

CHAPTER 1

MANAGING INPUT AND OUTPUT OPERATIONS

1. INTRODUCTION

- ✓ Reading, processing and writing of data are three essential functions of a computer program.
- ✓ There are two methods. The first method is to assign the values and the second method is to use input function scanf to read data and output function printf to write data.
- ✓ Ex: **#include<stdio.h>** Includes function calls such as printf() and scanf() into the source code when compiled.

2. READING A CHARACTER

- ✓ The simplest way of all input/output operations is reading a character from the 'standard input' unit (usually keyboard) and writing it to the 'standard output' (usually the screen).
- ✓ Reading a single character can be done by using the function getchar and it does not have any parameter.

Syntax: variable_name = getchar();

Where, variable_name is valid and has been declared in char type.

Example: char name;
name = getchar();

- ✓ The getchar function can also be used to read the characters successively until the return key is pressed.
- ✓ The character function are contained in the file type ctype.h and therefore the preprocessor directive #include<ctype.h> is included.
- ✓ And the functions of ctype.h returns 1 if (TRUE) and 0 if (FALSE).

Function	Test
isalnum(c)	Is c alphanumeric character?
isalpha(c)	Is c alphabetic character?
isdigit(c)	Is c a digit?
islower(c)	Is c a lower case letter?
isupper(c)	Is c a upper case letter?
isspace(c)	Is c a white case character?
ispunct(c)	Is c a punctuation mark?
isprint(c)	Is c a printable character?

Example Program:

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

```
#include<ctype.h>
```

```

void main( )
{
    char key;
    printf("Enter any key\n");
    key=getchar( );
    if(isalpha(key))
        printf("The entered key is a letter\n");
    else if(isdigit(key))
        printf("The entered key is a digit\n");
    else
        printf("The entered key is not alphanumeric\n");
}

```

OUTPUT:

Enter any key A The entered key is a letter	Enter any key 7 The entered key is a digit	Enter any key \$ The entered key is not alphanumeric
---	--	--

3. WRITING A CHARACTER

- ✓ Writing a single character can be done by using the function putchar.

Syntax: putchar(variable_name);

Where, variable_name is valid and has been declared in char type.

Example: name='y';
putchar(name);

Will display the character y on the screen

- ✓ The characters entered can be converted into reverse i.e., from upper to lower case and vice versa.
- ✓ The functions used to convert are toupper, tolower.

Example Program:

```

#include<stdio.h>
#include<ctype.h>
void main( )
{
    char alphabet;
    printf("Enter any alphabet\n");

```

```

alphabet=getchar( );
if(islower(alphabet))
    putchar(toupper(alphabet));
else
    putchar(tolower(alphabet));
}

```

OUTPUT:

Enter any alphabet A A	Enter any alphabet a A
------------------------------	------------------------------

4. FORMATTED INPUT

- ✓ Formatted input refers to an input data that has been arranged in particular format.
- ✓ Ex: 152 Rajesh
- ✓ In the above example the first part contains integer and the second part contains characters. This is possible with the help of scanf function.

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

- ✓ The control string specifies the field form in which data should be entered and the arguments arg1, arg2,.....,argn specifies the address of the location where the data is stored.
- ✓ The width of the integer number can be specified using %wd.

Example-2:

- scanf("%2d %2d", &a, &b);

%2d → two digit integer

%2d → two digit integer

4567 25

a=45 b=67

Note: 25 is not at all stored

- ✓ The input field may be skipped using * in the place of field width.

Example-3:

- scanf("%2d %*d %2d", &a, &b);

%2d → two digit integer

%*d → skip this integer

45 25 63

a=45

25 is skipped

b=63

- ✓ The scanf function can read single character or more than one character using %c or %s.

Rules for scanf:

- ✓ Each variable to be read must have a specification
- ✓ For each field specification, there must be a address of proper type.
- ✓ Never end the format string with whitespace. It is a fatal error.
- ✓ The scanf will read until:
 - A whitespace character is found in numeric specification.
 - The maximum number of characters have been read
 - An error is detected
 - The end of file is reached
- ✓ Any non-whitespace character used in the format string must have a matching character in the user input.
- ❖ The below table shows the scanf format codes commonly used.

Code	Meaning
%c	Read a single character
%d	Read a decimal integer
%e, %f, %g	Read a floating point value
%h	Read a short integer
%i	Read a decimal integer
%o	Read a octal integer
%s	Read a string
%u	Read an unsigned decimal integer
%x	Read a hexadecimal integer
%[]	Read a string of word

The l can be used as a prefix for long integer. Ex: %ld

5. FORMATTED OUTPUT

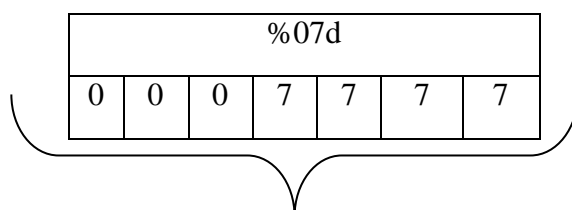
- ✓ The printf function is used for printing the captions and results.

Syntax: printf("control string", arg1, arg2,, argn);

- ✓ Control string contains three types of items:
 1. Characters that will be printed on the screen as they appear.
 2. Format specifications that define the output format for display of each item.
 3. Escape sequence characters such as \n, \t and \b.
- ✓ The printf can contain flags (0 + -)
 - 0 → fill in extra positions with 0's instead of spaces

a=7777

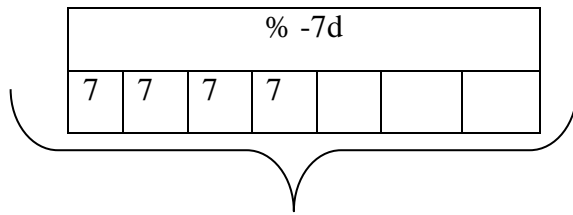
printf("a = %07d", a);



Width of 7 memory space is allocated and padded with leading zeroes

- → Left justify text instead of right justify text

```
a=7777;
```



```
printf("a = % -7d", a);
```

Width of 7 memory space is allocated and printed from left side (left justify)

+ → Print sign of number (whether positive or negative)

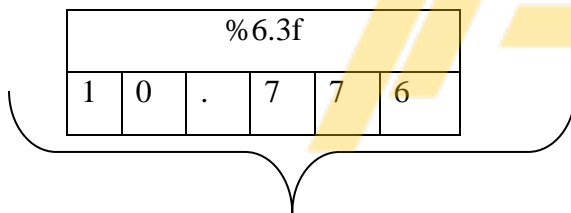
➤ Example with format specifier %d:

```
a=10, b=77;
printf("%d %d", a , b);
```

Output:
10 77

➤ Example with format specifier %f:

```
x=10.776;
printf("%6.3f%", x );
```



6.3f means totally 6 locations are allocated out of which 3 is for number of decimal places.

