CPROGRAMINING

BSC(IT) PART-II

UDAY KUMAR SINGH
DEPT. OF BCA, BD COLLEGE,PATNA

INTRODUCTION

C is mother of all programming languages. It is one of most popular programming languages. Due to its simplicity, structure and high level.

Initially “c” was based compliable with unix operating system , or we can say that it is develop for unix platform. But now these days , its compiler is available for almost all operating system .

History of c:

First program in c

```
# include <stdio.h>
Void main ( )
{
 Printf("welcome in c");
 Printf("How are you");
 Printf("Bye");
}
```

[# - pre-processor]
[<stdio.h> - header file]
[Void - Return type]

*Open the c program and type the above program in the editor .
*Save the above programme with any primary name but extension must be '.c'.
*After saving the file. Compile it using compile menu or Alt+79.
*After successful compilation we get another file with extension ".obj".

First.c —— Compile ——> First .obj

Source code object code

After compilation linking will be done . After linking we get another file with extension ".exe" .

Now Run the programme using Run Menu or Ctrl+79.

Common points related to 'c'.

1. Every programme in c must have a main function and it must be only one.
2. During the compilation , compiler will check the programme from top to bottom and produce syntax error if it is found.
3. During the execution programme will execute form main unction. Every programme must be terminated at the main function.
4. Every function must have a functionality. We write the functionality in the pair of curly bracket (brashes) called function block.
5. 'c' provides a no of libarary unction which can be used in the appropriate header file because the header file contains the definatio of related function.
6. To include the header file we use file inclusion direct followed by pre –processor symbol.
7. pre –processor indicate the statement which is executed before the programme execution.
8. Return types means the data type of value which will returned by data function.
9. If a function will not return any value we write void as a return type . In 'c' return type is void by default.
10. Every statement of 'c' must be terminated by semi column.
11. 'c' is case native programming language.

Formated output :

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

```
        Main ( )
{
    Clrscr( );
Printf("NSIT");
Printf("Tutorials of BCA&MCA");
Printf("\n kulharia complex");
Printf("\n Ashok RAj Path \n ");
Printf("Patna \n");
Printf("\n\n Pin 800004");
     Getch( );
}
```

Clrscr :- This function will clear the output and set the cursor at the top of screen .
\n (slace n) :- Black \n is use to create a new line but it must be inside the double code.
getch :- getch is an in pat function accept one key stroke and normally use to halt the output in output mode , until we process any key.

<u>NOTE</u> : - In 'c' some common feature of <stdio.h> and <conio.h>are include by default, hence we can use these common features without including header file.

'c' Tokens

All smallest individual components or unit of 'c' is called 'c' Tokens. These six are called 'c' Tokens :-
1. Variable
2. Constant
3. Key words
4. Operator
5. String
6. Special symbols

<u>Variable</u>

Variable is a memory location which has a name , a data type and a specific range and use to store a value which can be frequently modify during the programme execution.

According to data type variable can be categorized in three major categories.

1. Integer
2. Float
3. Character

1. **<u>Integer variable</u>** : - Integer variable is use to store a numeric value without any fractional part or decimal places.

<u>It has also three types.</u>

Name	**Memory size**	**Range**
Int	2 byte	-32768 to +32767
Short	1 byte or 2 byte	-128 or equal to +127
Long	4 byte	-2147483648 to +2147483647

<u>Note</u> :-

Normally , short variable gets one byte memory alocation and store -128 to +127 value. But some recent compiler also alocates 2 byte memory for short variable . In this cause the range of short variable is equal to range of int variable.

2. float variable :- Float variable is use to store the floating point value i.e. decimal value. It has also three types :-

Name	Memory size	Precesion value
Float	4 byte	6 digits
Double	8 byte	15 digits
long double	10 byte	19 digits

2. **character variable** :- Character variable is use to store a single character . It get one byte memory space and keyword is used "char".

NOTE : -

The memory alocation for integer variable ,some times depends on the machine configuration. Normaly 32 bits machie alocates 2 bits memory for int . But its higher machine may be alocate 4 bite memory for int.

Note : -

'c' follows the signed and usigned concept for integer variable. By default every decleared variable is signed . Although we can use a keywords signed. If a variable is decleared as signed , it can be hold positive value as well as negative value.

Ex. Int a
Signed int a
a = + 25 √
a = - 25 √

But if we want to respect a variable to hold only positive value , we can decleared it unsigned using a key word unsigned.

Ex. Unsigned int a
a = 25 √
a = - 25 ×

The another advantage of using unsigned variable is that we can get double range in positive side . It means the range of unsigned int becomes 0 to 65535.

Constant

Constant is also a memory allocation use to store a value but it can't be modify during the programme execution.

There are two types of constant: -

(i) Variable constant
(ii) Symbolic constant

(i) Variable constant: - It is a variable with constant nature**.** To decleared a variable constant, we use a key word cost.

Ex. Const int a = 50
a = a +10 ×

(ii) Symbolic constant: - It is a more flexible and powerful than variable constant. To create a symbolic constant , we use " # define" statement. To declared a symbolic constant.

The general format is
define name value

Ex:- 1. # define Max 100
2. # define pi 3.14
3. # define And &&
4. # define or / /

5. # define clear clrscr( )

This mechanism is also called ‘c’ macro.

Identifier:

An identifier is use to give the name of variable, constant or other user defined object to identify. An identifier may be combination of upper case and lower case letters, digits and a special symbol under scope. But first letter can’t be digit.

EX:-

int a √
int chandan √
Int chandan 5 √
int 5 chandan × (*)
int chandan kumar × (**)
Int chandan **.** Kumar × (***)
int chandan – kumar √

* Because first letter can’t be digit.
* * Because we can’t use space in between any two letter or symbols .
* * * We can use a special symbol under scope in identifier but can’t use any other symbols .

Keywords:

keywords are the reserved words of any programming language which has a cirtains meaning in that programming language .

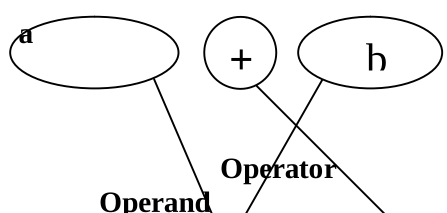
We can use any key word in the programme according to our need but we can’t redtined or modified it .

There are 32 keywords in c for programming and all are in small letters.

1. break
2. case
3. const
4.

Operator:

Operator are mode or symbol to do some operation on the operand.



Basically there are eight types of operator in ‘c’.

1. Arithmetic operator.
2. Assignment operator.
3. Increment – decrement operator.
4. Relational or comparison operator.
5. Logical operator.
6. Conditional operator.
7. Special operator.
8. Bit wise operator.

1. **<u>Arithmetic operator</u> :-** Arithmetic operator is used to perform some basic arithmetic operation like addition , subtraction etc. following five are knows as Arithmetic operation.

+	→	Add
−	→	Subtraction
×	→	multiplication
/	→	Div
÷	→	Remainder

On the basis of operant ,operator can be categories in two types.

(i) Uniary operator

(ii) Binary operator

(i) <u>Uniary operator</u> :- The operator which needs only one operant for operation it is called binary operator.

EX. +, −, (+ +,−−) - Uniary

(ii) **<u>Binary operator</u> :-** The operator which needs two operent for operation , it is called binary operater.

2. <u>Assignment operator</u> :- Assignement operater is used to give a new value to the variable or change the existing value of variable = is known as assignment operator.

We can also combined the Arithmetic operator and assignment operator to get anew operator arithmetic assignmet operator also called short hand operator.

Following are known as short hand operator.

+ =

- =

* =

/ =

÷ =

2. **<u>Increment / Decrement operator: -</u>**

When we want to increase the value of variable and decrease the value of variable with one, we use increment / decrement operator.

+ +	→	increment operator
− −	→	Decrement operator

Ex. a = 10 b = 10

a + + = 11 b − − = 19

a + + = 12 b − − = 18

We use increment / Decrement operator with two different ways.

a) Postfix b) prefix

a) <u>Postfix:</u> - when the operator appears after the variable it is called postfix.

Ex. a + +

When we use postfix the value of variable will be changed after the termination of expression.

Ex. a = 5

Ex. b = a + +

Print (b) → 5

Print (a) → 6

(b) <u>Prefix:</u> - when the operator appears before the variable it is called prefix.

Ex. + + a

When we use prefix the value of variable be changed in the current expression.

Ex. a = 5, `b = + + a

Print (b) → 6 & Print (a) → 6

Ex. main ( )
{
Int a = 5
Printf (“%d”, a); 5
Printf (“%d”, a++); 5
Printf (“%d”, a); 6
Printf (“%d”, ++a); 7
Printf (“%d”, a); 7
Printf (“%d”, a++); 7
Printf (“%d”, ++a); 9
Printf (“%d”, a); 9

3. **Relational operator : -**

It is use to chek the relation between two variable and return a bullion value. Bullion value means true or false.

‘c’ does not understand true or false for bullion value . it is consider 0 or false value and every non-zero or true value . by default it is one.

These are relational operator.

>	< =
> =	= =
<	! = (not equal to)

Ex. a = 5, b= 10,

(a < b) - 1 - True

(a > b) - 0 - False

(a = 5) - 1 - True

(b! =10) - 0 - False

Ex. main ( )
{
int a = 5;
int b = 10;
printf(“%d”,(a<b)); → 1
printf(“%d”,(a<=b)); → 1
printf(“%d”,(a>b)); → 0
printf(“%d”,(a>=10)); → 1
printf(“%d”,(a= =5)); → 1
printf(“%d”,(b! = 10)); → 0
printf(“%d”,(a! = 50)); → 1
}

4. **Logical operator : -**

It is use to check the relation between two or, more expression and return the bullion value.

These three are called logical operator.

(a) && → logical And
(b) !! → logical OR

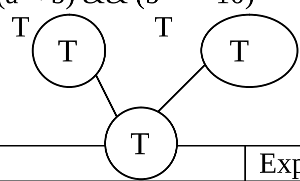
(c) ! → logical NOT

(a) **logical And (&&) : -** logical And operator the given expression true.

Ex. a = 5

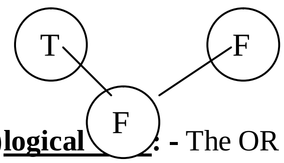
b = 10

(i) (a < b) && (b = = 10)



Exp 1	Exp 2	Result
T	T	T
T	F	F
F	T	F
F	F	F

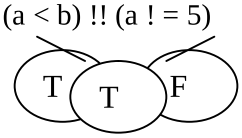
(ii) (a < b) && (a ! = 5)



(b)logical **: -** The OR operator will return true if any of given expression becomes true.

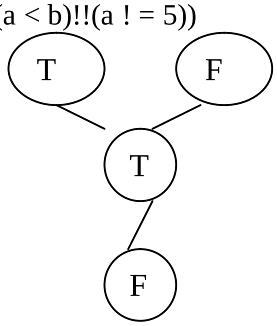
Ex. a = 5, b = 10

(a < b) !! (a ! = 5)

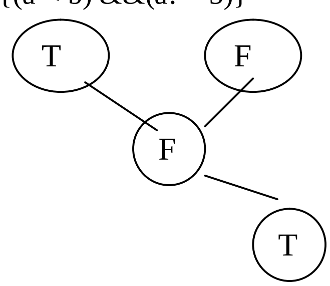


(C) Logical NOT:→ logical NOT operator willreverse the meaning of expression.

Ex.! ((a < b)!!(a ! = 5))



!{(a < b) &&(a! = 5)}



NOTE: - All relational and logical operators are binary in nature except logical NOT. Logical NOT is uniary operator.

(6) <u>Conditional operator</u> : - It is a special operator of nature. It returns true or false value on the basis of given condition. The general format is (condition)? True value: false value;

Ex. a = 5, b = 10

(i) (a<b)? a:b → **a** → **5**

(ii) (b! = 10)? a:b → B → 10

(7) <u>special operator</u> : - Following ar special ope

, . * [ ] ()

Precedence of operator

If more than one operators are involved in an expression then, C language has predefined rule of priority of operators. This rule of priority of operators is called operator precedence.

In C, precedence of arithmetic operators(*,%,/,+,-) is higher than relational operators(==,!=,>,<,>=,<=) and precedence of relational operator is higher than logical operators(&&, || and !). Suppose an expression:

```
(a>b+c&&d)

This expression is equivalent to:

((a>(b+c))&&d)

i.e, (b+c) executes first

then, (a>(b+c)) executes

then, (a>(b+c))&&d) executes
```

Associativity of operators

Associativity indicates in which order two operators of same precedence(priority) executes. Let us suppose an expression:

```
a==b!=c
```

Here, operators == and `!=` have same precedence. The associativity of both == and `!=` is left to right, i.e, the expression in left is executed first and execution take pale towards right. Thus, a==`b!=`cequivalent to :

```
(a==b)!=c
```

The table below shows all the operators in C with precedence and associativity.

Note: Precedence of operators decreases from top to bottom in the given table.

