Basics of C Programming and Conditional Strategies

- Basic Structure of ‘C’ Programs
- Variables, Constants, Tokens, Keywords and Identifiers
- Data Types and Declaration of Variables
- I/O Statements, Assignment Statement
- Operators and Expressions
- Precedence of Operators and Evaluation of Expressions
- Decision Making and Branching

Basic Structure of C program

Documentation section
Link section
Definition section
Global declaration section
Main() function section **{** **// body of main, declaration and executable part** **}**
subprogram section **f1() { … }** **f2() { . . .}** **….** **fn() { … }**

The **documentation section** consists of a set of comment lines, giving the name of a program, author name and other details.

Single line comments are specified by symbol **//**

Multiple line comments are specified within **/* …… */**

The **link section** provides instruction to the compiler to link functions from the system library. This section contains a list of header files. The name of header files are stdio.h, conio.h, string.h, math.h, ctype.h, etc.

Ex: #include<stdio.h>

The **definition section** defines all symbolic constants.

Ex: #define pi 3.14

Global declaration consists of global variables. Global variables are variables that are used in more then one function and are declared outside all the functions.

Every C program must have **one main function section**.

This section contains two parts

- Declaration part
- Executable part

The declaration part declares all the variables used in the executable part. There is at least one statement in the executable part. These two parts must appear between the opening and closing braces. The program execution begins at the opening bracers ` { `and ends at the closing braces ' } '. The closing brace of the main function is the logical end of the

program. All the statements in the declaration and other logic always ends with semicolon '**;** '.

The **subprogram section** contains all the user-defined functions. The function body of user-defined functions can appear in any order. All sections are optional except the main function.

Definition: **A Constant is value that cannot be changed during program execution.**

There are two simple ways in C to define constants −

- Using **#define** preprocessor.
- Using **const** keyword.

Using **#define** preprocessor.

```
Syntax:      #define identifier value
```

#define pi 3.15

#define length 12

Using **const** keyword.

```
Syntax:      const datatype variable = value;
```

```
const int LENGTH = 10;
const int  WIDTH = 5;
const char NEWLINE = '\n';
```

Tokens

A token is the smallest element of a program that is meaningful to the compiler. Tokens can be classified as follows:

1. Keywords
2. Identifiers
3. Constants
4. Strings
5. Special Symbols
6. Operators

Keywords : Keywords are pre-defined or reserved words in a programming language.

Each keyword is meant to perform a specific function in a program. Since keywords are referred names for a compiler, they can't be used as variable names because by doing so, we are trying to assign a new meaning to the keyword which is not allowed.

C language supports **32** keywords which are given below:

```
auto        double      int         struct
break       else        long        switch
case        enum        register    typedef
char        extern      return      union
const       float       short       unsigned
continue    for         signed      void
default     goto        sizeof      volatile
do          if          static      while
```

An identifier is nothing but a name assigned to an element in a program. Example, name of a variable, function, etc.

Data types:

C language supports four classes of data types

1. Primary (or fundamental) data types
2. User-defined data types
3. Derived data types and
4. Empty data set. (void)

[1] **Primary (or Fundamental) data type:**

C language supports the following basic data types:

char - one character

int - an integer

float - a single precision floating point number

double -a double precision floating point number

It also offers extended data types such as long int and long double.

Integer data type: C language has three classes of integer storage:

signed type	unsigned type
int	unsigned int
short int	unsigned short int
long int	unsigned long int

A signed integer uses 1 bit for signed and remaining bits for the magnitude of the number. But the unsigned integers use all the bits for the magnitude of the number and are always positive.

Type	Size(bits)	Range	Format Specifier
int or signed int	16	-32,768 to 32,767	%d
unsigned int	16	0 to 65535	%u
short int or signed short int	8	-128 to 127	%d
unsigned short int	8	0 to 255	%u
long int or sign long int	32	-2,147,483,648 to 2,147,483,647	%ld
unsigned long int	32	0 to 4,294,967,295	%lu

Character data type:

A single character can be defined as a character data type.

