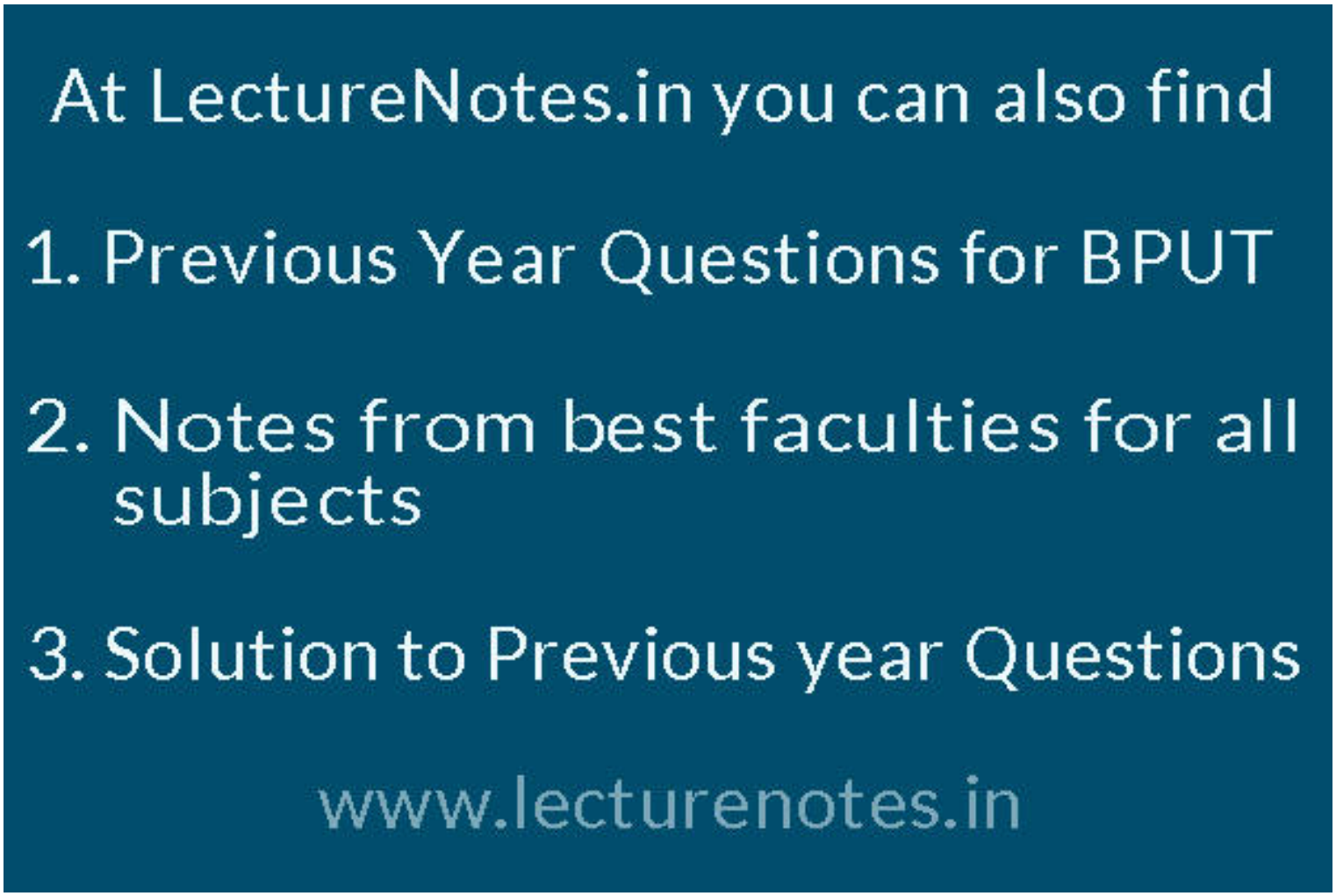
Programming In C

Contributed By:
Bibhuprasad Sahu

Disclaimer



Programming In C

Topic:

Introduction To C Programming

Contributed By:

Bibhuprasad Sahu

MODULE - 1

"Programming in 'C'"

Introduction to 'C' :-

→ 'C' is deeg designed and developed by Denis Ritche 'AT' and T's bell laboratary in the year 1972.

→ This is name as 'C' because most of the features from 'B' language. C als also middle level language because it reduces the gap between high level and low level language.

Why we use 'C' :-

→ 'C' is a robust language which consist because set of built in functions and operators that can be used to design complex program.

→ 'C' is highly portable because C program written for one computer can be run with another with little no change or no modification.

→ It is suited for structured programming language.

→ C Program is design such way that it utilise hardware when needed.

Program :-

→ Program is consist a set of instruction designed for a particular task.

Interpriter :-

- Interpriter needs only one line of the source Program at a time and converts in to object code. In case any error occurs the error will be indicated instantly.
- It is easier for the user to identify the error and debug it.
- It consumes more time for converting source Program to object Program.

Compiler :-

- A compiler reads whole Program at time then converts in to object code.
- It source the errors the whole Program.

Structure Programming Approach :-

→ 'C' is a structured language which follows top-down approach.

→ Top-down means be composing the Program in to no. of different task and the task are devided into sub task.

Task :-

→ Program / task is devided into multiple sub task known as modules, Proceolures and function.
→ Each module perform different task.
→ The modular structure is easy for debugging, testing and maintaintance.

Basic structure of 'C' Program :-

1. Comment
2. Pre Processor
3. Header file
4. Main function

- **Comment :-**
  comments are not necessary for a Program. It is used to understand the flow of the Program and meaning of Program statements. Two type of comments such as
  (a) single line //———
  (b) Multi line /*———*/
- **PreProcessor :-**
  some information are being used ~~for~~ by over Program which is Process before this is known as preprocessor.
- **Headerfile :-**
  The collection of some library function over a single file is known as headerfile.
  The extension of headerfile is .h.
- **Main function :-**
  In every C Program at least one function that is known as main function. This is the function which tells the compiler when the Programs starts.

```
main function
  {
    declaration Part
    execution Part
  }
```

• **Declaration Part :-**

The declaration Part declared the entire variable in the executable Part. The initialisation Part is done here. Initialisation means providing some value to particular variable.

Ex :-

```
main c
{
int a = 0, 4, 2 // Here I have declared 3 variables
}
```

int → Integer datatype

a → variable

• **Execution Part :-**

- This part contains statement of following implimentation of variable.
- This part may contain a set of statement or single statement and the statements are enclosed by curly braces ({ }).
- The executable session include built-in function user defined function, keywords, arithmetic or logic operation.
- Ex :-

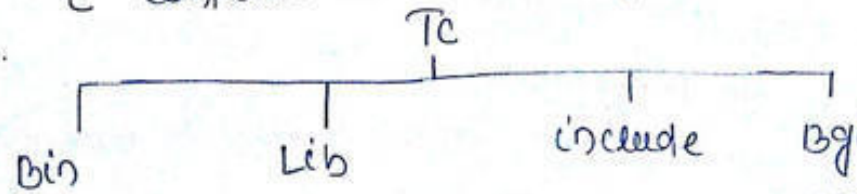
```
// My first 'C' Program
# include <stdio.h> (→ standard input output)
main ()
{
Printf (" welcome to c ");  (Printf is used to Print a statement.
}
```

o/p

Welcome to C.

Architechture of C :-

C compiler has following components



→ It we install C compiler in our system then the above director are automatically created under the root director.

- **TC :-** TC stands for turbo C. It is the main directory.
- **Bin :-** It is the binary directive which stores binary and executable files.
- **Lib :-** It is known library directives. It consists the library files.

Include - This directive contains all header file.

- **Bgi :-** Bgi stands for Binary graphics Interface. It is the collection of all graphical files and components.

Programming Rules :-

→ Every statements in C program should written lower case letter.
→ The upper case letter is used for symbolic constants.
Ex - symbolic constants PI = 3.142
→ Blank space can be inserted between 2 words.
→ Use beginning and ending curly braces for functions.
→ Each 'C' statement ended with a semicolon(;)
→ User defined function generally placed after the main function only.

Execution of a 'C' Program :-

* There are 4 basic statements

1. Creating the Programs
2. Compiling the Programs
3. Linking the Programs
4. Executing the Program

1. Creating the Program :-

→ The program written in C should be extension of C.
→ The default extension is C.
→ The program named depends upon the user only.

2. **Compiling the Program :-**
→ The source program statement are translated in to object program and it is done by the compiler.

3. **Linking the Program :-**
→ Linking means it is the process of putting together other programs files or functions i.e required to link with the program.

4. **Executing the Program :-**
→ After completion of executable object code will be loaded in to the computer in the main memory then the main memory program is executed.
→ After execution the object file changes in to exe. files.

Compilation	: Alt + Fg
Execution	: Ctrl + Fg
output	: Alt + Fs

'C' declaration :-
There are different elements in 'C' program.

1. 'C' character set
2. Delimeter
3. keyword
4. constants
5. Indentifier
6. Variable
7. data type
8. Operator

'C' character set :-
C character set are used form words and expressions depend on the program and the computer, letters, digits, special ~~chara~~ character, white space.

Letters : a to z and A to Z.
Digits : 0 to 9.
White space : Blank space, horizontal tab, vertical tab, new line.
special character : comma (,), dot (.), semicolon (;), colon (:), dollar ($), question mark (?), ', !, \, &, /, ~, ^, +, -, <, >, <=, >=, ( ),

[ ], { }, %, #, =, @, -, " .

- **Delimeter :-**

C uses special kind of character or unique character that is known as ~~code~~ delimeter.
Ex- colon, parenthesis, square braket, hash, comma, etc.

- **Token :-**

The smallest unit of C Program is known as token.

(i) Identifier
(ii) Keyword
(iii) strings
(iv) constants
(v) special symbols
(vi) operators.

- **Identifier :-**

→ The identifier are the name of the variable fun^s.
→ Identifier are written in both case, but lower case is preferrable.
→ An underscore (_) can be a identifier.

- **Keyword :-**

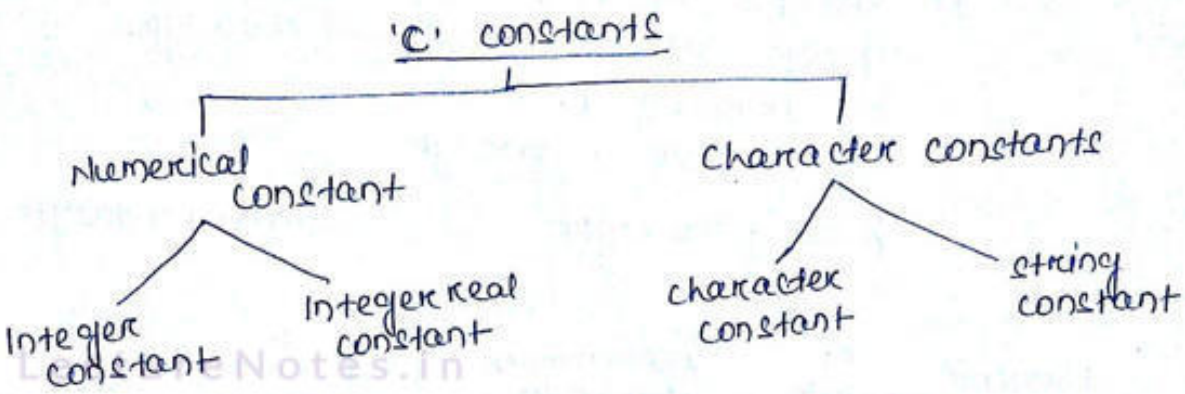
→ The C keyword reserve by the 'C' compiler.
→ All keywords has fixed meaning the meaning cannot be change by the user.
→ The user can use it cannot modify it.
→ Ex - auto, char, break, const, continue, do, else, if, double, enum, float, for, goto, int, long, short, sign, static, size of, struct, ~~union~~ switch, typedef, void, while, unsign.
→ All keywords are written in lower case letter.

- **constants :-**

→ The constants in C the fixed value which will not change through out the Program.
→ Ex - int const a = 10;

~~Classification of constants~~ :-

Classification of constants :-



→ **Integer constants :-**
→ These are the sequence of no. from the no. 0 to 9 without any decimal point.
→ It takes minimum two bits of memory maximum 4 bits of memory.
→ Integer constant may be negative and positive.
Ex int a = 16.

→ **Integer Real constants :-**
→ Real constants are also known as floating constants.
→ Ex - float a = 16.1, 4.5

→ ~~cha~~ **character constants :-**

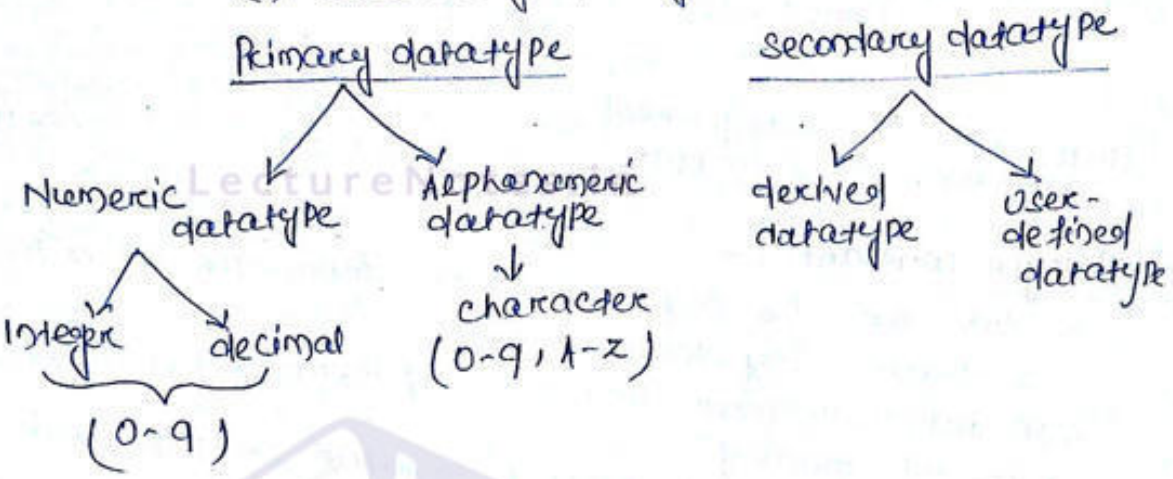
single character constants :-
character constants means a single character enclosed a single quote (' ').
Ex - char y = 'u'

string constant :-
string constant are the sequence of character enclosed with a double quote.
Ex - "abc".