Summary of C operators with precedence and associativity

Operator	Meaning of operator	Associativity
() [] -> .	Functional call Array element reference Indirect member selection Direct member selection	Left to right
! ~ + - ++ -- & * sizeof (type)	Logical negation Bitwise(1 's) complement Unary plus Unary minus Increment Decrement Dereference Operator(Address) Pointer reference Returns the size of an object Type cast(conversion)	Right to left
* / %	Multiply Divide Remainder	Left to right
+ -	Binary plus(Addition) Binary minus(subtraction)	Left to right
<< >>	Left shift Right shift	Left to right
< <= > >=	Less than Less than or equal Greater than Greater than or equal	Left to right
== !=	Equal to Not equal to	Left to right
&	Bitwise AND	Left to right
^	Bitwise exclusive OR	Left to right
|	Bitwise OR	Left to right
&&	Logical AND	Left to

Summary of C operators with precedence and associativity		
Operator	Meaning of operator	Associativity
		right
||	Logical OR	Left to right
?:	Conditional Operator	Left to right
= *= /= %= -= &= ^= |= <<= >>=	Simple assignment Assign product Assign quotient Assign remainder Assign sum Assign difference Assign bitwise AND Assign bitwise XOR Assign bitwise OR Assign left shift Assign right shift	Right to left
,	Separator of expressions	Left to right

Some special situation with increment operator.

```
Ex.     a = 5 ;
        b = a++ * a++ * a++;
printf("%d",b); = 125
printf("%d",a); = 8
ex.     a = 5 ;
        b = ++a * ++a* ++a;
printf("%d",b); → 216
printf("%d",a); → 8
```

Input output statement

'C' provides two common functions for input output.

(i) scanf( )

(ii) printf( )

(i) scanf( ) : → This function is use to input the value through the keyboard at run time.

The general format is scanf ("control string",& variable);

- control string indicates the nature of value inputted through the keyboard also called format specifier.

  Following are common format specifier.

  %d → int

  %f → Float

  %c → character

  %s → string

  %h → short int

  %u → unsigned int

  Ex. int a;

  Scanf("%d",&a);

  We can also input multiple value with single scanf.

  Ex. int a;

  Scanf("%d%d",&a,&b);

- int a;

  float b;

  Scanf("%d%f",&a,&b);

(ii) printf( ) : → This function is use to print the value of variable on the screen.

The general format is **printf("control string",variable);**

Ex. int a;

Printf("%d",a);

We can also print multiple value with single printf.

Ex. int a,b;

Printf("%d%d",a,b);

- write a programme to input two no from the user and print the condition.

```
Main( )
{ int a,b,c;
  Printf(" Enter two no");
 Scanf("%d%d",&a,&b);
 C = a+b;
Printf("%d",c);
}
```

- write a programme to input two no and print the addition, subtraction, multiplication and Division.

```
Main ( )
{
   Int a,b,c,d,e,f;
```

```
Printf("Enter two no.");
Scanf("%d%d", &a,&b);
c = a+b;
d = a-b;
e = a*b;
f = a/b;
printf("%d%d%d%d",c,d,e,f);
```

- write a program to swap the value of two variable.

```
Main ( )
{
       Int a, b, c;
       Printf(" Enter two no ");
       Scanf("%d%d",&a,&b);
       Printf("Before swaping");
       Printf("a=%d,b=%d",a,b);
       C=a;
       a=b;
       b=c;
       printf("After swap");
       printf("a=%d,b=%d",a,b);
       }
```

- write a pogramme to swap to the value of two varianle without using third variable.

```
# include <studio.h>
# include <conio.h>
Void main ( )
{
Int a,b;
Printf("enter two no");
Scanf("%d%d",&a,&b);
Printf("Before swaping");
Printf("a=%d,b=%d",a,b);
a=a+b;
b=a-b;
a=a-b;
printf("After swaping");
printf("a=%d,b=%d,a,b");
getch( );
}
```

TYPE CASTING

Type casting is the process to change the nature of variable. There are two types of type casting.

(i) Implicit
(ii) Explicit

(i) **Implicit :** → It is the process to change the data type of variable automatically. When we use more than one variable in a sigle expression , the lower types of variable will be change into the higher types automatically. This processes is called implicit type cast.

Following are the memory size priority of variables.

Long double

Double high
float
long
int
short low
char

```
ex. main ( )
{
        int a;
        float b,c;
        a =12;
        b = 13;
        c = (a+b)/2;
        printf("%f",c);
}
        Result = 12.50
```

(ii)Explicit:→ It is the process to change the data types of variable forcially.

```
Ex. main ( )
{
        Int a,b;
        Float c;
        a = 12;
        b = 13;
        c = (a+b)/2;
        printf("%f",c);
}
        Result = 12
```

The above program generates wrong output to solve this problem we have to need explicit type casting.

```
Ex. main ( )
    {
        int a,b;
        float c;
        a = 12;
        b = 13;
        c = (float)(a+b)/2;
        printf(" %f ",c);
    }
            Result = 12.50
```

MATHAMATICAL FUNCTION

'c' provides a no of predefined functions for mathematical calculations and all these function are available in math as header file.

Following are some common mathematical functions:-

	Function	**Meaning**
1.	Sin (x)	sin value of (x)
2.	cos (x)	cos value of (x)
3.	tan (x)	tan value of (x)
4.	sqart (x)	square root of (x)
5.	pow (x,y)	x to the power y (x)
6.	ceiling (x)	x rounded up to nearest integer
7.	floor (x)	x rounded down to nearest

integer

8. fmod (x,y) x remendery

9. fabs (x) absolute value of x.

User defined type declearation

We can assign a predefined or user defined data types in a valid identifier.

To increase the reliabilities for user defined type declaration, we use typed f statement.

The general format is typedef data type identifier

Ex.
1. Typedef int number;
   number n1;
2. Typedef float decimal;
   decimal d1;

CONTROL STATEMENT

The statement which is use to control the flow of control is called control statement.

There are three types of control statement.

1. selective statement/conditional statement /branching statement/decision making
2. Iterative statement/repeatative statement/loop
3. jumping statement

1. **selective statement or conditional statement or branching statement :-** The statement which is use to excute a selected statement on the basis of cirtain condition.

There are four type of selective statement.

a. Simple if
b. Nested if
c. If-------else ladder
d. Switch statement

a. Simple statement:- simple if statement is use to execute the statement on the basis of single condition.

The general format is –

```
if (condition)
true statement;
else
false statement;
```

ex. write a programme to input two numbers from the user and print the maximum.

```
Main()
{
Int a,b;
Printf (" Enter two no ");
Scanf ("%d%d",&a,&b);
If (a > b);
Printf ("a is max %d",a);
Printf("b is max %d",b);
}
```

- Write a programme to input the product rate and chek if it is greater than equal to five hundred give 20% discount other wise give 15% discount print the discount and net amount.

```
Main( )
```

```
{
        int product rate, discount , Netamount;
        printf ("Enter product rate");
        scanf ("%d",& product rate);
        if (product rat > = 500);
        discount = (500*20)/100;
        else
        discount = (500*15)/100;
        Netamount = product rate – discount;
        Printf ("%d%d",discount,Netamount);
}
```

NOTE:- By default if or else statement associate only one statement as its body. In this case to assocates multiple statement we use separate curly braces.
NOTE:- we can't use a separate condition for else.
NOTE:- we can also use if statement without else.

- Write a programme to input the employ salary and chek if it is > = 7000 give 40% bonous other wise give 32% bonous. Print the bonous and net salary.

```
        Main ( )
{
        Int employsalery , bonous , netsalary;
        Printf ("%d", Employ salery);
        Scanf(" %d",& employ salary);
        If (employ salary > = 7000);
        Bonous = (emplor salary*40)/100;
        Else
        bonous = (employ salary * 32)/100;
        net salary = (employ salary + boous);
        printf ("%d%d,bonous, net salary);
}
```

- Write a programme to input a no and check it is positive or negative.

```
        Main ( )
{
        Int a;
        Printf (" Enter a number ");
        Scanf (" %d",&a);
        If (a > 0);
        Pritf (" a is positive");
        else
        printf (" a is nigative");
}
```

- Write a programme to input a number and chek , it is odd or even.

```
        Main ( )
{
        Int a;
        Ppritf ("Enter a no.");
        Scanf ("%d",&a);
        if (a/2 = = 0);
        printf ("a is even%d",a);
        else
        printf ("a is odd %d", a);
```

```
}
```

b. Nested if :- we can write if statement inside the other if. It is called “nested if”. Nested if is useful when we want to give more than one condition.

- Write aprogramme to input three number and print the maximum.

```
Main ( )
{
int a,b,c;
printf(“Enter three number”);
scanf (“%d%d%d”,&a&b&c);
if (a > b)
```

Note:- In nested if , else is associated with its nearer if.

- Write a programme to input four no and prit the max.

```
Main ( )
{
int a,b,c,d;
printf(“Enter three number”);
scanf (“%d%d%d %d”,&a&b&c&d);
if (a > b)
if (a > c)
if (a > d)
printf(“a is max”);
else
printf(“d is max”);
else
if(c>d)
printf(“c is max”);
else
printf(“d is max”);
else
if (b > c)
if (b > d)
printf(“b is max”);
else
printf(“d is max”);
else
if (c > d)
printf(“c is max”);
else
printf(“d is max”);
printf (“%d”,max);
}
```

- Write a program to input five no. from the user and print max.

```
Main ( )
```

```
{
    int a,b,c,d,e;
    printf("Enter three number");
    scanf ("%d%d%d %d%d",&a&b&c&d&e);
    if (a > b)
    if (a > c)
    if (a > d)
    if (a > e)
    printf("a is max");
    else
    printf("e is max");
    else
    if(d > e)
    printf("d is max");
    else
    printf("e is max");
    else
    if (c > d)
    if (c > e)
    printf("c is max");
    else
    printf("e is max");
    else
    if (d > e)
    printf("d is max");
    else
    printf("e is max");
    else
    if(b > c)
    if (a > d)
    if (b > e)
    printf("b is max");
    else
    printf("e is max");
    else
    if (d > e)
    printf("d is max");
`   else
    printf("e is max");
    else
    if (c > d)
    if (c > e)
    printf("c is max");
    else
    printf("e is max");
    else
    if (d > e)
    printf("d is max");
    else
    printf("e is max");
    printf ("%d",max);
```

}

(c). If -------- else if ladder :- This format of if is also useful to give multiple condition.

The general format is

```
If (condition 1)
Statement ;
else
if (condition 2)
statement;
.
.
.
.
else
if (condition n)
statement;
else
statement;
```

Ex. Write a programme to input four number from the user and print the maximum.

```
Main ( )
{
Int a,b, c,d;
Printf(" Enter four number");
Scanf("%d%d%d%d",&a&b&c&d);
If ((a > b) && (a > c) && (a > d))
Max = a;
else
if((b > a) && (b > c) && (b > d))
max = b;
else
max = d ;
printf(" %d", max);

}
```

- Write a programmed to input the product rate and check if it is greater than equal to 1000, give 20% discount. If >= 500 &1000 give 25% discount and if product rate < 500give 12 % discount .print the discount and namt.

```
Main ( )
{
int prrate, discount, namt;
printf (" Enter product rate");
scanf ("%d",& prrate);
if (prrate > = 1000)
discount = (prrate * 20)/100;
else
discount = (prrate * 12)/10;
namt = prrate – discount;
printf (" %d%d", Namt , discount);
}
```

- Write a programme to input the marks of five subject and calculate total option marks percentage and division according to BIEC. Assume the full marks is 500.

```
Main ( )
{
    int math, physics, chem, eng, Hindi ;
    int total , percentage ;
    printf (" Enter marks of five subjects");
    scanf ("%d%d%d%d%d",&math&phy
    &chem&eng&hindi);
    Total = math + phy+ chem+ eng+ hindi;
    Percentage = ( total * 100)/500;
    Printf ("%d",percentage);
    If (percentage > = 60)
    Printf ("first div");
    else
    if (" percentage > = 45")
    printf (" 2nd div");
    else
    if (percentage > = 30)
    printf ("3rd div");
    else
    printf ("fail');
}
```

- Write a programme to input to the marks of 5 subject and calculate total marks , percentage and division you should must maintain that student must get 30 marks in each subjects.

```
Main ( )
{
    Int M,P,C,E,H;
    Printf ("Enter  marks of 5 subjects‘’);
    Scanf ("%d%d%d%d%d",&M&P&C
    &E&H);
    Total = M+P+C+E+H;
    Per = (total*100)/500;
    If ((M < 30) && ( P < 30) && (C < 30) && ( E < 30)
    && (H < 30))
    Printf ("fail");
    Else if (per > = 60);
    Printf (first div);
    else
    if (per > = 45)
    printf ("2nd div");
    else
    if (per > = 30)
    printf ("3nd div");
    else
    Printf ("fail");
}
```

- Write a programme to input the basic pay of employee and calculate the following allounces a/c to to the given condition.
    1. BP > = 8000
       DA   = 47%

TA = 15%
HRA = 18%
2. BP > = 5000 & < 8000
DA = 43%
TA = 13%
HRA = 16%
3. Bp < 5000
DA = 41%
TA = 12%
HRA = 650

Calculate the gross pay. calculate the Net pay.