➤ Example with \n and \t:

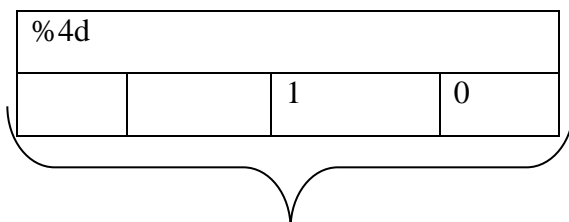
```
a=10, b=77, c=88;
printf("a=%d\n b=%d\tc=%d", a , b, c);
```

Output:

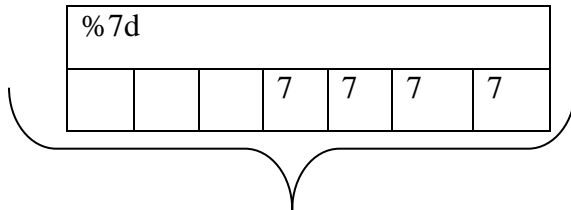
```
a=10
b=77
    c=88
```

➤ Example with size value:

```
a=10, b=7777, c=88;
printf("a=%4d\n b=%7d", a , b);
```



Width of 4 memory space is allocated and printed from right side



Width of 7 memory space is allocated and printed from right side (right justify)

❖ The below table shows the scanf format codes commonly used.

Code	Meaning
%c	Print a single character
%d	Print a decimal integer
%e	Print a floating point value in exponent form
%f	Print a floating point value without exponent
%g	Print a floating point value with or without exponent
%h	Print a short integer
%i	Print a decimal integer
%o	Print a octal integer
%s	Print a string
%u	Print an unsigned decimal integer
%x	Print a hexadecimal integer
%[]	Print a string of word

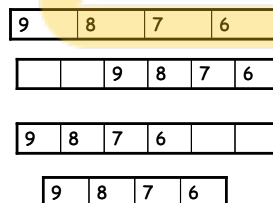
The l can be used as a prefix for long integer. Ex: %ld

➤ General syntax of field-width/size for different data types:

- %wd → Example: %6d
- %w.pf → Example %6.2f
- %w.ps → Example %10.4s
 - w means total field size/width
 - P means precision (tells how many decimal places to be printed in case of float) and
 - How many characters to be printed in case of strings

Examples:

```
a=9876;
printf("%d", a);
printf("%6d",a);
printf("%-6d",a);
printf("%3d",a);
```



NOTE: String input/output function: gets() and puts()

- gets and puts are line oriented input output functions available in the <stdio.h> header file.
- **gets():** Alternative method to read strings is *gets()* (**read it as get string**) function which is an unformatted string input function. Advantage of *gets()* is it can read multi-word strings.
- Example: char name[10];
gets(name);

Say input is:

N	E	W		Y	O	R	K
---	---	---	--	---	---	---	---

The complete string “NEW YORK” is read by *gets()*.

➤ **puts():** This function can be used to display entire string on monitor screen without the help of **%s** specifier.

➤ Example: `char name[] = “Bengaluru”`

`puts(name);`

`puts()` will display the string value stored in parameter passed to it on monitor. In this case it is—Bengaluru.

CHAPTER 2

Decision Making and Branching

1. INTRODUCTION

- ✓ C language possesses such decision making capabilities by supporting the following statements:
 1. **If** statement
 2. **Switch** statement
 3. Conditional operator statement
 4. **goto** statement
- ✓ These statements are popularly known as decision making statements. Also known as control statements.

2. DECISION MAKING (Two-Way Selection Statements)

- ✓ The basic decision statement in the computer is the two way selection.
- ✓ The decision is described to the computer as conditional statement that can be answered TRUE or FALSE.
- ✓ If the answer is TRUE, one or more action statements are executed.
- ✓ If answer is FALSE, the different action or set of actions are executed.
- ✓ Regardless of which set of actions is executed, the program continues with next statement.
- ✓ C language provides following two-way selection statements:
 1. if statement
 2. if – else statement
 3. Nested if else statement
 4. Cascaded if else (also called else-if ladder)

1. if statement: The general form of simple if statements is shown below.

```
if (Expression)
{
    Statement1;
}
Statement2;
```

- ✓ The Expression is evaluated first, if the value of Expression is true (or non zero) then Statement1 will be executed; otherwise if it is false (or zero), then Statement1 will be skipped and the execution will jump to the Statement2.
- ✓ Remember when condition is true, both the Statement1 and Statement2 are executed in sequence. This is illustrated in Figure1.

Note: Statement1 can be single statement or group of statements.

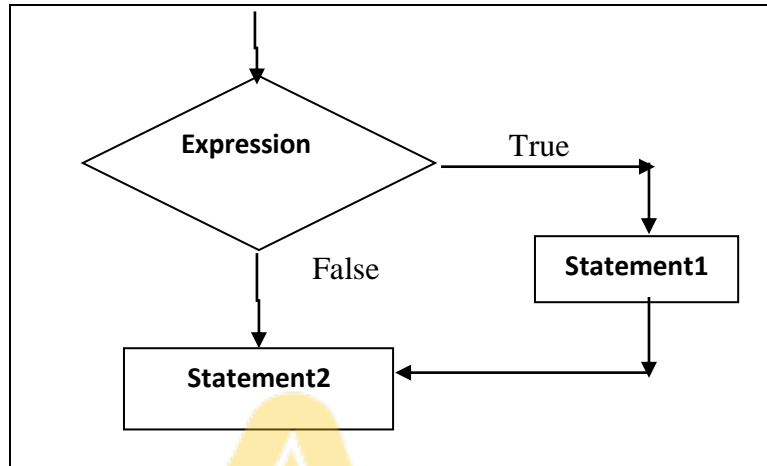


Figure 1: Flow chart of if statement

Example:

```

#include<stdio.h>
void main( )
{
    int a=20, b=11;
    if (a >b)
    {
        printf("A is greater\n");
    }
}
  
```

Output: A is greater

2. if..else statement: The if..else statement is an extension of simple if statement.

```

if (Expression)
{
    Statement1; → true-block
}
else
{
    Statement2; → true-block
}
Statement3;
  
```

- ✓ If the Expression is true (or non-zero) then Statement1 will be executed; otherwise if it is false (or zero), then Statement2 will be executed.

- ✓ In this case either true block or false block will be executed, but not both.
- ✓ This is illustrated in Figure 2. In both the cases, the control is transferred subsequently to the Statement3.

