
```

OR

```
#include<stdio.h>
int main()
{
   int n1,n2;
   int a=scanf("%d %d",&n1,&n2);
   printf("a=%d",a);
   return 0;
}
```

printf():

- The data stored in the memory area can be displayed on monitor or output device using the printf() function.
- The user can display integer values,floating-points,characters and strings using printf() function.
- printf() returns the total no. of digits or characters displayed on output screen.

Syntax 1:

```
printf("Format specifiers", variable_names);
```

Syntax 2:

```
printf("message to display");
```

Program to display the total no. of characters displayed using printf().

```
#include<stdio.h>
int main()
{
   printf("%d",printf("hello how are you?"));
   return 0;
}
```

Output:

```
hello how are you?
18
```

OR

```
#include<stdio.h>
int main()
{
   int a = printf ("GOOD MORNING");
   printf(" %d", a);
   return 0;
}
```

Output:

```
GOOD MORNING 12
```

Explanation: printf() prints the good morning first and then returns the number of characters it has printed and that no is stored in variable 'a' that is why its output is GOOD MORNING 12, 12 is returned by printf() function.

The following table shows the given values, for the statement to be executed and output obtained on the screen.

Input values	**Output statement to be executed**	**Output obtained**
a=1,b=2,c=3	printf("a=%d,b=%d,c=%d", a, b, c);	a=1,b=2,c=3
a=1,b=2,c=3	printf("a=%d\tb=%d\tc=%d",a,b,c);	a=1 b=2 c=3

a=1,b=2,c=3	printf("a=%d\nb=%d\nc=%d",a,b,c);	a=1
		b=2
		c=3

	printf("\"Hello\"");	"Hello"
	printf("Hello");	Hello
	Printf("New\tDelhi");	New Delhi

Field Width Specification for Integer:

The printf() function uses the field width to know the total number of spaces used on the screen while printing the value.

The specification is provided by means of a format string like **"%wk"** where,

- w indicates total number of spaces required to print the value(width specifier).
- k is the format specifier i.e. %d,%c etc.

Input Values	Output statement to be executed	Output Obtained
n=208	printf("%5d",n); **Note:** Right Justification printing because of positive value.	$\boxed{}\boxed{}\boxed{2}\boxed{0}\boxed{8}$
n=208	printf("%-6d",n); **Note:** Left Justification printing because of negative value.	$\boxed{2}\boxed{0}\boxed{8}\boxed{}\boxed{}\boxed{}$
n=208	printf("%2d",n); **Note:** if the space specified is less than the actual requirement then the width specifier is ignored.	$\boxed{2}\boxed{0}\boxed{8}$

Field Width Specification for Float:
The real numbers (floating point numbers) can be displayed using printf() statement with format specification.
Syntax:
"w.xf"

Where,
- w indicates total numbers of spaces required to print the values.
- x is the numbers of digits displayed after the decimal point.
- f is the format specifier for floating points.

Input Values	Statement to be executed	output
n=13.243	Printf("%7.2f",n); **Note:** Right justification printing	$\boxed{}\boxed{}\boxed{1}\boxed{3}\boxed{.}\boxed{2}\boxed{4}$
n=13.243	Printf("%-7.2f",n); **Note:** Left justification printing	$\boxed{1}\boxed{3}\boxed{.}\boxed{2}\boxed{4}\boxed{}\boxed{}$
n=13.243	Printf("%3.2f",n); **Note:** Field width ignored	$\boxed{1}\boxed{3}\boxed{.}\boxed{2}\boxed{4}$
n=13.243	Printf("%1.1f",n); **Note:** Field width ignored	$\boxed{1}\boxed{3}\boxed{.}\boxed{2}$
n=13.243	Printf("%2.0f",n); **Note:** Field width ignored	$\boxed{1}\boxed{3}$

Field Width Specification for string:
Strings can also be displayed using printf() function.

Syntax:
"%w.xs" **where,**
- w indicates the total spaces required to display the string.
- x indicates the total no. of characters to display.
- s is the format specifier for string.

Input Values	Statement to be executed	output
n="programming in c"	printf("%s",n); **Note:** Space also takes space	$\boxed{p}\boxed{r}\boxed{o}\boxed{g}\boxed{r}\boxed{a}\boxed{m}\boxed{m}\boxed{i}\boxed{n}\boxed{g}\boxed{}\boxed{i}\boxed{n}\boxed{}\boxed{c}$

n=”programming in c”	printf(“%15.10s”,n) **Note:** Right Justification						p	r	o	g	r	a	m	m	i	n
n=”programming in c”	printf(“%-15.10s”,n) **Note:** Left Justification	p	r	o	g	r	a	m	m	i	n					

Mathematical Functions:

- The C language provides number of library functions that carry out operations such as square root ,sin,cosine,tan,abs etc.These functions which are part of c mathematical library are called mathematical library functions.
- The definition for each mathematical function are available in “math.h” header file.Therefore anyone who uses these functions should include “math.h” in the beginning of the program.

Few mathematical functions are given below:

Functions	Syntax	Use
• abs()	• abs(integer_value/variable)	• Returns absolute value of integer number.
• fabs	• fabs(float_value/variable)	• Returns absolute value of floating number.
• sqrt()	• sqrt(integer_value/variable)	• returns square root of a number
• pow()	• pow(base,exponent)	• Returns power of a given number.
• round()	• round(float_value/variable)	• it returns the nearest integer value of float type.
• trunc()	• trunc(float_value/variable)	• it truncates the decimal value from floating point and returns integer value.

Programs for Mathematical Functions:

1. **abs() & fabs():**