• **variable :-**
→ A variable is a name used for storing the data value
→ The data values can be change during the Program execution.
→ It is not fixed.
→ syntax : datatype variable name
int a

→ The maximum length of a variable is 32 character.

- **Data types :-**
  → It specifies the data.
  → C language supports 2 types of data types.
    (I) Primary datatype
    (II) secondary datatype



→ **Integer datatype :-**
→ Integer datatype are the whole numbers with a range values supported by a particular values.
→ The datas are stored in the form of 0-9.
→ Integer short and long.
- **short Integer :-** The short integer takes 2 byte of memory location which ranges from -32768 to 32767 (-2^{32} to 2^{32}).
- **Long Integer :-** Long Integer occupies 4 bytes of memory. The ranges is -2147483648 to 2147483647.

→ The Integer datatypes is two types according to view control over the range.
(a) signed integer (-32768 to 32767)
(b) unsigned integer (0 to 65535)

→ **Floating point datatype :-**
→ In this data type integer as well as decimal data can be stored.
→ The ~~following~~ floating point variable is defined by the keyword float.
Ex - float a = 4.5

→ The floating point data type is also categories into 3 types according to the memory range.

(a) float
(b) double
(c) long double

float :- It takes 4 bytes of memory location. The formate specifies of floating point is (%.f).

double :- It takes 8 byte memory location.

Long double :- Long double represent the data in exponential form.

→ **character datatype :-**

→ The character datatype represents by the single character
→ It takes only one byte of memory location.
→ The string character datatype can represented %c.
→ It is 2 bytes.

(a) signed character
(b) unsigned character

signed character : It occupies 1 bytes of memory location which memory ranges from -2^7 to 2^7. The control string is %c.

unsigned character : It also takes 1 byte of memory location which ranges from 0-255.

Memory representation of data :-

→ The char type is a integral type able to store 8 bit signed no.s and 8 bit unsigned no.s.

→ Ex : char m = -4

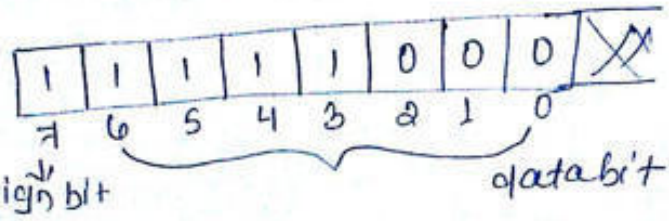
0000 0100 = 4

After 1's complement

1111 0111

Add 1 in 1's complement

1111 0111
1
1111 1000 (-8)



Standard input output function :-

There are 3 input output operations C environment uses standard out, stdin, stderr for accessing the devices.

Character based input output function :-

```
#include <stdio.h>
void main ( )
  {
     char ( );
     getchar ( );
     fflush (stdin);
     putchar ( );
  }
```

The function of getchar function accepts a character from input device such as keyboard and putchar accepts the parameter the character to display on output device (Monitor).

fflush ( ) :-

→ To create the buffer

write a func to input a character to display the character twice.

```
#include <stdio.h>
  void main( )
    {
      char A;
      A = getchar ( );
      fflush (stdin);
      putchar (A);
      putchar (A);
    }
```

write a Program to accept and display 2 characters

```
#include <stdio.h>
  void main ( )
    {
      char a, b;
      a = getchar ( );
      b = getchar ( );
      fflush (stdin);
      putchar (a);
      putchar ( b );
    }
```

String based input output function :-

gets ( )
Puts ( )

→ C language provides the fun^{c} gets ( ) and puts ( ) for string based input output.

```
#include <stdio.h>
    void main ( )
      {
        char str[];
        Puts (" Enter a string of length 20" );
        gets ( str );
        fflush (stdin);
         Puts (str );
      }
```

gets ( ) :- Accept the string variable into the location where the input string is to be stored.

Puts ( ) :- This function causes the cursor jump into next line after printing the string.

Write a function that prompts and accepts a name with max^{m} in 25 character and display the message.

```
Hi
name
#include <stdio.h>
  void main ( )
  {
    char str [25];
    Puts ("Enter a name with message 25 character");
    gets ( str );
    fflush ( stdin );
     Puts ( " hi");
     Puts (str );
  }
```

Write a function that prompts for name upto 20 character and address upto 30 character and display the name and address in the following length.

```
your name is ;
  (name)
your address is ;
  (address)
```

```
# include <stdio.h>
  void main ( )
   {
     char str1[20];
     char str2[20];
     Puts (" Enter your name ");
     Puts (" Enter your address ");
      gets (str1);
      gets (str2);
      fflush (stdin);
      Puts (" your name :" \n str1);
      Puts (" your address : " \n str2);
   }
```

Formated output :-

→ 'C' Provides standard function for formatted input and output.

→ The function Printf is used for formatied output to standard output based on the formate specification.

→ syntax - Printf ("formate", data 1, data 2 ---);

Ex - Printf (" %c ", Arpita);

%c is the formate specifier and Arpita is the name of the variable.

\b : It is use for back space.

\n : For new line

\t : For tab

Write a Program to show the uses of Printf.

```
# include <stdio.h>
void main ( )
   {
     char inp = 'A';
     int inum = 21;
     double dnum = 87.65;
     Printf (" char = %c \n", inp);
     Printf (" int = %d \n", inum);
     Print f (" double = %.2f \n", dnum);
   } getch ( ); (→ to return a character)
```

Output

char A
int 21
double 87.65

Formatted Input :-

Information - The organise form of data having specific meaning. The i/p and o/p device is known as peripherial device.

Plug & Play - Insert the pendrive.

1 byte = 8 bit
1 KB = 1024 bytes
1 MB = 1024 KB Bytes
1 GB = 1024 MB
1 TB = 1024 GB
Nano = $\frac{1}{10^{-12}}$ sec

Memory - Register (Temporary storage device)

Flow Chart :-

→ It is a type of diagram that represent an algorithm and process showing steps as boxes of various kind and their order connecting with arrow mark.
→ It is a diagramatic representation step by step solution for a particular Program.

→ Symbols of flowchart

→ Oval (start / End)

→ Rectangle (Processing function of a Program)

→ Rombus (Decision making)

→ Arrows (direction of flow)

Advantages of flowchart :-

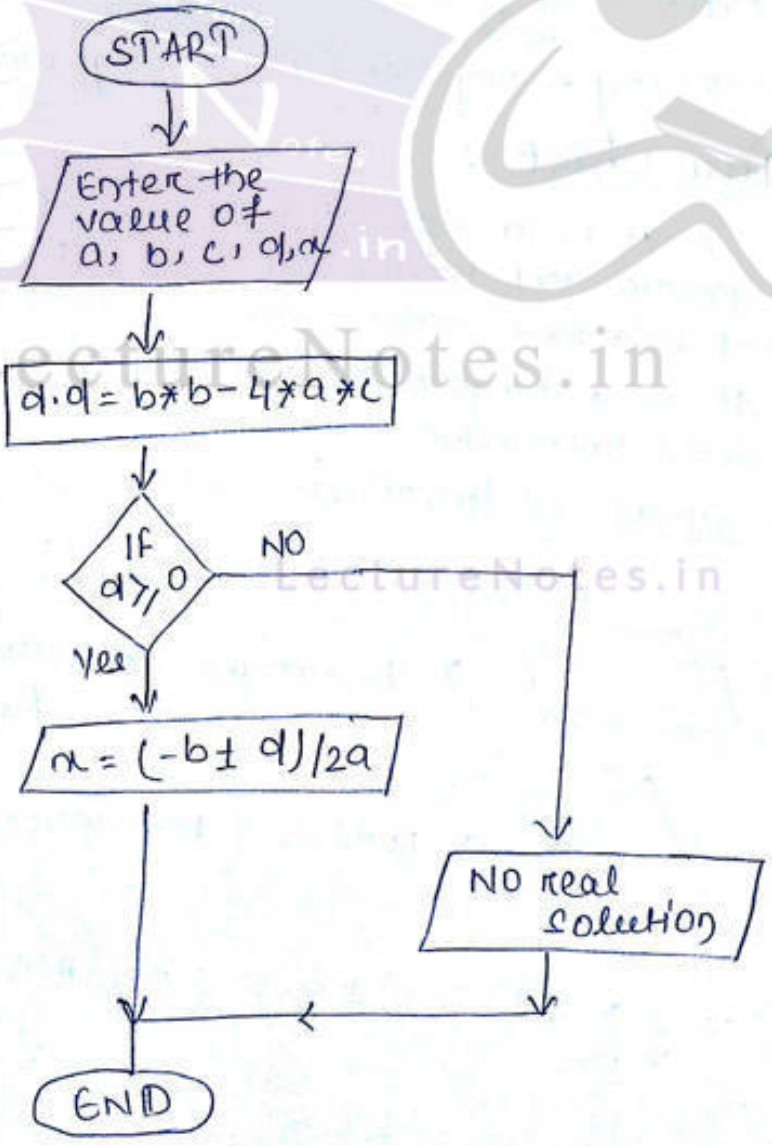
1. communication
2. Effective analysis
3. Proper documentation
4. Effective coding
5. Proper debugging (rectify the error - debug)
6. Easy Program maintainance

Disadvantages of flowchart :-

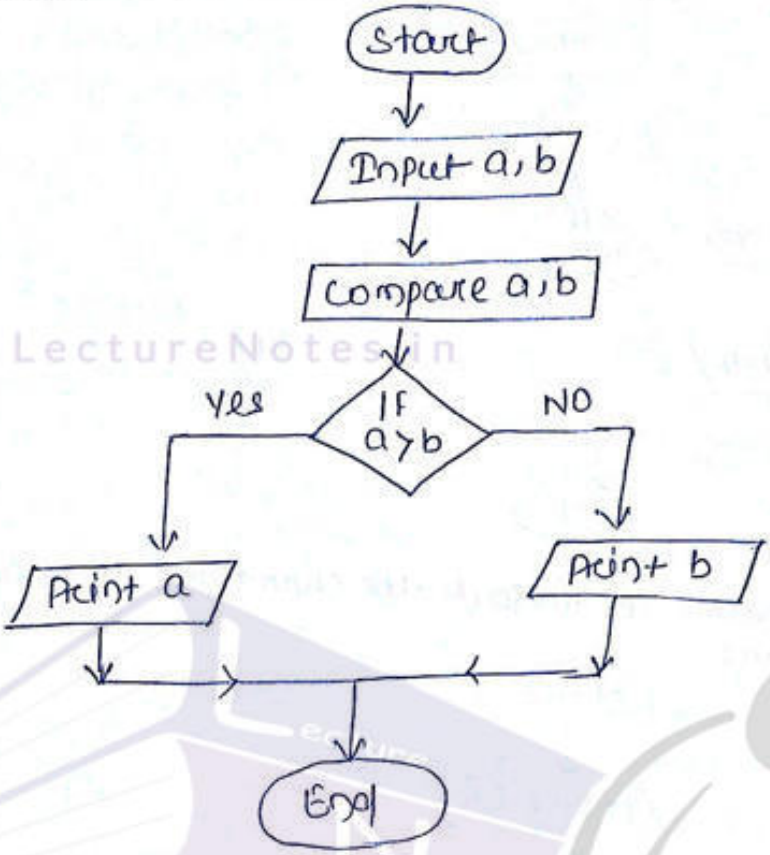
1. complex logic
2. Alteration and modification
3. Details lost how it's done

Ex-1

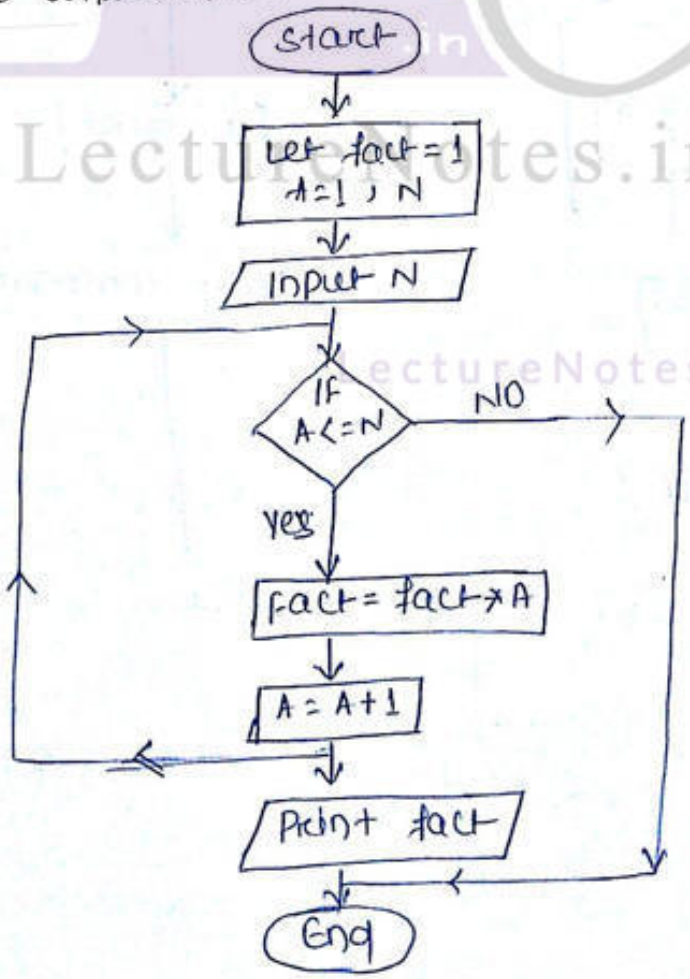
Draw a flowchart to find out roots of a quardtic equation.



