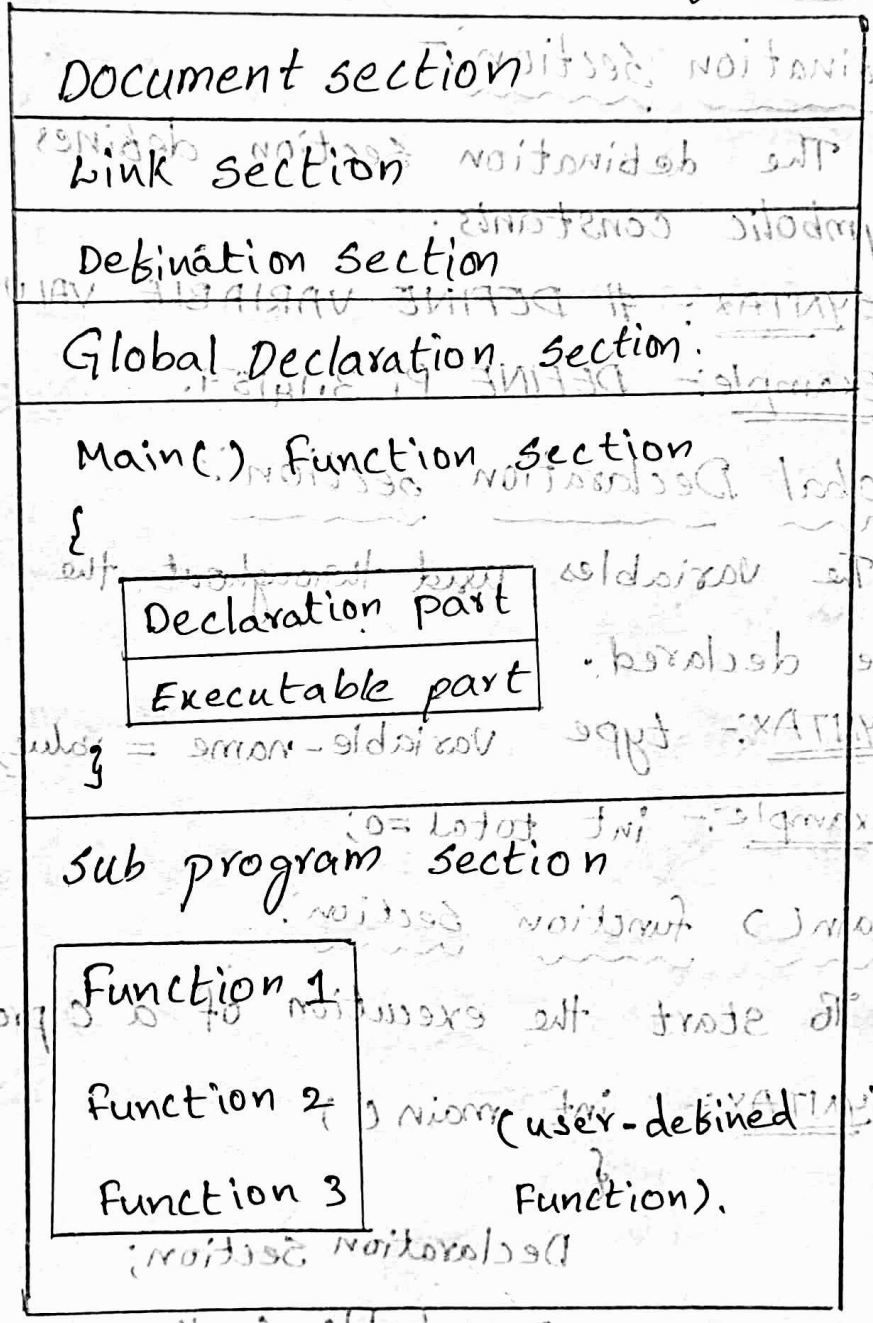
1. INTRODUCTION TO C LANGUAGE.

⇒ Basic Structure of C language:-



Document section:- Give the Brief explanation of the program.

SYNTAX:- /* To find n natural numbers */

Example:- /* comment lines */.

Link section:-

Includes header files with # include statements.

SYNTAX:- #include <st----.h>

Example:- #include <stdio.h>

Defination section:-

The defination section defines all symbolic constants.

SYNTAX:- # DEFINE VARIABLE VALUE

Example:- DEFINE Pi 3.14159.

Global Declaration section:-

The variables used throughout the program are declared.

SYNTAX:- type variable-name = value;

Example:- int total=0;

Main() function section:-

To start the execution of a C program.

SYNTAX:- int main()

{

Declaration section;

Executable section;

Return0;

}

Example:-

```
int main( )
{
    int x, y;
    Printf(" This is a C program\n");
    x=1;
    y=8;
    total = sum (x,y);
    printf(" Sum of two numbers : %d \n", total);
    return 0;
}
```

Sub program function:-

User can define his own function as per user requirement.

SYNTAX:-

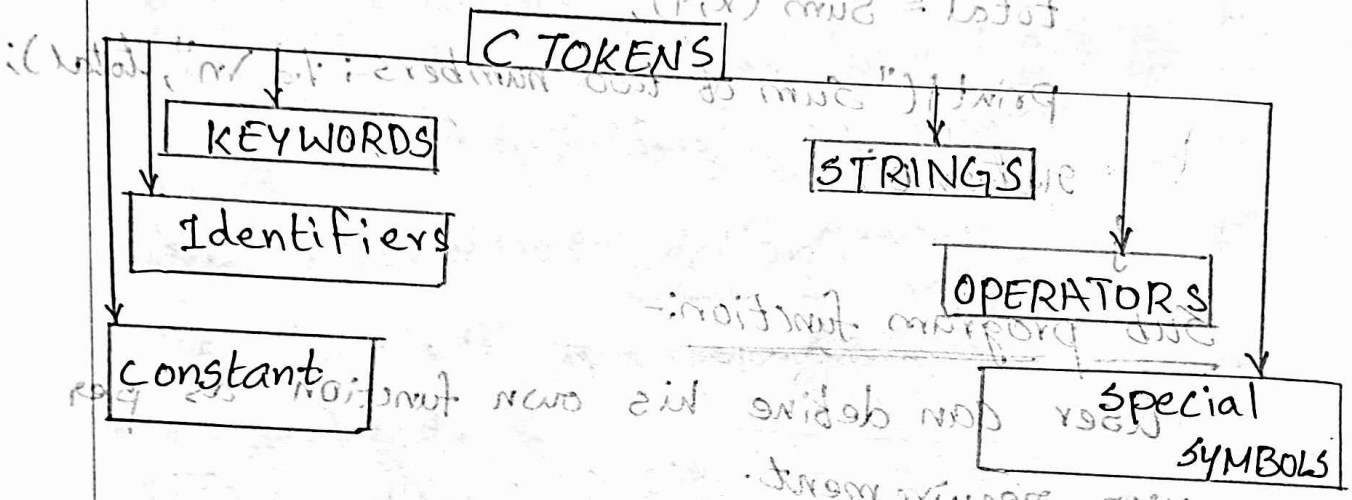
```
Return type function_name(type 1, type 2 ...)
{
Local Variable declaration section;
Defination section;
}
```