```
#include<stdio.h>
#include<math.h>
int main()
{
 int n=-12,m=11,r1,r2;
 float p=-10.432,q=12.43,p1,p2;
 r1=abs(n);
 printf("Absolute of %d is %d",n,r1);
 r2=abs(m);
 printf("\nAbsolute of %d is %d",m,r2);
 p1=fabs(p);
  printf("\nAbsolute of %f is %f",p,p1);
 p2=fabs(q);
 printf("\nAbsolute of %f is %f",q,p2);
return 0;}
```

Output:

```
Absolute of -12 is 12
Absolute of 11 is 11
Absolute of -10.432000 is 10.432000
Absolute of 12.430000 is 12.430000
```

2. sqrt and pow()

```
 int main()
{
 int a=2,b=4,c=8,s,p;
 s=sqrt(c);
 printf("square root of %d is %d",c,s);
 p=pow(a,b);//pow(2,4)
 printf("\npower=%d",p);
return 0;
}
```

Output:

```
Square root of 8 is 2
power=16
```

3. round() and trunc()

```
  #include<stdio.h>
  #include<math.h>
  int main()
  {
   float a=12.564,b=32.213,r1,r2,t;
   r1=round(a);
   printf("round of %f is %f",a,r1);
   r2=round(b);
   printf("\nround of %f is %f",b,r2);
   t=trunc(a);
    printf("\n%f after trunc is %f",a,t);
  return 0;
  }
```

Output:

```
  round of 12.564000 is 13.000000
  round of 32.213001 is 32.000000
  12.564000 after trunc is 12.000000
```

Additional Programs:

1. WAP to add 2 numbers(take input from user)

```
#include<stdio.h>
int main()
{
    int num1,num2,add;
    printf("Enter 1st number:\n");
    scanf("%d",&num1);//5
    printf("Enter 2nd number:\n");
    scanf("%d",&num2);//10
    add=num1+num2;
    printf("----------------------\n");
    printf("%d + %d = %d",num1,num2,add);
    printf("\n----------------------\n");
              return 0;
}
```

Output:

```
Enter 1st number:
12
Enter 2nd number:
10
----------------------
12 + 10 = 22
----------------------
```

2. WAP to check smallest no between 2 nos. using conditional operator(take input from user)

```
#include<stdio.h>
#include<math.h>
int main()
{
    int num1,num2,small;
    printf("Enter 1st number:\n");
    scanf("%d",&num1);//5
    printf("Enter 2nd number:\n");
    scanf("%d",&num2);//10

    num1<num2 ? printf("%d is the smallest no",num1) : printf("%d is the smallest no",num2);
   return 0;
}
```

output:

```
Enter 1st number:
23
Enter 2nd number:
56
23 is the smallest no
```

3. WAP to convert rupees into paise. (take input from user).

```
#include<stdio.h>
int main()
{
    int rs,paise;
    printf("Enter rupees:\n");
    scanf("%d",&rs);//3
        paise=rs*100;
    printf("%d RS=%d paise",rs,paise);
    return 0;
}
```

Output:

```
Enter rupees:
5
5 RS=500 paise
```

Task

- WAP to convert inch to cm. (take input from user) (1 inch=2.54 cm)
- WAP to find out sqrt of a number using sqrt() function(take input from user)
- WAP to find out square and cube of a number.(take input from user)

Branching Statements:

In sequential control all the statements are executed in the order specified in the program one after the other however computer can skip execution of some statements. This skipping facility is provided by the instructions called skip instructions. These skip instructions are also called branching statements. Thus, the statements that alter the sequence of execution of the program are called branching statements.

Branching Statements

Conditional Branching Statement	**Unconditional Branching Statements**
Simple if /if	goto
If-else	break
Nested-if	continue
else-if ladder	
switch	

Conditional Statements:

All the statements are executed one after the other, however computer can skip the execution of some statements based on some conditions. These statements that alter the sequence of execution of the program based on some condition are called conditional branching statements or selection statements or decision statements.

For Eg: if,if-else,else-if ladder and switch statements.

1. **Simple if / if statements**

The simple if or if statement is a simple decision statement.When a set of statement have to be executed or skipped when the condition is true or false,then if statement is used.

Syntax of if:

```
if(condition/expression)
{
   Statement-1;  ⎫
   Statement-2;  ⎪
         :       ⎬  Execute if condition is true
         :       ⎪
   Statement-n;  ⎭
   }
```

Working of if statement:

The if statement evaluates the expression/condition inside the parenthesis ().

- If the expression/condition is evaluated to true, statements inside the body of if are executed.
- If the expression/condition is evaluated to false, statements inside the body of if are not executed.

Note: If the body of an if...else statement has only one statement, you do not need to use brackets {} but if more than 1 statement is associated with if then {} is required.

Example:

```
if (condition)
  statement 1;
```

In this example, only statement 1 is associated with the if statement. so {} is not mandatory.

Program to display a message if user enters less than 10.

```
#include<stdio.h>
int main()
{
    int n;
    printf("Enter a number:\n");
    scanf("%d",&n);
    if(n<10)
  {
   printf("value is less than 10");
   }
return 0;
}
```

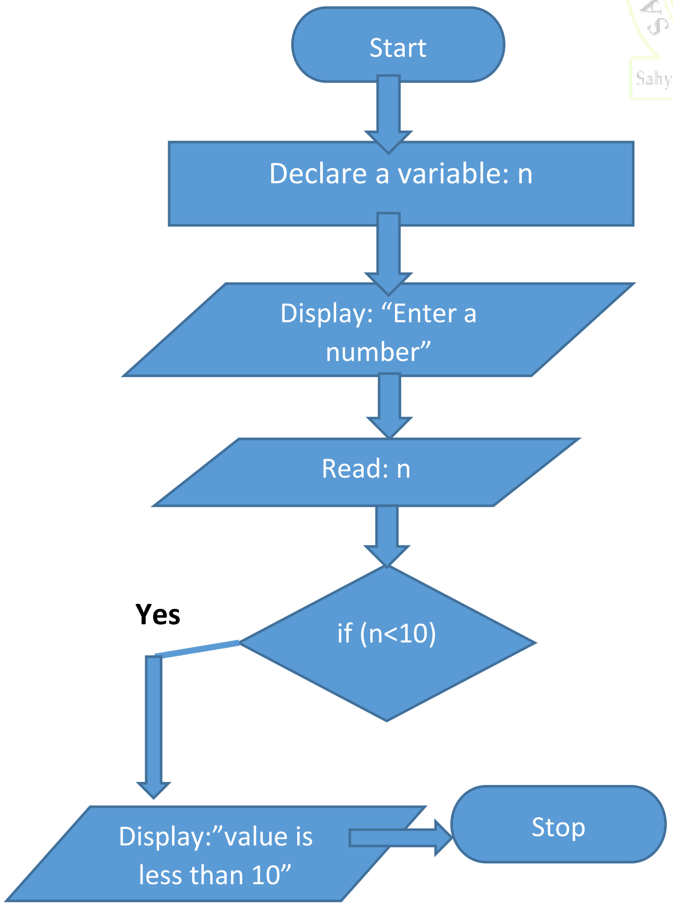
Output 1:

```
 Enter a number:
  4
Value is less than 10
```

Output 2:

```
Enter a number:
13
```

Flowchart for above program



Algorithm for above program

```
step 1: start
step 2: Declare variable:n
step 3: display:"Enter a number"
step 4: read: n
step 5: if(n<10)
            display:"value is less than 10"
step 6: stop
```

Program to check for positive number.