Type	Size(bits)	Range	Format Specifier
char or signed char	8	-128 to 127	%c
unsigned char	8	0 to 255	%c
String			%s

Floating point data type:

Floating point (or real) numbers are stored in 32 bits with 6 digits of precision. When the accuracy provided by a float number is not sufficient, the type double can be used. A double data type number uses 64 bits giving a precession of 14 digits; these are known as double precision numbers.

Type	Size(bits)	Range	Format Specifier
float	32	3.4E-38 to 3.4E+38	%f
double	64	1.7E-308 to 1.7E+308	%lf
long double	80	3.4E-4932 to 1.1E4932	%Lf

Void

The void type specifies that no value is available.

Variables

Definition: **A Variable is an entity whose value can be changed during program execution.**

A variable can hold only one value at a time during the program execution.

Variable name: Variable name is identifier used to give a name of the variable.

Rules for writing variable names:

A name of variable is consists of letters (a-z or A-Z), digits (0-9) and the underscore (_**)** character with the following rules.

1. They must begin with a letter. Some system permits underscore (_) as the first character.
2. The maximum length of the variable name is 31 characters. Normally, it should not be more than 8 characters.
3. It is case-sensitive. It means lower case and uppercase are different. E.g. the variable name Total and total are different.
4. The variable name should not be a keyword.
5. White space (Blank space) is not allowed.

Examples:

The following are some **valid** variable names:

Empno, name, basic_sal , sum1

The following are some **invalid** variable names with reasons.

Variable Name	**Reason**
student's	Illegal character (')
int	Keyword not allowed
Area circle	White(Blank) space not allowed
4sub	First character should be a letter.
$area	First character should be a letter
a++	+ not allowed
emp-sal	Hyphen (-) not allowed.

Declaration of variables

A variable can be used to store a value of any data type.

The general form variable declaration is

datatype variable1, variable2, ... VariableN **;**

Here, data type is any valid data type of java such as int, float, char etc. and variable1, variable2,…….VariableN are any valid variable names. Two or more variable names are separated by comma. A declaration statement must end with a semicolon.

Examples of declaration of variables:

```
float   price;
int     a, b, sum;
char    gender,choise,name[40];
```

Giving values to variables

There are two ways to give a value to a variable after its declaration.

They are:

- By using an assignment statement
- By using a scanf() statement

Initialization of variables

Variables are initialized (assigned an value) with an equal sign followed by a constant expression.

The general form of initialization is:

variable_name = value;

```
int d = 3, f = 5;              /* initializing d and f. */
double pi = 3.14159;           /* declares an approximation of pi. */
char x = 'x';                  /* the variable x has the value 'x'. */
int a=b=c=0;                   /* initializes a, b and c with value zero */
```

Variables can be initialized (assigned an initial value) in their declaration. The initializer consists of an equal sign followed by a constant expression as follows:

type variable_name = value;

I/O Statement:

printf() **(output statement)**

The printf() function is used to display a text or value of a variable or both on the computer screen.

Syntax:

prinf("Control String",variable1,variable2,.....);

Control string consist of 3 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.

Control string indicates how many arguments follow and what their types are.

Arguments **arg1,arg2,...,argn** are variables whose values are formatted and printed according to the specifications of the control string.

The argument should match in number, order and type with format specifications.

Format specification form: %w.p type-specifier

Where **w** specifies the field width for display and **p** instructs that

// Program to calculate simple interest.

```
void main ( )
{
                    int p, n ;
                    float  r, si ;
                    printf ( "Enter values of p, n, r" ) ;
                    scanf ( "%d %d %f", &p, &n, &r ) ;
                    si = p * n * r / 100 ;
                     printf ( " %f " , si ) ;
}
```

	Statements (example)	**Output**
1.	printf("Good Morning")	Good Morning
2.	printf("Good \nMorning")	Good Morning
3.	int a=5 ; printf(" %d ",a) ;	5
4.	int a=5 ; printf(" Value of A = %d ",a) ;	Value of A = 5
5.	int a=5 ; printf(" Square of %d is %d ",a , a*a) ;	Square of 5 is 25

Note: \n is newline character

Reading data from keyboard (scanf() or input statement):

The **scanf()** is a function of header file stdio.h and is used to read the input from the keyboard as any valid data type.

Syntax

scanf("control string",&variable1, &variable 2,...);

Control string contains the format in which data is to be entered.