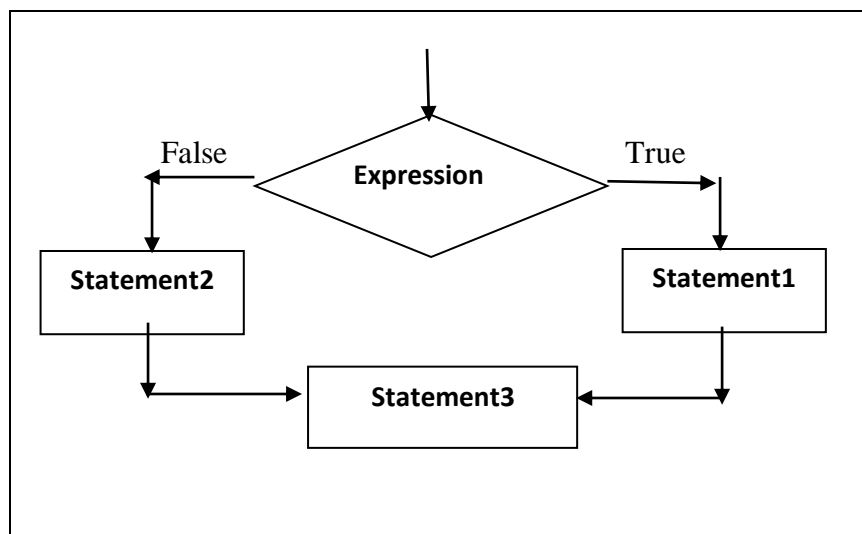


Figure 2: Flow chart of if-else statement

Example:

```

void main( )
{
    int a=10, b=11;
    if (a > b)
    {
        printf("A is greater\n");
    }
    else
    {
        printf("B is greater\n");
    }
}

```

Output: B is greater

3. Nested if .. else statement: When a series of decisions are involved, we have to use more than one if..else statement in nested form as shown below in the general syntax.

```

if (Expression1)
{
    if(Expression2)
    {
        Statement1;
    }
    else
    {
        Statement2;
    }
}

```

```

else if(Expression3)
{
    Statement3;
}
else
{
    Statement4;
}

```

- ✓ If Expression1 is true, check for Expression2, if it is also true then Statement1 is executed.
- ✓ If Expression1 is true, check for Expression2, if it is false then Statement2 is executed.
- ✓ If Expression1 is false, then Statement3 is executed.
- ✓ Once we start nesting if .. else statements, we may encounter a classic problem known as dangling else.
- ✓ This problem is created when no matching else for every if.
- ✓ C solution to this problem is a simple rule “always pair an else to most recent unpaired if in the current block”.
- ✓ Solution to the dangling else problem, a compound statement.
- ✓ In compound statement, we simply enclose true actions in braces to make the second if a compound statement.

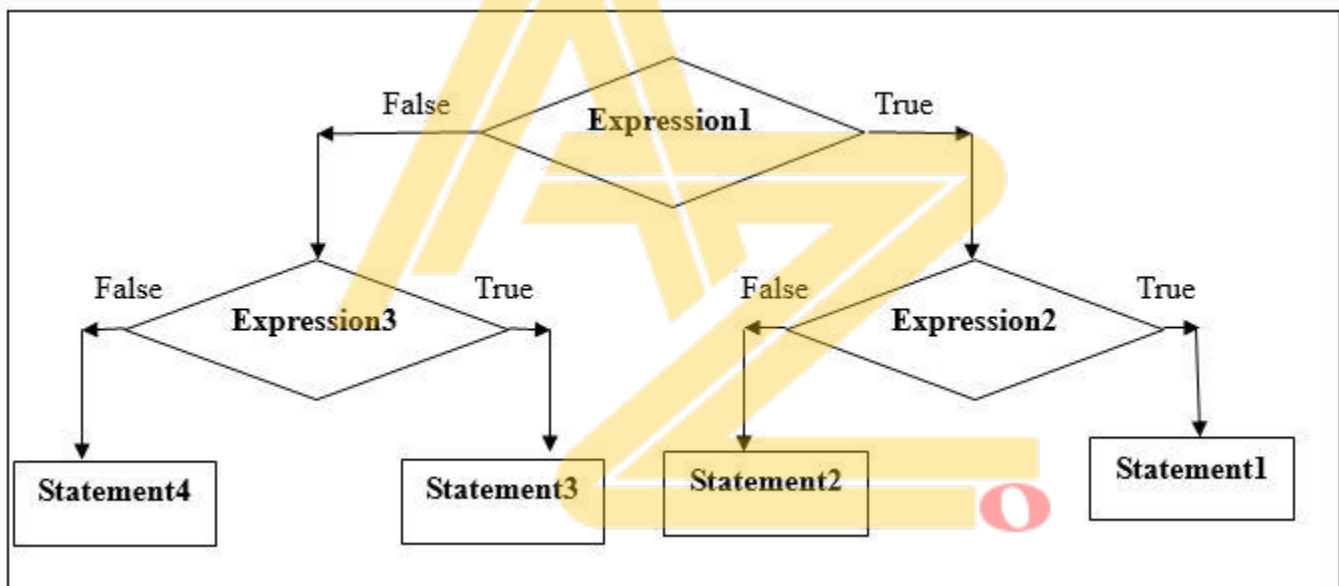


Figure 3: Flow chart of Nested if-else statement

Example:

```

#include<stdio.h>
void main( )
{
    int a = 20, b=15, c=3;
    if(a>b)
    {
        if(a>c)
        {
            printf("A is greater\n");
        }
        else
    }
}

```

```

    {
        printf("C is greater\n");
    }
}
else
{
    if(b>c)
    {
        printf("B is greater\n");
    }
    else
    {
        printf("C is greater\n");
    }
}

```

Output: A is greater

4. else-if ladder or cascaded if else: There is another way of putting ifs together when multipath decisions are involved. A multi path decision is a chain of ifs in which the statement associated with each else is an if. It takes the following form.

```

if (Expression1)
{
    Statement1;
}
else if(Expression2)
{
    Statement2;
}
else if(Expression3)
{
    Statement3;
}
else
{
    Statement4;
}
Next Statement;

```

- ✓ This construct is known as the else if ladder.
- ✓ The conditions are evaluated from the top (of the ladder), downwards. As soon as true condition is found, the statement associated with it is executed and control transferred to the Next statement skipping the rest of the ladder.
- ✓ When all conditions are false then the final else containing the default Statement4 will be executed.