Ex-2
Draw a flowchart findout largest between 2 number.

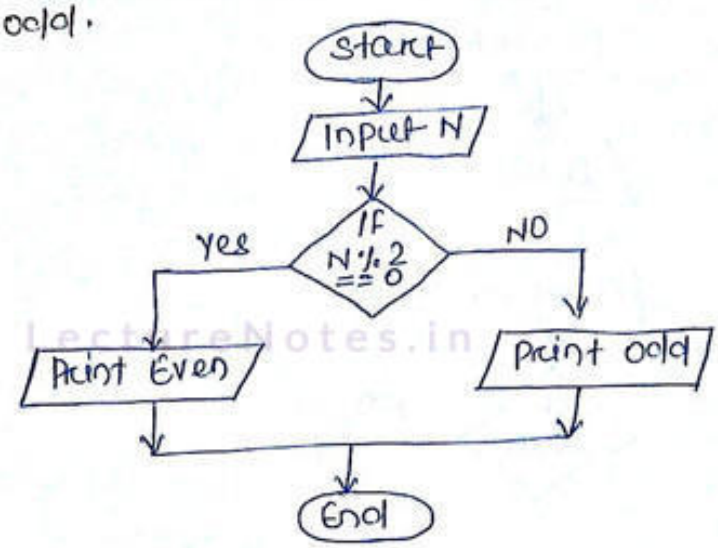


Ex-3
Draw a flowchart to findout the factorial of a no.



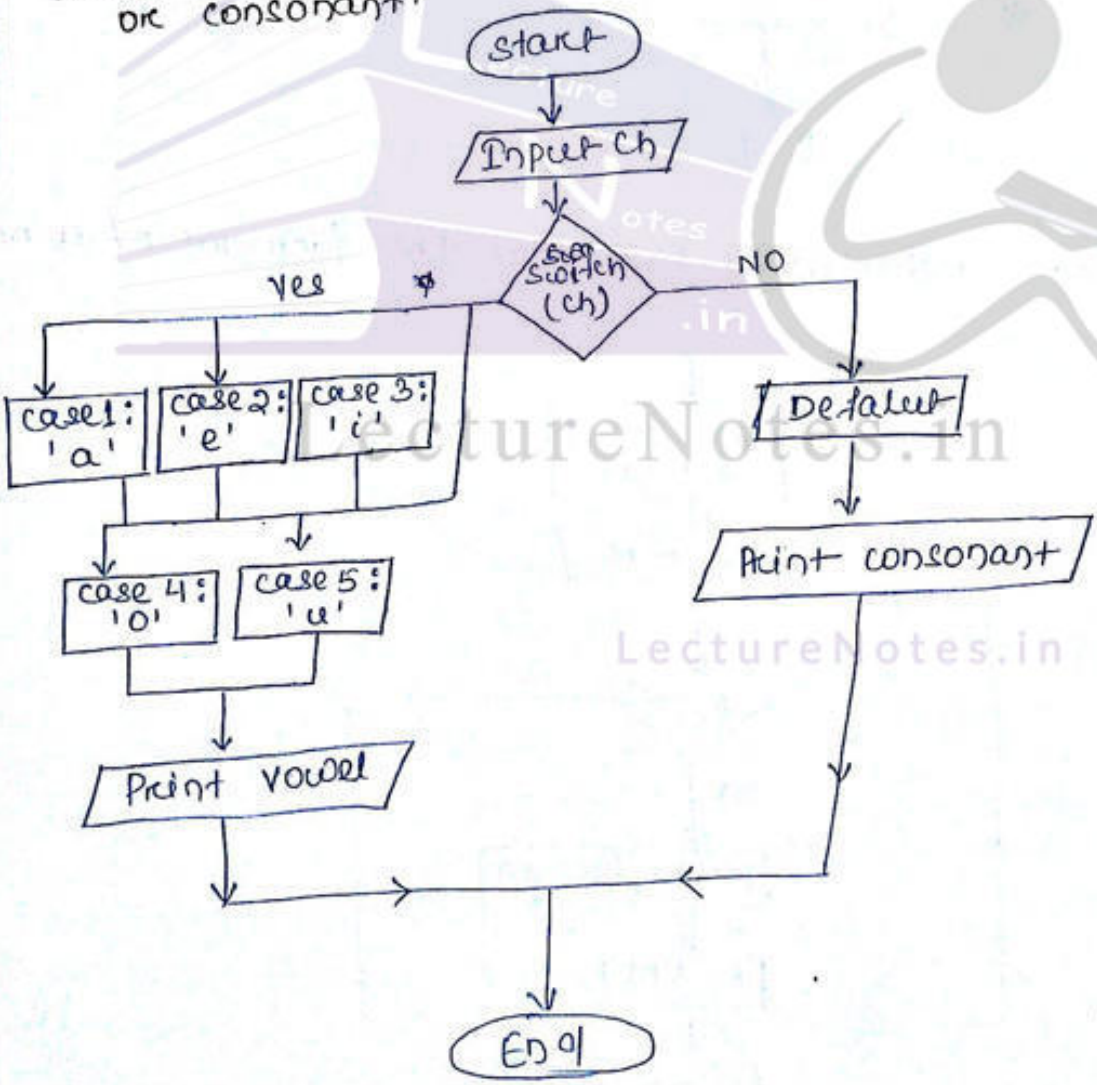
Ex – 4

Draw a flowchart to find out whether a no. is even or odd.



Ex-5

Draw a flowchart to find out the character is vowel or consonant.



Scanf :

→ Scanf func is used to read all types of data from key-board and stored it in a variable.

→ The scanf function requires a '&' operator called address operator, which stores the value of respective memory location of the variable.

→ By this operator we can identify memory address of the variable.

→ Syntax : scanf(" control string ", & variable);
Ex : scanf(" %d", &a);

* Write a Program to take two values through scanf statement

```
#include <stdio.h>
    void main ( )
      {
        int a, b ;
        Printf(" Enter 2 variable a and b" );
        Scanf(" %d %d ", &a, &b );
        Printf(" The value of a, b is %d %d", a, b );
      }
```

* Write a Program to Print the address of a variable.

```
#include<stdio.h>
    void main ( )
      {
        int a;
        Printf(" Enter a variable 'a ");
        Scanf(" %d", &a );
        Printf(" The value of a is %d ", &a );
      }
```

* Write a Program to Print the value of 2 variables using 3rd variable.

```
#include<stdio.h>
    void main ( )
      {
        int a, b, c=temp;
        Printf(" Enter two variable a, b");
        Scanf(" %d %d", &a, &b );
        Printf(" The value of a and b before
                                    swapping");
```

```
Printf(" %d %d ", a, b);
        temp = a;
        a = b;
        b = temp;
Printf(" After swapping the value of a and b is
        %d %d ", a, b);
    }
```

* Write a Program to swap 2 variable without using 3rd variable.

```
#include<stdio.h>
void main( )
{
    int a, b;
    Printf(" Enter two variable a and b");
    scanf(" %d %d", &a, &b);
    Printf(" The value of a and b before
            swapping ");
    Printf(" %d %d ", a, b );
    a = a*b;
    b = a/b;
    a = a/b;
    Printf(" After swap the value of a
            and b is %d %d ", a, b);
}
```

O/P

Enter two variable a and b
a = 5 b = 6
Before swapping the value of a and b
5 6
After swapping the value of a and b
6 5

* Write a Program to accept sides of a triangle to calculate the Perimeter of a triangle.

```
#include <stdio.h>
  void main ( )
    {
      int a, b, c, P;
      Printf (" Enter the 3 variable a, b, c ");
      Scanf(" %d %d %d ", &a, &b, &c );
       P= a+b+c/2
      Printf(" The value of the Perimeter = %d ", P);
    }
```

O/P

Enter the 3 variable a, b, c

a= 3, b= 4, c= 5

The value of Perimeter = 6

* Write a Program to receive a no. from keyboard and to Print the square and cube.

```
#include <stdio.h>
  void main( )
    {
      int a, s, c;
      Printf(" Enter the value from keyboard
                                  a ");
      scanf(" %d", &a );
       s= a*a;
       c= a*a*a;
       Printf(" The value of square = %d ", s);
       Printf(" The value of cube = %d ", c);
    }
```

O/P

Enter the value of a

a= 3

square = 3*3 = 9

cube = 3*3*3 = 27

Programming In C

Topic:

Operators

Contributed By:

Bibhuprasad Sahu

Operator	Meaning
+	Addition
-	Substraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decreament
&	address operator
sizeof	give size of data value

Relational operator :-

→ This type of operator are used to take certain decision by comparing the two data value.

→ This operator provide mechanism to find out the relation between two values.

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

* Write a Program to use various relational operator and their value.

```
# include <stdio.h>
  void main ()
   { Printf("\n 3==3 %d \n", 3==3);
   }
```

o/p = 1

```
# include <stdio.h>
  void main ()
  { Printf("\n 9>3 %d \n", 9>3);
  }
```

o/p = 1

Logical Operator :-

→ This operator are used when we want to test more than one condition.

→ Using this operator we join this expression.

Operator (symbol)	Meaning
&&	AND
||	OR
!=	NOT

AND GATE

OR GATE

NOT GATE

Tables for all Gate

A	B	A·B	A+B
1	1	1	1
1	0	0	1
0	1	0	1
0	0	0	0

NOT GATE

A	$\overline{A}$
0	1
1	0

Write a Program to use logical operator.

```
#include<stdio.h>
void main()
{
  Printf("\n 9>3 && 9>4 \n");
}
```

O/P

1

1

Increament and Decreament operator :-

Increament operator (++)

The increament operator is used to add 1 to its operand.

It is two types

* Write a Program to show multiple pre increament.

```
#include<stdio.h>
void main ( )
{
    int a=5
    a= ++a + ++a + ++a ;
    Printf(" The value of a = %d ", a);
}
```

O/P

24

* Write a Program to show multiple pre and post increament.

```
# include<stdio.h>
void main ( )
{
    int a=5
    a= ++a + ++a + ++a + a++ + a++;
    Printf(" The value of a is %d", a);
}
```

O/P

24 + 18 = 42

Assignment operator:-

Assignment operator is used to assign the result of an expression to the variable.

syntax- variable operator = expression;

```
a=5;
a+=5   = a+5
```

* Write a Program to use the assignment operator.

```
#include<stdio.h>
void main ( )
{
    Printf(" Enter a no. for variable a ");
    a+=5 ;
    Printf(" The value of a is %d", a );
}
```

O/P Enter the value for a 10

a = 10 + 5 = 15

Bitwise Operator :-

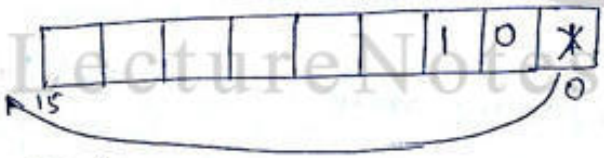
Operator	Meaning
>>	right shift
<<	left shift
^	Bitwise or or XOR
~ (tilde)	1's complement
&	Bitwise AND
|	Bitwise OR

$a = n/2^2$

* write a program to use right shift operator.

```
#include <stdio.h>
void main ( )
{
    int a;
    Printf(" Enter a value");
       a >> 1 ;
    Printf(" The value of a is %d", a);
}
```

O/P

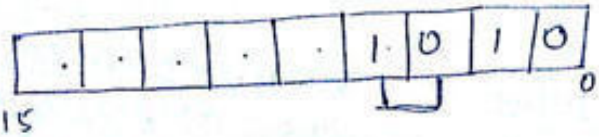


10 : 2 : a

* write a Program to use left shift operator.

```
#include <stdio.h>
    void main ( )
    {
        int a;
        Printf(" Enter a value for a ");
           a<<1;
        Printf(" The value of a is %d", a);
    }
```

O/P



1010 = 10 : a

Where a is the no. of the position to be shifted & n is the no.

$a = n \times 2^n$

* Write a Program to use Bitwise AND operator.

```
#include<stdio.h>
main()
{
  int a, b, c;
  Printf("Enter two variable a and b");
  Scanf("%d %d", &a, &b);
    c = a & b;
  Printf("The value of c is %d", c);
}
```

O/P

a = 5 — 101

b = 2 — 010

000 — c = 0

* Write a Program to use Bitwise OR operator.

```
#include<stdio.h>
void main()
{
  int a, b, c;
  Printf("Enter two variable");
    c = a | b;
  Printf("The value of c is %d", c);
}
```

O/P

a = 2 — 010

b = 3 — 011

011 = c = 3

Comma operator:—

This operator is used to separate two or more expression.

```
#include<stdio.h>
void main()
{
  Printf(" int a, b;
  Printf("The value of a is %d \t
          the value of b is %d,
          2+3, 5+3);
```

Programming In C

Topic:

Decision Making And Branching Statement

Contributed By:

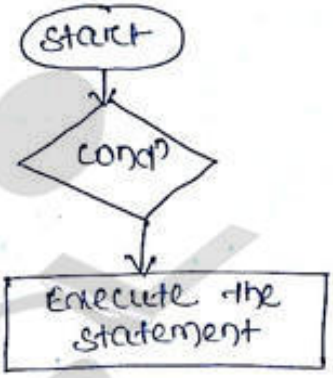
Bibhuprasad Sahu

Decision making and Branching statement :—

→ Depending on the condition the computer execute the statement accordingly.
→ Based on the condition we can branch it's called branching statement.
→ C language supports following decisions making statement
→ (i) simple if statement
(ii) if --- else statement
(iii) nested of if --- else statement
(iv) switch statement
(v) goto statement
→ Decision making statement are used control the flow of execution statement.