The ampersand symbol & before each variable name is an operator that specifies the variable name's address.

Example:

scanf ("%d", &number);

- When this statement is encountered by the computer, the execution stops and waits for the value of the variable **number** to be typed in.
- %d specifies that an integer value is to be read from the terminal.
- In the above example once the number is typed in, the value is assigned to variable **number.**

Input integer numbers: **%wd**

w is integer number that specifies field width of the number to be read and **d** as data type character.

```
scanf("%3d",&number);  //accepts number only upto 999
```

Input real numbers: (%wf)

```
scanf("%f",&number);
```

Input character:

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

Input string: (%ws)

```
scanf("%s",number);
```

Example : **//To print a square of a number.**

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

void   main()
{
        int   a, b;
        printf("Enter any number :");
        scanf("%d",&a);
        b = a*a;
        printf("Square of a given number %d is %d",a,b);
}
```

Output: Enter any number : 4

Square of a given number 4 is 16

The following symbols can be used in scanf() function to read different data type of a C languages.

Code	Meaning
%c	Read Single Character
%d	Read Decimal integer

%f	Read Floating point Value
%s	Read String a string
%u	Read Unsigned integer
%x	Read Hexadecimal
%ld	Read long integer

Assignment statement

A value can be assigned to a variable is by using assignment operator = . The value of the variable on the left of the 'equal sign' is set to the expression on the right.

The general form of this is:

variableName = constant ;

For example:

balance = 50.25 ;

sum=0;

ans = 'Y';

- Multiple assignments in one line are valid.
  - balance = 50.25 , sum=0; ans = 'Y';
- Assigning single value to multiple variables by a single statement:

  a = b = c = 0;

  Here, all three variables a, b, and c are initialized with zero.
- Assigning a value to a variable at the time of its declaration. The general form of this is:

  data type variableName = value ;

  For example:

  int i=0;

  char ans = 'Y';

  The process of giving initial value to variable at the time of its declaration is known as the ***initialization***.
- The automatic variables that are not initialized at declaration are automatically initialized by its garbage value.

  For example:

  int a, b=5; //Here, variable 'a' is initialize with garbage.
- External and static variables are initialized to ***zero*** by default.
  - static int a, b=5; //Here, variable 'a' is initialize with garbage.

- More than one variable can be initialized in one statement using multiple assignment operators.
- For example:

```
int   a, b,c =0;
        //initializes a, b and c  with zero.
```

Operator:

Definition: **An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations"**

C operators can be classified into a number of categories. They include

Sr.no	Operator Name	Operator Symbols
1.	**Arithmetic Operator**	**+ - * / %**
2.	**Relational Operator**	**>** **<** **>=** **<=** **!=** **==**
3.	**Logical Operator**	**&& || !**
4.	**Assignment Operator**	**=**
5.	**Increment and Decrement Operator**	**++** **--**
6.	**Conditional Operator**	**?:**

Arithmetic Operator:

The C language has both **unary** and **binary** arithmetic operators. Unary operators are those which operate on a single operand while binary operators are operate on two operands.

The arithmetic operators and their meaning are shown below:

Operator	Meaning
+	Addition or unary plus

-	Subtraction or unary minus
x	Multiplication
/	Division
%	Modulo division

Integer Arithmetic: When both the operands in a single arithmetic expression (such as a + b) are integers, the expression is called integer expression, and the operation is called integer arithmetic. Integer arithmetic always results an integer value.

Example: If a & b are integer & its value is a = 9 and b=5 then we have the following results:

a -b = 4

a + b = 14

a * b = 45

a / b = 1 (decimal part truncated quotient)

a % b = 4 (remainder).

Example: During division if one of operator is negative value result value depends on machine,

-6/7 (Either 0 or -1 depends on machine)

Example: During division if both operators are positive/negative value result value is always positive,

6/7 = 0

-6/-7 = 0

Example: During modulo division , sign of result is always sign of first operands.

-14 % 3 = - 2

-14 % -3 = - 2

14 % -3 = 2

Real Arithmetic: An arithmetic operation involving only real operands is called real arithmetic.

Example: If a, b and c are floats then we will have

a = 6.0/7.0 (a becomes 0.857143)

b = 1.0/3.0 (b becomes 0.33333)

c = -2.0/3.0 (c becomes 0.66667)

The operator % can not be used with real operand.

Mixed-Mode Arithmetic: When one of the operand is real and the other is integer the expression is called Mixed-mode arithmetic expression.

If either operand is of the real type, then only the real operation is performed and the result is always a real number.

Thus 15/10.0 = 1.5

Arithmetic Instruction	Result	Arithmetic Instruction	Result
k = 2 / 9	0	a = 2 / 9	0.0
k = 2.0 / 9	0	a = 2.0 / 9	0.2222
k = 2 / 9.0	0	a = 2 / 9.0	0.2222
k = 2.0 / 9.0	0	a = 2.0 / 9.0	0.2222
k = 9 / 2	4	a = 9 / 2	4.0
k = 9.0 / 2	4	a = 9.0 / 2	4.5
k = 9 / 2.0	4	a = 9 / 2.0	4.5
k = 9.0 / 2.0	4	a = 9.0 / 2.0	4.5

2. Relational Operator

A relation operator is used to make comparisons between two expressions.

C language supports six relational operators.

These operators and their meaning are shown below.

Operator	Meaning
<	Is less than
<=	Is less than or equal to
>	Is greater than
>=	Is greater than or equal to
==	Is equal to
!=	Is not equal to

An expression containing a relational operator is termed as relational expressions. The value of relational expression is either one (true) or zero (false).

Example:

A = 10;

B = 20;

(A < B) True

(B < A) False

3. Logical Operator

An expressions combining two or more relational expressions are known as logical expressions or compound relational expressions. C language has the following three logical operators.

Operator	Meaning
&&	Logical AND
||	Logical OR
!	Logical NOT

Truth Table

operand 1	operand 2	op1 && op2	op1 || op2
Non - Zero	Non - Zero	1	1
Non - Zero	Zero	0	1
Zero	Non - Zero	0	1
Zero	Zero	0	0

The value of logical expression is either one (true) or zero (false).

Example on Logical AND:

A =5; B =10; c = 15;

The condition (A > B && A< C)

(False || True) Result : False

The condition (A < B && A< C)

(True || True) Result : True

Example on Logical OR:

A =5; B =10; c = 15;

The condition (A > B || A< C)

(False || False) Result : False

The condition (A < B || A > C)

(True || False) Result : True

Example on Logical NOT:

The condition !(A <B) evaluates false and !(A> B)evaluates true

4. **Assignment Operator(=)**

Assignment operators are used to assign the result of an expression to a variable.

The general form of assignment operator is

variable = expression;

C has a set of "Shorthand" assignment operators of the form

V op= exp ;

Where, V is a variable, exp is an expression and op is a binary arithmetic operator.

The operator op= is known as the shorthand assignment operator.

The assignment V op= exp; is equivalent to V = V op (exp);

Example:

Statement with simple assignment operator	***Statement with shorthand assignment operator***
a = a + 1;	a += 1;
b = b - 1;	b-= 1;
a = a + (n + 3);	a += n + 3;
a = a * b;	a *= b;
a = a / b;	a /= b;
a = a / (n +1) ;	a /= n+1 ;
a = a % b;	a %= b;

Advantages of short hand assignment operator:

1. What appears on the left hand side need not be repeated and therefore it becomes easier to write.
2. The statement is more concise and easier to read.
3. The statement is more efficient

5. **Increment and Decrement Operator**

The C language uses two operators for incrementing and decrementing variables.

Increment Operator (++)

- A symbol for an increment operator is ++.
- It increases the value of a variable by 1.
- It is a unary operator.

- It can be prefix or postfix.
- When we used a prefix in an expression, the value of a variable is incremented before used in expression. But when used as a postfix, its value is first used in expression and then the value is incremented.

  Example:

Prefix	**Postfix**

int x = 0, y = 5:
x = ++y;
printf("x=%d, y=%d",x,y);
	int x = 0, y = 5:
	x = y++;
	printf("x=%d, y=%d",x,y);

output: x = 6, y = 6	output: a = 5, y = 6

Decrement Operator : (--)

- A symbol for a decrement operator is --.
- It decreases the value of a variable by 1.
- It is a unary operator.
- It can be prefix or postfix.
- When we used a prefix in an expression, the value of a variable is decremented before used in expression. But when used as a postfix, its value is first used in expression and then the value is decremented.

  Example:

prefix	**postfix**

int x = 0, y = 5:
x = --y;
printf("x=%d, y=%d",x,y);
	int x = 0, y = 5:
	x = y- -;
	printf("x=%d, y=%d",x,y);

output: x = 4, y = 4	output: x = 5, y = 4

6. Conditional Operator

It is also known as **Ternary Operator**.

The general form of ternary operator is

exp 1 ? exp2 : exp3 ;

Where exp1, exp2, exp3 are expressions.

The operator **? :** Works as follows :

- exp1 is evaluated first.
- If it is non-zero (true), then the expression exp2 is evaluated and becomes the value of the expression.
- If it is zero (false), then the expression exp3 is evaluated and becomes the value of the expression.

Example:
```
a = 10;
b = 15;
x = (a>b) ? a : b;
```

In this example, x will be assigned the value of b. This can be achieved using the if...else statements as follows.