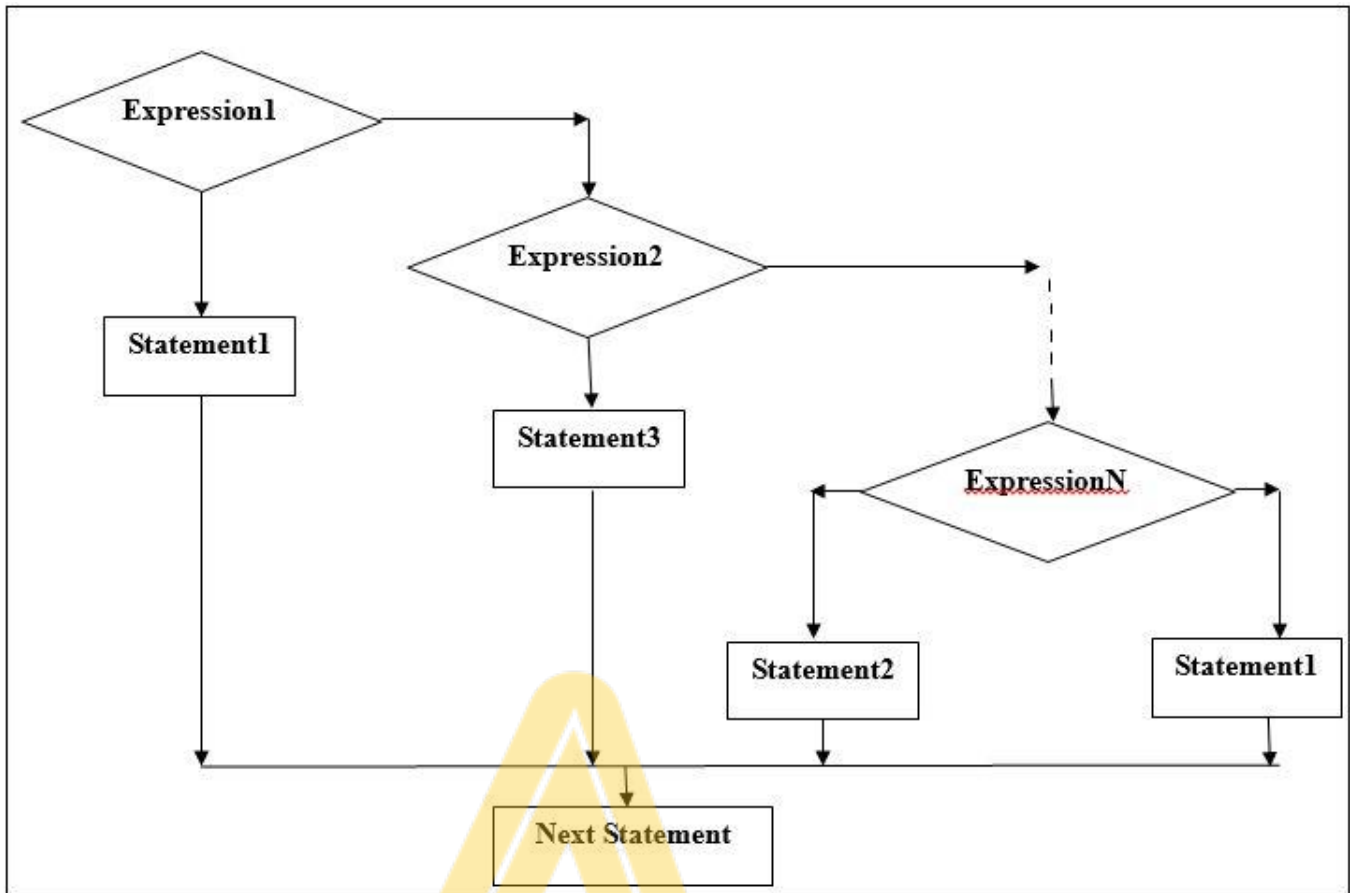


Figure 4: Flow chart of else if ladder statement

Example:

```

#include<stdio.h>
void main( )
{
    int a=20, b=5, c=3;
    if((a>b) && (a>c))
        printf("A is greater\n");
    else if((b>a) && (b>c))
        printf("B is greater\n");
    else if((c>a) && (c>b))
        printf("C is greater\n");
    else
        printf("All are equal\n");
}
  
```

Output: A is greater

3. SWITCH STATEMENT

✓ C language provides a multi-way decision statement so that complex else-if statements can be easily replaced by it. C language's multi-way decision statement is called switch.

General syntax of switch statement is as follows:

```

switch(choice)
{
    case label1:  block1;
                  break;
    case label2:  block2;
                  break;
    case label3:  block-3;
                  break;
    default:      default-block;
                  break;
}

```

- ✓ Here *switch*, *case*, *break* and *default* are built-in C language words.
- ✓ If the choice matches to label1 then block1 will be executed else if it evaluates to label2 then block2 will be executed and so on.
- ✓ If choice does not matches with any case labels, then default block will be executed.