Simple if statement :—

Syntax - if (condition)
{
statement 1
statement 2
|
statement 10
} statement n)



* Write a Program input two no. and check which it's the greater no.

```
#include<stdio.h>
    main ( )
     {
       int a, b ;
       Printf (" Enter two variable a and b");
       scanf (" %d %d ", &a, &b);
        if(a>b)
        {
          Printf(" The greater value of a is %d",
                                  a);
        }
    }
```

If ---- else statement :-

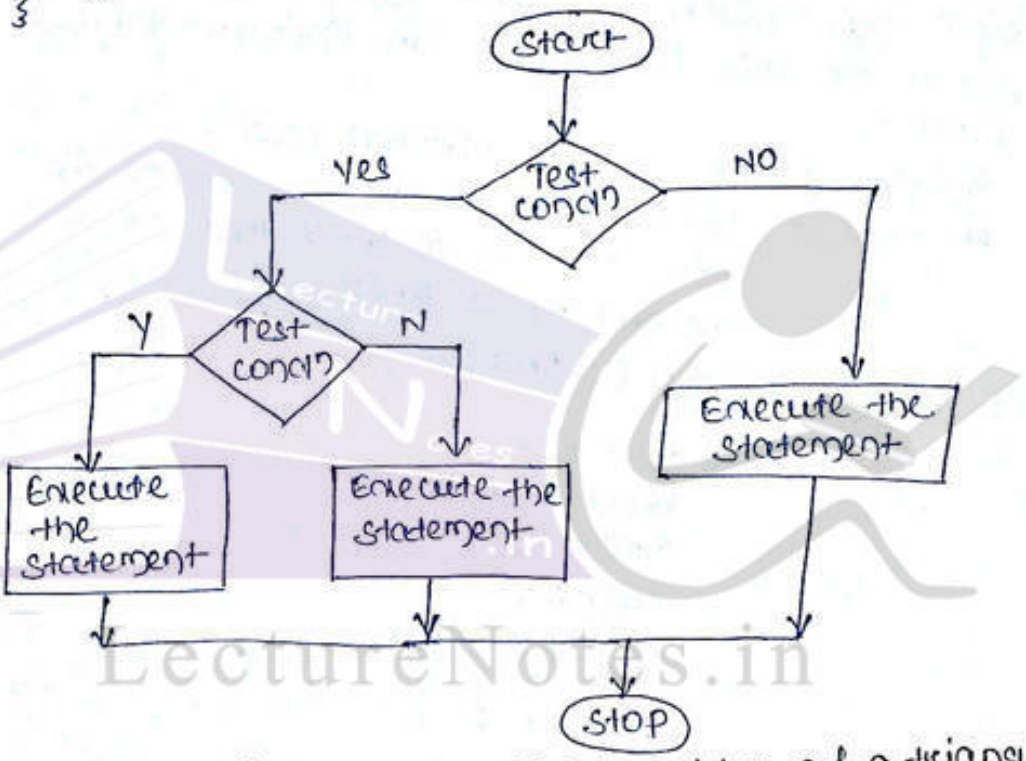
Syntax :- if (condition)
{
statement 1
statement 2
}
else
{
statement 1
statement 2
}

* Write a Program using if-else statement to find out whether the year is leap year or not.

```
#include<stdio.h>
void main()
{
int y;
Printf("Enter the year");
scanf("%d",&y);
if(y%4==0)
{
printf("leap year");
}
else
{
Printf("not leap year");
}
}
```

Nested if :-

```
if (condn)
{
 else if (condn)
    {
      statement
    }
 else
    {
     statement
    }
}
```



* Write a Program to accept sides of a triangle and check what type of triangle this is.

```
# include<stdio.h>
  void main ( )
    {
      int  a, b, c ;
      Printf(" Enter the 3 sides of a triangle ");
      scanf(" %d %d %d ", &a, &b, &c );
      if(a==b && b==c &&c==a)
      { Printf(" Equilateral triangle ");
      }
       else if(a==b || b==c || c==a)
```

```
    {
      Printf(" isolated triangle ");
    }
      else
        {
          Printf(" isolates triangle ");
        }
```

Switch case :-

→ This is a multi-way conditional control statement which helps us to make choices among a no. of alternatives.

→ The switch statement body consists of a series of case levels and an optional default level.

→ default : This is the optional level.

→ break : break statement is use end of the case signals to prove that it is the end of a particular case.

→ syntax :

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

* Write a Program using switch case findout the days of a week.

```
#include<stdio.h>
  void main( )
    {
      char d;
      Printf(" Enter days of a week ");
      scanf(" %c ", &d);
      switch (d)
        {
          case 1 :
          Printf(" first day is sunday");
          break;
```

```
        case 2 :
            Printf(" Second day is Monday ");
            break;
        case 3 :
            Printf(" Third day is Thursday ");
            break;
        case 4 :
            Printf(" Forth day is Wednesday ");
            break;
        case 5 :
            Printf(" Fifth day is Thursday " );
            break;
        case 6 :
            Printf(" Sixth day is Friday ");
            break;
        case 7 :
            Printf(" Seventh day is Saturday ");
            break;
        default :
            Printf(" Invalid day ");
    }
```

O/P

Enter 1 : First day is Sunday
5 : Fifth day is Thursday.

Goto Statement :

→ 'C' Provides /supports unconditional statement known as goto statement.
→ This statement can be Present anywhere in the Program.
→ the place where the branch is to be called execution must.
→ The label name must be start with a character.
→ Syntax - goto level
* Write a Program to find out square & cube.

```
#include <stdio.h>
void main ()
{
    int a, b;
    Printf(" Enter two Variable a and b");
```

```
scanf("%d %d", &a, &b);
    if(a > b)
     goto greater;
     else
     goto lesser;
greater:
printf(" a is greater %d ", a);
else
lesser:
Printf(" b is lesser %d", b);
}
```

—x—

Decision Making / Looping :—

→ Looping is defined as a block of statement which are repeatedly executed a certain no. of time.

→ or. Looping can be defined as another way repeating instruction until set until the condition satisfy.

→ 'C' concepts provides 3 types of loops.

(I) while loop
(II) Do while loop
(III) for loop

While loop :- In such case the condition check first, if the the condition is true then the body of while loop is execute until condition is false. ~~If the condition is false the body of while does not execute.~~

→ If the condition is false then the body of while does not execute.

→ while loop does not ended with semicolon (;)

→ syntax. while(cond^n)
{
execute the statement 1
execute the statement 2
}

* Write a Program using while loop to find out the factorial of a no.

```
#include<stdio.h>
main ( )
  {
    int n, fact=1 ;
    Printf(" Enter the value of n");
    Scanf(" %d", &n);
     while(n>=1)
      {
        fact = fact * n;
          n--;
      }
       Printf(" The value of factorial is %d", fact);
  }
```

O/P

Enter the value of n

5

The value of factorial = 120

* Write a Program input a no. and check whether it is Palindrome or not.

```
#include<stdio.h>
void main ( )
  {
    int n, rv=0, dg, temp ;
    Printf(" Enter a number");
    Scanf(" %d", &n);
     temp = n ;
     while(n>0)
      {
        dg = n % 10 ;
        rv = rv * 10 + dg ;
          n = n/10 ;
      }
      if( rv == temp )
       {
         Printf(" no is Palindrome");
       }
       else
```

```
    }
        Printf(" no. is not Palindrome ");
    }
}
```

O/P

(i) Enter a no.
111
No. is Palindrom

(ii) Enter a no.
312
No. is not Palindrome.

Do While :—

→ In such case the body execute first then check the condition, if the condition is true then it execute until condition is false.

→ It must execute once.

→ do while loop ended with semicolon (;).

→ syntax :

```
do
{
  ------
}------
while(condn);
  {
    execute statement 1
    ------
    execute statement n
  }
```

* write a program to input 3 digit no. and reverse it.

```
#include<stdio.h>
  void main( )
    {
      int n, rv=0, dg
      Printf(" Enter a no. ");
      Scanf(" %d", &n);
        while(n!=0)do
          {
            dg = n%10;
            rv= rv*10+dg
            n= n/10;
          }
```

```
    while(n>0);
    Printf(" reverse of a no. is %d", rv);
}
```

O/P

Enter a no.

123

Reverse of a no. = 321

For LOOP :- for loops first initialisation of a variable and repeats certain statement based on condition checked and also helps in increament / decreament of the variable.

→ syntax :-

```
for(initialization ; condition ; increament/decreament)
```

* write a Program to generate a fibonassi series.

```
#include<stdio.h>
void main()
{
    int f1=0, f2=1, f3=0, i, n;
    Printf(" Enter a no.");
    scanf("%d", &n);
    for(i=0; i<=n; i++)
    {
        f1=f2;
        f2=f3;
        f3=f1+f2;
    }
    Printf(" The value of fibbonasi series is %d",
                f3);
}
```

O/P

Enter a no.

13

0 1 1 2 3 5 8 13

* write a Program to solve the following series.

$$S = \frac{1}{1!} + \frac{2}{2!} + \frac{3}{3!} + \text{----} - \frac{n}{n!}$$

```
#include <stdio.h>
  void main( )
    {
      int i, n, fact = 1 ;
       float s=0 ;
       Printf(" Enter a no.");
       Scanf(" %d", &n );
       for (i=01 ; i<=n; i++)
       {
         fact = fact * i ;
         s = s+ (float) i / fact ;
       }
       Printf(" Sum of factorial is %f",
                    s);
    }
```

Nested for loop :-

```
for ( ;initialization; condition ; increament / decreament )
   for ( )
    {
    for ( )
     {
     ------
     }
    ------
    }
```

* Write a Program to Print the following structure

```
1
2  3
4  5  6
7  8  9  10
```

```
#include <stdio.h>
  void main ( )
    {
      int i, j, k=1 ;
      for (i=01 ; i<=5 ; i++)
      {
       for ( j= 1 ; j<=i ; j++ )
```

```
    {
        Printf("%d ", k++);
    }
    }
    Printf("\n");
}
```

* Write a Program to Print the following Structure.

```
        1
      1 2 1
    1 2 3 2 1
  1 2 3 4 3 2 1
1 2 3 4 5 4 3 2 1
  1 2 3 4 3 2 1
    1 2 3 2 1
      1 2 1
        1
```

```
#include<stdio.h>
void main()
{
    int i,j,s;
    for(i=1;i<=5;i++)
    {
    for(s=1;s<=5-i;s++)
    Printf(" ");
    for(j=1;j<=i;j++)
    Printf("%d",j);
    for(j=j-2;j>=1;j--)
    Printf("%d",j);
    Printf("\n");
    }
}
```

* Write a Program to Print the following structure.

```
        1
      1 2 1
    1 2 3 2 1
  1 2 3 4 3 2 1
1 2 3 4 5 4 3 2 1
  1 2 3 4 3 2 1
    1 2 3 2 1
      1 2 1
        1
```

```
# include <stdio.h>
  void main ( )
    {
      int i, j, s ;
      for( i = 1 ; i <= 5 ; i++)
      {
      for(s = 1 ; s <= 5-i ; s++ )
       Printf("  ");
      for( j = 1 ; j <= i ; j++ )
       Printf("%d", j);
      for( j = j-2 ; j >= 1 ; j-- )
       Printf ("%d", j);
       Printf("\n");
      }
      for( i = 4 ; i >= 1 ; i--)
      {
      for( s = 1 ; s <= 5-i ; s++ )
       Printf("  ");
      for( j = 1 ; j <= i ; j++ )
       Printf("%d", j);
      for( j = j-2 ; j >= 1 ; j--)
       Printf("%d", j);
       Printf("\n");
      }
  }
```

Programming In C

Topic:

Array

Contributed By:

Bibhuprasad Sahu

Array :-

Array is a homogeneous sequencial collection of data item over a single variable name.

Characteristics of array

→ In array are always store in consequitive memory location.
→ An array can store multiple value of similar type which can refferred by a single name.
→ Array name acctually a pointer to the first location of memory block allocated to the name of the array.
→ An array either be an integer, character or float data-type can be initilise only during the declaration but not after words.
→ Any particular element of an array can be modify separately without distributing other element.
→ All elements of an array said the same name and they are distinguish with each other with the help of the element no.

Operation of an array :-

1. Traversing : Processing each and every element in the array sequencially.
2. Searching : Searching an element to findout the element is present or not.
3. Sorting : Arranging the element in an array in a particular sequence.
4. Inserting : To insert the element to the array.
5. Deleting : To delete the element to the array.

Types of Array :-

There are 3 types of array such as
1. single dimensional array
2. Double dimensional array
3. Multi dimensional array

One - dimensional array :-

→ one dimensional array is an array which have only one subscript specification is needed to specify a particular element of an array.

→ Syntax -- Data type _ array name [size];

Initialization of one one dimensional array :-

An array can be initialise at either following states.

1. At compiling time (static initilization)
2. Dynamic initialization.

Compiling time Initialization :-

→ The compile time initialization means the array of elements are initialise at the time of program written or array declaration.