```
#include<stdio.h>
int main()
{
int n;
printf("Enter a number\n");
scanf("%d",&n);
if ( n>0)
{
printf("%d is a positive number");
}
return 0;
}
```

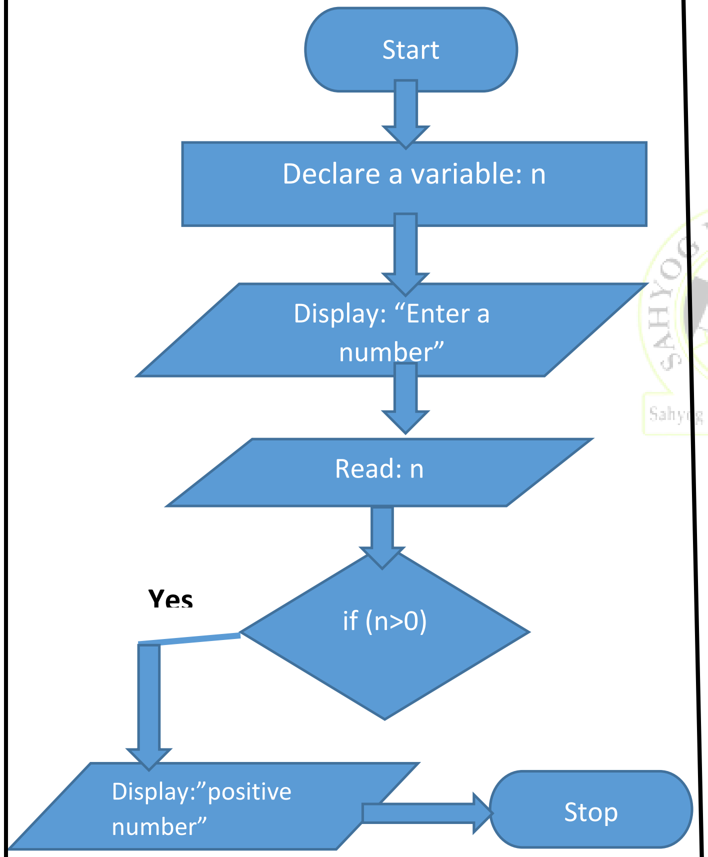
Output 1:

```
  Enter a number:
   20
20 is a positive number
```

Output 2:

```
Enter a number:
 -10
```

Flowchart for above program



Algorithm for above program

```
step 1: start
step 2: Declare variable:n
step 3: Display:"Enter a number"
step 4: Read: n
step 5: if(n>0)
            Display:"positive number"
step 6: stop
```

2. if-else statement

- It is a two way decision statement which executes one statement (or block of statements) or another statement (or block of statements).
- one of these two is done when condition is true and the other one when the condition is false for this purpose if-else statement is used.
- For true part if takes care and for the false part else takes care.

Syntax for if-else:

$$\begin{array}{l}
\text{if(condition)}\\
\{\\
\left.\begin{array}{l}\text{Statement-1;}\\ \text{Statement-2;}\\ \quad :\\ \quad :\\ \text{Statement-n;}\end{array}\right\}\ \textbf{Execute if condition is true}\\
\}\\
\text{else}\\
\{\\
\left.\begin{array}{l}\text{Statement-1;}\\ \quad :\\ \text{Statement-n;}\end{array}\right\}\ \textbf{Execute if condition is false}\\
\}
\end{array}$$

Working of if-else statement

If the expression is evaluated to true,

- statements inside the body of if are executed.
- statements inside the body of else are skipped from execution.

If the test expression is evaluated to false,

- statements inside the body of else are executed
- statements inside the body of if are skipped from execution.

Program to check smallest no between 2 numbers using if-else

```
#include<stdio.h>
int main()
{
int a,b;
printf("Enter 2 numbers\n");
scanf("%d%d",&a,&b);
if(a<b)
{
printf("%d is the smallest number",a);
}
else
{
printf("%d is the smallest number",b);
}
return 0;
}
```

Output 1:

```
Enter 2 numbers
12
24
12 is the smallest number
```

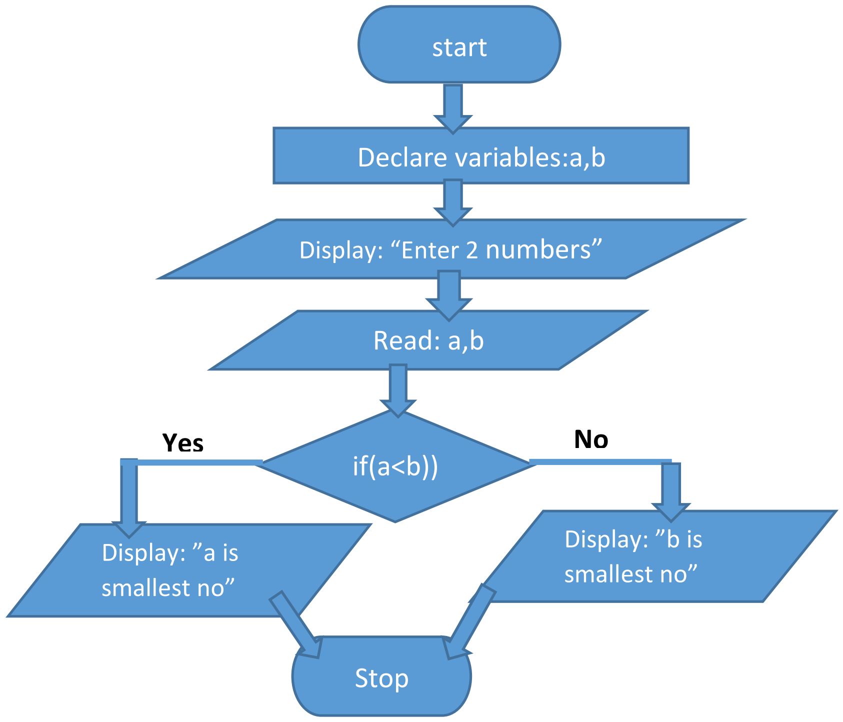
Output 2:

```
Enter 2 numbers
34
87
34 is the smallest number
```

Algorithm for above program

```
step 1: start
step 2: Declare variables:a,b
step: Display:"Enter 2 numbers"
step 4: Read: a,b
step 5: if(a>b)
            Display:"a is smallest number"
             else
           Display:"a is smallest number"
step 6: stop
```

Flowchart


start
Declare variables:a,b
Display: "Enter 2 numbers"
Read: a,b
if(a<b))
Yes
No
Display: "a is smallest no"
Display: "b is smallest no"
Stop

Program to check whether given no is even or odd.

```
#include<stdio.h>
int main()
{
int num;
printf("Enter number:\n");
scanf("%d",&num);
if(num%2==0)
{
printf("Number is Even");
}
else
{
printf("Number is odd");
}
return 0;
}
```

```
Output 1:                    Output 2:

Enter 2 numbers                Enter 2 numbers
12                             13
Number is Even                 Number id odd
```

Program to check whether year is leap year or not.

What is a leap year?

Definition: A year which satisfies one of the following two cases:

- It is divisible by 4 and should not be divisibly by 100
- Divisible by 400

If one of the conditions is true, it is a leap year. But, if both conditions are false, the year is not leap year.

```
#include <stdio.h>
int main()
{
int year;
printf("Enter the year");
scanf("%d",&year);
if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0))
printf("%d is a leap year",year);
else
printf("%d is not a leap year",year);
return 0;
}
```

Output:

```
Enter the year
2020
2020 is a leap year
```

Task:

- Prepare Flowchart and Algorithm for above program.
- Write a C program to check whether a number is divisible by 5 and 11 or not using if else.
- Write a C program to check whether a character is alphabet or not.
- Write a C program to input any alphabet and check whether it is vowel or consonant.
- Write a C program to input any character and check whether it is alphabet, digit or special character.
- Write a C program to check whether a character is uppercase or lowercase alphabet.
- Write a C program to input all sides of a triangle and check whether triangle is valid or not.

Note:

- We can directly put any integer number or zero 0 at condition place.
- Integer number indicates than condition is true and zero indicates that the condition is false.

Program 1:

```
#include<stdio.h>
int main()
{
        if(1)
        printf("hi");
        else
        printf("hello");
        return 0;
}
```

Output:

```
hi
```

Explanation for above program:

- if(1) indicates that condition is true so here the statements inside the if block will execute and else block will get skipped.

program 2:

```
#include<stdio.h>
int main()
{
        if(0)
        printf("hi");
        else
        printf("hello");
        return 0;
}
```

Output:

```
hello
```

Explanation for above program:

- if(0) indicates that the condition is false so here the statements inside the else block will execute and if block will get skipped.

program 3:

```
  #include<stdio.h>
  int main()
  {
        if(4)
        printf("hi");
        else
        printf("hello");
        return 0;
  }
```

Output:

```
hi
```

Explanation for above program:

- if(4) indicates that condition is true so here the statements inside the if block will execute and else block will get skipped.

3. else-if ladder

- The if...else statement executes two different codes depending upon whether the test expression is true or false. Sometimes, a choice has to be made from more than 2 possibilities.
- The if...else ladder allows you to check between multiple test expressions and execute different statements.
- else-if ladder helps user decide from among multiple options. The statements are executed from the top down.

syntax for else-if ladder

```
if(condition)
{
statements;
}
else if(condition)
{
statements;
}
else if(condition)
{
statement;
}
else
{
statements;
}
```

Program to stimulate calculator using else if ladder:

```
#include<stdio.h>
int main()
{
        int a,b,c,ch;
        printf("Enter 1 for addition\nEnter 2 for subtraction\n");
        printf("Enter 3 for Multiplication\nEnter 4 for division");
        printf("\n===================\n");
        printf("Enter Your Choice:");
        scanf("%d",&ch);
        if(ch==1)
        {
                printf("Enter 2 numbers\n");
                scanf("%d%d",&a,&b);
                c=a+b;
                printf("%d+%d=%d",a,b,c);
        }

        else if(ch==2)
        {
            printf("Enter 2 numbers\n");
                scanf("%d%d",&a,&b);
                c=a-b;
                printf("%d-%d=%d",a,b,c);
        }
        else if(ch==3)
        {
            printf("Enter 2 numbers\n");
                scanf("%d%d",&a,&b);
```

OUTPUT