Example:-

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

2. CONSTANTS, VARIABLES AND DATA TYPES IN C

⇒ C TOKENS : Individual words and punctuation marks are called tokens. The smallest (individual) units are called c Tokens.



KEYWORDS :- The all keywords are have fixed meanings and these meanings can't be change.

Ex:- auto, do, double etc...,

IDENTIFIERS :- The names of functions, variables and user defined items are called as an Identifiers.

Ex:- variables, pi, radius, area etc.,

CONSTANT :- Constants refer to fixed values that do not change during the execution of a program. C supports three types of constants. These are

1. Numeric constants
2. character constants
3. String constants.

① Numeric constants :-

A numeric constant is made up of a sequence of numeric digits with optional presence of a decimal point. These are positive or negative numbers. There are two types :-

① integer constants
② Floating point constants.

Integer constants :- It is a whole Number without a decimal point.

Ex :- 23, 25, 40 etc.,

Floating constants :- It is a number with a decimal point. Ex :- 12.5, 2.35, 19.935 etc.

② Character constants :-

Any character enclosed in between single quotes.

Ex :- 'A', 'a', '.', $ etc.,

③ String constants :-

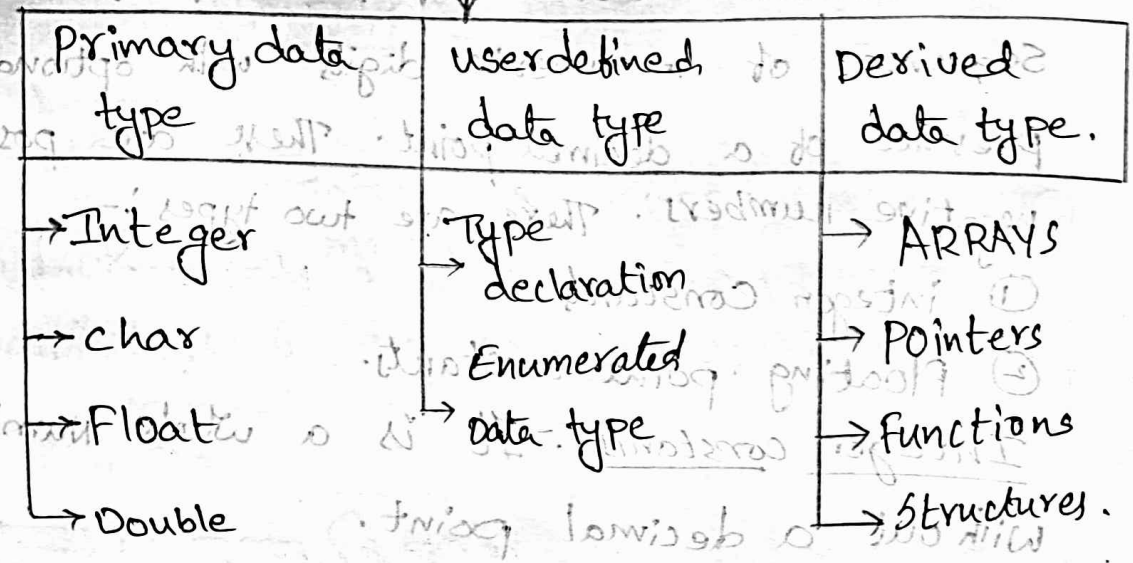
A group of characters enclosed with double Quotes. Ex :- "Diploma", "Computer", "PRAKASH" etc.,

⇒ DATA TYPES :-

The data types classified in to 3 types.

① primary Data types.
② User- Defined Data types.
③ Derived Data types.

C DATA TYPES



Integer :- To store integer values.

Int type:

SYNTAX :- int variable name;

Example :- int a, b, c, g = 10;

Character :- To store character values.

Char type:

SYNTAX :- char variable name;

Example :- char m, n, s[50];

Float :- To store real values.

Float data type :-

SYNTAX :- Float variable name;

Example :- float b, g, c;

Double data type :- To store large float values beyond float data type

SYNTAX :- double variable name;

Example :- double p, s;

DECLARATION OF A VARIABLE

A variable can be used to store a value of any data type. That is, the name has nothing to do with its type.

SYNTAX:-

Data type v1, v2, ..., vn;

Variables are seperated by commas. A declaration statement must end with a semicolon.

Example:-

```
int sum;
int number, marks;
double rate;
```

ASSIGINING VALUES TO VARIABLES:-

Values can be assigned to variables using the assignment operator = as follows:

SYNTAX:-

Data_type variable_name = constant;

Example:-

```
int a=10;
float x=75.64;
char yes='Y';
```

3. OPERATORS AND EXPRESSION

IN 'C'

⇒ **Operator:-**

The c operator is defined as a symbol that is used to perform operations on identifiers. Operators are grouped in to 8 Types. They are:-

1. Arithmetic operator
2. Relational operator
3. Logical operator
4. Assignment operator
5. Conditional operator
6. Increment and decrement operator
7. Bitwise operator
8. Special operator.

1. **Arithmetic operator:-** These are used to perform mathematical calculations like addition, substraction, multiplication, division and modulus.
   Ex:- +, -, *, /, %

2. **Relational operator:-** These operators are used to compare the value of two variables.
   Ex:- >, <, >=, <=, ==, !=

3. **Logical operator:-** These operators are used to perform logical operations. There are 3 logical operators in c language. They are
   (1) Logical AND &&
   (2) Logical OR ||
   (3) Logical NOT !