→ syntax - datatype _ array name [size] = (list of element of an array)

→ Ex - int num [5] = {0, 1, 2, 3, 4}

* write a Program declare to an array and initialise the value at compile time and display.

```
#include <stdio.h>
main ( )
{
    int a [5] = {0, 1, 2, 3, 4};
    Printf(" %d", a[0]);
    Printf(" %d", a[1]);
    Printf(" %d", a[2]);
    Printf(" %d", a[3]);
    Printf(" %d", a[4]);
}
```

Runtime initialisation :-

Runtime initialisation means the array can be initialise at runtime. That means array elements are initialise after the compilation of the Program.

* write a Program to input the data from keyboard and display it.

```
#include <stdio.h>
    main( )
     {
       for(i=0; i<5; i++)
       {
        for( int a[5], i;
         for (i=0; i<5; i++)
          {
          scanf(" %d", &a[i]);
          }
          Printf(" display");
          {
           for(i=0; i<5; i++)
           {
            Printf(" %d", a[i]);
           }
          }
       }
```

* Write a Program to accept 5 no. from keyboard and Print them reverse order on the screen.

```
  #include <stdio.h>
    main( )
      {  int a[5], i ;
         for (i=0; i<5; i++)
         {
         Printf(" Enter 5 no. from keyboard ");
         scanf(" %d", &a[i]);
         }
         Printf(" Display the no.");
         for (i=4; i>=0; i--)
          {
           Printf(" %d", a[i]);
          }
      }
```

* Write a Program to count the even and odd in an array.

```
   #include<stdio.h>
     main ( )
       {
          int a[5], odd=0, even=0, i ;
          for(i=0; i<5; i++)
          {
```

```
Printf(" Enter the no. of an array ");
scanf(" %d ", &a[i]);
  if(a[i]%2 ==0)
   {
     even = even ++ ;
   }
    else
   {
     odd = odd ++ ;
   }
  }
 Printf(" Display the even and odd number %d %d ",
                    even , odd );
}
```

* Write a Program to find out minimum and maximum no. of an array.

```
#include <stdio.h>
      main ( )
           {
        int a[5] = {0, 1, 2, 3, 4}, i,
            a[0]= max = a[0],
                  min = a[0];
        for(i=0; i<5; i++)
        {
         Printf(" Enter the value ");
         Scanf("%d", &a[i]);
        }
          if(a[i] < min)
          Printf(" minimum %d", a[i]);
          else
           a[i] > max)
          Printf(" Maximum", a[i]);
        Printf(" Max and min is %d %d ",
                          a[i] );
        }
```

Two dimensional array :-

→ Two dimensional array is nothing but a table with rows and columns.

→ A two dimensional array can be expressed as a contigious and tabular block in memory where the data can be stored in table structure.

→ Syntax : ~~datatype arrayname [no. of row size][~~

datatype arrayname [no. of row size][no. of col. size]

→ Ex : int a[3][3]

Initialisation of two dimensional array

Compile time / static initialise

Implementation of two diamentional array :-

A two dimensional array can be implemented 2 ways

(i) row major implementation

(ii) column major implementation

* Write a Program to initialisation values to a 2D array and display it.

```
#include<stdio.h>
main()
{
    int a[2][3] = {1,2,3,4,5,6};
    Printf("%d", a[0][0]);
    Printf("%d", a[0][1]);
    Printf("%d", a[0][2]);
    Printf("%d", a[1][0]);
    Printf("%d", a[1][1]);
    Printf("%d", a[1][2]);
}
```

* Write a Program using two dimensional addition of 2D array two matrix.

```
#include<stdio.h>
main()
{
    int a[3][3], b[3][3], c[3][3], i, j;
    for(i=0; i<3; i++)
```

```
{
for(j=0; j<3; j++)
{
 Printf(" Enter the ~~for~~ number for a ");
 Scanf(" %d", &a[i][j]);
}
}
 for(i=0; i<3; i++)
 {
  for(j=0; j<3; j++)
  {
   Printf(" Enter the number for b");
   Scanf(" %d", &b[i][j]);
  }
 }
 Printf(" The value of an array c");
    for(i=0; i<3; i++)
    {
     for(j=0; j<3; j++)
     {
      c[i][j] = a[i][j] + b[i][j];
      Printf(" The value of c is %d", c[i][j]);
     }
    }
}
```

Multidimensional array:-

→ When the diamensional subscript of array more than two
--this is called multidiamensional array.

→ syntax : datatype arrayname size[][][].

→ Ex : int a[3][3][3]

* write a Program

```
#include <stdio.h>
    main( )
        {
          int a[3][3][3], b[3][3][3], c[3][3][3],
              i, j, k;
          for(i=0; i<3; i++)
          {
            for(j=0; j<3; j++)
            {
```

```
for (k=0;k<3; k++)
{
 Printf(" Enter the value for a");
 Scanf(" %d", a[i][j][k]);
}


 for(i=0; i<3; i++)
 {
  for(j=0; j<3; j++)
  {
   for(k=0; k<3; k++)
   {
   Printf(" Enter the value for b");
   Scanf(" %d" , &b[i][j][k]);
   }
   }
   }
   Printf (" The value of an array c");
     for(i=0; i<3; i++)
     {
     for(j=0; j<3; j++)
     {
     for(k=0; k<3; k++)
     {
     Printf(" The value of c is %d",
                        c[i][j][k]);
      }
      }
      }

}
```

Programming In C

Topic:

Strings

Contributed By:

Bibhuprasad Sahu

Strings :-

→ A string constant is one dimensional array of character terminated by a null.

→ A group of character define between double quote mark is a string constant.

Declaration and initialisation of string variables

→ C does not support string datatype thats why it allows us to represents string as a character arrays.

→ syntax - char string name [size];

→ Ex - char book [10];

→ A null character (\0) is assign to the string automatically when the compile assign a string to a character array. so the size of the array becomes max no. array + 1.

Initialisation

There are two type of initialisation of string

(I) compile time

(II) Run time

compile time :-

char name [10] = "Nishaan"

char name [10] = {'N', 'i', 's', 'h', 'a', 'a', 'n', '\0'}

conio.h for string purpose

(connection input output)

* write a Program to initialise a character array in compile time and display.

```
# include <stdio.h>
# include <conio.h>
   main ( )
        {
          char name [10] ;
          Printf(" char name is %s", name);
        }
```

getchar ( ) :-

→ getchar ( ) is used to accept or read a single character from standard input output device from the keyboard.

Syntax :

```
char ch ;
ch = getchar ( );
```

* write a Program the character from the user using getchar ( )'

```
# include <stdio.h>
# include <conio.h>
  main ( )
    {
      char name [10] ;
      Printf(" Enter a character" );
      name = getchar ( );
      Printf(" char name is %c", name );
    }
```

* write a Program which is accept from a user and will Print the string.

```
# include <stdio.h>
# include <conio.h>
  main ( )
    {
      char str [10] ;
      Printf(" Enter a string" );
       getchar (string );
      Printf(" string is %s ", str );
    }
```

String Manipulation function :-

→ C supports a set up library functions for manipulating the strings :

→ The string handling function define with the header file string.h.

→ strlen ( ) : This function is used to find the length of the string.

→ strcpy ( ) : This function is used to copies one string into another string.

→ strcmp ( ) : This function is used to compare two strings lexicographically.

→ strcat ( ) : This function is used to concate two strings.

→ strrev ( ) : This function is used to reverses a string.

→ strlwr ( ) : This function is used to convert uppercase letter in a string to lowercase.

→ strupr ( ) : This function is used to convert lower letter in a string to uppercase.

* Write a Program which will accept a string from the user and Print the length of the string without using library function.

```
# include <stdio.h>
# include <conio.h>
# include <string.h>
   main ( )
      {
        char s1 [10]; string is .
        int i , l=0;
        Printf(" Enter a string name ");
        gets ( a );
        for(i=0; a[i] != '\0'; i++)
          l++;
        Printf(" Length of %s", %d ",
                              a,l);
      }
```

* Write a Program to copy the string without using library function.

```
# include<stdio.h>
# include<conio.h>
# include <string.h>
    main ( )
      {
        char s1[5], s2[5];
         int i ;
        Printf(" Enter a string ");
          gets (s1 );
        Printf(" Enter another string ");
          gets (s1 );
        for ( i=0; i != '\0' ; i++)
        {
          s1[i] = s2[i];
        }
        Printf(" The original string is %s",
                              s1);
        Printf(" The copy string is %s ",
                              s2 );
      }
```

* write a program to accepts two strings and print two strings without using strcmp ( ).

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
  main( )
    {
      char s1[5], s2[5];
       int i;
      Printf(" Enter a string ");
       gets(s1);
      Printf(" Enter another string ");
       gets(s2);
      while(s1 != '\0' && s2 != '\0' &&
                  s1[i] == s2[i])
        {
          i++;
        }
      if(s1[i] == s2[i])
        {
          Printf(" strings are equal ");
        }
      else
        {
          Printf(" strings are not equal ");
        }
    }
```

* write to accept two string and concate two strings without using strcat ( ).

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
  main( )
    {
      char s1[5], s2[5];
       int k=0, l1, l2, i;
      Printf(" Enter two strings ");
       gets(s1);
       gets(s2);
      for(i=l1; i<=l1+l2; i++)
       {
```

```
    s1[i] = s2[k++];
  }
  Printf(" Two strings are concatenate d.s ", s1[i]);
}
```

* Write a Program to input a string and Print it to reverse order without using strrev( ).

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
    main( )
      {
        char s[10];
        int l, k=0;
        Printf(" Enter a string ");
        gets(string);
        for(i=l-1; i>=0; i--)
        {
          k++=l;
        }
        Printf(" Reverse of the string is d.s ", l);
      }
```

* Write a Program to input a string in uppercase and Print it to lowercase.

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
    main( )
      {
        char str[5];
          int i;
        Printf(" Enter a string ");
        gets(str);
        for( i=0; str[i]!<='\0'; i++)
        {
          if( str[i] >=65 && str[i]<=90)
            str[i] = str[i]+32;
        }
      Printf(" The string is convert to lower
                    case is d.s ", str[i]);
  }
```

Programming In C

Topic:

Function

Contributed By:

Bibhuprasad Sahu

Function :-

→ C Program is nothing but one or more function.
→ A C Program at least one main function.
→ If a Program contain only one function it may be main().
→ If there are more than one function Present in C Program one of those function should be main().
→ There is no limit on the no. of function that Present in a Program main().
→ Each function in a Program is called in the sequence specified by the function calls in the main().
→ After each function has done its thing then the control returns to the main().
→ When the main runs out of func calls then the Program ends.

Defn : A function is a self contain block of code instruction that Perform a specific task and it always returns a value.

Types of func :-

(I) Library function
(II) User defined function

Library function :-

→ This function is defined by C compiler.
→ The name and meaning of the function cannot be change by user.
→ Ex. strcmp( )

User defined function :-

→ This function defined by the user according to their requirement so, the user can modify the func.
→ The user can understand the internal working of a Particular function.
→ abs( ).

Need func in C Programming :-

→ The following benifit can be achieve by the using function in C Program.

→

1. Program Portability :-
This benefit arises from reusability of the code because the program will be becomes smaller in size. So sub-sequent packaging will be small and faster.

2. Code reusability :-
If you want to perform a job repeatedly then it is not necessary to write a particular block of code again and again.
In otherwords we can tell the function which is written once can be use to diff. Program without having rewrite function.
Maintain a large program become easier due to following reason.
(1) Different modules work independently each other.
(2) changing one modules or functions code should have limited effect other parts.

3. Modularising the Program :-
It is very easy to debug the error.

Elements of user defined func as follows :-
1. function defition or called function
2. Function called or calling function
3. function declaration

function definition :-
→ This function defition is an independent module that separately to return the implement.
→ The function definition consists of following elements.
- function name
- function type
- its a Parameter / argument
- local variable declaration
- return statement

Syntax : datatype function name (argument)
{
local variable declaration
statement 1
statement 2
return statement
}

→ The function header contains function name, function type and list of parameter.
→ The parameter / arguments in function definition is called formal arguments.
→ There is no semicolon (;) is used after the function header.
→ The function types specifies the type of the function or which type the value of the function is return.
→ The function body contains local variable, statements and return statements.
→ The function body starts with curly brase and ends with curly brash.
→ The local variable specifies the variable needed by the function.
→ If a variable is declared as a local then it lifetime should be within the particular function.

Function calling :-

→ A function calls simply using the function name followed by the list of parameter.
→ The function is ended with a semicolon (;).
→ Ex -:

```
void main ()
{
    int a=5, b=6;
    add(a,b); / sum(a,b);
}
```

function declaration / Prototype :-

→ Function declaration consists of 4 parts
  (•) function type
  (•) function name
  (•) Parameter list
  (•) terminating semicolon (;)
→ It is same as the function header the only difference is function is ended with a semicolon (;).
→ syntax :- function type function name (argument / Parameter list)
  Ex - int sum (int, int, int);
→ The function prototype indicate the name of function what type of value it return and

→ The Parameter list should be separated by a comma.(,)
→ The type must match with the type of Parameter in the function definition.
→ Use of Parameter in function declaration is optional but datatype must be there.
→ If the function does not return any value then the datatype must be void.

Arguments / Parameters :-

→ It is a variable that communicates between calling function to called function.
→ There are two types of argument
  (i) Actual Argument
  (ii) Formal Argument

Actual Arguments :-

→ The arguments which defines inside the calling func is known as actual arguments.

Formal Arguments :-

→ The arguments that is defines in called function is known as formal arguments.

Types of User defined function :-

There are 4 types of user defined function
(I) function with no argument, no return value.
(II) function with no argument with return value.
(III) function with argument with no return value.
(IV) function with argument with return value.