```
Enter 1 for Addition
Enter 2 for Subtraction
Enter 3 for Multiplication
Enter 4 for Division
==================
Enter Your Choice:
3
Enter 2 numbers
12
5

12*5=60
```

```
                    c=a*b;
                    printf("%d*%d=%d",a,b,c);
          }
          else if(ch==4)
          {
             printf("Enter 2 numbers\n");
             scanf("%d%d",&a,&b);
                    c=a/b;
                    printf("%d/%d=%d",a,b,c);
          }
          else
          {
                    printf("Invalid Choice");
          }
          return 0;
  }
```

Additional Programs:

Program to find out smallest no between 3 numbers.

```
  #include<stdio.h>
  int main()
  {
          int n1,n2,n3;
          printf("Enter 3 numbers:\n");
          scanf("%d%d%d",&n1,&n2,&n3);

    if(n1<n2 && n1<n3)
          {
             printf("%d is the smallest number\n",n1);
          }
          else if(n2<n1 && n2<n3)
          {
             printf("%d is the smallest number\n",n2);
          }
     else if(n3<n1 && n3<n2)
          {
             printf("%d is the smallest number\n",n3);
          }
          else
          {
             printf("All are equal");
          }
          return 0;
  }
```

OUTPUT

```
Enter 3 numbers:
23
11
65
11 is the smallest number
```

Program to check whether entered number is positive, negative or zero using else if ladder.

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

```
#include<conio.h>
int main()
{
    int a;
    printf("Enter a Number: ");
    scanf("%d",&a);
    if(a > 0)
    {
        printf("%d is Positive number",a);
    }
    else if(a == 0)
    {
        printf("Entered Number is Zero");
    }
    else if(a < 0)
    {
        printf("%d is Negative number",a);
    }
   return 0;
}
```

Output 1:

```
Enter a Number: -9
-9 is Negative number
```

Output 2:

```
Enter a Number: 11
11 is Positive number
```

4. Nested-if:

- Nested if statements means an if statement inside another if statement and It is possible to include an if...else statement inside the body of another if...else statement.
- When a series of decision is required, nested if-else is used. in nested if, conditions are checked in series that in sequence and here condition checking is depend on each other i.e. if first condition is true then only second condition will get checked otherwise not.

syntax:

```
 if(condition1)
{
       if(condition2)
       {
            statements;
        }
       else
      {
                 statements;
      }
   }
  else
  {
        statements;
    }
```

Program to check whether candidate is eligible for the interview or not using nested-if

conditions: 1. if candidate has scored more than 60 in ssc then only he/she will be eligible for second round
2. if candidate has scored more than 80 in C then he/she will be selected for the interview else not.

Note: if ssc marks is not more than 60 then second condition should not be checked, directly candidate will be out of the interview.

Program:

```
#include<stdio.h>
int main()
{
        int ssc,c;
        printf("Enter ssc score:\n");
        scanf("%d",&ssc);
        printf("Enter c score:\n");
        scanf("%d",&c);
        if(ssc>60)
        {
                if(c>80)
                {
                        printf("Congratulations!!! you are selected");
                }
                else
                {
                        printf("Sorry!!!Try to score more in C");
                }
        }
        else
        {
                printf("sorry!! try to score more in ssc");
        }
        return 0;
}
```

Output 1

```
Enter ssc score:
34
Enter c score:
90
sorry!! try to score more in ssc
```

Output 2

```
Enter ssc score:
90
Enter c score:
60
Sorry!!!Try to score more in C
```

Output 3