4. Assignment operator:- The values for the variables are assigned using assignment operator. Assignment operator is =.

SYNTAX:- Variable = expression;

Example:- x=4, y=5, z=15.

5. Conditional operator:-

conditional operators are also known as ternary operators. There are only two conditional operators. They are "? :". Condition operator returns one value if condition is true and returns value if condition is false.

SYNTAX:- (Condition ? true_expression : false_expression)

Example:- x = (A > 100 ? 0 : 1);

6. Increment & Decrement operators:-

Increment operators are used to increase the value of the variable by one and decrement operators are used to decrease the value of the variable by one in c programs.

symbols of increment & Decrement

++i ; i++

--i ; i--

7. Bitwise operators:-

These operators are used to perform bit operations. There are various bitwise operators in c language. They are

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
<<	Shift left
>>	Shift Right
~	ones complement.

8. Special operators :-

C supports some special operators such as comma operator, sizeof operator, pointer operators (& and *) and member section operators (. and ->).

PRECEDENCE AND ASSOCIATIVITY OF OPERATORS.

C has a precedence associated with it. This precedence is used to determine how an expression involving more than one operator is evaluated.

The operators of the same precedence are evaluated either from left to right or from right to left depending on the level. This is known as the associativity property of an operator.

4. MANAGING INPUT AND OUTPUT OPERATIONS

GETCHAR( ) of INPUT

This function is used to accept a single character from the keyboard and stores it in a variable of type char.

SYNTAX:- char - variable = get char();

Example- char name;

name = getchar();

SCANF( ) of INPUT

It is used to accept the data of all types from the keyboard. To make use of scanf() in any program one should include stdio.h file.

SYNTAX:-

scanf("format_string1, ... format_stringn", &variable1, ..., &variable n);

Example:- scanf("%d", &a, &b, &z);

PUTCHAR( ) of OUTPUT

This function is used to display a single character stored in a variable of type char on to the monitor. The variable should be declared as a char data type.

SYNTAX:- putchar (char-variable);

Example:- putchar(ch);

PRINTF() of OUTPUT:-

It is used to display the data of all types on the standard out put device. To make use of printf() in any c program one should include stdio.h file.

SYNTAX:-

printf("format_string1, ..., format_string n",
expr1, expr2 ... exprn);

Example:-

printf("e value will be neglected) but a = %d \n", e, a);

5. DECISION MAKING

→ Decision making in program :-

* C program is a set of statements which are normally executed sequentially in the order.
* We have a number of situations where we may have to change the order of execution of statements based on certain conditions.

→ Decision making statements :-

* Decision making statements are also called as control statements (or) selection statements
* Decision making statements in c language. they are :-
  * If Statement
  * Else if statement.
  * Else if ladder statement
  * Nested if else statement
  * switch statement

→ Simple If statement :-

* If is used to make decisions.
* The If statement takes logical condition and evaluates it.
* The Statement based on whether the condition evaluates it. it to true or false.

SYNTAX :-

```
If (expression)
{statement ;

}
next - statement.
```

Example:-

```
if (a>b)
{
 printf ("a >b\n");
}
printf ("Over\n");
```

Example Program:-

```
/* C Program to check the given number is even
   or odd number */
   #include <stdio.h>
   main( )
   {
      int a;
      printf (" Enter a value:\n");
      scanf (" %d", &a);
      if (a %2 == 0)
      printf("Number is even");
   }
```

Working:- The condition is checked first, If it is true then control executes true-block and then control goes to next-statement, if the condition is false then control directly goes to next-statement.

2 → IF ELSE statement :-

* If else statement is a bidirectional if statement, in this statements consider two condition cases true and false also.

SYNTAX:-

```
If (expression)
{
  Statement-1;
 else
  {
   Statement-2;
  }
}
next Statement;
```

Example:-

```
If (a>b)
{
printf ("a>b\n");
}
Else
{
printf ("b>a\n");
}
printf ("over\n");
```

Working:- First check the condition. If it is true then executes statement-1 and then go to next statement; If the condition is false then control executes statement-2 and control go to next statement.

Example:-

```
/* program to check for positive or negative */
#include <stdio.h>
#include <conio.h>
int main()
{
  int a;
  clrscr();
  printf("Enter a number");
  scanf ("%d", &a);
  if (a>0)
  {
  printf ("The number %d is positive", a);
```

```
    }
    else
    {
     printf ("The number %d is negative", a);
    }
    getch();
    return(0);
}
```

3 → Else if ladder :-

* Multi-way decisions arise when there are multiple conditions and different statements are to be executed under each condition.

SYNTAX :-

```
If (expression 1)
{
  Statement-1;
  Statement-2;
}
else if (expression 2)
{
  Statement-1;
  Statement-2;
}
else if
{
  Statement-1;
  Statement-2;
}
else
{
  Statement-1;
```

Working :-

* It is a multiple branching statement.
* In which condition is satisfied in that block enter and execute the block.

Example :-

```
If (wday ==1)
{
 printf ("Sunday\n");
}
else if (wday ==2)
{
 printf ("monday,\n");
}
else if (wday == 3)
 {
 printf ("Tuesday,\n");
 }
 else if (wday ==4)
 {
 printf ("Wednesday,\n");
 }else if (wday ==5)
 {
 printf ("Thursday,\n");
 }
 else if (wday ==6)
 {
 printf ("Friday,\n");
 }
 else if(wday ==7)
 {
 printf("saturday,\n");
 }
else
```

```
{
printf ("Invalid Day of the week,\n");
}
```

Program:-

```
/* Program to print the week day based on day
   number using if ladder */
#include <stdio.h>
#include <conio.h>
int main ()
{
 int wday;
 clrscr ();
 printf ("Enter the week day number ");
 scanf (" %d", &wday);
 If (wday ==1)
 {
 printf ("Sunday\n");
 }
 else if (wday ==2)
 {
 printf ("Monday\n");
 }
elseif (wday ==3)
{printf ("Tuesday\n");
 }
 else if (wday ==4)
 {
 printf ("Wednesday\n");
 }
 else if (wday ==5)
 {printf("Thursday\n");
```

```
}
else if (wday == 6)
{
printf ("Friday\n");
}
else if (wday == 7)
{
printf ("saturday\n");
}
else
{
printf ("Invalid day of the week \n");
}
getch();
return(0);
}
```

→ Nested If Else Statement :-

* If or If else statements placed within other if or if-else statements is called as nested if else statement.
* Nested If else statement is also called nested conditional statement.