PF = 7% of gross pay
ESI = 3.5% of gross pay

```
Main()
{
    Float BP,TA DA,HRA,GP,PE,ESI,ND;
    Printf ("Enter basic pay");
    Scanf ("%d",& BP);
    If (Bp <  5000)
    {
        DA = (BP*47)/100;
        TA = (BP*15)/100;
        HRA = (BP*18)/100;
        Else
        If ( BP > = 5000)
        {
            DA = (BP * 43)/100;
            TA =  (BP * 12)/100;
            HRA = 650;
        }
    GP = BP + TA + DA + HRA;
    PF = (GP*7)/100;
    ESI = (GP*3.5)/100;
    NP = GP – (PF + ESI);
    Printf ("%f%f%f%f%f%f%f",
    TA,DA,HRA,GP,PE,ESI,NP);

    }

}
```

SWITCH STATEMENT

Point related to switch

1. Switch is also a selective statement use to compare one value with multiple constant values.
2. Switch is simple solution of complex if else structure.
3. The data type of switch either in the expression or the case value must be int of char. case valu must be constant value.
4. We can't use same case value more than ones in the programme.
5. We can't use only relational or logical operator as case value.
6. Break is optional in switch .we use break statement to terminate the switch forcely.
7. In switch, the value used in the expression will be compare with case value from top to bottom. As soon as it finds the first matched case value , the body associated with that case value will be executed and this execution will be continue until it break statement or end of switch.
8. If there is no any matched case found default will be executed if it is given.
9. Default is also optional in switch.
10. We can also write default statement in between two case values.
11. To declare a switch, we use a key words switch.

The general format is-

```
        Switch (Expression)
           {
            Case value 1;
            Statement;
            break;
                    .
                    .
                    .

     Case value n:
        Statement;
            break;
     default:
          Statement
}
```

Ex:-

```
  main (     )
{
        int num;
        printf ("ENTER A NUMBER );
        scanf ("% d", &num);
```

```
{
        case5:
    printf ("five");
        break;
       case 10:
        printf ("ten");
       break;
        case 15:
       printf ("fifteen");
         break;
        default:
        printf ("Bye");
}
  }
```

- Write a programme to input day number and print the day name.

```
         Main (  )
    {
         int num;
         printf ("Enter a number");
        Scanf ("%d" ,& num );
        Switch (num )
    {
        Case 1:
        printf ("monday");
        break;
        Case2:
        printf ("tuesday");
        break;
        Case3:
         printf ("Wednesday");
        break;
        Case4:
         printf ("Thursday");
         break;
        Case5:
         printf ("Friday");
         break;
        Case6:
         printf ("saturday");
         break;
        Case7:
        printf ("sunday");
    break;
        default to ("none");
```

```
                printf (none);
        }
        }
```

- Write a programme to input the month no and print the month name.

```
Main ( )
        {
                int num;
                printf ("Enter a number");
                Scanf ("%d" ,& num );
                Switch (num )
        {
                Case 1:
                printf ("January");
                break;
                Case2:
                printf ("February");
                break;
                Case3:
                printf ("March");
                break;
                Case4:
                printf ("April");
                break;
                Case5:
                printf ("May");
                break;
                Case6:
                printf ("June");
                break;
                Case7:
                printf ("July");
        break;
                Case 8:
                printf ("August");
                break
                Case9:
                printf ("September");
                break;
                Case10:
                printf ("October");
                break;
                Case11:
                printf ("November");
                break;
```

```
Case12:
 printf ("December");
 break;
default to ("none");
printf (none);
}
}
```

Iterative statement or Repeatative statement or loop

When we want repeat a particular statement no of times on the basic of certain conditions, we use iterative statement also called loop.

There are three types of loop in 'c'.

1. for loop (counter loop)
2. while loop
3. Do – while loop

1. **for loop:** → It is a counter loop normally useful to when we known the exact no of repeatation.

The general format is –

```
For (initialization; condition; increment/decrement)
{
body to repeat;
}
```

Ex. Write a programme to print your name 100 times.

```
Main ( )
{
Int i;
for ( i =1; i <=100; i++)
    {
        Printf ("MR.DINKAR");
    }
}
```

- write a programme to print 1 to 100.

```
main ()
{
    int i;
    for( i = 1; i <=100; i++)
    {
        Printf ("%d",i);
    }
}
```

- Write a programme to print 100 to 1.

```
main( )
{
    int i;
    for (i=100;i>=1; i--)
    {
        Printf("%d"i);
    }
    }
```

- Write a programme to print all odd number below hundred.

```
main( )
{
    int i;
    for(i=100; i<=100; i++)
    {
        if(i/2!=0)
        printf("%d",i);
    }
}
```

→

In the for loop, first the variable will be initialize and check the condition. If the given condition is true the body associated with that condition will be executed other wise loop will be terminate for the next time and own word the control will be transferred in increment / decrement section where the value of variable will be changed and again check the condition. If the given condition will be true the body associated with that condition will be executed other wise loop will be terminated.

- Write a programme to print all even no in between 1 to 100 and also print their sum.

```
main ()
{
    int i; sum=0;
    for(i=1; i<=100;i++)
    {
        If(i/2 = = 0)
        {
            printf("%d",i);
            sum = sum+i;
        }
        Printf("%d",sum);
    }
}
```

- Write a programme to print the table of five.

```
main ( )
{
```

```
        int i ;
        for (i=5;i<=50;i=i+5)
        {
                Printf("%d",i);
        }
}
        Or
main ( )
{
        int i ;
        for (i=1;i<=10;i=i+5)
        {
                Printf("%d",(i*5));
        }
}
```

- Write a program to input no and print the table.

```
        main ()
{
        int i,a;
        printf("Enter a number");
        scanf("%d",&a);
        for(i=a;i<=a*10; i+a)
        {
                Printf("%d",i);
        }
}
                Or
        for(i=1;i<=10; i++)
        {
                Printf("%d",(i*a));
        }
}
```

- Write a program to input a no and print their table. Also print their sum.

```
                main( )
        {
                int i,a;sum=0;
                printf("Enter a number");
                scanf("%d",&a);
                for(i=1;i<=10; i++)
                {
                        Printf("%d",(i*a));
                        Sum = sum +(i*a);
                }
                Printf("%d",sum);
```

```
}
```

- Write a programme to input a number and print the factorial.

```
main( )
{
    int i, a, fact=1;
    printf("Enter a number");
    scanf("%d",&a);
    for(i=a;i<=1; i--)
    {
        fact = fact * i;
        printf("%d",fact);
    }
}
```

- Write a programme to input a number and check it is odd or even if it is odd print the factorial , otherwise print the table.

```
main( )
{
    int i, a, fact=1;
    printf("Enter a number");
    scanf("%d",&a);
    if (a/2!=0)
```

- Write a programme to print the following series.

O, 1, 4, 9, 16, -------------------n terms.

```
main()
{
    int i,n;
    printf("Enter a no. of terms");
    scanf("%d",&n);
    for(i=0,i<=n;i++)
    {
        Printf("%d",((i*i)+i));
    }
}
```

- 0, 1, 1, 2, 3, 5, 8, ----------------n

```
main ()
{
    int i, n, a, b, c;
```

```
a=0;b=1;
printf("Enter no. of terms");
scanf("%d",&n);
printf("%d%d",a,b");
for("i=I,i<=n-2,i++");
{
        C=a+b;
        Printf("%d",c);
        a=b;
        b=c;
}
}
```

Dry run

N	I	o.p	C	a	b
10	1	1	1	1	1
-	2	2	2	1	2
-	3	3	3	2	3
-	4	5	5	3	5
-	5	8	8	5	8
-	-	-	-	-	-
-	-	-	-	-	-

- Write a programme to input a no and check it is prime or not.

```
Main()
{
        int i, num;
        char prime = 't';
        printf("Enter a number");
        scanf("%d",&num);
        for(i=2;i<num/2;i++)
```

```
        {
            if(num%i==0)
            {
                Prime = 'f ';
                break;
            }
            if(prime =='t')
            printf("No is prime");
            else
            printf("No is not prime");
        }
    }
```

2. **Nested for loop:-** If we want we can write another loop inside one loop. It is called nesting of loop. Nested for loop is useful when we worked with more than one direction.

    Ex. Write a programme to print the following series.

    1
    1 2
    1 2 3
    1 2 3 4

```
main( )
{
int i,j;
for(i=1,i<=4;i++)
{
   for(j=1;j<=i;j++)
   printf("%d",j);
   printf("\n");
}
}
```

Note: - In nested for loop generally outer loop is used for row and inner loop is used for column.

- Write a programme to print the following series.

    1 2 3 4
    1 2 3
    1 2
    1

```
main( )
{
int i,j;
for(i=4,i>=1;i--)
{
   for(j=1;j<=i;j++)
   printf("%d",j);
```

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

- Write a programme to print the following series.

4
4 3
4 3 2
4 3 2 1

```
main( )
{
int i,j;
for(i=4,i>=1;i--)
{
    for(j=4;j>=i;j--)
    printf("%d",j);
    printf("\n");
}
}
```

- Write a programme to print the following series.

4 3 2 1
4 3 2
4 3
4

```
main( )
{
int i,j;
for(i=1,i<=4;i++)
{
    for(j=4;j>=i;j--)
    printf("%d",j);
    printf("\n");
}
}
```

- Write a programme to print the following series.

1
2 2
3 3 3
4 4 4 4

```
main( )
{
int i,j;
for(i=1,i<=4;i++)
```

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

- Write a programme to print the following series.

4 4 4 4
3 3 3 3
2 2
1

```
main( )
{
int i,j;
for(i=4,i>=1;i--)
{
    for(j=1;j<=i;j++)
    printf("%d",i);
    printf("\n");
}
}
```

- Write a programme to print the following series.

0 1 4 9
0 1 4
0 1
0

```
main( )
{
int i,j;
for(i=3,i>=0;i--)
{
    for(j=0;j<=i;j++)
    printf("%d",(j*j));
    printf("\n");
}
}
```

- Write a programme to print the following series.

0
0 1
0 1 4
0 1 4 9

```
main( )
{
int i,j;
```

```
for(i=0,i<=3;i++)
{
    for(j=0;j<=i;j++)
    printf("%d",j);
    printf("\n");
}
}
```

- Write a programme to print the following series.

*****
******
****
****

```
main()
{
int i,j,T;
for(i=1;i<=4;i++)
{
    for(j=1;j<=i;j++)
    T=*;
    Printf("%d",T);
    Printf("\n");
}
}
```

- Write a programme to print the following series.

****
****
****** **I = row**
***** **j = column**

```
main()
{
int i,j,T;
for(i=4;i>=1;i--)
{
    for(j=1;j<=i;j++)
    T=*;
    Printf("%d",T);
    Printf("\n");
}               }
```

- Write a programme to print the following series.

******
****
****
****

```
main()
```

```
{
int i,j,T;
for(i=2;i<=8;i=i++)
{
    for(j=1;j<=i;j++)
    T=*;
    Printf("%d",T);
    Printf("\n");
}
}
```

- Write a programme to print the following series.

```
*
***
*****
*******
main()
{
int i,j,T;
for(i=1;i<=7;i=i+2)
{
    for(j=1;j<=i;j++)
    T=*;
    Printf("%d",T);
    Printf("\n");
}
}
```

- Write a programme to print the following series.

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

1. for(i=1;i<=100;i++);
printf(“%d”,i);
If we use semicolumn, it means there is no any bodys associated with for loop.
Output =101
2. i=1;
for(i<=100;i++)
printf(“%d”i,);
we can leave any section of for loop has empty but we must use to semi column.
3. for(i=100;i;i--)
Printf(“%d”,i);
Output=100-1
4. for(i=1;j=5;i<=10;i++;j--)
Printf(“%d%d”i,j;);
If we want we can use more than one expression in a single section of for loop but we must be separated by the couma.
1,5,3,4,3,3,4,25,1,6,0,7-1,8,-2,9,-3,10,-4

Special situation with for loop

5. for(i=1;j=5;i>=0;i<=10;i++;j--)
Printf(“%d%d”,i,j);
When we usew more than one condition in the single for loop, loop will be terminated according to right hand side condition.

While loop:→

It is a conditional loop useful when we don’t known the exact number of repetation.In while loop we get only condition in the clause but if we want we can give the initialize and increment decrement section externally.