Function with no argument no return value :-

→ In this type of function the calling func does not pass the argument when it is calls the calling function also the called function does not return value to the calling function.
→ There is no data transfer between calling and called function.
→ Here each function are independent. They read the data, data values, calculate the result and display the same block.

Working Principle of func with no argument no return value -

Calling function

```
main ( )
{
  sum ( )
  ------
  ------
}
```

Called function

```
sum ( )
{
  ---
  ---
}
```

Ex -
```
#include <stdio.h>
void call func ( )
main ( )
{
  call func ( )
  Printf (" I am in calling
           function);
}
```

Ex -
```
#include <stdio.h>
void call func ( )
{
  Printf (" I am in
    called func ");
}
```

* write a program to calculate factorial of a no. using no. argument no return value.

```
#include <stdio.h>
void fact ( );
main ( )
{
  fact ( )
}
void fact ( )
{
  int n, fact = 1 ;
  Printf (" Enter a number ");
  scanf (" %d ", &n);
  while (n > 0)
  {
    fact = fact * n;
    n--;
  }
  Printf (" factorial of a number is %d ",
           fact);
}
```

Function with no argument with return value :-

→ In this type of function the no. of arguements are pass through the calling function to the called function but the called function returns value.

→ The variables required for the called function that are declared and initialise the same called function module.

→ The called function is independent.
→ Here both function communicate partially.

Calling function

```
void main ( )
 {
   int x;
   x = sum ( );
 }
```

Called function

```
sum ( )
 {
  int p=4, q=3,
      s;
  s = p+q;
  return (s);
 }
```

→ The output is printed at the calling function side.
→ The return statement is required for returning a value from called side to calling side.
→ The datatype of returning value and the function return type must be same.

* write a Program to calculate factorial of a no..

```
#include <stdio.h>
 void main ( )
  {
    int n;
    n = fact ( );
    Printf(" factorial of a no = %d ", fact);
  }
 fact ( )
  {
   int n, fact = 1;
   Printf(" Enter a no. ");
   Scanf(" %d", &n);
    while (n>0)
     {
       fact = fact *n;
         n--;
     }
    Printf("
    return (fact);
  }
```

Function with argument no return value :-

→ This type of function the argument are passed through the calling function to called function but the called function does not return value
→ Such type of function partially depended with each other.

Calling function

```
void main ()
{
    int a,b;
    sum (a,b);
}
```

Called function

```
void sum (int x, int y)
{
    int add;
    add = x+y;
    Printf("%d", add);
}
```

* Write a Program to input 3 digit no. and reverse it.

```
#include <stdio.h>
void main ()
{
    int n;
    Printf(" Enter the no ");
    Scanf ("%d", &n);
    reverse (n);
}
void reverse (int n)
{
    int reverse=0, rdg;
    while (n>0)
    {
        rdg = n%10;
        rv = rv * 10 + rdg;
        n = n/10;
    }
    Printf(" reverse of a no. is %d",
            reverse );
}
```

function with argument with return value :-

→ In this type of function the calling function Pass the arguments to the called function and called function send back value to the calling function.

→ Both functions are dependent only each other.

Calling function

```
void main ()
{
    int a=c, b=d, y;
    y= sum (a,b);
    Printf("%d", y);
}
```

Called function

```
sum (int a, int b)
{
    int add;
    add = a+b;
    return (add);
}
```

* write a program to input 3 digit no and reverse it.

```
#include<stdio.h>
  void main ( )
    {
      int n , z ;
      Printf(" Enter a no.");
      scanf(" %d", &n);
      z = reverse (n);
      Printf(" reverse of a no. is %d", z);
    }
  reverse (int n)
    {
      int rv = 0, dg ;
       while (n>0)
        {
          dg = n % 10;
          rv = rv * 10 + dg ;
          n = n/10 ;
        }
       return (reverse) ;
    }
```

There are ~~one~~ two types of calling function.

(I) call by value

(II) call by reference

call by value :-

→ when calling function is called the called function by passing by the value of the actual arguments is known as call by value.

→ In otherwords in this case the value of the variable is passed as function argument.

→ If any change made in formal argument then does not effect the actual argument because the formal argument are photocopy of actual argument.

→ The argument with return value and argument with no return value are call by value concept.

* write a program to swap two no. using call by value concept.

```
# include <stdio.h>
void swap (int , int ) ;
  main ( )
   {
     int a, b ;
      Printf(" Enter two no. a and b ");
      scanf (" %d %d ", &a, &b );
      Printf (" before swapping two variable = %d %d "
                                  a, b );
      swap (a , b) ;
   }
   void swap ( int x , int y )
     {
       x = x+y ;
       y = x-y ;
       x = x-y ;
       Printf(" After swapping two variable = %d %d ",
                                  x, y );
     }
```

call by reference :-

→ When the calling function is calls the called function by passing the address address of the actual argument this is is called by reference.

→ In otherwords the address of the variable can pass the argument.

→ Here the formal argument are pointer to the actual argument.

→ The formal argument must be same type pointer variable because the calling function pass the address of the actual argument.

→ Here the changes made in arguments in parameters.

→ Through call by reference a function can return more than one values.

* write a program to swap two no using call by reference concept.

```
# include<stdio.h>
  void swap (int , int ) ;
   main ( )
    {
```

```
int a, b;
Printf(" Enter two variable a and b");
scanf(" %d %d ", &a, &b);
Printf(" before swapping two variable = %d %d",
        a, b);
}
void swap ( int *x, int *y)
        {
        int temp;
        temp = *x;
        *x = *y;
        *y = temp;
Printf(" After swapping the value is %d %d",
                *x, *y);
        }
```

Variable of function :-

(I) Local variable
(II) Global variable
(III) static variable

Local variable :-

→ The variable that is define inside the body of func on block one called local variable for the particular function.

```
Ex - abc ( )
     {
     int a, b, c;
     }
```

→ Here the variable a, b are define with in the body of abc.
→ Local variable are used only these function of block.
→ The some variables name may be use in different concept.

```
Ex - abc ( )
     {
     int a=5, b=6;
     }
     xyz ( )
     {
     int a=8, b=9;
     }
```

Global variable :-

→ The variable that are define outside any function is called global variable.
→ All function in the Programe can access and modify global variable.
→ It is useful to declare a global variable in a function if it is use many function in the Program.
→ Global variable are automatically initialise to zero at the time of the declaration.

```
# include <stdio.h>
  void function 1 (void);
  void function 2 (void);
    int a, b = 6;
      main ( )
        {
          Printf(" inside main : a = %d, b = %d ",
                    a, b);
          function 1 ( );
          function 2 ( );
        }
      void function 1 (void)
        {
          Printf(" inside function ( ); a is %d "
                  b is %d", \n", a, b);
        }
    void function 2 (void)
      {
        ~~int~~ a int a = 7;
          Printf(" inside function 2 ( ) : a
          is %d ; b is %d \n", a, b);
      }
```

Static variable :-

→ static variable are declared by writting a keyword static infront of declaration.
→ syntax - static type :- variable name
→ A static variable is initialise only once and the value of variable written called function.
→ If the static variable is not initialise then it is automatically initialise to zero.

→ Ex: #include<stdio.h>
void function (void)
main ( )
{
function ( );
function ( );
function ( );
}
void function (void)
{
int a = 10;
static int b = 10;
Printf(" a = %d", b = %d", a, b);
a++;
b++;
}

O/P

	a	b
1	10	10
2	10	11
3	10	12

O/P

10 10
10 11
10 12

→ Here the variable b is static variable.
→ When a function called b 1st time it is initialise value 10. But inside function the value of b becomes 11. Next time the function the value of b becomes 12. But the variable a which is not static initialise an can call the value remain same.

Recursion :—

→ This is the function which calls itself repeatedly until certain condition is satisfy is called as a recursion.
→ In a recursion looping concept is not used.
→ only if else statement can be used. fact = no * fact (n-1)
→ write a program to calculate a factorial of a no using recursion function.

33

```
#include<stdio.h>
void main()
{
    int a, n;
    Printf(" Enter a no. for a ");
    Scanf(" %d", &a);
}
fact(int no)
{
    int no, fact = 1;
    if(no==1)
    {
        return(1);
    }
else
    {
        fact = fact(n-1) * no;
    }
    return(fact);
}
```

Programming In C

Topic:

Storage Class

Contributed By:

Bibhuprasad Sahu

Storage class :-

→

→ When a storage class is not present in the declaration the declaration the compiler assumes a default storage class based on the place of declaration.

→ The storage class besides the following ~~variab~~ various aspect of a variables.

(I) Lifetime : Time between the creation and destruction of a variable.

(II) Scope : Locations where the variable is available for use.

(III) Initial value : Default value taken by all initial variables.

(IV) Place of storage : Place in memory where the storage is allocated for the variables.

→ syntax : class data type variable name ;

Need of storage class in 'C' :-

By the Purpose use of storage class makes our Program efficient and fiest..

Types of storage class :-

(I) Automatic (auto)
(II) External (Extern)
(III) Static (static)
(IV) Register (register)

Automatic :-

→ All the variables inside block or function without any storage class specifier is called automatic variables.
→ The keyword 'auto' is used.
→ Ex:
```
function ( )
{
    int a, b;
}
function ( )
{
    auto int a, b;
}
```
→ This is called automatic because the storage is reserve automatically each time when the control enter the function or block and the released automatically when the function are block is terminated.
* Ex. Write a Program to illustrate default initial value of an automatic storage class.

```
main ( )
{
    auto int sum;
    Printf (" %d", sum)
}
```

O/P
Garbage value

→ In this Program the variable sum is automatic storage class.
→ Here the variable sum is not initialised.
→ Still we want to Print the value of the sum. So compiler it will print and unPredictable value. The value is known as garbage value.

* write a program to illustrate default initial value of an automatic storage class.

```
#include <stdio.h>
main( )
{
    Auto int sum ;
    Printf(" %d", sum);
}
```

→ In this program the variable sum is automatically storage data / class.
→ Here the variable sum is not initialised.
→ so we want Print the value of sum so (compiler) it will Print on unpredictable value. This value is known as garbage value.

Register storage class :-

→ The feature of register storage class are given below.
- Storage area : CPU register
- Default initial value : Garbage value
- Scope : Local to the block in which the variable is declared.

* write a program to illustrate register storage class.

```
#include <stdio.h>
#include <conio.h>
main( )
{
    register int x;
    Printf(" Enter a no.");
    for ( x=1; x<=4; x++ )
    Printf(" %d", x);
}
```

O/P
1 2 3 4

Static storage class :-

→ Here the keyword static is used to declare static variable
→ The static variables are not reinitialised but the auto variable are reinitialised. The small difference of auto is initial value of auto is garbage value but static is zero.

* write a program to illustrate the work of static storage class.

```
# include <stdio.h>
# include <conio.h>
  main ( )
    {
       xyz ( );
       xyz ( );
       xyz ( );
    }
     xyz ( );
     {static
        static int x;
        Printf(" xyz ");
        x++;
     }
```

O/P

0 1 2

```
# include <stdio.h>
# include <conio.h>
  main ( )
    {
       xyz ( );
       xyz ( );
       xyz ( );
    }
     xyz ( );
      {
        auto int x;
        Printf (" xyz ");
        x++;
      }
```

O/P

1 1 1

External storage class :

→ The variables that are active and alive throughout our entire program.
→ External variable are also known as global variables.
→ Here the keyword extern is used to declare external variable.
→ The external variable is used to outside in function.

```
# include <stdio.h>
# include <conio.h>
  main ( )
   {
     extern  xyz ( );
             xyz ( );
             xyz ( );
   }
  xyz ( );
  {
    Printf (" xyz " );
     x++;
  }
```

O/P

0

```
# include <stdio.h>
# include <conio.h>
  extern x=5
   main ( )
    {
       xyz ( );
       xyz ( );
       xyz ( );
    }
  xyz ( );
   {
     Printf (" xyz " );
      x++;
   }
```

O/P

5 6 7

Programming In C

Topic:

Pointer

Contributed By:

Bibhuprasad Sahu

Pointer :-

→ Pointer is a derived data types.
→ A Pointer holds address not value.
→ A Pointer can be defined as it is a memory variable that stores a memory address.
→ It is denoted by '*' operator.

Advantages of using pointer :-

→ Pointer is used to call a function by reference.
→ Pointer is used to dynamic memory allocation.
→ Using pointer memory space can be save.
→ Execution time with pointer is faster than the data is manipulated with address
→ Memory access can be done efficiently.
→ Pointer in C Providing an alternative way to access information from array.

Declaration of pointer :-

syntax : datatype * <variable name>
Ex : int * P;
Here the * mark before the variable indicate the it is a pointer variable.

Operator used in pointer :-

There are two operator are used in pointer.
(i) *
(ii) &

'*' (indirection operator) :-

→ It is a unary operator,
→ It is used to create a pointer variable.
→ It is used to findout the value of the address.
→ The indirection operator is known as the dereference dereference operator. when a pointer is dereference then the value stored by the pointer at that address is retrived.

'&' operator :-

→ This is a unary operator.
→ This is used to findout the address of a variable.

* Write a Program to Print the address of a variable.

```
#include <stdio.h>
main( )
{
    int a;
    Printf(" Enter a variable ");
    scanf(" %d", &a );
    Printf(" The value of a is %d ", &a);
}
```

Types of Pointer :-

Depending on memory models and segment pointer is classified into 3 types.