SYNTAX :-

```
If (expression1)
{
if (expression 2)
{
statement-1;
}
else
{
statement-2;
}
}
else
{
if (expression(3))
{
statement-3;
}
else
{statement-4;
}
}
```

Example :-

```
If (a>=b)
{
 if(a>=c)
  big=a;
 else
  big=c;
}
else
{
 if(b>=c)
  big=b;
 else
  big=c;
}
```

Program :-

```
/* C program to find largest of 3 number using nested
 if statement */
#include<stdio.h>
int main()
float a, b, c, big;
clrscr();
printf("Enter three numbers");
scanf("%f%f%f", &a, &b, &c);
if (a>=b)
{
 if(a>=c)
  big = a;
 else
  big = c;
```

```
    }
    else
    {
      if (b>=c)
      big=b;
      else
      big=c;
    }
  printf("The biggest of %.2f, %.2f and %.2f is %.2f
        \n",a,b,c,big);
}
```

5 Switch condition :-

* It is used to select a set of statements based on the value of an expression instead of checking N conditions.
* It is also called as selection statement.

SYNTAX :-

```
switch (expression)
{
case -1;
      statement-1;
      break;
case -2;
      statement-2;
      break;
case -3;
      statement-3;
      break;
case-n;
      statement-n;
      break;
      default block;
      default statement;
      break;
}
next statement;
```

Program:-

```
/* Program to print the week day based on day
   number using switch-case statement */
#include <stdio.h>
#include <conio.h>
int main()
{
  int wday;
  clrscr();
  printf("Enter the weekday number");
  scanf("%d",&Wday);
  switch(Wday)
  {
  case-1;
        {printf("sunday\n");
        break;
  case-2;
        {printf("Monday\n");
        }break;
  case-3;
        {printf("Tuesday\n");
        }
        break;
  case-4;
        printf("Wednesday\n");
        break;
  case-5;
        printf("Thursday\n");
        break;
  case-6;
        printf("Friday\n");
        break;
```

```
    case-7;
            printf ("Saturday \n");
            break;
    default;
            printf ("Invalid day of the week \n");
  }
```

Example:-

```
switch(Wday)
{
  case-1;
        printf ("Sunday \n");
        break;
  case-2; printf (" Monday \n");
        break;
  case-3; printf ("Tuesday \n");
        break;
  case-4; printf ("Wednesday \n");
        break;
  case-5; printf ("Thursday \n");
        break;
  case-6;
        printf ("Friday \n");
        break;
  case-7;
        printf ("Saturday \n");
        break;
  default; printf ("Invalid day of the week \n");
}
```

6. Looping Concept:-

For loop:-

→ This is also called iterative loop.

→ It is preconditioned loop that checks for the condition first.

→ If it is satisfied then only the body of the loop gets executed. This process is continued as long as the condition remains true.

Syntax:-

```
for(initialization; test condition; increment/decrement)
{
.....
Body of for loop
.....
}
```

Example:-

```
for (i=1; i<=n; i++)
  fact=fact*i;
  Printf("%d\n", fact);
```

Program:-

```
main()
{
  int i,n, fact=1;
  printf("Enter n value");
  Scanf("%d", &n);
  for (i=1;i<=n;i++)
    fact = fact*i;
    Printf("%d\n", fact);
}
```

Working:-

1. Initialize
2. check for condition if true then ~~execute~~ execute body update and go to Step 2 else, exit the loop.

While ~~statement~~ loop:-

SYNTAX:-

```
While (condition)
{
  body of the loop;
}
```

Example:-

```
While (i<=n)
{
  sum = sum + i;
  i = i++;
}
```

Program:-

```
main()
{
  int i=1, n, sum=0;
  printf ("Enter n value \n");
  scanf ("%d", &n);
  while (i <= n)
  {
    sum = sum + i;
    i = i++;
  }
  printf ("%d", sum);
}
```

→ The simplest of all the looping structures in c is the while statement.

→ The while is an entry-controlled loop statement. The test condition is evaluated if the condition is true. Then the body of the loop is executed. After execution is once again. This process is continue until the test-condition finally becomes false.

do-while :-

→ The program proceeds to evaluate the body of the loop first. At the end of the loop, the test condition in the while statement is evaluated.

→ It is the exist-controlled loop statement.

SYNTAX :-

```
do
{
 Body of the loop;
}
while (condition);
```

Example :-

```
do
{
  sum = sum+i;
    i = i++;
}
while (i<=n);
```

Program :-

```
#include<stdio.h>
#include<conio.h>
main()
{
 int i=1,n,sum=0;
 printf("Enter n value\n");
 scanf("%d",&n);
 i=1;
 do
 {
  sum = sum+i;
   i=i++;
 };
 while (i<=n);
 printf("%d",sum);
}
```

17. ARRAYS

ARRAY DEFINATION:- An array is a collection of elements of same data data type under a single name.

SYNTAX:-

Data type array-name [array size]

ARRAY Initialization AND DECLARATION:-

Array can be declared and initialized in to two types:-

1. During its declaration with a single statement.
2. Declare it and initialize seperately.

SYNTAX:-

① DATA-TYPE ARRAY-NAME [SIZE] = { 1, 2, 3, 4, 5 }

② DATA-TYPE ARRAY-NAME [ARRAY-SIZE];

Array name [0] = value 1;

Array name [1] = value 2;

⋮

Array name [n-1] = value n;

Examples:-

① int age[5] = { 15, 20, 10, 25, 5 };

② int age [5];

age [0] = 15;

age [1] = 20;

age [2] = 10;

age [3] = 25;

age [4] = 5;

ONE DIMENSIONAL ARRAY.