```
Enter ssc score:
98
Enter c score:
87
Congratulations!!! you are selected
```

1.else and else..if are optional statements, a program having only “if” statement would run fine.
2. else and else..if cannot be used without the “if”.
3. There can be any number of else..if statement in an if else..if block.
4. If none of the conditions are met then the statements in else block gets executed.
5. Just like relational operators, we can also use logical operators such as AND (&&), OR(||) and NOT(!).

5. **switch():**

- The switch statement allows us to execute one code block among many alternatives.
- You can do the same thing with the if...else..if ladder. However, the syntax of the switch statement is much easier to read and write.
- Switch statement in C checks the value of a variable and compares it with multiple cases. Once the case match is found, a block of statements associated with that particular case is executed.
- Each case in a block of a switch has a different name/number which is referred to as an identifier. The value provided by the user is compared with all the cases inside the switch block until the match is found.
- If a case match is NOT found, then the default statement is executed, and the control goes out of the switch block.

syntax:

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

Note:

- The expression can be integer or a character only, floating points and strings are not allowed in switch expression.
- Duplicate case values are not allowed.
- The case value must be an integer or character constant

How does the switch statement work?

- The expression is evaluated once and compared with the values of each case label.
- If there is a match, the corresponding statements after the matching label are executed. For example, if the value of the expression is equal to value-2, statements after case value-2 are executed until break is encountered.
- If there is no match, the default statements are executed.
- If we do not use break, all statements after the matching label are executed until a break is reached.
- By the way, the default clause inside the switch statement is optional.

Program to stimulate a calculator using switch:

```
#include <stdio.h>
int main()
{
   char op;
   int n1, n2;
   printf("Enter an operator (+, -, *, /):\n ");
```

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

    switch(op)
    {
        case '+':
             printf("Enter two operands:\n ");
             scanf("%d %d",&n1, &n2);
             printf("\n==============================\n");
             printf("%d + %d = %d",n1, n2, n1+n2);
             printf("\n==============================\n");
             break;

        case '-':
            printf("Enter two operands:\n ");
             scanf("%d %d",&n1, &n2);
             printf("\n==============================\n");
             printf("%d - %d = %d",n1, n2, n1-n2);
             printf("\n==============================\n");
             break;

        case '*':
            printf("Enter two operands:\n ");
             scanf("%d %d",&n1, &n2);
             printf("\n==============================\n");
             printf("%d * %d = %d",n1, n2, n1*n2);
             printf("\n==============================\n");
             break;

        case '/':
              printf("Enter two operands:\n ");
             scanf("%d %d",&n1, &n2);
             printf("\n==============================\n");
             printf("%d / %d = %d",n1, n2, n1/n2);
             printf("\n==============================\n");
             break;

        default:
            printf("Error! operator is not correct");
    }
    return 0;
}
```

Output:

```
Enter an operator (+, -, *, /):
-
Enter two operands:
 12
 3
==============================
12 - 3 = 9
```

Program to execute more than 1 case at a time.

- We can also execute multiple cases based on a single value given by the user.

```
#include<stdio.h>
int main()
{
        int n;
        printf("Enter value\n");
        scanf("%d",&n);
        switch(n)
        {
                case 1:
                case 2:
                case 3:
                        printf("User entered 1,2 or 3");
                        break;
                case 4:
                case 5:
                case 6:
                        printf("User entered 4,5 or 6");
                        break;
                default:
                        printf("User entered more than 6 or some other value");
        }
        return 0;
}
```

Output 1:

```
Enter value
3
User entered 1, 2 or 3
```

Output 2:

```
Enter value
8
User entered more than 6 or some other value
```

Additional Programs:

C program to read weekday number and print weekday name using switch.

```
#include <stdio.h>
int main()
{
    int wDay;

    printf("Enter weekday number (1-7):\n ");
    scanf("%d",&wDay);

    switch(wDay)
    {
        case 1:
            printf("Sunday");
            break;
        case 2:
            printf("Monday");
            break;
        case 3:
            printf("Tuesday");
            break;
        case 4:
            printf("Wednesday");
            break;
        case 5:
            printf("Thursday");
            break;
        case 6:
            printf("Friday");
            break;
        case 7:
            printf("Saturday");
            break;
        default:
            printf("Invalid weekday number.");
    }
    return 0;
}
```

Output:

```
Enter weekday number (1-7):
 3
Tuesday
```

C program to check whether a character is VOWEL or CONSONANT using switch.

```
#include<stdio.h>
int main()
{
    char ch;
    printf("Enter a character: ");
    scanf("%c",&ch);
    if((ch>='A' && ch<='Z') || (ch>='a' && ch<='z')) //condition to check character is alphabet or not
    {
        switch(ch)
        {
            case 'A':
            case 'E':
            case 'I':
            case 'O':
            case 'U':
            case 'a':
            case 'e':
            case 'i':
```

```
            case 'o':
            case 'u':
                printf("%c is a VOWEL.\n",ch);
                break;
            default:
                printf("%c is a CONSONANT.\n",ch);
        }
    }
    else
    {
        printf("%c is not an alphabet.\n",ch);
    }
    return 0;
}
```

Output :

```
Enter a character: U
U is a VOWEL.
```

C program to check whether number is EVEN or ODD using switch.

```
#include <stdio.h>
int main()
{
    int number;
    printf("Enter a positive integer number: ");
    scanf("%d",&number);

    switch(number%2) //this will return either 0 or 1
    {
        case 0:
            printf("%d is an EVEN number.\n",number);
            break;
        case 1:
            printf("%d is an ODD number.\n",number);
            break;
    }
    return 0;
}
```

Output 1:

```
Enter a number:
3
3 is an ODD number.
```

Output 2:

```
Enter a number:
12
12 is an EVEN number.
```

Valid and invalid switch expression.