```
if ( a > b)
        x=a;
else
        x = b;
```

Example : Find whether given number is odd or even

```
#include<stdio.h>
void main()
{       int a;
        printf('\n Enter value of a : ");
        scanf("%d" ,&a);
        ( a%2 ==0 ) ? printf ("%d is even ",a)  :  printf ("%d is odd",a);
}
```

Input:

```
Enter value of a : 23
```

Output:

```
23 is odd
```

Expressions and Manipulation

An arithmetic expression is a combination of variables, constants, and operators arranged as per the syntax of the language.

Example of C expressions

```
a * b –c
(m+n)*(x+y)
A * b / c
3 * x * x + 2 * x + 1
x  / y + c
```

Evaluation of Expressions

Expressions are evaluated using an assignment statement of the form:

Variable= expression;

When the statement is encountered, the expression is evaluated first and the result then replaces the previous value of the variable on the left-hand side.

Example:

```
x=a * b – c;
y = b / c * a;
```

Precedence of arithmetic operators:

An arithmetic expression without parentheses will be evaluated from **left to right** using the rules of precedence of operator.

High priority: * / %

Low priority: + _

For example:

X=a-b/3+c*2-1

When a=9, b=12,c=3 the statement becomes

X=9-12/3+3*2-1

Is evaluated as follows:

First pass

Step 1: x=9-4+3*2-1

Step 2: x=9-4+6-1

Second pass

Step 3 : x=5+6-1

Step 4: x=11-1

Step 5: x=10

The order of evaluation can be changed by introducing parentheses into an expression.

For example:

9-12/(3+3)*(2-1)

The expressions within parentheses assume highest priority.

First pass

Step 1: 9-12/6*(2-1)

Step 2: 9-12/6*1

Second pass

Step 3: 9-2*1

Step 4: 9-2

Third pass

Step 5: 7

Parenthesis may be nested, and in such cases, evaluation of the expression will proceed outward from the innermost set of parentheses.

Rules for evaluation of expression

- First parenthesized sub expression from left to right are evaluated.
- If parentheses are nested, the evaluation begins with the innermost sub-expression.
- The precedence rule is applied in determining the order of application of operators in evaluating sub-expressions.
- The associativity rule is applied when two or more operators of the same precedence level appear in a sub-expression.
- Arithmetic expressions are evaluated from left to right using the rules of precedence.
- When parentheses are used, the expressions within parentheses assume highest priority.

Control strategies -Decision making

In number of situations where we may have to change the order of execution of statements based on certain conditions, or repeat a group of statements until certain specified conditions are met.

- Decision making statements used in 'c' :
    1. If statement
    2. Switch statement
    3. Conditional operator
    4. Goto statement

These statements are known as ***decision-making statements***. Since these statements control flow of execution, they are also known as control statements.

- **Looping statements**
    1. Do while statement
    2. While statement
    3. For statement

These statements are known as ***looping statements***. Since a sequence of statements are executed until some conditions for termination of the loop are satisfied.

Conditions

Branching statement: It alters (change) sequential execution of program statements. They are

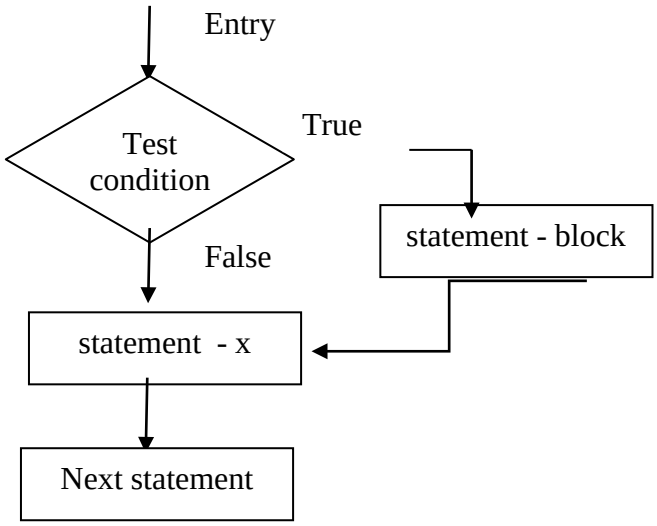
1. if statement
2. if…else statement
3. Nested if statement
4. The else if ladder
5. switch statement

if statement: It is decision making statement. It is used to control the sequence of the execution of statements.

Syntax:

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

Description: statement(s) are executed only if condition is true (nonzero).Flowchart of simple if control:



Example: To find whether given number is positive, negative or zero.