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int i=0;
 int arr[6];
  arr[0] = 1;
  arr[1] = 2;
  arr[2] = 3;
  arr[3] = 4;
  arr[4] = 5;
  arr[5] = 6;
 for(i=0; i<6; i++)
 {
  printf("%d", arr[i]);
 }
 getch();
}
```

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int a[5] = {1,2,3,4,5};
 int i;
 clrscr();
 for(i=1; i<=5; i++)
 {
 printf("%d", a[i]);
 }
 printf("\n");
 getch();
}
```

TWO DIMENSIONAL ARRAY.

```
#include<stdio.h>
void main()
{
 int a[2][2] = {1,2,3,4};
 int i,j;
 clrscr();
for(i=0; i<=1; i++)
{
for(j=0; j<=1; j++)
{
printf("\n%d", a[i][j]);
}
getch();
}
```

TWO DIMENSIONAL ARRAY.

```
#include <stdio.h>
#include <conio.h>
int main()
{
 float a[2][2], b[2][2], result[2][2];
 printf("Enter element of 1st matrix\n");
 for(int i=0; i<2; ++i)
 for(int j=0; j<2; ++j)
 {
  printf("Enter a%d%d:", i+1, j+1);
   scanf("%f", &a[i][j]);
 }
 printf("Enter element of 2nd matrix\n");
 for(int i=0; i<2; ++i)
 for(int j=0; j<2; ++j)
 {
 printf("Enter b%d%d:", i+1, j+1);
  scanf("%f", &b[i][j]);
 }
 for(int i=0; i<2; ++i)
 for(int j=0; j<2; ++j)
 {
  Result[i][j] = a[i][j] + b[i][j];
 }
 printf("\n Sum of matrix :");
 for(int i=0; i<2; ++i)
 for(int j=0; j<2; ++j)
 {
  printf("%.1f\t", result[i][j]);
   if(j==1)
   printf("\n");
 }getch();
 }return(0);
```

MULTI DIMENSIONAL ARRAY

```
#include <stdio.h>
#include <conio.h>
int main()
{
 int test [2][3][2];
 clrscr();
 printf("Enter 12 values:\n");
 for (int i=0; i<2 ; ++i)
 {
  for (int j=0; j<3; ++j)
  {
   for (int k=0; k<2; ++k)
   {
 scanf("%d", &test [i] [j] [k]);
   }
  }
 }
 printf("\n displaying values: \n");
 for (int i=0 ; i<2; ++i)
 {
  for (int j=0; j<3; ++j)
  {
  for (int k=0; k<2; ++k)
  {
   printf("test [%d] [%d] [%d] = %d \n", i, j, k, test [i] [j] [k]);
  }
 }
}
 getch();
 return (0);
}
```

8 UNDERSTANING STRINGS

Defination:-

A group of characters that stored in a single varible that related to character datatype is knows as string.

SYNTAX:- char (string Name [size of string].

Initialization:- Here we can denote a group of characters with double quotation (or) single character will be enclosed with single quotation marks.

Ex:- char a[15] = "Jyothi prakash"

char a[15] = {'J','y','o','T','H','I',' ','P','R','A','K','A','S','H'}

READINGS OF STRINGS:-

(1) scanf (2) get char (3) gets.

(1) **scanf:-** It is used with %s format to read in a string of character. It is input function.

Ex:- scanf ("%s", name);

(2) **getchar( ):-** It is used for to read a single characters from the input and character array.

Ex:-
```
char ch;
ch = getchar( );
```

(3) gets() :- This file is available in <stdio.h> header file. It is also read the whole double quotation marks on a single line between

SYNTAX :- gets(string)

Ex:-
```
char a[20];
gets(a);
printf("%s",a);
```

OUTPUT FUNCTIONS IN STRINGS :-

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

(1) printf() :- It is used for to display the array of characters in the output.

Ex:- printf("%s",a);

(2) putchar() :- It is worked as oppositely to getchar(). This function requires parameter.

Ex:-
```
char ch=A;
putchar(ch);
```

(3) puts() :- It is worked as oppositely to gets().

Ex:-
```
char Name[20];
puts(name);
```

STRING HANDLING FUNCTIONS

String Handling functions are various types.
They are:-

	SYNTAX
① String copy	① Strcpy (S1, S2);
② String concatenate	② Strcat (S1, S2);
③ String n concatenate	③ Strncat (S1, S2, n);
④ String length	④ Strlen (S1);
⑤ String compare	⑤ Strcmp (S1, S2);
⑥ String n compare	⑥ Strncmp (S1, S2);
⑦ String compare ignore	⑦ Strcmpi (S1, S2);
⑧ String lower case	⑧ Strlwr (S1);
⑨ String upper case	⑨ Strupr (S1);
⑩ String reverse.	⑩ Strrev (S1);

EXAMPLES

① char s1[] = "paper";
char s2[] = "document";
strcpy(s1, s2); Make both s1 and s2 as "Document.

① String copy (Strcpy (s1, s2));
copies string 2 in to string 1.

② String concatenates .(strcat(s1, s2));

The strings2 is concatenates with strings1 on the end.

Examples

~~char[]~~

char s1[] = "R. Jyothi"

char s2[] = "PRAKASH"

strcat(s1, s2); The making of string s1 is "R. Jyothi PRAKASH". and string as "PRAKASH".

③ (Strncat) String n concatenates:-

(s1, s2, n).

concatenates n characters of string s2 at the end of the string s1.

Example :-

char s1[] = "Hello";

char s2[] = "friends"

strncat(s1, s2, 5); Makes the string s1 as "Hellofrien" and ~~s2~~ String s2 as "friends."

4. String length (strlen(A1));

The length of the string A1 is returns.

char A1[] = "POLYTECHNIC", ~~←~~ Example

Strlen(A1) ⁼ gives 11.

5. String compare (strcmp(s1, s2));

The strcmp function compares two strings and identify the differences between two strings and display the which not have in two strings that is the display on the strcmp (s1, s2);

Example:-

```
strcmp(s1,s2);
    char s1[] = "PRAKASH";
    char s2[] = "PR.AKASH";
```

strcmp(s1,s2); gives a Negative value, the difference between 'a' and 'p' is 15.

6. strncmp(s1,s2,n):— String n compares:- compares first n characters of string s1 and s2, Returns 0 if n characters of s1 and s2 are the same; less than 0 if n characters of s1 < n characters of s2; greater than 0 if n characters of s1 > n characters of s2.

Examples:-

```
char s1[] = "PRAKASH";
char s2[] = "PRAKASH";
```

Strncmp(s1,s2,7) gives a 0, since first 7 characters of both the strings are same.