The general format is
While(condition)
{
body of repeat
{

It is a general rule that we can before the loop and give increment decrement section inside the loop as a last statement.

initilize
while (condition)
{
Increment/ decrement

```
{
```

Ex. Write a programme to input to print 1 to 100.

```
main()
{
int i;
i=1;
while(i<=100)
{
Printf("%d",i);
i++;
}
}
```

- Write a programme to input a number and print the table.

```
main()
{
int,i,num;
printf("Enter a number ",);
scanf("%d",&num);
i=1;
while (i <=10);
{
T= num*i;
printf("%d",T);
i++;
}
}
```

- Write a programme to input a number and print the table in reverse order and also print their sum.

```
main()
{
int i,num, T, sum=0;
printf("Enter a number ",);
scanf("%d",&num);
i=10;
while (i >=1)
{
T= num*i;
printf("%d",T);
i--
sum = sum+T;
printf("%d",sum);
}
}
```

- Write a programme to input a number and print the maximum and minimum digit.

```
main()
{
int num, temp,max,min;
printf("Enter a number",);
scanf("%d",&num);
max=min=num%10;
while(num>0)
{
temp = num %10
if(max<temp)
max = temp;
if(min>temp)
```

```
num = num/10;
}
printf("Max digit =%d",max);
printf("Min digit =%d",min);
}
```

Dry run

Num	Temp	5max	5min
1785	5		
178	8	8	
17	7		
1			
0			

- Write a programme to input a number and reverse the digit.

```
main()
{
int num, temp,rev=0;
printf("Enter a number",);
scanf("%d",&num);
while(num>0)
{
temp = num %10
rev=rev*10 + temp
num = num/10;
}
printf("reverse of number is %d",rev);
}
```

Dry run

Num	Temp	Rev
1234	4	4

123	3	43
12	2	432
1	1	4321

```
main()
{
int num, temp,rev=0,T;
printf("Enter a number",);
scanf("%d",&num);
T=num;
while(num>0)
{
temp = num %10
rev=rev*10 + temp
num = num/10;
}
if(T==rev)
printf("Number is polidram");
else
printf("Number is not polidram");
}
```

- Write a programme to print the table of 5 untill you gate the sum of series.

```
main()
{
int i, T, sum=0;
i =1;
While (sum<=1260)
{
T=5*i;
printf("%d",T);
Sum= sum+T;
i++;
}
}
```

- Write a programme to print the following series.

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

```
Main()
{
int i, j;
i =1;
while (i<=4)
{
J=1;
While(j<=i)
{
Printf("%d"j,);
j ++;
{
Printf("\n",);
i++;
}
}
```

- Write a programme to print first 10 tebonikiseries using while loop.

```
Main()
{
int a ,b, c, i;
a=0;
b=1;
printf("%d%d",a,b);
i=1;
While(i<=8)
{
c=a+b;
printf("%d",c);
a  =b;
b=c;
i++;
}
}
```

Do while loop :→ when it is necessary to ascute a loop atleast ones , we use do – while loop in do – while loop first the body will be executed then check the condition.

The general body formate is

```
do
{
   body to repeat
}
While (condition);
```

Ex.

- Write a programme to print 1 to 100.

```
main()
{
int i;
do
{
printf("%d",i);
i++;
}
While(i<=100);
}
}
```

- Write a programme to input one number and print the factorial.

```
main( )
{
        int i, num, fact=1;
        printf("Enter a number");
        scanf("%d",&num);
        i=num;
        do
        {
                fact = fact * i;
                i--;
        }
        While(i>1)
                printf("%d",fact);
}
```

- Write a programme to input a number and reverse the digit.

```
main()
{
int num, temp,rev=0;
printf("Enter a number",);
scanf("%d",&num);
do
```

```
{
temp = num %10;
rev=rev*10 + temp;
num = num/10;
}
While(num>0);
printf("reverse of number is %d",rev);
}
```

The major adavantage using do-while loop is that we can make a programme Genric.

- Write a programme to print 1 to 500 page ways.

```
main()
{
int i;
for(i=1;i<=500;i++)
printf("%d\n",i);
if(i/25= =0)
{
printf("press  any key",);
getch();
}
}
```

Genric means to take output more than ones in the same running model.

ex.

- write a programme to input a no and print the table . you should write a programme to such a mainer use can take output more than ones.

```
main()
{
int num,i;
chaar ch;
do
{
printf("Enter a number",);
scanf("%d",&num);
for(i=1;i<=10;i++)
printf("%d",(num*i));
printf("do you want to continue (Y/N)");
ch=getch();
}
while ((ch= = 'y') !! (ch= = 'Y'));
```

- write a generic prog to input a no and print the factorial.

```
main()
{
```

```
int num,i,f;
cha ch;
do
{
f = 1
printf("Enter a number",);
scanf("%d",&num);
for(i=1;num;i>1;i--)
f = f*i
printf("%d",f);
printf("do you want to continue (Y/N)");
ch=getch();
}
while ((ch= = 'y') !! (ch= = 'Y'));
```

<u>Jumping statement</u>

jumping statement is used to jumps the control from one location to another location in the programme.

following three are called **jumping** statement.

1. Break
2. Continue
3. Go to

1. Break statement :→ Break statement is generally used in switch but we can also use break to terminate the loop forcely.

Ex.

```
1. for(i=1;i<=100;i++)
     if (i > = 50)
     break;
else
    printf("%d",i);
    output = 1 to 49
2. for(i=1;i<=100;i++)
     if (i > = 50) && (i < = 75)
     break;
else
    printf ("%d",i);
    output = 1 to 49, 76-100
3. for(i=1;i<=100;i++)
     if (i > = 50) && (i < = 75)
     continue;
else
    printf ("%d",i);
    output = X
```

2. continue statement :→ When we use continue statement with for loop , after the execution of continue , control will be transfer in increment , decrement section.

Ex.

```
i=1;
while(i<=100)
{
   if(i>=25)
   continue;
else
printf("%d",i);
i++;
}
```

3. Go to statement :→ Go to statement is use to transfer the control from one label to another label in the programmer.

Label is any valid identifier.

The general format is :-

```
 go to label;
   label:
```

we can go to statement with two different mechanism.

1. un conditional goto

2. conditional goto

1. unconditional go to :→ when we use goto statement without any condition then it is called unconditional goto.

ex.

```
main()
{
Hello
printf("How are you.");
printf("Bye.");
go to hello;
}
```

2. conditional goto :→ when we use go to statement with a separate condition , then it is called conditional go to .

ex.

```
stlib.h
main()
{
int a,b;
printf("Enter two no.");
scanf("%d%d",&a,&b);
if (a>b)
go to first;
else
go to second;
{
first :
```

```
printf("a is max");
exit(0);
second:
printf("b is max");
}
```

*** stlib.h**

```
or
first : printf("a is max");
go to last;
second:
printf("b is max");
last:
```

Note:→ The 'c' avoid the use of go to
Note:→ stlib.h → The definition of exit(0) is available in this header file.

<u>Array</u>

Array is the collection of more than one variables having same name and same data types but with different index value. Basically is useful to store multiple values at contiguous memory. the general format is:

Data type	**Array Name**	**[Size];**
↓	↓	↓
Any predefined OR user defind	Any valid identifier	Always integer constant

Eg. **Int A[5];**

101	**103**	**105**	**107**	**109**
A[0]	**A[1]**	**A[2]**	**A[3]**	**A[4]**

The index of array will be started from 0 hence to access the first element we use A[0] and to access the last element we use A[n-1] where n is size of array.

The memory allocation of array will be done in contiguous manner hence if we know the address of first element (base element) we can easily find the address of next element.

The name of the array is a constant pointer which will hold the address of base element.

Eg.
Write a program to input 10 integer values in the array and print it.

```
main()
{
int A[10];
Int I;
for(i=0;i<10;i++)
{
printf("enter a number ; ");
scanf("%d",&A[i]);
}
printf("---- elements of array-------");
for(i=0;i<10;i++)
printf("%d",A[i]);
 }
```

Write a program to input 10 integer values in the array and find the sum.

```
main()
{
int A[10];
Int I,sum=0;
for(i=0;i<10;i++)
{
printf("enter a number ; ");
scanf("%d",&A[i]);
}
printf("---- elements of array-------");
for(i=0;i<10;i++)
{
printf("%d",A[i]);
sum=sum+A[i];
 }
Printf("sum of elements= %d",sum);
}
```

Write a program to input 10 integer values in the array and find the sum of elements,average,largest element and smallest element.

```
main()
{
int A[10];
Int I,sum=0,max,min,avg;
for(i=0;i<10;i++)
{
```

```
printf("enter a number ; ");
scanf("%d",&A[i]);
}
Max=min=A[0];
printf("---- elements of array-------");
for(i=0;i<10;i++)
{
printf("%d",A[i]);
sum=sum+A[i];
if(max<A[i])
max=A[i];
if(min>A[i])
min=A[i];
 }
Avg= sum/10;
Printf("sum of elements=%d",sum);
Printf("largest element=%d",max);
Printf("smallest element= %d",min);
Printf("average= %d",avg);
}
```

INITILIZATION OF ARRAY

We can initialize the array in the following manner:

1. int A[5]={10,20,30,40,50};

printf("%d",A[1]);	-	20
printf("%d",A[4]);	-	50

2. int A[]={10,20,30,40,50};

printf("%d",A[1]);	-	20
printf("%d",A[4]);	-	50

if we want , we can leave the size of array but in this case we must initialize all the elements of array at the time of declaration.

3. int A[5]={10,20,30,40};

printf("%d",A[1]);	-	20
printf("%d",A[4]);	-	GV

if we will not initialize any element of array, compiler will not initialize it hence it produce garbage value.

4. static int A[5]={10,20,30,40};

printf("%d",A[1]);	-	20
printf("%d",A[4]);	-	0

if we use static keyword with array declaration, compiler will initialize all empty elements with 0.

5. int A[5]={10,20,30,40,50};

printf("%d",A[1]);	-	20
printf("%d",A[4]);	-	50
printf("%d",A[5]);	-	GV

in c, there is no any boundary checking for the elements of array it means we can access the element of array beyond its size. Neither compiler nor run time system will complain but it is the responsibility of the programmer to take care about the boundary of array otherwise entire system may be damaged.

- Write a program to input n number in the array and sort it .

```
# Define  N   10
Main (  )
{
  int  A[N] ,i,j,temp;
  for ( i=0; i<N; i++)
 {
        print f ("Enter a no ") ;
        Scan f(" % d" ,& A [i]) ;
```

```
}
/ * printing original Array * /
Print f("________Elements Array _______") ;
for(i=0;i<N;i++)
        Print f ("% d " ,A[i]) ;
/ * Sorting begins */
for (i=0; i<N-1 ;i++)
for (j=0 ; j<N-1-i ; j++)
  if ( A[j]> A [j+1])
{
temp = A [j] ;
A [j] = A [j +1 ] ;
A [j+1] = temp ;
}
/ * Printing sorted Array */
Print (" after sorting ")
For (i=0; i<N ; i++)
Print f ("% d " ,A[i] ;
}
```

- Write a program to search a particular item in the array .

```
# define N    10
Main ( )
{
int A [N] ,i , item , loc =0 ;
for (i=0 ; i<N ; i++)
{
        Print f ("Enter a no ") ;
        scan f("%d" ,&A [i] ;

    }

        / * Printing Array elements * /
         Print f (___elements of array__)
        For (i=0 ; i<N ; i++)
        Print f ("% d '' , A[i]);
        / * searching begins * /
        Print f ("Enter items to search") ;
        Scan f ("% d '' ,& item ) ;
         For (i=0 ; i<N ; i++)
        If (A[i] == item)
        {
                loc =i+1;
                break ;

         }
```

```
/ * checking  errors * /
If (loc > 0)
{
Print f (" searching successful");
Printf("item found at location %d",loc);

}
else
  {
        printf("searching fallure")
        printf ("Iteam doesnot exist n the list");
  }
}
```

- Write a program to delete a particular element from the array.

```
# define N 100
main (  )
{
int A[N], i , item , loc , size;
Printf( "Enter the Array size");
Scanf (" % d" , & size );
/ * input Array * /
for (i=0; i<size; i++)
{
        printf ("Enter a no");
        Scan f (" % d" ,&A[i]) ;

 }
/ * printing Array * /
Print f ("elements of Array");
for (i=0; i<seze; i++)
Print f (" % d" , A[i]);
/ * Deleting begins * /
Printf ("enter item to Delete");
Scan f ("%d" , & item );
10c=  -1;
for (i=0; i<size; i++)
if (A[i] = = item )
{
        loc  = i;
        break;
}

if (loc> =0)
{
        A [loc] =0
        / * shifting elements backwword * /
        For (i =loc; i<size -1; i++)
        A[i] =A[i+1];
        Print f ("Deletion successful");
Print f ("item is deleted from %d position" ,locc);
```

```
        Printf("After deletion Array is");
        For (i=0; i<size -1; i++)
        Print f(" % d" , A[i]);
}
        Else
{
        print f ("Deletion fallures");
        Print f ("item is not found");
}
}
```

- write a program to insert a particular item at the given location of the array .

```
#define N 100
main (  )
{
        int A[N] , i, item , loc;
        Printf ("Enter the Array size");
        Scanf ("% d" , & size);
        / * input Array * /
        for (i=0; i<size; i++)
{
        printf("Enter a no");
        Scanf ("%d" , &A [i]);
}
        / * printing Array * /
        Print f ("elements of Array");
        for (i=0; i<size; i++)
        printf ("%d" , A[i]);
        / * inserting begins * /
        printf ("Enter item to insert");
        scanf (" % d" , & item);
        Printf ("enter location from 0 to %d" ,
                                  size -1);
        scanf(" % d" , & loc);

        / * shifting elements * /
        for (i=size; i>=loc; i--)
        A[i] =A [i-1];
        A[loc] =item;
        / *printing array * /
        Print f ("After insertion, list is");
        for (i=0; i<size +1 ; i++)
        Print f (" % d" , A[i]);
}
```

Two Dimensional Array

}

When we use two subscript with array name to represent data in two direction, it is called two dimensional array or multidimensional array.Two dimensional array is called matrix.