```
void main()
{
   int n;
        printf("\n Enter Number : ");
        scanf("%d",&n);
        if ( n > 0 )
                printf ("Positive Number…");
        if ( n> 0 )
                printf ("Negative Number…");
        if ( n == 0 )
                printf ("Zero Number…");
}
```

if…else statement:

Syntax:

```
        if ( condition )
        {
                statement-1;
        }
        else
              {
                statement-2;
                }
```

statement x;

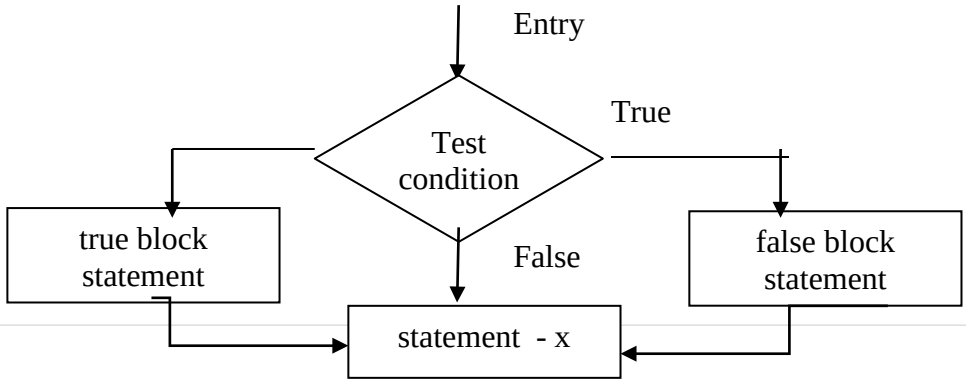
Description:

When condition is true (nonzero), then if-part is executed and control passes to the next statement following the *if* statement. Otherwise the else-part is executed and control passes to the next statement.

Initially condition is evaluated.

If condition is true then statement-1 and then statement-x is executed, Otherwise statement-2 is executed and then statement-x is executed.

Flowchart of if…else control:



Example: Write a C program to find maximum number from two integer numbers.

```
void main()
{
   int a, b;
        printf("Enter first number : ");
        scanf("%d",&a);
        printf("Enter second number : ");
        scanf("%d",&b);
        if ( a > b )
                printf ("\n %d is maximum",a);
        else
```