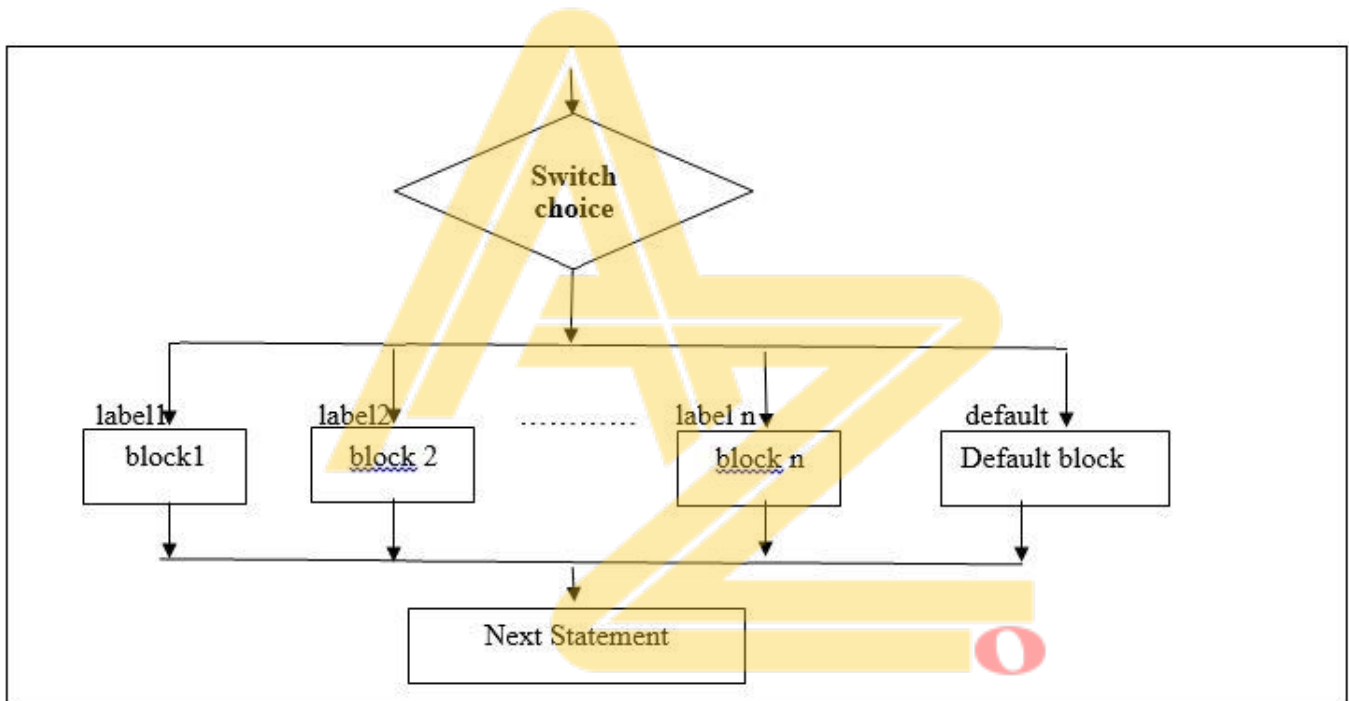


Figure 5: Flow chart of switch case statement

- ✓ The choice is an integer expression or characters.
- ✓ The label1, label2, label3,... are constants or constant expression evaluate to integer constants.
- ✓ Each of these case labels should be unique within the switch statement. block1, block2, block3, ... are statement lists and may contain zero or more statements.
- ✓ There is no need to put braces around these blocks. Note that case labels end with colon(:).
- ✓ Break statement at the end of each block signals end of a particular case and causes an exit from switch statement.
- ✓ The default is an optional case when present, it will execute if the value of the choice does not match with any of the case labels.

Example:

Label → Number	Label → Character
<pre>#include<stdio.h> #include<stdlib.h> void main() { int ch,a,b,res; float div; printf("Enter two numbers:\n"); scanf("%d%d",&a,&b); printf("1.Addition\n 2.Subtraction\n 3.Multiplication\n 4.Division\n 5.Remainder\n"); printf("Enter your choice:\n"); scanf("%d",&ch); switch(ch) { case 1: res=a+b; break; case 2: res=a-b; break; case 3: res=a*b; break; case 4: div=(float)a/b; break; case 5: res=a%b; break; default: printf("Wrong choice!!\n"); } printf("Result=%d\n",res); }</pre>	<pre>#include<stdio.h> #include<stdlib.h> void main() { int a,b,res; char ch; float div; printf("Enter two numbers:\n"); scanf("%d%d",&a,&b); printf("a.Addition\n b.Subtraction\n c.Multiplication\n d.Division\n e.Remainder\n"); printf("Enter your choice:\n"); scanf("%c",&ch); switch(ch) { case 'a': res=a+b; break; case 'b': res=a-b; break; case 'c': res=a*b; break; case 'd': div=(float)a/b; break; case 'e' : res=a%b; break; default: printf("Wrong choice!!\n"); } printf("Result=%d\n",res); }</pre>

In this program if ch=1 case '1' gets executed and if ch=2, case '2' gets executed and so on.

4. TERNARY OPERATOR OR CONDITIONAL OPERATOR (?:)

- ✓ C Language has an unusual operator, useful for making two way decisions.
- ✓ This operator is combination of two tokens ? and : and takes three operands.
- ✓ This operator is known as ternary operator or conditional operator.

Syntax:

Expression1 ? Expression2 : Expression3

Where,

Expression1 is Condition

Expression2 is Statement Followed if Condition is True

Expression3 is Statement Followed if Condition is False

Meaning of Syntax:

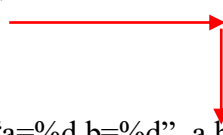
1. Expression1 is nothing but Boolean Condition i.e. it results into either TRUE or FALSE
2. If result of expression1 is TRUE then expression2 is executed
3. Expression1 is said to be TRUE if its result is NON-ZERO
4. If result of expression1 is FALSE then expression3 is executed
5. Expression1 is said to be FALSE if its result is ZERO

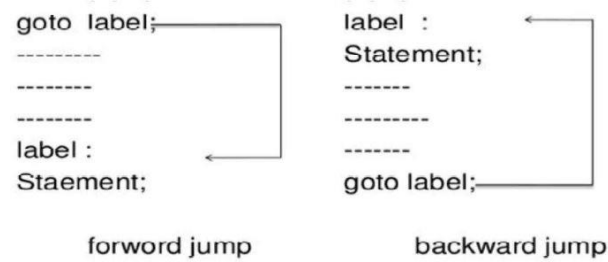
Example: Check whether Number is Odd or Even

```
#include<stdio.h>
void main()
{
    int num;
    printf("Enter the Number : ");
    scanf("%d",&num);
    flag = ((num%2==0)?1:0);
    if(flag==0)
        printf("\nEven");
    else
        printf("\nOdd");
}
```

5. GOTO STATEMENT

- ✓ goto is an unconditional branching statement. The syntax of goto is as follows:

Syntax	Example
<pre>goto label; statement1; statement2; label:</pre>	<pre>void main() { int a=5, b=7; goto end; a=a+1; b=b+1; end: printf("a=%d b=%d", a,b); }</pre> 



- ✓ A label is a valid variable name.
- ✓ Label need not be declared and must be followed by colon.
- ✓ Label should be used along with a statement to which control is transferred.
- ✓ Label can be anywhere in the program either before or after the goto label.
- ✓ Here control jumps to *label* skipping statement1 and statement2 without verifying any condition that is the reason we call it unconditional jumping statement.

CHAPTER 3

Decision Making and Looping

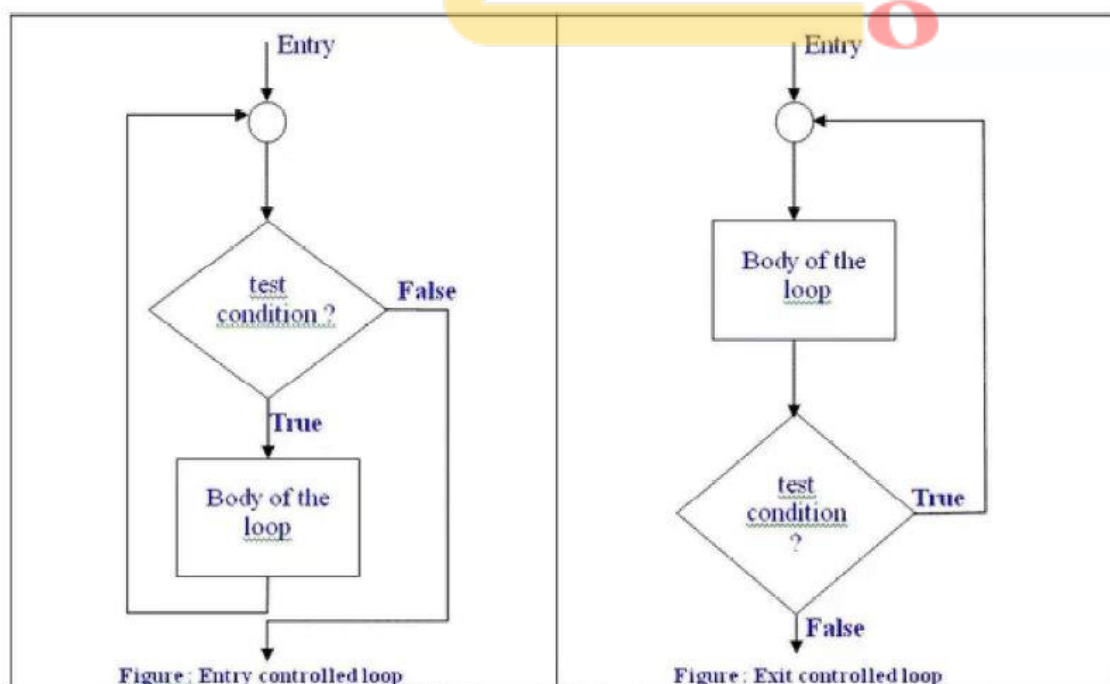
1. INTRODUCTION

Definition of Loop: It is a programming structure used to repeatedly carry out a particular instruction/statement until a condition is true. Each single repetition of the loop is known as an iteration of the loop.