The general format is

Data type Arrayname [row] [col];

int A[2] [3]

A[0][0] A[0][1] A[0][2]

1.			
2.			

A[1][0] A[1][1] A[1][2]

To access the element of two dimensional array .We use Array name with row and column.

EX:- If we want to access the element of third row and four column we use A[2] [3].

EX:- write a programe to input item in 2×3 array and print it

```
main ( )
        {
                int A [2] [3] , i, j;
                 for  (i=0; i<2; i++)
                 for  (j=0; j<3; j++)
        {
                printf("Enter a number");
                Scanf ("%d" , & A [i] [j]);
    }
                Print f ("__ out put___");
                for (i=0; i<2; i++)
        {
                 for (j=0; j<3; j++)
                Printf ("%d" ,  A [i] [j]);
                Print f ("\n");

        }
        }
```

- Write a programe to input icon in 3*3 matrix and print the sum row wise.

```
main ( )
 {

    int A[3] [3] , i, j; T, sum=0;
    for (i=0; i<3; i++)
    for (j=0; j<3; j++)
 {
    printf ("Enter a no");
    Scanf ("%d" , & A [i] [j]);

 }

    Printf ("----- out put ___");
    for (i=0; i<3; i++)
 {
    for (j=0; j<3; j++)
 {
    print f ("%d" , A [i] [j]);
    Sum = sum + A [i] [j];

 }
    Print f ("%d" , sum );
    Print f ("\n")
    Sum=0;
}
}
```

- Write a program to add the element of two array and assign in third array . Both array are same size.

```
    main (    )
 {
    Int  A[3][3] , B [3][3] , C[3][3];
    Int I,j;
    for (i=0; i<3; i++)
    for(j=0;j<3;j++)
 {
    printf ("Enter element for 1st matrix ");
    scanf ("%d" ,& A[i][j]);
}
    for (i=0; i<3; i++)
    for(j=o;j<3;j++)
```

```
{
        print f ("Enter element for 2nd matrix ");
        scanf ("%d" , & B[i][j]);

}       /* find addition of A & B and assign in C */
        for (i=0; i<3; i++)
        for(j=0;j<3;j++)
         C[i][j] = A[i][j] +B[i][j];
        Printf ("elements of 3rd array");
        for (i=0; i<3; i++)
{
        for(j=0;j<3;j++)
        Printf ("%d" , C[i][j]);
         printf("\n");
    }
}
```

- Write a program to find the subtraction of two matrices.

```
        main (    )
    {
        Int  A[3][3] , B [3][3] , C[3][3];
        Int I,j;
        for (i=0; i<3; i++)
        for(j=0;j<3;j++)
    {
        printf ("Enter element for 1st matrix ");
        scanf ("%d" ,& A[i][j]);
}
        for (i=0; i<3; i++)
        for(j=o;j<3;j++)
    {
        print f ("Enter element for 2nd matrix ");
        scanf ("%d" , & B[i][j]);

}   /* find subtraction of A & B and assign in C */
        for (i=0; i<3; i++)
        for(j=0;j<3;j++)
         C[i][j] = A[i][j] -B[i][j];
        Printf ("elements of 3rd array");
        for (i=0; i<3; i++)
{
        for(j=0;j<3;j++)
```

```
                    Printf ("%d" , C[i][j]);
                     printf("\n");
              }
       }
```

- Write a program to find the transpose of matrix.

```
                    main (    )
              {
                    Int  A[3][3] , B [3][3] ;
                    Int I,j;
                    for (i=0; i<3; i++)
                    for(j=0;j<3;j++)
              {
                    printf ("Enter element for  matrix ");
                    scanf ("%d" ,& A[i][j]);
          }

                    for (i=0; i<3; i++)
                    for(j=0;j<3;j++)
                     B[i][j] = A[j][i];
                    Printf ("elements of  2nd matrix");
                    for (i=0; i<3; i++)
          {
                    for(j=0;j<3;j++)
                    Printf ("%d" , B[i][j]);
                     printf("\n");
              }
       }
```

Initialization of 2 Dimensional array

We can initialize the two dimensional array in the following manner.

```
(I)       int A[2] [3] = {
                  {10, 15, 20}
                  {30, 40, 50 }

                 };
```

```
(ii)   int A [2] [3]  = {10, 20, 30, 40, 50, };
(iii)  int A [ ] [3]  = {10, 20, 30, 40, 50,60 };
```

If we want we can leave the row size but we must be column size.

MEMORY ARRANGEMENT OF MATRIX

Although matrix represent data in two direction but it store in the computer memory straight forward because computer memory cell is designed in linear fashion .
There are two common arrangement of matrix data in computer memory cell

1. Row major order
2. Column major order
   1. Row major order- In row major order, elements are stored row by row that is first row stored first, then second row and so on…

A[0][0] → A[0][1] → A[0][2]
A[1][0] → A[1][1] → A[1][2]
A[2][0] → A[2][1] → A[2][2]

If we want to calculate the address of A[i][j] in row major order, we can use the following formula.

B+(I*C*S)+(J*S)

Where B is base address of matrix, C is column size and S is byte size of each elements

For example, if we want to calculate the address of A[3][4] in matrix A[4][6] we use formula-

101+(3*6*2)+(4*2)
= 101+36+8
=145 (ans)

101	103	105	107	109	111
113	115	117	119	121	123
125	127	129	131	133	135
137	139	141	143	**145**	

A[3][4]

1. Column major order- In column major order, elements are stored column by column that is first column stored first, then second column and so on…

A[0][0] A[0][1] A[0][2]
A[1][0] A[1][1] A[1][2]
A[2][0] A[2][1] A[2][2]

If we want to calculate the address of A[i][j] in column major order, we can use the following formula.

B+(J*R*S)+(I*S)

Where B is base address of matrix, R is row size and S is byte size of each elements

For example, if we want to calculate the address of A[3][4] in matrix A[4][6] we use formula-

101+(4*4*2)+(3*2)
= 101+32+6
=139 (ans)

101	109	117	125	133	
103	111	119	127	135	
105	113	121	129	137	
107	115	123	131	**139**	

A[3][4]

STRING

An array of character is known as string. String is basically useful to store multiple characters at contiguous memory.

The general format is

Char stringname[size];

Eg. Char name[20];

Here name is a string variable of size 20 in which we can store 20 characters but we should use only 19 characters because last character is null terminator. Null terminator indicates end f string and represented by ‘\0’.

Initialization of string

We can initialize the string in the following manner.

1. Char name[20]= { ‘a’,’m’,’I’,’t’,’\0’};
2. Char name[ ]= { ‘a’,’m’,’I’,’t’,’\0’};
3. Char name[20]= “amit”;

   In this case null terminator is given by the compiler automatically.
4. Char name[ ]= “amit”;

String input function

There are three common function for string input

- Scanf()
- Gets()
- Getchar()

Scanf() :

Scanf() function can be used to input string through the keyboard with format specifier %s.

Eg.

```
Main()
{
Char name[20];
Printf(“enter your name “);
Scanf(“%s”,name);                 ← amit
Printf(“%s”,name);                → amit
Printf(“enter your full name “);
```

Scanf("%s",name); ← amit kumar
Printf("%s",name); → amit
}

Scanf() function will not accept blank space as input. As soon as we it get blank space , it will stop the reading.

To remove this problem we use gets().

Gets() :

This function is specially design for string input. It can take string with blank space as input.

The general format is:

Gets(string);

Eg.

```
        Main()
{
Char name[20];
Printf("enter your name ");
Gets(name);                 ← amit
Printf("%s",name);                  → amit
Printf("enter your full name ");
Gets(name);                 ← amit kumar
Printf("%s",name);                  → amit kumar
}
```

Getchar():

This function can input string character by character. When we input string character by character we must give null terminator externally.

The general format is :

Variable= getchar();

Eg.

```
        Main()
{
Char name[20],ch;
Int i=0;
Printf("enter your name ");
Do
{
ch= getchar();
name[i]=ch;
i++;
}
While(ch!='\n');
Name[--i]= '\0';
```

```
Printf("%s",name);
}
```

String output function

There are three common function of string output.

1. printf( )
2. puts( )
3. putchar( )

1. **Printf ( ) :-** The function can be print the string with the help of format specifier % s.

   Eg.

```
      Main()
{
Char name[20];
Printf("enter your name ");
Gets(name);                          ← amit
Printf("%s",name);                   → amit
  }
```

2. **puts ( ) :-** This function is specially design to print the string on the screen . It is just similar to the printf only difference is that puts print the string and change the line automatically.

   The general format is

   Puts (string);

   EX:-

```
main ( )
{
        char name [20] , add [20];
        Print f ("Enter your neme");
        gets (name);                         ← amit
        Print f ("Enter address");
        Gets (add);                          ← patna
        Print f ("%s" , name);
        Puts (add);
        Print f ("%s" , name);
}
```

Output Amit patna amit

3. Putchar ( ):- This function is used to print the string character by character.

The general format is

Putchar (char);

EX:-

```
main (  )
{
        char name [20] ;
        int i;
        printf ("Enter your name");
gets(name);
        i  = 0;
        While (name [i] ! =  '\0');
{       putchar(name[i]);
        I++;
}

}
```

String manipulation function

'c' provides a number of library function used for string manipulation like concatenating two strings ,copy strings ,reverting string etc. All these functions are defined in string.h header file.

following are some common string manipulation function.

1. strlen( ):→ The strlen ( ) function with count the length of strings.

The general format is

var = Strlen (string);

ex.

```
main()
{
        char name [10] = "Amit"
        int num ;
        num = strlen (names);
        printf("%d",num);               → 4
        printf("%d",strlen("names"));         → 5
}
```

NOTE: → The strlen function will not count the null terminator.

2. Strcat ( ):→ This function is used to concatinate two strings.

The general format is

stract (string1,string2);

+

Here string 2 is source string and string 1 is destination string. string2 will be added with string1.

ex.

```
main()
{
        char  fname [20] = "Amit";
        char  lname [10] = "Kumar";
        printf("%s",fname);             → amit
        printf("%s",lname);             → kumar
        strcat(fname,lname);
        printf("%s",fname);             → amitkumar
        printf("%s",lname);             → kumar
        stract(fname, "roy");
        printf("%s",fname);             amitkumarroy

}
```

NOTE :→ The stract function will not provide any blank space automatically between the strings. Actually first character of source string will be appear at the null terminator of target string.

3. strcpy ( ):→ This function is used to copy one string into the another string.

The general format is

strcpy (string 1, string2);

Here string2 will be copyed on string1

ex.

```
main()
{
        char  fname [20] = "Amit";
        char  lname [10] = "Kumar";
        printf("%s",fname);             →amit
        printf("%s",Lname);             →kumar
        strcpy(fname,Lname);
        printf("%s",fname);             → kumar
        printf("%s",Lname);             →kumar
        strcpy(fname, "patna");
        printf("%s",fname);             → patna

}
```

4. Strcmp ( ):→ This function is used to compare two strings . Strcmp compare two strings character by character and returns the difference of ASCII value. If strcmp returns 0 it means both string are equal.

The general format is

Strcmp(string1,string2);

ex.

Strcmp("Anil" ,"Amit");

n-m

= 1

If strcmp returns non zero ,it means strings are not equal.

5. strrev( ):→ This function is used to reverse the string.

The general format is

Strrev(string);

ex.

Strrev("Amit"); **→ timA**

- write a programme to count the length of string without using library function.

```
void main( )
{
        int count = 0,
        int i=0;
        char string [20];
        printf("Enter one strings");
        gets(string);
        while(string[i] != '\0')
{   i++;
Count++;
}
printf("Number of character in given string  = %d", count);
}
```

- write a programme to reverse the string without using library function.

```
        void main( )
    {
        int length,i;
        char string [20];
        printf("Enter one strings");
        gets(string);
        length= strlen(string);
        for(i=length-1; i>=0; i- -)
        putchar(string[i]);
}
```

Function

Function is an object which has a certain functionality to perform a particular task. The main advantage of function is it's reusability and we can divide a large and complex program into several small modules to reduce the complexity.

There are two types of function.

1. predefined function
   or
   library function
   or
   Built in function
2. user defined function

1. predefined function:→ A function which is all ready defined and readymade available for use, is called predefined function or library function. we can use any library function but we can't modify it's functionality. The library function is defined in the related header file, hence before using it we must include appropriate header file.

following are some common header file used in 'c'.

(a) <stdio.h> :→ This header file contains all standard input output function such as printf(), scanf(), puts(), gets(), putchar() etc.