Expression	Valid	Invalid	Reason for Invalidity
switch(12)	valid	-	-
switch(2.3)	-	Invalid	Floating points can’t be used as switch expression
switch(“name”)	-	Invalid	String can’t be used as switch expression
switch(a%2)	valid		
switch(1+2)	Valid		
switch(‘c’)	Valid		

TASK:

- C program to find number of days in a month using switch case.
- C program to read gender (M/F) and print corresponding gender using switch.

Type Casting and Conversion in C:

- In a programming language, the expression contains data values of the same datatype or different data types.
- When the expression contains similar datatype values then it is evaluated without any problem. But if the expression contains two or more different datatype values then they must be converted to the single datatype of destination datatype.
- Here, the destination is the location where the final result of that expression is stored. For example, the multiplication of an integer data value with the float data value and storing the result into a float variable. In this case, the integer value must be converted to float value so that the final result is a float datatype value. In c programming language, the data conversion is performed in two different methods as follows.

1. **Type Conversion**
2. **Type Casting**

- **The type conversion** is the process of converting a data value from one data type to another data type automatically by the compiler. Sometimes type conversion is also called **implicit type conversion**.
- The implicit type conversion is automatically performed by the compiler. For example, in c programming language, when we assign an integer value to a float variable the integer value automatically gets converted to float value by adding decimal value 0. And when a float value is assigned to an integer variable the float value automatically gets converted to an integer value by removing the decimal value. To understand more about type conversion observe the following.

```
int i=10;
float x=15.5;
char ch=’A’;
```

i = x ; ===> x value 15.5 is converted as 15 and assigned to variable i

x = i ; ===> Here i value 10 is converted as 10.000000 and assigned to variable x

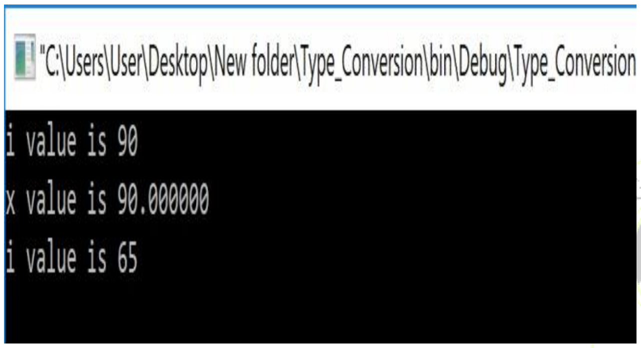
i = ch ; ===> Here the ASCII value of A (65) is assigned to i

Program:

```
#include<stdio.h>
int main()
{
  int i = 95 ;
  float x = 90.99 ;
  char ch = 'A' ;

  i = x ;
  printf("i value is %d\n",i);
  x = i ;
  printf("x value is %f\n",x);
  i = ch ;
  printf("i value is %d\n",i);
}
```

Output:



```
"C:\Users\User\Desktop\New folder\Type_Conversion\bin\Debug\Type_Conversion
i value is 90
x value is 90.000000
i value is 65
```

- In the above program, we assign **i = x**, i.e., float variable value is assigned to the integer variable. Here, the compiler automatically converts the float value (90.99) into integer value (90) by removing the decimal part of the float value (90.99) and then it is assigned to variable **i**. Similarly, when we assign **x = i**, the integer value (90) gets converted to float value (90.000000) by adding zero as the decimal part.

3. Typecasting is also called an **explicit type conversion**. Compiler converts data from one data type to another data type implicitly. When compiler converts implicitly, there may be a data loss. In such a case, we convert the data from one data type to another data type using explicit type conversion. To perform this we use the **unary cast operator**. To convert data from one type to another type we specify the target data type in parenthesis as a prefix to the data value that has to be converted. The general syntax of typecasting is as follows.
(TargetDataType) DataValue;

Program:

```
#include<stdio.h>
int main()
{
int totalMarks = 450, maxMarks = 600 ;
float avg;
avg = totalMarks / maxMarks * 100 ;
printf("Average without cast=%f",avg);
```

```
   avg = (float) totalMarks / maxMarks * 100 ;
printf("\nAverage with cast=%f",avg);
return 0;
}
```

Output:

```
Average without cast=0.000000
Average with cast=75.000000
```

***