Three important components of any loop are:

1. Initialization (example: `ctr=1, i=0` etc)
2. Test Condition (example: `ctr<=500, i != 0` etc)
3. Updating loop control values (example: `ctr=ctr+1, i=i-1`)

Pre-test and Post-test loops



- ✓ Loops can be classified into two types based on the placement of test-condition.
- ✓ If the test-condition is given in the beginning such loops are called pre-test loops (also known as entry-controlled loops).
- ✓ Otherwise if test condition is given at the end of the loop such loops are termed as post-test loops (or exit controlled loops).

Note Figure 1:

1. Here condition is at the beginning of loop. That is why it is called pre-test loop
2. It is also called as entry controlled loop because condition is tested before entering into the loop.
3. while is a keyword which can be used here.

Note Figure 2:

1. Here condition is at the end of the loop. That is why it is called post-test loop.
2. It is also called as exit controlled loop because condition is tested after body of the loop is executed at least once.
3. do and while are keywords which can be used here.

2. LOOPS IN C

C language provides 3 looping structures namely:

1. while loop
2. do....while loop
3. for loop

The loops may be classified into two general types. Namely:

1. Counter-controlled loop
2. Sentinel-controlled loop

✓ When we know in advance exactly how many times the loop will be executed, we use a counter-controlled loop. A counter controlled loop is sometimes called definite repetition loop.

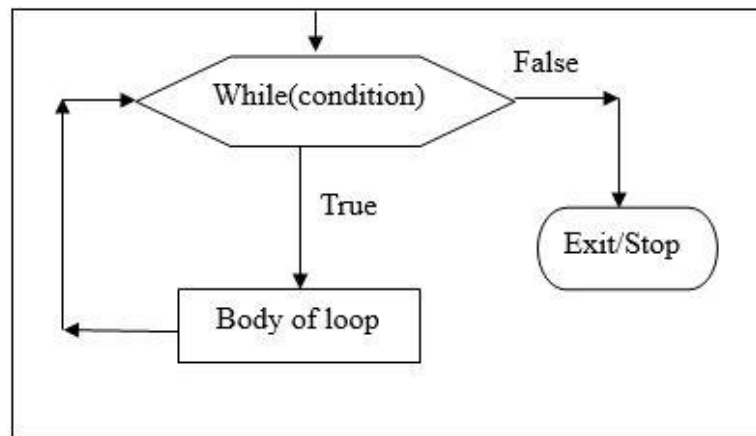
✓ In a sentinel-controlled loop, a special value called a sentinel value is used to change the loop control expression from true to false. A sentinel-controlled loop is often called indefinite repetition loop.

i. while loop:

- ✓ It is a pre-test loop (also known as entry controlled loop). This loop has following **syntax**:

```
while (condition)
{
    statement-block;
}
```

- ✓ In the syntax given above 'while' is a key word and *condition* is at beginning of the loop.
- ✓ If the test condition is true the body of while loop will be executed.
- ✓ After execution of the body, the test condition is once again evaluated and if it is true, the body is executed once again.
- ✓ This process is repeated until condition finally becomes false and control comes out of the body of the loop.

Flowchart:

✓ Here is an example program using while loop for finding sum of 1 to 10.

Example: WAP to find sum of 1 to 5 using while.

```

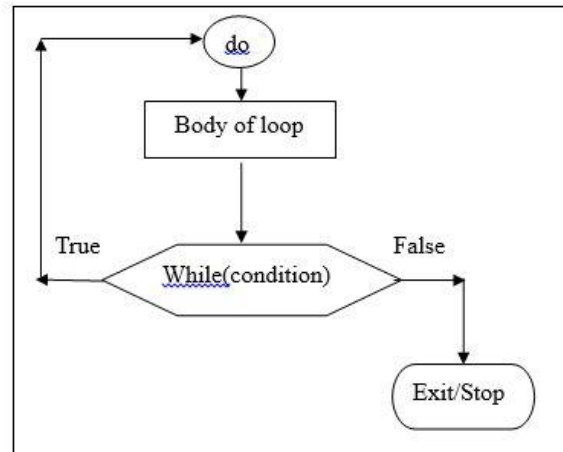
#include<stdio.h>
void main()
{
    int i=1, sum=0;
    while (i<=5)
    {
        sum=sum+i;
        i=i++;
    }
    printf("%d", sum);
}
  
```

ii. do.... while loop: It is a post-test loop (also called exit controlled loop) it has two keywords *do* and *while*. The General syntax:

```

do
{
    statement-block;
} while (condition);
  
```

- ✓ In this loop the body of the loop is executed first and then test condition is evaluated.
- ✓ If the condition is true, then the body of loop will be executed once again. This process continues as long as condition is true.
- ✓ When condition becomes false, the loop will be terminated and control comes out of the loop.

Flowchart:

Example: WAP to find sum of 1 to 5 using do... while

```

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

iii. for loop:

- ✓ It is a pre test loop and also known as entry controlled loop.
- ✓ “for” keyword is used here.
- ✓ Here, the head of the for loop contains all the three components that is initialization, condition and Updation.

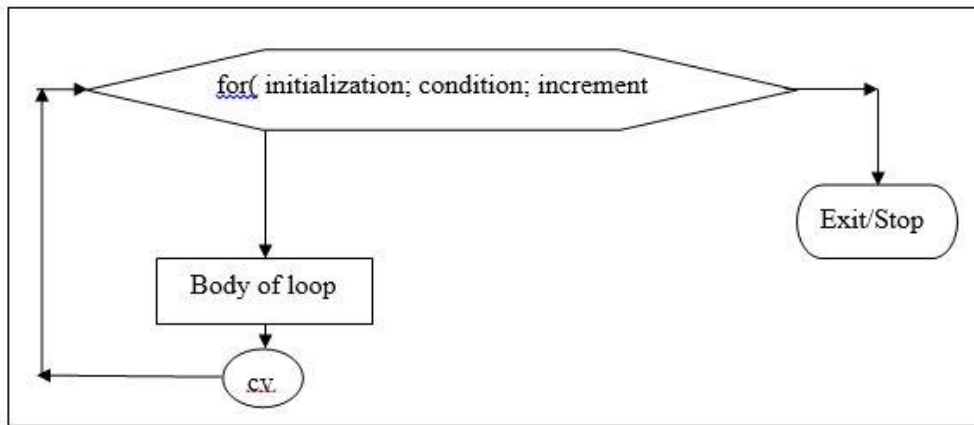
General syntax:

```

for( initialization; test-condition; updation)
{
    Statements;
}
  
```

The execution of for statement is as follows.

1. First the control variable will be initialized. (Example i=1)
2. Next the value of control variable is tested using test condition. (i<=n)
3. If the condition is true the body of the loop is executed else terminated.
4. If loop is executed then control variable will be updated (i++) then, the condition is checked again.

Flowchart:

Example: WAP to find the sum of 10 numbers using for loop.

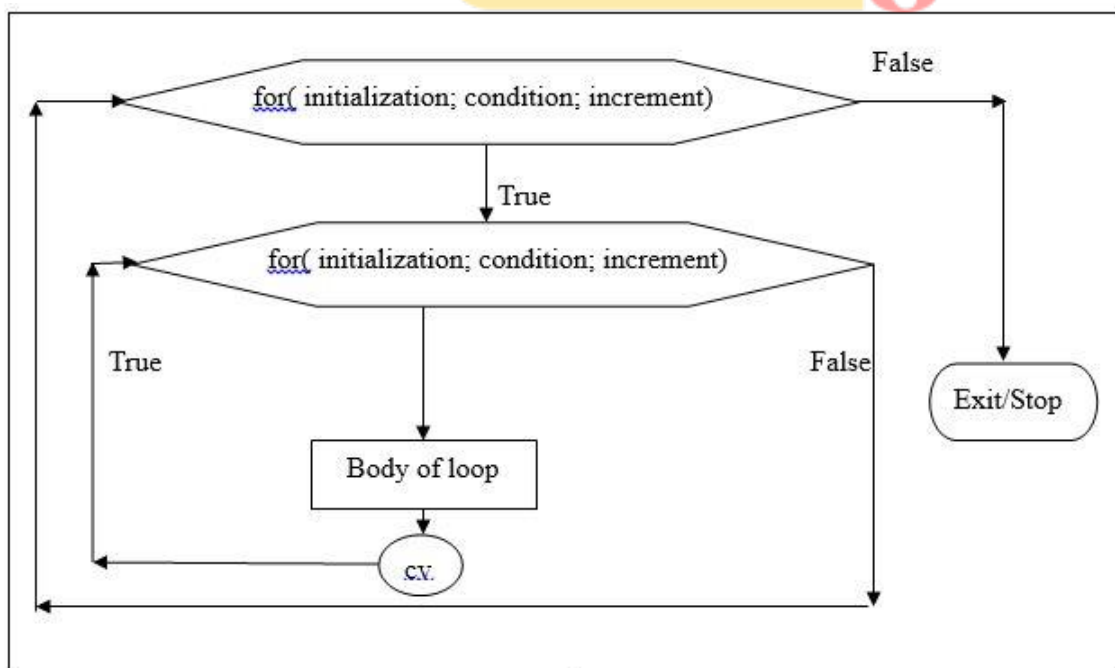
```

#include<stdio.h>
void main( )
{
    int i, sum=0;
    for (i=1; i<=10; i++)
    {
        sum=sum+i;
    }
    printf("%d", sum);
}
  
```

Note: In for loops whether both `i++` or `++i` operations will be treated as pre-increment only.

Nested for loops:

- ✓ A for loop inside a for loop is called Nested for loop.



Example:

```
for(i=0; i<2; i++)
{
    for(j=0; j<2; j++)
    {
        scanf("%d", &a[i][j])
    }
}
```

Note: If updation is not present in loops then, it will execute infinite times.

If initialization is not given then, program prints nothing.

while	do... while
It is a pre test loop.	It is a post test loop.
It is an entry controlled loop.	It is an exit controlled loop.
The condition is at top.	The condition is at bottom.
There is no semi colon at the end of while.	The semi colon is compulsory at the end of while.
It is used when condition is important.	It is used when process is important.
Here, the body of loop gets executed if and only if condition is true.	Here, the body of loop gets executed atleast once even if condition is false.
SYNTAX, FLOWCHART, EXAMPLE (Same as in explanation)	SYNTAX, FLOWCHART, EXAMPLE (Same as in explanation)

3. JUMPS IN LOOPS

✓ **Break and continue:** break and continue are unconditional control construct.

1. Break

- ✓ It terminates the execution of remaining iteration of loop.
- ✓ A break can appear in both switch and looping statements.

Syntax	Flowchart
<pre>while(condition) { Statements; if(condition) break; Statements; }</pre>	<pre>#include<stdio.h> void main() { int i; for(i=1; i<=5; i++) { if(i==3) break; printf("%d", i) } }</pre> <p>OUTPUT 1 2</p>

2. Continue

- ✓ It terminates only the current iteration of the loop.
- ✓ Continue can appear in looping statements.

Syntax	Flowchart
<pre>while(condition) { Statements; if(condition) continue; Statements; }</pre>	<pre>#include<stdio.h> void main() { int i; for(i=1; i<=5; i++) { if(i==3) continue; printf("%d", i); } }</pre> <p>OUTPUT 1 2 4 5</p>

Program 1: Computation of Binomial co-efficient

Problem: Binomial coefficients are used in the study of binomial distributions and readability of multicomponent redundant systems. It is given by:

$$B(m, x) = \binom{m}{x} = \frac{m!}{x!(m-x)!}, m \geq x$$

A table of binomial coefficients is required to determine the binomial coefficient for any set of m and x.

Problem Analysis: The binomial coefficient can be recursively calculated as follows:

$$B(m, 0) = 1$$

$$B(m, x) = B(m, x-1) \left[\frac{m-x+1}{x} \right], X = 1, 2, 3, \dots, m$$

Further,

$$B(0, 0) = 1$$

That is, the binomial coefficient is one when either x is zero or m is zero. The program below prints the table of binomial coefficients for m=10. The program employs one do loop and one while loop.

Program:

```
#include<stdio.h>
#define MAX 10
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:

mx	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

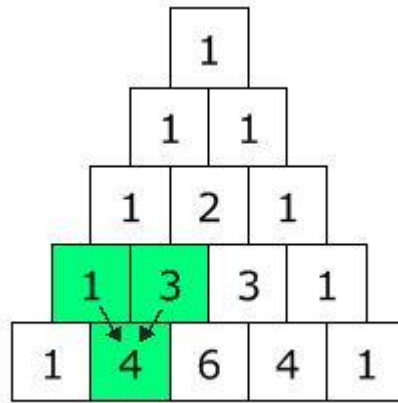
Program 2: Pascal's Triangle

✓ One of the most interesting Number Patterns is Pascal's Triangle (named after *Blaise Pascal*, a famous French Mathematician and Philosopher).