(b) <conio.h>:→ This header file contains various console input output function such as clrscr(), getch(), gotoxy( ) etc.

gotoxy():→ This function is used to set the cursor at the specified row and column on the output screen.

syntax- gotoxy(row,column);

gotoxy(20,10);

(c) <string.h>:→ This header file contains various string manipulation function like strcat(),strlen(),strcmp().

(d) <math.h>:→ This function contains the definition of various mathematical function.

Following are some common mathematical functions:-

Function	**Meaning**
Sin (x)	sin value of (x)
cos (x)	cos value of (x)
tan (x)	tan value of (x)
sqart (x)	square root of (x)
pow (x,y)	x to the power y (x)
ceiling (x)	x rounded up to nearest integer
floor (x)	x rounded down to nearest integer
fmod (x,y)	x remendery
fabs (x)	absolute value of x.

(e) <dos.h>:→ This header file contains various Dos based function like delay( )

delay( )→ This function hold the cursor in output mode specified time. It accept time in millisecond.

ex:→
```
for(i=1;i<=100;i++);
{
printf("%d",i);
delay(1500);
}
```

(f) <aloc.h>:→ This header file contains the definition of various dynamic memory allocation function like

calloc( )
malloc( )
free( )

(g) <stdlib.h>:→ This header file contains the definition of various character manipulation function.

ex. exit(0);

(h) <ctype.h>:→ this header file contains the definition of various character manipulation function.

ex.

isupper() → To check the given character is in uppercase?

Isupper('a') → 0

Isupper('P') → 1

islower() →To chek the given argument in lower case?

isspace() → To check the give argument is blank space?
Isalnum() → check the given argument is alphanumeric?
toupper() → Convert the character in upper case.
Eg. toupper (‘a’) → A
tolower() → Convert character in lower case.
tolower (‘A’) → a
toascii() → converts character to its ASCII value.
atoi( ):→ Convert string to integer.

User defined function

An user defined function is defined by the user according to their need .The main advantage of user defined unction is its reliability.

ex.

```
        void show ( );          ——————→ Declaration/prototyping
        main ( )
        {
                printf (“In main”);
                printf (“Go in show”);
                        show ( );
calling         printf (“Again go in show”);
                        show ( );
                printf (“Bye”);
        }

        void show ( )           ——————→ Decelerator/definition
    {
                printf (“Welcome in show”);
                printf (“How are you”);
                printf (“Thanks”);
        }
```

In userdefind function we get three major points.

1. Declaration: → We know that we can’t refer anything without its proper declaration. hence before using user defined function we must declare it also called prototyping.

The general format is
Returntype fun_name( );

2.calling:→ Calling means to transfer the control from calling function to called function and After the execution of called function ,control will be again transfer from called to calling .To call a function we use calling statement.

The general format is
Fun_name ( );

3. Decelerator:→ Decelerator is used to define a function. it indicates what operation to be performed by the function.

The general format is

```
Return_type  function_name()
        {
          Function
                        body
          }
```

NOTE:→ Decelerator must be similar to the declaration but it can't be terminated by semi colon(;)

Note1:→ Decelerator must be similar to the declaration but it can't be terminated by semi colon(;)

Note2:→ We can also declare a function inside the other unction . In this case the scope of function becomes local.

Note3:→ We can also define a function before the calling function In this case declaration is not needed.

Note4:→ We can't define a function inside the other function .

Note5:→ A function can call it self inside it's body , it is called recursive function.

Type of function

According to return type and arguments, function can be categories in four major categories.

1. no argument and no returns
2. accept argument but not return
3. no argument but return
4. accept argument and also returns

1. no argument and no return

A function which will not accept any value as an argument and also not return any value to its caller. In this type of function we write void as a return type and parenthesis will be empty.

ex.

```
void main( )
{
        sum ( )
}
void sum ( )
{
        int a, b, c;
        printf ("Enter two no");
        scanf ("%d%d",&a,&b);
                c = a+b;
        printf ("%d",c);
}
```

2.) accept argument but not return

A function which will accept some values as argument but not return any value.
In this type of function we write void as a return type but parenthesis can't be empty. we write the data type of arguments in the parenthesis.

ex.

```
void sum( int, int);
main ( )
{
        int a, b, ;
        printf ("Enter two no");
        scanf ("%d%d",&a,&b);
        sum(a,b);
}

        void sum( int p, int q)
        {
                int r;
                r = p+q;
                printf ("%d",r);
        }
```

Note:→ we can use the same name of argument in the calling function and in the called function or we can change it but we can't change number of arguments, data types of argument and sequence of argument.

- write a program to input two number and print the addition , subtraction, multiplication and division using separate function for it.

```
        void sum (int, int);
        void sub (int, int);
        void mul (int, int);
        void Div (int, int);
main ( )
{
        int a,b;
        printf("Enter two no.");
        scanf ("%d%d",&a,&b);
                sum (a,b);
                sub (a,b);
                mul (a,b);
                Div (a,b);
}
        void sum (int a , int b)
                {
                        int c;
                        c = a+b;
                        printf("sum=%d",c);
                }
        void sub (int a , int b)
                {
```

```
                    int c;
                    c = a-b;
                    printf("subtraction = %d",c);
                }
        void mult (int a , int b)
                {
                    int c;
                    c = a*b;
                    printf("multiplication=%d",c);
                }
        void Div (int a , int b)
                {
                    int c;
                    c = a/b;
                    printf("Division= %d",c);
                }
```

- Write a program to input a number and print the table using function.

```
void table ( int);
main ( )
{
        int num ;
        printf ("Enter a number ");
        scanf ("%d",&num );
                table (num);
}
        void table ( int num);
        {
                int I;
                for(i=1;i<=10;i++)
                printf ("%d",(num*i));
                }
```

- write a program to define a function which will accept three number and print the largest value.

```
void max (int , int , int,);
main (   )
{
            int a, b , c;
            print f ("Enter three number")
            scan f ("%d %d %d" , & a, &b, &c);
            max (a, b, c);
    }
            void max (int a, int b, int c)
     {
    Int m;
            if  (a> b)
            if (a>c)
```

```
                m=a;
            else
            m = c;
        else  if  (b>c)
                m = b;
            else
                m =c;
                print f ("Largest value is %d" , m);
            }
```

- write a program to input first and second term and print the Fibonacci series using function.

```
    void fibo (int , int );
        main (   )
    {
        int a , b;
         print f ("Enter first and second terms");
        scanf ("%d %d" , &a , &b);
        fibo (a, b);
        void fibo (int a , int b)
        {
        int c , i;
        print f ("%d %d" , a, b );
        for (i=1; i< =8; i++ )
        {
                c = a+b;
                print f ("%d" , c );
                a = b;
                b = c;
 }
   }
```

3 . Accept argument and also return

A function which will accept some values as an argument and also return a value. Generally in this of function, we can't write void as an return type . We write the data type of value which will be return by the function as return type.

Ex:-

```
            int sum ( int , int );
            main (  )
    {
                int a , b, c;
                print f ("Enter two no");
                scan f ("%d %d" , &a , &b );
                c  = sum (a , b );
```

```
print f (" %d" , c );

}
int sum (int a, int b )
{
    int c;
c =a +b ;
    retutn  c ;
}
```

Point related to return type :-

>> To return a value from a function, We use return keyword. These are valid format of return.

```
return;
return 0;
return 50;
return a;
return  ( a + b );
```

- Although a function may accept more than one value as argument but it always return only one value.
- We should write a return statement as the last statement in the called function because as soon as return is executed control will be transfer from called to calling.
- Although we can write multiple return statement but only one return is executed.

Eg.

Write a program to define a function which will accept the radius of circle and return the area.

```
Float area (int );
main (  )
{
        int  r ;
        float a;
        print f ("Enter the radius" );
scanf  ("% d" , & r ) ;
a = area ( r );
        print f ("area of circle= % f" , a);
}
        float area (int r)
{
        Float a;
a = 3.14*r*r;
        return  a;

}
```

- Write a program to define a function which will accept a number and return the reverse of digits.

```
              int  rev (int);
main (   )
              {
                        int num , r;
              print f (“Enter a number”);
              scan f ( “%d” , & num );
                         r = rev (num );
                        print f (“%d” , r );
   }
                        int rev (int num )
              {
                        int r =0, temp;
                        While ( num > 0)
              {
                        temp = num % 10;
                        r = (r * 10) + temp;
                        num = num / 10;
    }
                         return  r;
              }
```

- WAP to input a no and check it is odd or even. if it is odd call the factorial function otherwise call the table function.

```
               int fact (int);
void Table (int);
main (   )
     {
          int num ;
          print f (“enter a no”);
          scan f (“%d” , & num );
     if (num % 2 ! = 0)
     print f (“ %d” , fact (num) );
else
          Table  ( num );
 }
          int Fact (int num )
     {
          int i , f = 1;
          for ( i = num; i>1; i-- )
           f = f * i;
```

```
        return f;
}
        void table (int num )
    {
        int i;
        for ( i = 1; i<=10; i++)
        print f (“%d” , ( num * i) );
}
```

Write a program to define a function Which will accept a number and return the factorial.

```
        int fact (int);
         main (  )
    {
         int , num , f;

         print f (“Enter a no”);
        scan f (“%d” , & num );
        f = fact ( num );
        print f (“%d” , f );
  }
         int fact (num )
    {
        int i , f = 1;
        for ( i = num ; i>1; i-- )
        f=f* i;
         return f;
 }
```

Write a program to define a function which will accept a number and return the sum of digit.

```
         int sum ( int );
        main (  )
    {
        int num, s;
        printf (“Enter a no”);
         scanf (“ %d” , &num)
        s = sum( num );
         print f (“ sum = % d” , s );
  }
        int sum (int num)
     {
        int  temp , s = 0;
        While  ( anum> o )
{
        temp = num%10;
        s= s + temp;
```

```
                num=num/10;
        }
                return s;
            }
```

4 . No argument but return value

A function which will not accept any value argument but can return a value. Generally in this types of function we can't write void as a return type, we use data type as a return type.
Eg.

```
int sum();
void main()
{
int c;
c= sum();
printf("%d",c);
}

int sum()
{
int a,b,c;
printf("enter two number :");
scanf("%d%d",&a,&b);
c= a+b;
return c;
}
```

CALLING A FUNCTION

We can call a function with two different mechanism.
(i) call by value
(ii) call by address (call by reference)

(1) call by value :-

```
        void change (int , int );
         main  (  )
    {
                int a , b;
                printf ("Enter two no");
                scanf ( " % d % d" , &a, &b);
                 printf (" % d % d" , a , b );
```

```
                change(a , b);
                printf(" % d % d" , a , b );
        }
                 void change (int a , int b )
    {
                print f (" %d % d" , a ,b );
                a = a + 10;
                 b = b +20;
                print f (" % d %d" , a , b );
      }
```

Output
10 15
10 15
20 35
10 15

When we use call by value mechanism a Carbon copy of argument will be passed in the called function which is stored in separate variable, Hence any changes occurred in the called function will not effect the original variable in the calling function.

(ii) Call by address: - When we use call by reference mechanism actually we pass the address of variable Hence any changes occurred in the called function will also effect the original variable in the calling function.

To implement the call by refresh mechanism ,we use concept of pointer.

Ex:- void change (int * ,int *);

```
    main (  )
   {
            int a , b;
            a = 10;
            b = 15;
            print f (" % d % d" , a , b);
            change ( &a , &b );
            print f (" %d % d" , a , b );
   }
            void change (int *a , int *b)
{
            print f (" %d %d" , *a , *b);

            *a = *a + 10;
            *b = *b + 20;
    print f (" % d % d" , *a , *b );
    }
```

Output
10 15
10 15
20 35
10 15

Array and function

As any normal variable we can also pass on entire array as function argument.

When we pass an array as function argument , we pass only the array name . The name of the array is the constant pointer which will hold the address of base element and we know that the memory allocation for array will be done in continuous manner , Hence function can access next element automatically .

When we pass an array , actually we pass the address of first element hence it is always case of call by address it means any changes in the called function will also change the original array of the calling function .