(i) near pointer
(ii) far pointer
(iii) huge pointer

Near pointer :-

→ A near pointer is a pointer which works within 64 Kb data segment of memory.
→ It cannot access address beyond the data segment.
→ These under normal circumstances all pointer that we create near pointer.
→ It stored the upset part of the address in the memory.
→ A near pointer can be increamented or decreamented through the entire address range using arithmetic operator.

syntax - <datatype> near <pointer definition>
<datatype> near <function definition>

Ex - char near *s
int (near (*P) ip)

Far pointer :-

→ A far pointer stores both upset and segment of the address to which the pointer is differencing.
→ A far pointer address ranges in 0 to 1 MB.
→ When the far pointer is increamented or decreamented only the upset part is increamented or decreamented.
→ syntax. <datatype> far <pointer definition>
<datatype> far <function definition>

Ex. char far *s ;
int (far (*P) 10) ;

Huge Pointer :—

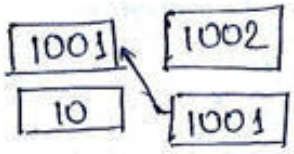
→ The huge pointer is similar to far pointer, in term of size because both are 32 bit address.

→ The huge pointer can be increamented without suffering segment wise record.

→ Ex int P = 10 ; *P

P = &a

10 = *(&a)
*(P)
*(1002)
10



Accessing value through pointer :—

→ This is can be done by using indirection operator so this operator is also known as value at address operator.

* Write a Program to access a value to a variable through pointer.

```
#include<stdio.h>
#include<conio.h>
    main()
      {
        int a, *ptr;
        Printf("Enter a value of a");
        Scanf("%d", &a)
          ptr = &a;
        Printf("%d", *ptr);
      }
```

Chain Pointer :- (Pointer to Pointer)

→ when a pointer variable point to another pointer variable is called chain pointer.

→ It also known as Pointer to pointer.

→ syntax - * (*P)

address 2 → address 1 → value

Ex - int p = 10, *p, **p ;

$p_1 = \&a_1$

$p_2 = \&\&a_1 \;/\; \&p_1$

```
# include <stdio.h>
# include <conio.h>
   main ( )
     {
        int a1, *ptr1, **ptr2 ;
        Printf (" Enter a value ");
        Scanf (" %d", &a1 );
         ptr1 = &a1 ;
         ptr2 = &&a2;
        Printf (" %d", *ptr1 );
        Printf (" %d", **ptr2 );
     }
```

Pointer Arithmetic :-

```
# include <stdio.h>
# include <conio.h>
  main ( )
    {
      int a, b, S, X, M, D, ptr1, ptr2 ;
      Printf (" Enter two variable " );
      Scanf (" %d %d", &a, &b);
         ptr1 = &a ;
         ptr2 = &b ;
      S = *ptr1 + *ptr2 ;
      X = *ptr1 - *ptr2 ;
      M = *ptr1 * *ptr2 ;
      D = *ptr1 / *ptr2 ;
      Printf (" %d %d %d %d", S, X, M, D );
    }
```

Pointer and Array :-

```
# include <stdio.h>
# include <conio.h>
   main ( )
     {
        int a[3][3] , i , j ;
          int *ptr ;
```

```
        ptr = a;
        for(i=0; i<=3; i++)
        {
        for(j=0; j<=3; j++)
        {
        printf("%d", *ptr, ptr);
          ptr++;
        }
        }
}
```

Pointer and characterstics :-

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
    main()
      {
          char str[10];
          char *ptr;
          printf("Enter a name");
            gets(str);
            puts(str);
            ptr = str;
          printf("name = %c", *ptr);
      }
```

Pointer and function :-

The arguments to the function can be pass two way

(i) call by value
(ii) call by reference

call by value :-

→ In call by value only values of the arguments are sent to the function.

→ Any change made in formal argument donot change.

* Write a Program accept 2 no. and print increament by 1. using call by value concept.

```
#include <stdio.h>
void main ( )
{
    int a, b;
    Printf(" Enter two numbers ");
    scanf(" %d %d ", &a, &b);
    Printf(" before increament the value = %d %d ",
                                    a, b);
    increament (a, b);
    Printf(" After increament the value = %d %d",
                        a, b);
}
void increament (int x, int y);
  {
     x++ ;
     y++ ;
     Printf(" After increament the value = %d %d",
                        x, y);
```

call by reference :-

→ In call by reference only pass the address of the argument.

→ If any modification is done in formal argument it will affect in actual argument.

* Write a Program to accept 2 no. and print increament by 1 using call by reference concept.

```
#include<stdio.h>
voidmain ( )
{
    int a, b;
    Printf(" Enter two number ");
    scanf(" %d %d", &a, &b);
    Printf(" before increament the value = %d %d",
                        a, b);
    increament (&a, &b);
    Printf(" After increament the value = %d %d",
                        a, b);
}
void increament (int*x, int*y);
  {
    (*x)++ ;
    (*y)++ ;
    Printf(" After increament the value = %d %d",
```

Dynamic memory allocation :-

Memory allocation in C :-

(I) Compile time (static)
(II) Runtime (dynamic)

Compile time memory allocation :-

→ In compile time memory allocation the required amount of memory is allocated to the element at the start of the program.
→ Otherwise we can tell the allocation of memory is done before compilation of the program
→ Ex - int a[3]
2*3 = 6 (memory is required)
because integer takes 2 bytes.

Limitation of compile time memory allocation :-

- can't grow or shrink of the size of declared memory.
- In static memory allocation if you store less no. of element then the no. of elements for which we have declared memory then memory will be wasted.

Dynamic memory allocation :-

→ To develop the overcome the limitation of compile time memory allocation.
→ The process of allocating memory at the time of execute is called dynamic memory allocation.
→ We can access the dynamic memory allocation pointers only.
→ C programming allowing dynamic allocation and d- allocation.

(I) Malloc ( )
(II) Calloc ( ) } allocation purpose
(III) Realloc ( )
(IV) free alloc ( ) → d- allocation purpose

Malloc( ) :-

→ Malloc ( ) is used to allocated a block of memory in bytes.
→ It takes only one argument.
→ The user should explicitly give the block size it required for use.
→ The malloc function request RAM of the system to allocate memory, if the request is granted it returns a pointer to the first block of memory.
→ The type of pointer it returns void which means we can assign it any type of pointer.
→ when malloc function fail to allocate it returns null.
→ The memory allocated by malloc function contains Garvage value by default.
→ The headerfile <malloc.h> or <stdlib.h> both used in turbo c.
→ syntax - malloc (5 * size of (int));
Ex - int * ptr;
ptr = malloc (5 * size of (int));
ptr = (int *) malloc (5 * size of(int));
ptr (datatype *) malloc (specified size * size of (datatype));

* write a Program to find 5 integer no. using dynamic memory allocation.

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
main ( )
{
    int n, i;
    int * ptr;
    Printf(" Enter the elements");
    Scanf(" %d ", &n);
    ptr = malloc (int *) malloc (size of (int));
    if( ptr == null )
    {
        Printf(" The required memory is not
                                available ");
    }
    exit(0);
    else
```

```
{
    Printf(" Enter the elements ");
    for (i=0; i<5; i++)
    {
        Printf("%d", ptr+i);
    }
}
```

calloc ( ) :-

→ The calloc ( ) is used to allocate multiple block of memory.

→ syntax - void * calloc (size_t, size_esize)
(no. of element) (size of no. of element)

The basic difference betn malloc ( ) and calloc ( ) :-

→ It takes two arguments

→ The memory ~~allocation~~ allocated by calloc ( ) contains all zero.

→ The calloc ( ) also returns null if no memory is available.

Realloc ( ) :-

→ This function is used to resize the memory function. This allocated by calloc ( ) and malloc ( ).

→ This function is needed in following situations.
(i) If the allocated memory block is insufficient for current application.
(ii) If the allocated memory is much more sufficient then the actual requirement.

syntax :- ptr
ptr_variable = realloc ( ) (ptr_var, newsize);

```
ptr = int * (malloc size);
ptr = int * (realloc (ptr newsize);
```

free alloc ( ) :-

→ This function is used deallocate the memory which is previously allocated.

→ syntax - void free (void * p);
free (ptr);

40

Write a Program to use the 3 function.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
   main ( )
      {
         char *ptr;
         char a;
         ptr = malloc (char * )malloc (size of (char ));
         strcpy (ptr, "GIFT");
         Printf(" new string is %c ", ptr);
         void calloc (size of element, size of no. of
                        element );
         strcpy (ptr, "COLLAGE");
         Printf(" new string is %c ", COLLAGE ");
          free (ptr);
          strcpy( ptr, "calloc");
          Printf(" new string is %c", ptr );
      }
```

void Pointer :—

→ A pointer to void is a generic pointer that can point to any datatype.

→ syntax - void * ptr ;

Programming In C

Topic:

Structure

Contributed By:

Bibhuprasad Sahu

STRUCTURE

Structure :-

→ A structure is a ~~datatype~~ collection of one or more variable of different datatype grouped together in a single name.

→ structure is a collection of hetrogeneous datatype.

→ syntax - struct structure name
{
datatype 1 member 1
datatype 2 member 2
};

→ Ex - struct student
{
int regd no;
int char name;
}; int mark;

NOTE :- Within structure we can't declare func.
With in structure we can't initialise the value of structure member.

Declaration of structure variable :-
After declaration of the structure formate we can declare the variable that type.

Instructure declaration :-

```
struct structure name
    {
      int pages;
      char book name [5];
      float price;
    }; b1
```

Here b1 is the variable of book value.

Initialisation of structure :- (@ compile time)

(1)
```
struct
    {
      int pages;
      char name [5];
      float price;
    };
    b1;
  struct book b1 = { Das, 10, 100 };
```

This concept is declaration of variables.

Using (.) operator :- (runtime)

```
struct book
  {
    char book name;
     int pages;
    float prices;
  };
struct book b1;
  {
    b1. name
    b1. Page
    b1. price
  }
```

* write a program to enter to declare a structure and initialise the values for the members and display it.

```
#include <stdio.h>
#include <conio.h>
  main ( )
    {
      struct student
        {
          int rollno;
          char name;
          int mark;
        };
      struct student s1 = {01, sony, 100}
        printf(" rollno is %d ", s1.rollno);
        printf (" The name is %c", s1.name);
        printf(" The mark is %d ", s1.mark);
    }
```

Assign a structure variable :—

```
#include <stdio.h>
#include <conio.h>
    struct student s1
      {
        int rollno;
        char name;
        int mark;
      };
      main ( )
        {
          struct student s1 = {01, chandini, 100};
          struct student s2
            s1 = s2
          Printf(" The s1 student of rollno, name &
                  mark is %d, %c %d ", s1.rollno,
                  s1.name, s1.mark);
          Printf(" The s2 student of rollno, name &
                  mark is %d %c %d ", s2.rollno,
                  s2.name, s2.mark);
        }
```

Array of structure :-

```
# include <stdio.h>
#include <conio.h>
   main ( )
   {
     struct stud
        int rollno;
        char name;
        int mark;
   };
struct student S[3];
    int i;
   for (i=0; i<3; i++)
   {
     Printf(" Enter the student rollno, name, mark");
     scanf(" %d %c %d", &S[i], &S[i] rollno,
                S[i]. name, S[i]. mark);
   }
   for (i=0; i<3; i++)
   {
     Printf(" The students rollno, name and
          mark is %d %c %d", S[i]. rollno,
          S[i]. name, S[i]. mark);
   }
}
```

Nested structure :-

→ Declaration of variable of one structure to an another structure is known as nested structure.

→ syntax -

```
struct time
{
   int second;
   int minute;
   int hour;
};
struct xyz
{
   int x;
   struct time st;
};
struct xyz var;
```

112

* Write a Program to illustrate nested structure.

```
#include <stdio.h>
#include <conio.h>
    main ( )
      {
        struct time
         {
           int second;
           int minute;
           int hour;
         };
        struct xyz
         { int x;
           struct time st;
         };
        struct xyz x;
        Printf the values for x
        Printf(" Enter the value of x");
        Scanf(" %d", &x.x);
        Printf(" Enter the value of second, minute,
                 hour ");
        Scanf(" %d %d %d", &x.st.second, &x.st.
              minute, &x.st.hour);
        Printf(" %d %d %d", x.st.second, x.st.
              minute, x.st.hour);
      }
```

Pointer to structure :-

→ pointer to structure means we can assign the address of a structure to the structure type pointer variable.

→ It points to the starting address of the a structure.

→ To access the member of a structure by the structure pointer we use arrow (→) operator.

syntax - struct <name> * ptr;

Ex - struct student * ptr;

* Write a Program to illustrate concept of pointer to structure.

```
#include <stdio.h>
struct student
{
    int rollno;
    int marks;
    char name;
};
struct student s,*s1;
{
    Printf("Enter the roll no");
    Scanf("%d", &s.rollno);
    Printf("Enter the marks");
    Scanf("%d", &s.marks);
    Printf("Enter the name");
    gets(s.name);
    s1 = &s;
    Printf("%d %d %c", &s.rollno,
    (s1)->marks, (s1)->name);
}
```

Structure and function :-
Like standard data type variable the structure variable can passed as argument in function argument in two ways.

(i) call by value
(ii) call by reference

Call by value :-
→ when the value of the structure variable is passed as argument of calling function are called call by value.

* write a program to pass the structure variable two user defined function and display using call by value.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
struct student
{
    int rollno;
    int marks;
    char name;
};
main()
```

```
struct student s;
Printf(" Enter the student rollno, mark, name ");
Scanf(" %d %d %c " , &s.rollno , s.mark , s.name );
  stud (s) ;
};
  stud (struct stud s);
   {
    Printf(" The student rollno, mark, name is %d %d
      %c ", s→ rollno, s→ mark, s→ name );
   }
```

call by reference :-

→ when the value of the address structure variable is passed as argument of calling function are called call by reference.