7. String compare ignores (strcmpi(s1,s2));
Same as string compare (strcmp(s1,s2)); but ignores case.

Example:-

strcmpi("Hello","HELLO"); returns 0

8. String lower case (strlwr(s1));
It converts all the characters of the string1 into lower case letters.

Example

Printf("strlwr("HELLO!")

print # hello!

String uppercase :- (struprcs1);

It converts all the characters of the string 1 in to uppercase letters.

Examples

Printf("%s\n", strupr("hello!"));

print HELLO!

String Reverse (strrev(s1)) :-

Reverses the string s1 in the same location.

Example :-

puts (strrev("PRAKASH!"));

print"!HSAKARP"

PROGRAMS

```
#include <stdio.h>
#include <conio.h>
void main()
{
char ch1[5] = { "A','B','H','I','\0' }
printf("%s", ch1);
getch();
}
```

2.
```
#include <stdio.h>
#include <conio.h>
void main()
{
 char ch1[] = "ABHISHEK";
 Printf("%.", ch);
 getch();
}
```

3.
```
#include <stdio.h>
#include <conio.h>
void main()
{
 char name [] = "R.JYOTHI PRAKASH";
 printf("%.s \n", strlwr(name));
 printf("%.s\n", strupr(name));
 printf("%.s", strrev(name));
 printf("%.s", strlen(name));
 getch();
}
```

<u>Output:-</u>

r.jyothi prakash
R.JYOTHI PRAKASH
HSAKARP IHTOYJ.R.
length of string is 16

9. USER DEFINED FUNCTIONS

Elements of function.

1. Function Name
2. Function type
3. List of parameters.

} Function header

4. Local variable declarations
5. Function statements
6. Return statement.

} function body.

function header :- (Function name, type, list of parameters)

function body :- (local variables, statements and return statement)

Ex:-

```
int num(int a, int b, int c)
{
  int sum;
  sum = a+b+c;
  printf("%d", sum);
  return;
}
```

Syntax:-

```
function_type function_name (parameter list)
{
  local variable declaration;
  Statement 1;
  Statement 2;
  return statement;
}
```

9. USER DEFINED FUNCTIONS

Defination of function:

A function is defined that contains a group of statements to perform a specific task.

Parameters with return values.

It is a two way data communication between the calling function and the called function.

* Functions have parameters, the calling function can be send data to the called function.
* It can also return any value to the calling function with the use of return statement.

Example program:

```
#include <stdio.h>
#include <conio.h>
int sum(int,int);
main()
{
  int x,y,z;
  clrscr();
  printf("Enter the first number:");
  scanf("%d",&x);
  printf("Enter the second number:");
  scanf("%d",&y);
  z = sum(x,y);
  printf("Sum of two no.s is:%d",z);
}
int sum (int a, int b)
{
    int sum=0;
    sum =a+b;
    return (sum);
}
```

Functions With NO Argument and NO Return Values

Functions which have no arguments and NO return values.

Example

```
#include <stdio.h>
#include <conio.h>
void fun();
main()
{
 clrscr();
 fun();
}
void fun()
{
 int num;
 printf("Enter the number:");
 scanf("%d",&num);
 printf("the number you entered is %d",num);
}
```

NOP&NOR.c

Functions with arguments but no return values

Functions can have parameters, hence the calling functions can send data to the called function but it cannot return any value to the calling functions as it has no return statement.

Example:-

```
#include <stdio.h>
#include <conio.h>
int Sum (int,int);
main().
{
  int x,y;
  printf("Enter first number:");
  Scanf("%d",&x);
  Printf(" Enter Second Number:");
  Scanf("%d",&y);
  Sum(x, y);
}
int Sum (int a, int b)
{
  int sum=0;
  Sum=a+b;
  Printf(" Sum of two no.s is :%d", Sum);
}
```

Functions with NO Arguments with return values :-

* It take not any arguments but returns a value to the calling function.

Example-

```
    else
    {
      printf("%d is not a prime Number \n", a);
    }
    getch();
    return(0);
}
int prime()
{
  int flag, i;
  printf("Enter a Number:");
  scanf("%d", &a);
  flag=1;
  for (i=2; i<a/2; i++)
  {
    if (a%i==0)
    {
      flag=0;
      break;
    }
  }
  printf("This function is written without arguments and
          with return values\n");
  return(flag);
}
```

```
#include<stdio.h>
#include<conio.h>
void sum(int,int);
void main()
{
  sum(10,20);
  getch();
}
void sum(int i, int j)
{
  int s;
  s = i+j)
  printf("%d", s);
  return(sum);
}
```

Advantages of functions:

1. The program will be easier to understand.
2. By using functions to the maintainence of the program will be easy.
3. By using functions we can debug the errors easily.
4. The large program can be divided in to smaller modules.
5. Saving memory space.

Storage classes or Scope, visibility and lifetime of variables.

Scope & visibility:- The area of the program where the variable can be accessed or available to the user.

Like time:- The area of the program where the value of the variable is alive.

While writing functions based on the position of declaration of variables their scope, visibility and lifetime will be decided.

Each variable has a storage class which decides its scope, visibility and lifetime of that variable.

1. Automatic storage class variables
2. External storage class variables
3. Static storage class variables
4. Register storage class variables.

1. Automatic storage class variables:-

A variable declared inside of a function without any storage class specification, is by default an automatic variable. They are created when function is called and are destroyed automatically when the function exits. [auto]

```
void test();  ------> /* prototype */
void main()
{
  test();  ------> /* Function Call 1 */
  test();  ------> /* function call 2 */
  test();  ------> /* function call 3 */
}
void test()
{
auto int a=0;  ------> /* local variable */
a=a+1;
printf("%d\t",a);
}
```

External (or) Global variable.

The variable which we have declare before the ~~main~~ any function is called External variable. External variables used through out the program. The external variable can changed the value in any function. [Extern].

Example:-

(Normal)

```
#include<stdio.h>
int g=100;
void main()
{
G--;
printf("%d",g);
:
:
}
```

[External]

```
#include<stdio.h>
void main()
{
Extern int g;
:
:
}
```

Static variable:- The value of static variable persists until the end of the program. A variable can be declared static using of keyword [static].