Eg:-
```
void change ( int [  ] );
main  (   )
{
    int A [5] = { 10,20,30,40,50};
    int i;
    for ( i = 0; i<5; i++ )
    printf (" %d" , A [i] );
    change( A );
    for ( i = 0;  i<5; i++ )
    printf(" %d" , A [i] );
}
void change ( int A[ ] )
{
    int i;
    for ( i = 0; i<5; i++ )
{
    A [i] = A [i] + 5
    print f (" %d" , A [i] );
    }
}
```

- Write a program to define a function which will accept an array and return the sum of elements .

```
int sum ( int [ ]);
main ( )
{
    int A [10] , i , s;
    for ( i = 0; i<10; i++ )
{
    print f ("Enter a no");
    scan f (" %d" , & A [i] );
}
```

```
print f ("____ element of Array  ____" );
 for (i = 0; i<10; i++ )
 print f (" % d" , A [i]);
s = sum (A);   }
print f ("sum of elements = %d" , s );
}
int sum (int A [ ] )
{
 int s = 0 , i;
 for ( i = 0; i<10; i++)
s = s+A[i];
return s;
}
```

- Write a program to define a function which wll sort an array.

```
# define N 10
void sort ( int [ ] );
main (  )
{
int A [N] , i;
 for ( i = 0; i<N; i++ )
{        print f (" Enter a no" )
scan f (" % d" , & A [ i ] );
}
print f ("original Array" );
for ( i = 0; i<N;  i++ )
print f (" % d" , A [i] );
sort ( A );
print f ("After sorting");
 for ( i = 0; i<N; i++ )
print f (" % d" , A [ i ] )
}
void sort ( int A [ ] )
{
int i , j , temp;
 for ( i = 0; i<N-1;  i++ )
 for ( j = 0; j<N-1-i;  j++ )
if  ( A [j]> A [j+1] )
{
 temp = A [j];
A [j] = A[j+1];
 A [j+1]  =  temp;
}
```

```
}
```

- Write a program to define a function which accept an item and search it in the array.

```
# define N 10
int search ( int [ ] , int );
void main ( )
{
int A [ N ] , i , loc, item;
for ( i = 0; i<N; i++ )
{ printf ("Enter a number");
scanf ("%d" , & A [i] );
}
printf (" elements of Array");
for ( i = 0; i<N; i++)
 printf (" %d" , A [i]);
printf ( " Enter item to search" );
scan f ("%d" , & item );
loc = search ( A , item );
if ( loc > 0 )
{printf ( " searching successful" );
printf ("item is present in %d location",loc );
}
else
{ printf ("searching failure");
printf ("Item is not present in Array");
}
}
int search (int A [ ], int item )
{
int i , loc = 0;

for ( i = 0; i <N; i++ )
if ( A [i] = = item )
{ loc = i+1;
break,
}
return loc;
}
```

- Write a programme to define a function which will delete on item from thr array

```
# define N 10
void delete ( int [ ] , int );
```

```
main (   )
{
int A [ N ] , i , item;
for ( i = 0; i<N; i++ )
{ printf (“Enter a no”);
scanf (“ % d” , & A [i] );
}
printf (“ original Array”);
for ( i = 0; i<N; i++)
printf (“ %d” , A [i]);
printf ( “ Enter item to delete” );
scan f (“%d” , & item );
Delete ( A , item );
}
void Delete (  int A [] , int item )
{ int i , loc = -1;
for ( i = 0; i<N; i++ )
if ( A [i] = = item )
{ loc = I;
break ;
}
if ( i>=0 )
{ A [loc] = 0;
for ( i = 10c; i<N-1; i++ )
A [i] = A [i+1]
A [i] = 0;
printf (“Deletion successful”);
printf (“item is deleted from % d location”, loc + 1 );
printf (“After Deletion” );
for ( i = 0; i<N-1; i++ )
printf (“ %d”, A[i] );
}
else
{  printf (“Deletion fallure” );
print f (“Item is not present”);

}
}
```

- Write a program to define a function which will insert an item in the array in the given location.

```
# define max 100
void insert ( int [ ] , int , int );
```

```
void     main  (  )
{
    int A [ max ] , i , loc ,N , item;
    print f ( " How many item you want to store" );
    scan f ( " %d", &N );
    for ( i = 0; i<N ; i++ )
    {print f ( "Enter a no" );
    scan f ( " %d" , & A [i] );
    }
    print f ( " original Array");
    for ( i = 0; i<N; i++ )
    print f (" %d", A [i] );
    print f ("Enter item Which you want to insert" )
    scan f (" %d" , & item );
    print f (  "Enter location Where you want to insert" );
    scan f (" % d", & loc );
    if ( (loc < 0 ) | | ( loc > N ) )
    {
        printf ( "Insertion failure");
        print f ( "Invalid location" );
        exit ( 0 );
    }
    insert( A ,loc , item );
    print f ("After Insertion" );
    for ( i = 0; i < N+1; i++ )
    print f ( " % d", A [i] );
}
void insert ( int A [ ], int locc ,int item )


{ int i;
for ( i = N; i > loc; i-- )
A [i] = A [i-1];
A [loc] = item;
}
```

Storage class

Storage class indicates the prime behavior of variables like memory allocation of the variable , scope and visibility of variable, life time of variable and about the initialization of variable .

To declare a variable we use following general format.

Stoage_class Data_type variable_ name;

There are five types of storage class.
(i) Automatic or local.
(ii) static
(iii) Global
(iv) Register
(v) external

1. Automatic or local:→ By default every declare variable is automatic or local . although we can use a keyword auto .

ex.
```
        int a; √ (Automatic variable)
        auto int a; √ (Automatic variable)

        void show();
        main( )
{
        int a,b;
        a =10;
        b =15;
        printf("%d%d" ,a,b);            →10  15
        show ();
        printf("%d",c);                 → error
}
        void show ()
{
        Int b,c
        C=50;
        Printf("%d",a);         → error
        Printf("%d",b);         → garbage value
        Printf("%d",c);         → 50
}
```
Local variable gets a temporary memory allocation . The scope and visibility of local variable is restricted only inside the function block where it is declared. The life time of local variable is equal to the life time of function.

If we will not initialize a local variable , compiler will not initialize it automatically ,hence it produce garbage value.

(2.) static :→
```
void change ( );
main ( )
```

```
{
        change ( );
        change ( );
        change ( );
}
        void change ( )
{
        int a = 50;
        a = a+10;
        printf ("%d",a);
}
```

Output

60
60
60

```
        void change ( );
        main ( )
{
        change ( );
        change ( );
        change ( );
}
        void change ( )
{
        static int a = 50;
        a = a +10;
        printf ("%d",a);
}
```

Output

60
70
80

static variable gets permanent memory allocation . The scope and visibility of static variable is also restricted inside the function block where it is declared. The life time of static variable is equal to the life time of program.

static variable initialize only ones in the program . If we will not initialize the static variable compiler will automatically initialize it with 0.

static variable is useful when we want a variable which will retain the value after the termination of function.

(3.) Global :→

```
int a;
void show ( );
main ( )
printf ("%d",a); →0
a = 50;
print f ("%d",a);        →50
show ();
print f ("%d",a);        →90
```

```
}
        void show ( );
{
        printf ("%d",a); →50
        a = 90;
        print f ("%d",a);        →90

}
```

A variable which is declared outside the function, it is called global variable . global variable gets permanent memory allocation .The scope and visibility of global variable is throughout the program. The life time of global variable is equal to the life time of program.

If will not initialize the global variable compiler will automatically initialize it with 0.

Global variable can be accessed by all the function which is declared after the declaration of global variable.

(4.) Register:→

when we declare a variable sufficient memory will, be allocated for the variable in systems memory. Each time when we want to access the variable ,compiler will such it in the memory . Memory is large area may be compiler take some time to search it, due to this program execution becomes slow.

If we want we can save this time if we declare a variable as Register. Register is small storage area which hold some important data temporarily.

ex. Register int a;

Note :→ We can declare only int or char as register variable .

Note :→ We can't declare more the two or three variable as register.

5) external

External variables are used in multi programming environment.

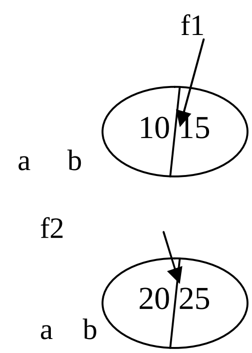
External variable declared in one program can be accessed in another program.

Eg.

extern float pi= 3.14;

STRUCTURE

```
        struct first
{
        int a;
        float b;
};
        main ( )
{
        struct first f1, f2;
        printf ("Enter one int and one float");
        scanf ("%d%f",&f1.a,&f1.b); → 10 15
printf ("Enter one int and one float");
scanf ("%d%f",&f2.a,&f2.b); →20 25
printf("------output------");
printf ("%d%f",f1.a,f1.b);→10 15
printf ("%d%f",f2.a,f2.b);→ 20 25
}
```



Points related to structure :

1. structure is collection of more than one variables which may be some type or different types.
2. structure creates an user defined data type of its own type.
3. To declare a structure , we have to use a keywords struct.

The general format is

```
        Struct  struct_name
{
        Data type var1;
        Data type var2;
                .
                .
                .
        Data type varn;
};
```

4. The declaration of structure is treated as a single statement hence it must be terminated by semi colon.
5. The declaration of structure is only a blue print or template. There is no any memory allocation for the member of structure.
6. To use the member of structure we have to create structure variable. To create structure variable we use structure name as a data type. The general format is: Struct struct_name variable1,variable2….. variable n;
7. As soon as structure variable is created, sufficient memory will be allocated for the variable.
8. The amount of memory for structure variable is depend on the member of structure.
9. Every structure variable gets a separate memory allocation according to its member which is not overlapped by each other.
10. To access the member of structure we use structure variable with the help of dot operator.

.

Characteristics of structure

1. Initialization of structure- we can initialize the structure in the following manner.

```
Struct first
{
Int a;
 float b;
};

Void main()
{
Struct first f1= { 10,15.50};
Struct first f2= { 20,25.50};
Printf("%d%f",f1.a,f1.b);     → 10 15.50
Printf("%d%f",f2.a,f2.b);     →20 25.50
}
```

2. we can also create structure variable at the time of structure declaration.

```
   Struct first
{
Int a;
 float b;
}f1,f2;

Void main()
{
F1.a=10;
F1.b=15.50;
F2.a=20;
F2.b=f1.a+f1.b+f2.a
Printf("%f",f2.b);            →45.50
}
```

3. we can also create structure without its name in this case we must create all the structure variable at the time of structure declaration.

```
        Struct
{
Int a;
 float b;
}f1,f2;

Void main()
{
F1.a=10;
F1.b=15.50;
F2.a=20;
F2.b=f1.a+f1.b+f2.a
Printf("%f",f2.b);              →45.50
}
```

Array of structure

We can also create an array of structure variable so that it can store Multiple set of values.

Ex:- write a program to input roll , Name and fee of 10 students and print it in tabular format.

```
# define N 10
   struct student
{
        int roll;
   char name [20];
        floot fee;
 }
     Main (    )
 {
```

```
struct student S[N]
int i;
for ( i = 0; i<N; i++ )
{
    printf (“Enter roll” );
    scanf ( “ %d” , &S[i].roll );
    printf ( “ Enter Name” );
    scanf (“ % s”, s [i].name);
    printf (“Enter fee” );
    scanf (“ % f” , &S[i].fee );
}
printf (“___ student details ____” )
printf ( “\ n Roll \t\ t name \t\ t Fee” );
for ( i = 0; i<N; i++)
    printf (“ \n % d \t\t %S\t\ t % f” ,S[i].roll, S[i]
                                   .name , S[i].fee );
}
```

structure and function

We can pass a structure variable as function argument.

Ex:-

```
struct first
{
    int a;
    floa b;
};
void sum ( struct first , struct first );
main ( )
{
    struct first F1 = { 10, 15.50 };
    struct First F2 = { 20, 25.50 };
    print f ( “ % d % f” , F1.a , F2.b )
    print f ( “ % d % f” , F2.a, F2.b );
    sum ( F1, F2 );
}
void sum ( struct first F1 , struct first F2 )
{
    struct first F3;
    F3.a = F1.a + F2.a;
    F3.b = F1.b + F2.b;
    print f (“ % d % f” , F3.a , F3.b ); →30 41.00

}
```

We can also return a structure variable through the function.

```
struct first
{
    int a;
    float b;
} ;

struct first sum ( struct first , struct first );
```

```
main (  )
{
     struct first F1 = { 10,15.50 };
     struct first F2 = { 20 , 25.50 };
     struct first F3;
     F3 = sum( F1, F2 )
     printf (  " % d % f", F1.a , F1.b ) →10 15.50
     print f ( " % d % f", F2.a , F2.b ) → 20 25.50
     print f (" % d % f", F3.a, F3.b ) → 30 41.00
}
     struct first sum ( struct first F1 , struct first f2 )
{
     struct fist F3;
     F3.a = F1.a + F2.a;
F3.b = F1.b + F2.b;
     return F3;
}
```

Array structure and function

We can also pass an array of structure variable as function argument.