✓ To build the triangle, start with "1" at the top, then continue placing numbers below it in a triangular pattern.

✓ Each number is the numbers directly above it added together.

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$



```
#include <stdio.h>

long factorial(int);

int main()
{
    int i, n, c;

    printf("Enter the number of rows you wish to see in pascal triangle\n");
    scanf("%d",&n);

    for (i = 0; i < n; i++)
    {
        for (c = 0; c <= (n - i - 2); c++)
            printf(" ");

        for (c = 0 ; c <= i; c++)
            printf("%ld ",factorial(i)/(factorial(c)*factorial(i-c)));

        printf("\n");
    }

    return 0;
}

long factorial(int n)
{
    int c;
    long result = 1;

    for (c = 1; c <= n; c++)
        result = result*c;

    return result;
}
```


Program 3: Quadratic Equation

✓ The Quadratic Formula uses the "a", "b", and "c" from " $ax^2 + bx + c$ ", where "a", "b", and "c" are just numbers; they are the "numerical coefficients" of the quadratic equation they've given you to solve.

If determinant > 0,	$\text{root1} = \frac{-b + \sqrt{(b^2 - 4ac)}}{2a}$ $\text{root2} = \frac{-b - \sqrt{(b^2 - 4ac)}}{2a}$
If determinant = 0,	$\text{root1} = \text{root2} = \frac{-b}{2a}$
If determinant < 0,	$\text{root1} = \frac{-b}{2a} + i \frac{\sqrt{-(b^2 - 4ac)}}{2a}$ $\text{root2} = \frac{-b}{2a} - i \frac{\sqrt{-(b^2 - 4ac)}}{2a}$

```
#include<stdio.h>
#include<math.h>
void main()
{
    float a, b, c, root1, root2, rpart, ipart, disc;
    printf("Enter the 3 coefficients:\n");
    scanf("%f%f%f", &a, &b, &c);
    if((a*b*c) == 0)
    {
        printf("Roots cannot be Determined:\n");
        exit(0);
    }
    disc=(b*b) - (4*a*c);
    if(disc == 0)
    {
        printf("Roots are equal\n");
        root1= -b / (2*a);
        root2=root1;
        printf ("root1 = %f \n", root1);
        printf ("root2 = %f \n", root2);
    }
    else if(disc>0)
    {
```

```

printf("Roots are real and distinct\n");
root1= (-b + sqrt(disc)) / (2*a);
root2= (-b - sqrt(disc)) / (2*a);
printf ("root1 = %f \n", root1);
printf ("root2 = %f \n", root2);
}
else
{
printf("Roots are complex\n");
rpart = -b / (2*a);
ipart = (sqrt (-disc)) / (2*a);
printf("root1 = %f + i %f \n", rpart, ipart);
printf("root2 = %f - i %f \n", rpart, ipart);
}
}

```

Output 1	Output 2	Output 3
Enter the 3 coefficients: 1 2 1 Roots are equal Root1= -1.000000 Root2= -1.000000	Enter the 3 coefficients: 2 3 2 Roots are real and equal Root1=-0.500000 Root2=-2.000000	Enter the 3 coefficients: 2 2 2 Roots are complex Root1=-0.500000+i 0.866025 Root2=-0.500000- i 0.866025

VTU SOLVED QUESTIONS

1.	<p>Evaluate:</p> <pre> i=1 1 : if(i >2) { printf("saturday"); i = i-1; goto 1; } printf("sunday"); </pre> <p>Explain your result briefly.</p> <p>Output → Sunday The i is initialized to 1 and in if condition when checked $1 > 2$ returns false. Hence, it doesn't enter inside loop and just prints the Sunday.</p>
----	---

2. Write a C program that prints the following output:

```

    "I am
    an"
    Student'
    Engineering
  
```

```

#include<stdio.h>
void main( )
{
    printf("\nI am\n");
    printf("an\n" \t 'Engineering\n");
    printf("student\n");
}
  
```

3. State the drawback of ladder if-else. Explain how do you resolve with suitable with suitable example.

- ✓ If there are more than one **if** statement and only one else occurs, this situation is called as dangling else problem.
- ✓ This problem is created **when no** matching else exists for every if.

Ex:

```

if(condition1)
{
}
if(condition2)
{
}
if(condition3)
{
}
else
    printf("Dangling Problem\n");
  
```

Solutions:

1. The first approach is to follow simple rule i.e., "Always pair on else to most recent unpaired if the current block."

```

    if (condition1)
    if (condition2)
    if(condition3)
    └─ else
        printf("Dangling Problem\n");
  
```

2. The second approach is a compound statement. Here, we make else to belong for the desired if statement using brackets. (In the below example else belongs to first if statement)

```

    if (condition1)
    {
        if (condition2)
        if (condition3)
        }
    else
  
```

```

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

4. WACP to get the triangle of numbers as a result:

```

1
1 2
1 2 3
1 2 3 4

#include<stdio.h>
void main( )
{
    int n, i, j;
    printf("Enter the number of rows\n");
    scanf("%d",&n);
    for(i=1 ;i<=n; i++)
    {
        for(j=1; j<=i; j++)
        {
            printf("%d\t", j);
        }
        printf("\n");
    }
}

```

5. Write a program in C to display the 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

```

#include<stdio.h>
void main( )
{
    int marks;
    printf("Enter the marks\n");
    scanf("%d", &marks);
    if(marks >= 0 && marks <= 39)
        printf("F");
    else if(marks >= 40 && marks <= 49)
        printf("E");
    else if(marks >= 50 && marks <= 59)
        printf("D");
    else if(marks >= 60 && marks <= 69)
        printf("C");
    else if(marks >= 70 && marks <= 79)
        printf("B");
    else if(marks >= 80 && marks <= 89)
        printf("A");
    else
        printf("O");
}

```

6.	<p>WACP to find the sum of natural numbers from 1 to N using while loop.</p> <pre> #include<stdio.h> void main() { int n, i=1, sum=0; printf("Enter the value of n\n"); scanf("%d", &n); while (i<=n) { sum=sum+i; i=i++; } printf("%d", sum); } </pre>
7.	<p>WAP to find sum of odd numbers from 1 to N using do-while loop.</p> <pre> include<stdio.h> void main() { int n, i=1, sum=0; printf("Enter the value of n\n"); scanf("%d", &n); do { sum=sum+i; i = i + 2; } while (i<=n); printf("Sum of odd numbers is %d", sum); } </pre>
8.	<p>WACP to check whether a given number is even or odd using if-else statement.</p> <pre> include<stdio.h> void main() { int n; printf("Enter a number\n"); scanf("%d", &n); if(n%2==0) printf("The given number is Even\n"); else printf("The given number is Odd\n"); } </pre>