```
}
        printf ("\n %d is maximum",b);
```

Nested if...else statement:

One if…else statement written inside another if…else statement is known as nested if ... else statement.

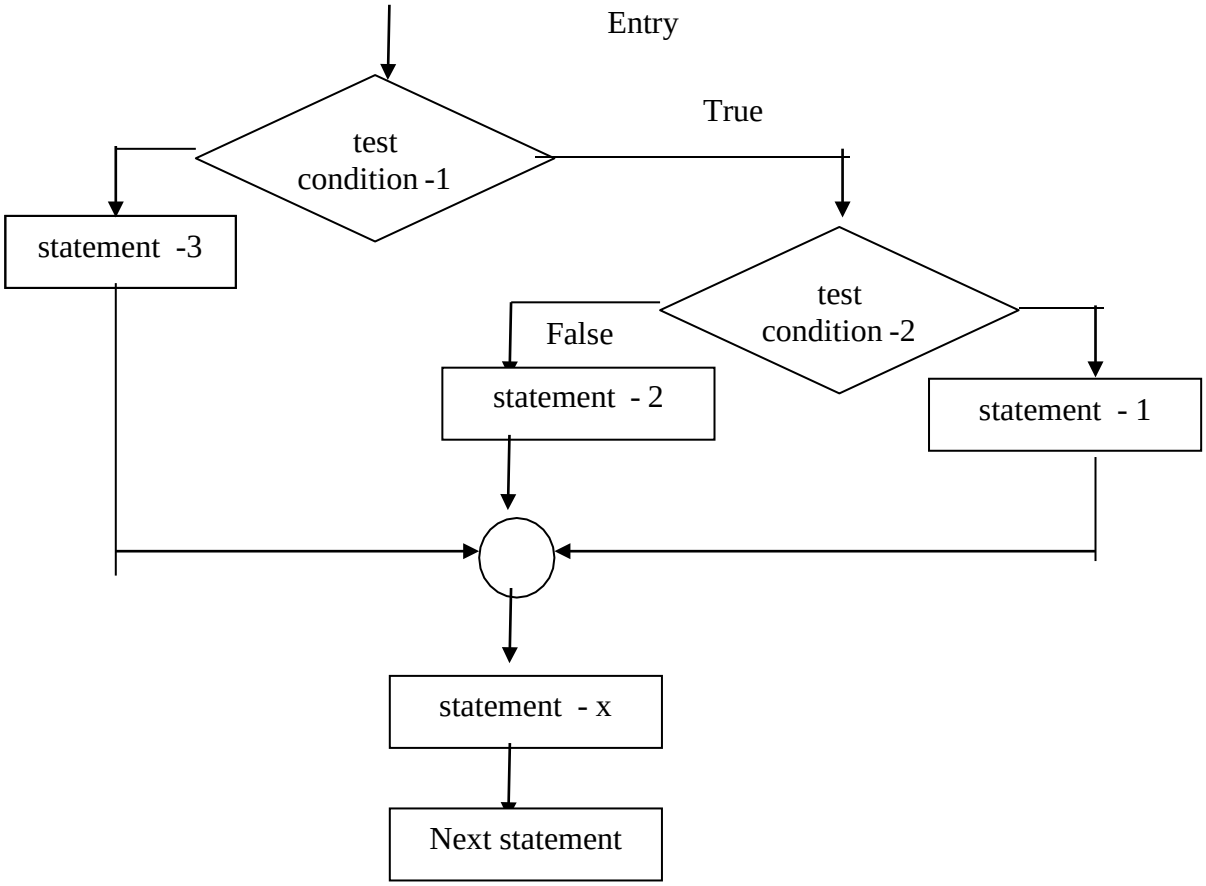
Syntax:

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

If condition-1 is true then statement1 block is executed and else-part is skipped. The control passes to the condition -2. If condition-2 is true then statement-2 block is executed and else-part is skipped.

If condition-1 is false then statement1 block is skipped and else-part is executed.

Flowchart of Nested if…else control:



Example: Find maximum number from three integer numbers.

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int a, b, c;
        clrscr();
        printf("Enter first number : ");
        scanf("%d",&a);
        printf("Enter second number : ");
        scanf("%d",&b);
        printf("Enter third number : ");
        scanf("%d",&c);
```

```
        if ( a > b )
        {
                if ( a> c )
                        printf ("\n %d is maximum",a);
                else
                        printf ("\n %d is maximum",c);
        }
        else
        {
                if ( c > b )
                        printf ("\n %d is maximum",c);
                else
                        printf ("\n %d is maximum",b);
        }
getch();
}
```

switch statement: It is a multi-way decision making statement the allows to select one of the many alternatives.