Ex:- write a program to to define a function Which will accept Roll , name and fee of 10 student and sort it According to Roll.

```
# include < string . h >
# define N 10
struct student
{
        int roll;
    char name [20];
    float fee;
};

    void sort ( struct student [ ] );
    main (    )

    {
        struct student S [10];
        int i;
    for ( i = 0; i<N; i++ )
    {
        printf ( "Enter roll name and fee");
scanf ( "%d%s%f",&S[i].roll,S[i].name,&S[i].fee );

        }
        printf ( "___ original data base____" );
        print f ( "\n Roll \t Name\t fee" );
        for ( i = 0;i<N; i++ )
printf (" \n%d\t%s\t%f\t",S[i].roll,S[i].name,S[i].fee);
```

```
sort (S);
print f ( "___ sortest data base ____" )
print f ( "\n Roll \t Name\t fee" );
for ( i = 0;i<N; i++ )
printf (" \n%d\t%s\t%f\t",S[i].roll,S[i].name,S[i].fee);
}

void sort ( struct student S[N] )
{
struct temp;
int i,j;
for (i = 0; i<N-1; i++ )
for ( j = 0; j<N-1-i; j++ )
if (S[j].roll >S[j+1].roll )
{
temp. roll = S[j] . roll;
strcpy (temp.name ,S[j].name );
temp.fee = S[j] . fee;
S [j]. roll = S[j+1] . roll;
strcpy (S[j].name , S[j+1].name );
S[j].fee = S[j+1] .fee;
S [j+1] . roll = temp. roll;
strcpy (S[j+1] . name , temp .name );
S [ j+1 ] . fee = temp . fee;
}
```

* WAP to input employee code , employee Name and salary . Define a function Which will accept the employee data base and check the salary one by one. if salary > = 7000 , Give increment for 20 % ,if salary > 5000 Give increment 15 % other wise give increment 10 % .

```
#define  N  10
struct emp
{
int code;
char name [20];
int sal;
};
void increment (struct emp []);
main ( )
{
int i;
struct emp E[N]
for (i=0; i<N; i++)
{
printf ("Enter ecode name and salary");
scanf ("%d%s%d"&E[i].code,&E[i].name,&E[i].sal);

}
printf ("----------original employe record------------");
printf ("\n emp_code\t name\t salary");
for (i=0;i<n;i++)
```

```
printf("\n%d\t%s\t%d",E[i].code,E[i].name,E[i].sal);
                    increment (E);
printf ("----------after increment records are---------------");
                    for (i=0;i<n;i++)
printf("\n%d\t%s\t%d",E[i].code,E[i].name,E[i].sal);
}

void increment (struct emp E [])
               {
                              int i, bon;
                              for (i=0;i<N; i++)
                        {
                    if (E[i].sal >=7000)
                              E[i].sal = (E[i].sal+(E[i].sal *20)/100;
                    Else if(E[i].sal>=5000)
                              E[i].sal = (E[i].sal+(E[i].sal *15)/100;
                    Else
                              E[i].sal = (E[i].sal+(E[i].sal *10)/100;

                    }
                    }
```

Structure within structure
or
Nested structure

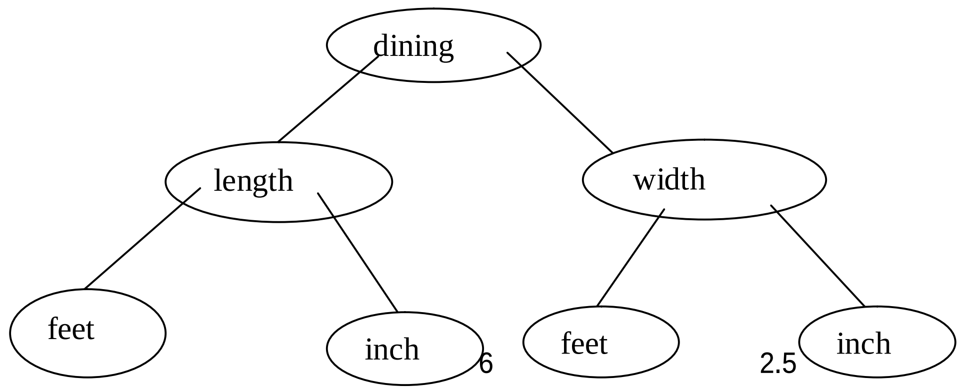
If structure is called structure within structure.

```
                              structure distance
                    {
                              int feet;
                              float inch;
                    };

                    struct room;
                    {         struct distance length;
                              struct distance width;
                        };
                              main ( )
          {
                              struct room dining
                              float Area;
                              dining. Length. feet = 7;
                              dining. Length.inch = 3.5;
                              dining.width. feet = 6;
                              dining.width. inch = 2.5;
Area = ((dining. Length. Feet*12) + dining. Length. inch) * ((dining. Width. Feet*12) + (dining.
Width. inch))
printf("Area of room = %f", Area);
                    }
```

In a case of Nested structure to access the target, We use dot operator twice.

Fig 1.



UNION

Union is a similar to the structure only difference in the case of memory allocation. Structure variable gets a separate memory for its each member Where as union gets a common memory which is shared by its all member.

Eg.

```
        struct first
    {
        int a;
        float b;
    };
        main( )
    {
        struct first f1;
```

```
                f1.a =10;
                f1.b = 15.50;
                printf ("%d%f",f1.a,f1.b); → 10, 15.50
}
                union first
        {
                int a;
                float b;
        };
                main( )
        {
                union first f1
                f1.a = 10;
                f1.b =15.50;
                printf ("%d%f",f1.a,f1.b); → 15, 15.50
        }
```

DIFFRENCE BETWEEN STRUCTURE AND UNION

1. structure variable gets a separate memory allocation according to its member where as union variable gets a common memory which is shared by all its member.
2. Union variable basically access its member one by one
3. union variable provides the maximum utilization of memory rather than structure variable.

POINTER

Pointer is a variable which will hold the address of other variable of type for which it is declared.

The general format is
Data type * pv;
Ex. int*p;

Here P is a pointer variable which will hold the address of integer variable.

float*q;

Here q is a pointer variable which will hold the address of float variable.

The data type of pointer is pointer itself .normally pointer variable gets two byte memory to hold the address.

Address operator

when we creat[illegible], suficient memory will be allocated to the variable. If we want we can know the address of variable with the help of address operator.

```
main ( )
{
        int a;
        a = 10;
        b = 15.50;
        printf ("%d%f",a,b); → 10,15.50
printf ("%u%u",&a,&b); → add of a,add of b
}
```

NOTE:→ The address of variable will be given in hexadecimal format.
To store the address of variable we use pointer variable

```
main ( )
{
int a,*p;
float b,*q;
a = 10;
b = 15.50;
p = &a;
q= &b;
printf ("%d%f",a,b);        10,15,50
printf ("%u%u",&a,&b);          add of a,add of b
printf ("%u%u",&p,&q); → add of a, add of b
}
```

Dereferencing with pointer

we can also access the value of variable through the pointer with the help of dereferencing (*) operator.

ex.

```
main ( )
{
    int a,*p;
    float b,*q;
    a = 10;
    b = 15.50;
    p = &a;
    q= &b;
    printf ("%d%f",a,b);          → 10, 15.50
    printf ("%u%u",p,q);→ add of a, add of b
    printf ("%d%f",*p,*q);        → 10,15.50
        *q = *q+*p;
    printf("%f",*q);              → 25.50
    printf("%f",b);               → 25.50
        *p = *p + 50;
    printf ("%d",*p);             → 60
    printf ("%d",a);              → 60
}
```

Array of pointer

we can also create an array of pointer to hold the multiple address.

The general format is

```
Data type *p [size];
main ( )
{
    int A[5] = {10,20,30,40,50};
    int *p[5],i;
    for (i=0;i<5;i++);
        p[i] = &A[i];
        for ( i= 0; i<5; i++)
        printf("\n%u\t%d", p[i],*p[i]);
}
```

pointer and function

As any normal variable we can also pass a pointer variable as function argument. when we pass a pointer as function argument , actually we pass the address of variable hence it is always case of call by address it means any changes by the called function will also change the original value of calling function.

```
void change (int*,int*);
    main ( )
    {
        int a,b;
        int *p,*q;
        p = &a;
        q = &b;
        printf("Enter two no");
        scanf ("%d%d",&a,&b);   ← 10, 15
        printf("%d%d",a,b);  → 10 , 15
```

```
        change (p,q);
    printf ("%d%d",a,b); → 20 35
}
  void change (int *a, int *b)
  {
    printf ("%d%d",*a,*b); →10, 15
           *a = *a +10;
           *b = *b +20;
    printf("%d%d",*a,*b); → 20, 35
  }
```

- write a program to swap the value of variable using pointer and function.

```
void swap (int*, int*);
        main( )
{
          int a,b;
        printf ("Enter two no");
        scanf ("%d%d",&a,&b);
        printf ("a=%d,b=%d",a,b);
             swap (&a,&b)
  printf("After swaping");
        printf ("a=%d,b=%d",a,b);
}
          void swap (int*a, int*b);
{
          int c;
          c = *a;
          *a = *b;
          *b = c;
}
```

Array pointer and function

Array and pointer are highly associated because name of the array is also a constant pointer which will hold the address of base element.we can also pass an array as the function argument with the help of pointer.

Ex. void change(int*);

```
main ( )
{
int A [5] ={10,20,30,40,50};
  int i, *p;
for (i=0; i<5; i++)
printf ("%d",A[i]);
       p = A;
change (p);
for (i=0; i<5; i++)
printf ("%d",A[i]);
 }
void change (int *p)
{
```

```
   int i;
for (i=0;i<5;i++)
{
 *(p+i) = *(p+i)+5
printf ("%d",*(p+i));
}
```

output
10 20 30 40 50
15 25 35 45 55
15 25 35 45 55

Pointer Arthmetic

we can perform some arithmetic operation with the pointer variable also . To understand the pointer arithmetic considers the following example.

Suppose 'p' is a pointer variable of type int which will hold the address 500. Then following operation can be possible.

Int *P; ← 500

1. If we use p++ , it will point the next address that is 502.
2. If we use p-- , it will point the previous address that is 498.
3. If we use p+5 , it will point previous 5th address , that is 510.
4. If we use p-5 , it will point previous 5th address that is 490.
5. we can substract two pointer variables to know the difference between two addresses.

Ex.

int *p = 500
int *q = 650
q-p → 75 (150 byte)

6. We can't add two pointer variable.
7. We can't use multiplication, division and reminder operator with pointer.
8. We can use relational operator with two pointer variable to compare it.

void pointer

when we want a pointer which will hold the address of any types of variable , we create void pointer.

The general format is
void * pv;

```
main ( )
{
        int a,*p;
        float b,*a;
        void *r;
        p = &a;
        p = &b;  → X
        q = &b;
  q = &a;  → X
r = &a;
         r = &b;
}
```

pointer to structure

As any normal pointer , we can also create a pointer to structure which will store the address of structure variable. We can also access the member of structure through the pointer to structure with the help of → operator.
Eg.

```
Struct first
{
Int a;
Float b;
};
Void main()
{
Struct first f1={10,15.50};
Struct first *p1;
P1=&f1;
Printf("%d%f",f1.a,f1.b);          → 10, 15.50
Printf("%u",p1);                   → address of f1
Printf("%d%f",p1->a,p1->b);        → 10,15.50
}
```

pointer to pointer (p to p)

As any normal variable , pointer variable has also an address .

If we want we can store the address of pointer variable in pointer to pointer.

The general format is

Datatype ** ptop;

ex.

```
main ( )
{
  int a =10;
  int *p;
  int **p;
  p = &a;
  q = &p;
printf("%d",a);  → 10
printf("%u",p);  → add of a
printf("%u",q);  → add of p
printf("%d",*p);  → 10
printf("%d",**q);  → 10
   **q = *p+50;
printf("%d",a); → 60
printf("%d",p);  → 60
printf("%d",**q);  → 60
}
```

DYNAMIC MEMORY ALLOCATION

When we create an array, we must give the size of array at the time of declaration it means use has no any role to decide the size of array. This generates two types of problem.

- Insufficient memory
- Wastage of memory

To remove this problem we use run time memory management.

C provides following common library function for dynamic memory allocation

Malloc()

Calloc()
Realloc()
Free()

Malloc():- this function allocates a single block of memory at run time. The general format is:

Ptr= (caste type*)malloc(byte_size);

Eg.

>> Int *p;
P= (int*)malloc(sizeof(int)

Here malloc creates 2 byte memory of size int and assign the address in pointer p. now p can be act as integer variable.

P= (int*)malloc(10*sizeof(int);

Here malloc creates 10 memory each of size 2 byte and assign the address of first memory in pointer p. now p can be act as array of integer of size 10.

P= (int*)malloc(n*sizeof(int);
Where n can be decided at run time.

Calloc() :- calloc() is also a dynamic memory allocation and similar to malloc() only difference is that calloc() allocates multiple block of memory of same size and set each byte as zero.

The general format is :

Ptr= (caste type*) calloc(num_of_block, byte_size);

Eg.

Int *A;
A= (int*) calloc(n,sizeof(int);

Here calloc allocates n block of memory of size int for pointer A.

Free() :-

This function deallocates memory at run time.

The general format is-
Free(ptr);

Eg.

Free(p);

Write a program to input n number in the array and find the sum.

```
main()
{
Int n,i,sum=0;
int  *A;
printf(" enter the array size ");
scanf("%d",&n);
A= (int*)malloc(n*sizeof(int);
for(i=0;i<n;i++)
{
printf("enter a number ; ");
```

```
scanf("%d",&A[i]);
}
printf("---- elements of array-------");
for(i=0;i<n;i++)
{
printf("%d",A[i]);
sum=sum+A[i];
 }
Printf("sum of elements= %d",sum);
}
```

Realloc() :-
Using this function we can increase or decrease the size of memory at run time.

The general format is

Ptr= realloc(new_size);