Example

```
Static int x;
Static float y;

void test();
void main()
{
 test();
 test();
 test();
}
void test()
{
 Static auto int a = 0;
 a = a + 1;
 printf("%d\t", a);
}
```

Register variable:- Register variable informs the compiler to store the variable in register instead of memory. Register variable has faster access than normal variable. **Syntax:-** `Register int number;`

Define A function call:-

A function can be called by simply using the function name followed by a list of actual parameters enclosed in parentheses.

Example

```
Main
{
 int a;
 a = add(10,15); /*function call*/
 printf("%d\n", a);
}

int add(int x, int y)
{
 int z; /* local variable */
 z = x + y; /* x = 10; y = 15 */
 return(z);
}
```

Passing Arrays to Functions with program :-

A single array element or an entire array can be passed to a function. Also both one-dimensional and multi-dimensional array can be passed to function as argument.

Example :-

```
Void sort (int m, int x[ ]);
main()
{
 int i;
 int marks[6] = {87, 89, 83, 82, 95, 93};
 for(i=0; i<6; i++)
  printf("%d", marks[i]);
  printf("\n");
  sort(6, marks);
 for(i=0; i<6; i++)
  printf("%d", marks[i]);
}
 void sort (int m, int x[ ])
 {
  int i, j, temp;
  for(i=1; i<=m-1; i++)
  for(j=1; j<=m-i; j++)
   if(x[j-1] >= x[j])
   {
    temp = x[j-1];
    x[j-1] = x[j];
    x[j] = temp;
   }
 }
```

10. Basics of Pointers :-

Defination:- pointer variable are use to store the address of another same data type of variable. The pointer point the address of memory locations of given variable.

Declaration:-

Syntax :- Data type * variable name;

Ex :- int * a;

Initialization of pointer:-

Syntax :- Data type * variable name = &another variable of same data type;

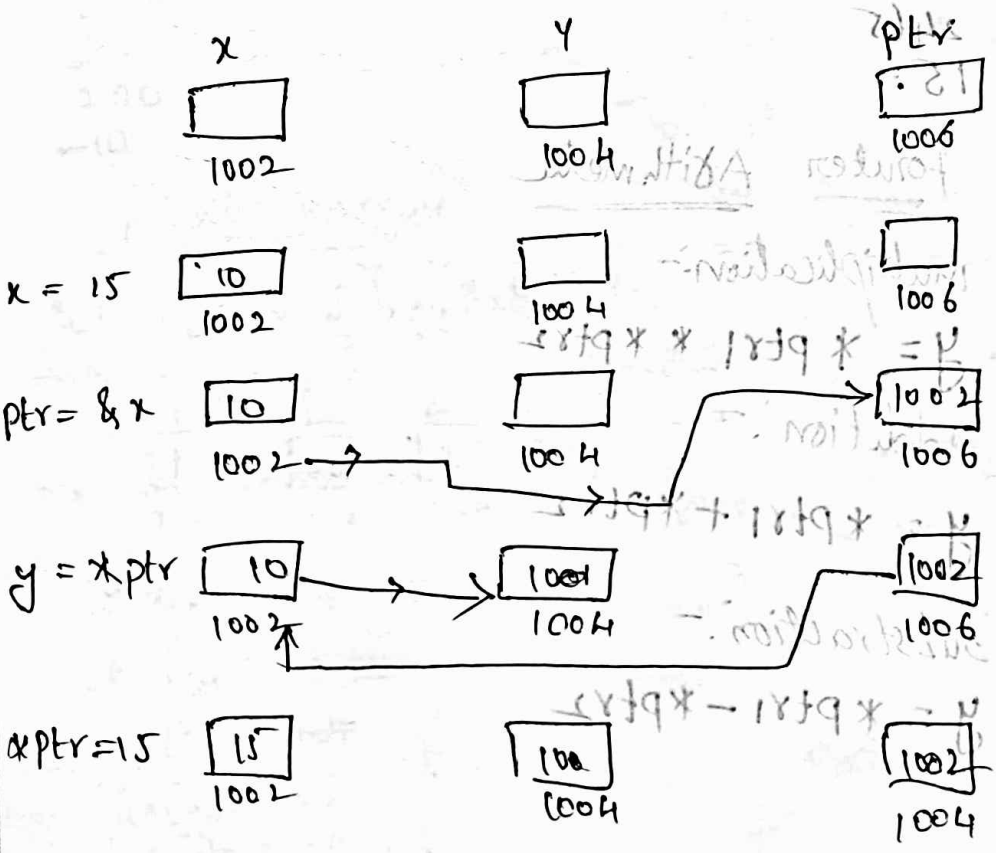
Ex:- int * p = &r;