Syntax:

```
        switch ( expression )
        {
                case value1 :
                                block-1;
                                break;
                case value2 :
                            block-2;
                                break;
                                :
                                :
                  default :  block;
                                break;
        }
        statement-x;
```

Description:

switch statement test the value of given expression with list of case values. When a matching case is found , the block of statements associated with that case are executed.

- Expression is an integer or character expression.
- Value1,value2…are constants or constant expressions known as case labels.
- Each case label must be unique.
- block-1, block-2…can consist if zero or more statements.
- There is no need to put braces around the blocks.
- Each case label must end with colon (:)

Break statement

- Break statement at the end of each block indicates end of that particular case and exits form switch statement and transfers control to statement-x following switch statement;

Default case

- Default case is optional case.
- If it is present it will be executed when no matching case is found.
- If it is not present and no matching case is found, no action takes place and control is transferred to statement-x following switch statement;

Rules for switch case:

1. The switch expression must be integral type.
2. Case labels must be constants or constant expressions.
3. Case labels must be unique. No two case label can have same. value.
4. Break statement transfers control out of switch statement.
5. Break statement is optional. Two or more case lables may belong to the same statements.
6. Default case is optional case. If it is present it will be executed when expression does not find matching case.
7. There can be only one default label.
8. Default label can be placed anywhere but usually at the end.
9. It is permitted to nest switch statements.

Example :

```
switch(a)
{
        case 1 :
        case 2 :
        case 3 : printf("\n Case One or Two or Three");
                 break;
default : printf("\n Default case");
}
```

Example:

```
#include<stdio.h>
#include<conio.h>
void main()
{
        int   a;
        clrscr();
        printf("\n Enter Choise :");
        scanf("%d",  &a);
        switch(a)
        {
                case 1 : printf("\n One");
                          break;
                case 2 : printf("\n Two");
                         break;
                case 3 : printf("\n Three");
                         break;
                case 4 : printf("\n Four");
                         break;
        default : printf("\n Invalid choise");
        }
        getch();
}
```

Output :

```
Enter Choice : 4
Four
```

For Practical knowledge purpose

Output of integer number

Format specification for printing an integer number is:

%wd

Where **w** specifies minimum field width for the output. if the number is greater than specified field width, it will be printed in full.

d specifies that the value to be printed is an integer.

Default : right-justified in the given field width.

To print left-justified place minus sign directly after ' % ' character.

It is also possible to pad with zeros the leading blanks by placing a zero(0) before field width specifier.

Format	Output
printf("%d",9876)	**9 8 7 6**
printf("%6d",9876)	**9 8 7 6**
printf("%2d",9876)	**9 8 7 6**
printf("%-6d",9876)	**9 8 7 6**
printf("%06d",9876)	**0 0 9 8 7 6**

Output of real number

Format specification for printing an real number is:

%w.p f

Integer **w** indicates minimum number of positions that are to be used for display of value and integer **p** indicates the number of digits to be displayed after decimal point.

When displayed, is rounded to **p** decimal places and printed right-justified in field of **w** columns

Default precision is 6 decimal places.

Y=98.7654

Format	**Output**
printf("%7.4f",y)	**9 8 . 7 6 5 4**
printf("%7.2f",y)	**9 8 . 7 6 5 4**
printf("%-7.2f",y)	**9 8 . 7 7**
printf("%f",y)	**9 8 . 7 6 5 4**
printf("%10.2e",y)	**9 . 8 8 e + 0 1**
printf("%11.4e",-y)	**- 9 . 8 7 6 5 e + 0 1**
printf("%-10.2e",y)	**9 . 8 8 e + 0 1**
printf("%e",y)	**9 . 8 7 6 5 4 0 e + 0 1**

Printing of a single character:

%wc

The character will be displayed right-justified in the field of **w** columns.

We can make left justified by placing a minus sign before integer **w**. The default value for **w is 1**.

Printing of string:

Format specification for outputting strings:

%w.ps

Where **w** specifies the field width for display and **p** instructs that only the first p characters of the string are to be displayed. The display is **right-justified.**

S= NEW DELHI 110001

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**	**0**	**1**
%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**				