* write a Program to pass the structure variable two user defined function and display using call by reference.

```
# include <stdio.h>
# include <conio.h>
# include <stdlib.h>
    struct student
        {
          int rollno;
          int mark;
          char name;
        };
        main ()
         {
           struct student s;
           Printf(" Enter the rollno, mark, name ");
           Scanf(" %d %d %c ", s.rollno, s.mark,
                    s.name );

           stud (&s);
         }
        stud (struct stud *s);
         {
           Printf(" The student rollno, mark, name =
                    %d %d %c ", s→ rollno, s→ mark,
                      s→ name );

         }
```

NOTE :

When a structure element is to be pass to any other function it is essential to declare the structure outside the main function as global.

User defined datatype :- (typedef)

syntax - typedef <type> type name ; or new type ;

Using typedef we can create new data types.

NOTE-1 :- Type refers when existing data type which can be either a standard datatype or previous user defined datatype.

NOTE-2 :- New type refers the new user defined data type.

NOTE-3 :- The new datatype will be new in the name only actually the new datatype will not different from one of the data type.

Ex - Typedef int integer

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

It is same as int a, b, c ;

```
struct book
{
  ------
  ------
  ------
};
```

typedef struct book b1 ;

or typedef struct book book1

book s1

* write a program to create a userdefined data type GIFT or character data type and print your name.

```
# include <stdio.h>
# include <conio.h>
# include <stdlib.h>
  main ( )
    {
      typedef char GIFT ;
        GIFT name;
      Printf(" Enter your name " );
      Scanf(" %c ", &name );
```

printf(" your name %c ", name);
}

Difference Between Array and structure :-

1. Array is the collection of related data element of same type of element where as structure is the collection of different data type.
2. Array is derived datatype where as structure is Programming defining one.

Union :-

→ It is heterogeneous user defined datatype.
→ The keyword union is used.
→ syntax- union union name < datatype > variable name.

Difference Between structure and union :-

→ structure is same as union but the structure holds many object at a time.
→ In structure his member has own memory location members of union has same memory location.
→ Example syntax- union union name
{
datatype 1 variable name 1 ;
datatype variable name n ;
};
→ Example - union book
{
int page;
float price;
};

Ex -

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
union student
{
int roll no;
int mark;
char name;
};
```

```
Union stud S ;
  {
    Printf(" Enter the rollno");
    Scanf(" %d", &S.rollno );
    Printf(" Enter the mark");
    Scanf(" %d", &S. mark );
    Printf(" Enter the name");
     Scanf(" %c", &S. name );
    Printf(" The student rollno, mark, name is - %d %d
            %c ", S. rollno, S. mark, S. name );
  }
```

File Management :—

File :- • A file is a complex datatype i.e stored to the external to the main memory.
• It is a storage area of memory where the data are stored in a organised form.

Stream :- • A stream is a full duplex connection between Process and a device driver.
• A stream linked to a file using open ( ) and disconnected to the file using close ( ).

File formates :-
There are two types of formate to formate the data.
(I) Binary formate
(II) Text formate

Steps of file operation :—
→ 'C' Programming supports a no. of functions that have ability to perform basic file operation.
→ which includes naming a file, opening a file, reading a file, writing a file, closing a file.
But there are mainly 3 operation.
(I) Opening a file
(II) Reading / writing a file
(III) closing a file

Opening a file :-

→ A file must be opened before co operation can be

Defn :-

A process of establishing connection between in Program and file c's called opening file.

syntax - FILE *FP
FP = Fopen ("file name", "mode");

Ex - FILE *FPtr
FPtr = Fopen (result.dat", r)

Reading a file :-

Once a file c's opened with a f-open function it contains are load in memory.
Now we can use to getc ( ) to con

Closing a file :-

The file that c's opened by fopen ( ) should be close after the work c's over.
To close a file following functions are used.

1. fclose ( )
2. fclose all ( )

fclose ( ) :-

→ It c's use to close a specia specified file.
→ syntax - fclose (FILE *)
Ex - File *FP;
FP = fopen ("GIFT.Txt", "w");
fclose (FP);

fclose all ( ) :-

→ It closed all files that are Previously open.

Eof :- (End of file)

→ A stream terminates with an end of file maker.
→ An Eof may be written like a boolean variable ! (true) indicates it has be found and 0 (false) indicates it has not found.

Modes of accessing :-

There are different modes in file can be access.
(1) Text mode

→ Text mode is 2 types.
(a) write mode (w)
(b) Read mode (R)

Write Mode :-
→ This mode is used to open a file for writing.
→ If the concern file is not available then a new file is created.
→ If the file is already exist the data will be over written without any convertation.
→ syntax - fopen ("file name", "w");
Ex - fopen ("GIFT.Txt", "w");
Here the .Txt is the existing file and 'w' is the mode.

```
#include <stdio.h>
#include <conio.h>
  main ( )
   {
     FILE , *FP;
       char C;
     FP = fopen ("Arpita.Txt", "w");
      while((C = getchar ( )) != EOF( ))
        {
          fputs (C, FP);
        }
        fclose (FP);
   }
```

Read Mode :-
→ This mode is used to data to read a file.
If the file is opened successfully then the file pointer points to the 1st character of the file doesnot exist the compiler will reen return to the ncell of the pointer.
→ syntax -
FP = fopen ("file name", Mode);
Ex - FP = fopen ("Arpita.txt", "R");
fclose (FP);

```
#include <stdio.h>
#include <conio.h>
  main ( )
    {
      FILE * FP ;
       char C;
       FP = Fopen ("Arpita . Txt","w" );
       while ((c = getchar ( ))! = EOF( ))
         {
          fputs (c, FP);
         }
          fclose (FP);
       FP = Fopen ("Arpita . Txt", " R");
        while ((c = gets (FP)! = EOf( ))
         {
           fputs ( c, FP);
         }
          Printf(" %c", c);
         fclose (FP);
    }
```

Append Mode:-

→ This mode opened the existing file adding new data at the end of the file.

→ If file opened successfully then the file pointer points to the last character of the file.

→ If the file is not exist then any file is created.

→ syntax - FP = Fopen ("Filename","a");

* write a program to write a data text file and read it using append mode.

```
#include <stdio.h>
#include <conio.h>
   main ( )
     {
       file * FtP ;
        char C;
       FP = Fopen ("Arpita . txt", "R");
       while (c = gets(FtP)! = Eof( ))
         {
          fputs (c, FtP);
         }
```

```
Printf("%c", c);
  fclose(FtP);
   if(FtP == NULL)
    {
     Printf("The file cannot be append");
     exit(0);
    }
    while(c != '.')
     {
      c = getche()
      fputs c(FtP)
       fclose(FtP);
     }
    fp = fopen("Arpita.txt", "R");
      felose(FtP);
  }
```

W+ : (write + read)

→ This mode opens a file for both write and read.
→ Writing new contains reading them and modify.
→ syntax fp = fopen("file name", "w+");

R+ : (Read + write)

→ This mode is used for both read and write.
→ syntax - FP = fopen("file name", "R+");

A+ : (Append + Read)

→ In this mode file can be read and record can be added end of the file.

Binary Modes :-

WB : (write Binary)

→ This modes opens a binary file in write mode.
→ syntax - FP = fopen("file name", "WB");
→ Its extension is .bat.
→ Ex - FP = fopen("Arpita.bat", "WB");

WR : (write read)

→ This mode is open a binary file in read mode.
→ syntax - FP = fopen("file name", "RB");
→ Ex - FP = fopen("Arpita.bat", "RB");

AB (write) :-

→ This mode can be opened a binary file in append mode and data can be added in write mode.

→ syntax : FP = fopen ("file name", "ab");

→ Ex :- : FP= fopen ("Arpita.bat", "ab");

W+B : (Read + write)

→ This mode creates a new file read and write mode.

→ syntax - FP = fopen ("file name", "w+B");

→ Ex - FP =fopen ("Arpita.bat", "w+B");

R+B : (write + Read)

→ This mode opens a pre existing file in read and write mode.

→ syntax - FP = fopen ("file name", "R+B");

→ Ex - FP= fopen ("Arpita.bat", "R+B");

Write a Program to open a for read and write in binary mode.

Read write the new information in the file.

```
#include <stdio.h>
#include <conio.h>
main( )
{
    FILE * FP;
FP=fopen ("Arpita.bat", "RB");
```

Predefined file pointer :-

→ The Predefined constant file are opened automatically when the Program is executed when the Program is executed.

file pointer	Device
1. stdin	standard input device
2. stdout	standard output device
3. stderr	standard error

function used for c/o :-

1. character type
   - fgetc ( )
   - fputc ( )
   - getc ( )
   - putc ( )
2. String type c/o :
   - fgets ( )
   - fputs ( )
3. Integer c/o :-
   - getw ( )
   - putw ( )
4. Formatted c/o :-
   - fscanf ( )
   - fprintf ( )
5. Unformatted c/o :-
   - fread ( )
   - fwrite ( )

character c/o :-

Fputc ( ) :

→ This function writes a character to the specified file at current file position and increament the position of the file pointer the file.

→ syntax : fputc (int c , file * ptr);

* write a Program to increament fputc ( ).

```
# include <stdio.h>
# include <conio.h>
  main ( )
    {
      FILE *FP
       char c;
      FP= FOpen ("Arpita.txt", "w");
      if( FP = FOpen ("Arpita.txt", "w") == null )
       {
          ch = getchar ( ) != EOf ( )
           fPutc (ch, FP);
       }
        else
        {
```

fgetc ( )

→ This function reads a single character from a given file and increamented the file pointer position.

→ Ex-

```
# include <stdio.h>
# include <conio.h>
   main ( )
     {
       FILE *FP ;
        char c;
       FP= FOpen (" Arpita.txt", "r");
       if( FP= FOpen("Arpita.txt", "r") ==
                                  Null )

        Printf(" The file does not exist " );

       else
        Printf(" Enter the character" );
        while (ch = getc (FP)) != EOf )
        {
          fgetc ( "1.c", ch)
        }
     }  fclose ( FP);
```

Putw( ) :-

→ This function is used to read

→ syntax : int putw (int , FILE * FP);

getw( ) :-

int getw (FILE *FP);

This function is used to read an integer data from file.

Write a Program to implement getw( ).

```
#include<stdio.h>
#include <conio.h>
   main( )
     {
        FILE *FP ;
        int no ;
        FP= fopen("Arpita. dat ","WB");
        if( FP == NULL)
          {
            Printf(" Error in reading");
            return 0 ;
          }
          else
          while (no = getw(FP) ! = Eof( ))
          {
            Printf(" %d", no);
          }
            fclose (FP);
     }
```

Formatted I/O :-

→ **fPrintf ( ) :-**

→ This is same as Printf but it writes formatted data into the file into the file.

→ It contains more than one parameter i.e file pointer points the the

* Write a Program to open a text file.

```
#include<stdio.h>
#include <conio.h>
   main( )
     {
        FILE  *FP;
        char text [30] ;
        FP = fopen ("GIFt. txt ","w");
```

```
// clrscr( );
 Printf(" int
  gets (text);
 fPrintf("FP, %s", "text");
  fclose (FP);
}
```

fscanf( ) :-

→ This function reads character, string, integer, float etc, from the ~~function~~ file pointer pointed by the file pointer.

```
# include<stdio.h>
# include<conio.h>
  main ( )
   {
      FILE *FP;
      char text[30];
      FP = FOPEN ("GIFT txt", "w");
      Printf ("name\t page");
      scanf ("%c %d", &name, &page);
      fPrintf ("FP, %s %d", name, page);
      Printf ("Name \t page);
      fscanf ("FP, %s %d", &name, &page);
       fclose (FP);
   }
```

fwrite( ) :-

→ This function is used for writing a structure variable in your file.

→ syntax: fwrite (const, void *ptr, size of (data), no. of data, file * FP);

ptr :-

ptr is the pointer which points the block of memory information to be refers of file.

size of data :-

It indicates the length of each item

No. of item) :

It indicates the no. of items to be refers to the file.

*FP:
→ It indicates the file where data are return.
→ This is used to read the entire block from a given file.
fread (const void *ptr, size of (data), no. of data file *FP);

Accessing of files :-
To access a particular record following functions are used.
(I) fseek( )
(II) ftell ( )
(III) rewind ( )

fseek( ):
→ This is used to move the file pointer to the particular position.
→ fseek (file *FP, long size, mode);
*FP = file pointer
long size = It is the positive long integer to move reposition of file pointer towards backward or forward direction.
Mode = It is the current position of the file pointer. (Mode)
It is 3 types
(I) seek - set 0
(II) seek - current 1
(III) seek - end 2

Beginning of file :-
Ex - seek (FILE *FP, 10, seek - set);

ftell ( ):-
It returns the current position of the file pointer.
syntax - long file (file *FP);

rewind ( ):
→ It is used to move the file pointer position to the beginning of the file.
→ syntax - void rewind (file *FP);
→ Ex - fseek (FILE *FP, seek_set 0);

Command Line argument :-

→ This argument are issued from the operating system.
→ some argument are to be associated with the commands hence, the arguments are known as command line arguments.
→ syntax - void main (int argc, char * argv []);
Here argc it is the integer type count total no. of Parameter pass in the command Prompt.

argv [] - It is the character the pointer array that the contain address of Parameter pass in command pointer.

```
Ex - main (int argc, char * argv [])
     {
       int i;
       Printf(" Total no. of argument %d",
                argc);
       for(i=0; i<argc; i++)
       {
         Printf("%d", argv [] );
       }
     }.
```

Error Handling During i/o Operation :-

while reading or writing some errors may be occure.
(I) opening a file with an invalid file name.
(II) write attempt to write a write Protection file.
(III) Trying to use a file that has been opened.
(IV) Trying to read beyond to end of the file mark.

syntax : int feof (FILE * stream)

```
Ex -  if (feof (fp))
        Printf(" End of data ");
```