(or)

```
int *p;
p = &r;
```

Example of pointer:-

Declaration of x, y, ptr

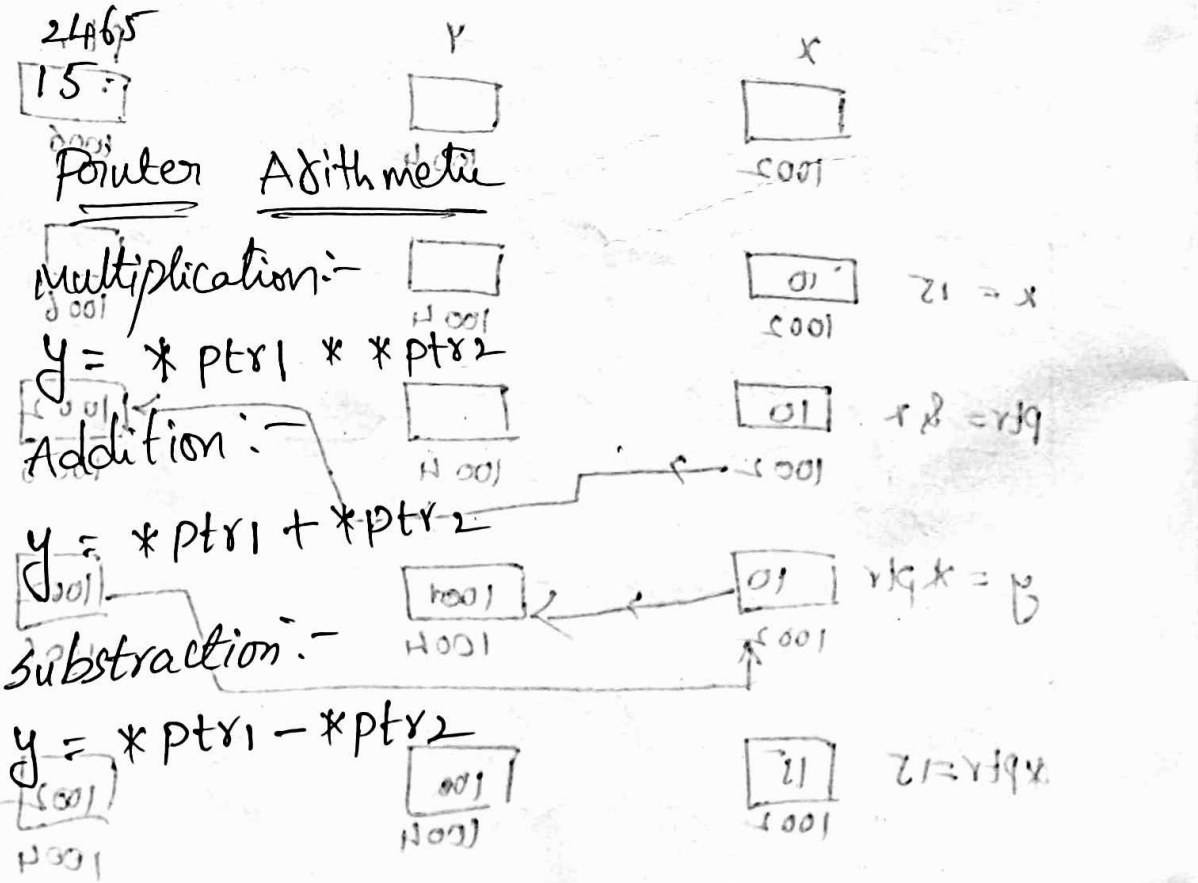


Program:-

```
#include <stdio.h>
#include <conio.h>
main()
{
  int *p, r;
  p=&r;
  r=15;
  clrscr();
  printf("%d is stored at address %u", r, &r);
  printf("%u", p);
  printf("%u", &*p);
  getch();
  return(0);
}
```

Output:-

15 is stored at address 2465



Pointer Arithmetic

Multiplication:-

y = *ptr1 * *ptr2

Addition:-

y = *ptr1 + *ptr2

Substraction:-

y = *ptr1 - *ptr2

Program:-

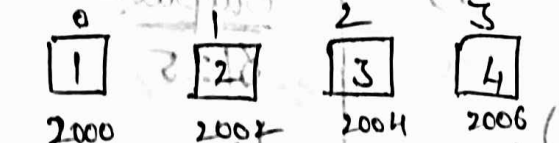
```
#include <stdio.h>
#include <conio.h>
main()
{
 int *ptr1 = &n;
 int *ptr2 = &m;
 int x, y, z;
 n=10;
 m=20;
 x = *ptr1 + *ptr2;
 y = *ptr1 ** ptr2;
 z = *ptr1 - *ptr2;
 printf("%d\n%d\n%d\n%d\n", x, y, z);
 getch();
 return 0;
}
```

Output:-

```
30
200
-10
```

Pointer in arrays

```
int x[5] = { 1, 2, 3, 4, 5};
```




```
&x[0] = 2000
&x[1] = 2002
&x[2] = 2004
&x[3] = 2006
&x[4] = 2008
```

Program:-

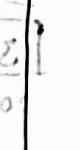
```
#include <stdio.h>
#include <conio.h>
main()
{
 int a[5];
 int i;
 clrscr();
 printf("Enter 5 numbers");
 for(i=0; i<=5 ; i++)
 {
   scanf("%d", &a[i]);
 }
 printf("The address of given numbers");
 for(i=0; i<=5 ;i++)
 {
  printf("%u\n", &a[i])
 }
 getch();
 return(0);
}
```

Output

```
Enter 5 numbers
2 4 6 5 3
The address of given numbers
1000 1002 1004 1006 1008
```

```
#include <stdio.h>
#include <conio.h>
main()
{
 int a[5]={1,2,3,4,5};
 clrscr();
 printf("%u", &a);
 printf("%u", *a);
 getch();
 return(0);
}
```

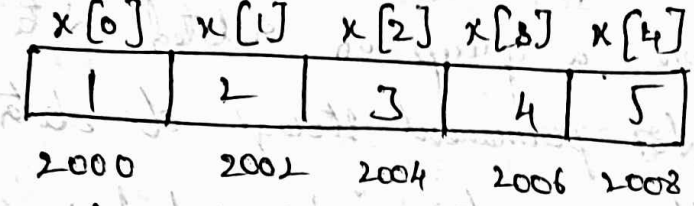
Output:-




```
61:25
1000
```

pointer array:-

```
int *a = &x[5];
x[5] = {1, 2, 3, 4, 5};
```



} Single dimensional

Multidimensional:-

```
int *a[4][3] = {{1,2,3} {4,5,6} {7,8,9} {10,11,12}};
```

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9
3	10	11	12

a[0][0] = 1
a[0][1] = 2
a[0][2] = 3
a[1][0] = 4
a[1][1] = 5
a[1][3]

11. FILE MANAGEMENT IN C

File:- A file represent a sequence of byte on the disk where a group of related data is stored. File is created for permanent storage of data. The contents of a file can be interpreted as characters, words, lines, paragraphs and pages from a textual document.

⇒ The basic file operations are:-

→ Naming a file.
→ Opening a file → closing a file.
→ Reading data from a file
→ Writing data from a file.

Declaration of file pointer to a file:-

→ The all files should be declared as data type "FILE" before they are used. The FILE MUST be in capital.

Syntax:-

```
FILE pointer variable;
variable = fopen("filename","mode");
```

Ex:-

```
FILE *P1
P1 = fopen("data","P");
```

⇒ Here p1 file will be opened as a filename "data" and it will read by the mode "P".

The modes:-

⇒ The mode is used to determine how the file may be accessed.

⇒ They are six modes to open a file.

Mode	Description :-
r	open text file for reading
w	create text file for writing.
a	open the new file for appending.
w+	create a file for write / read
r+	open a file for Read / write
a+	open or) create a file for append.

Opening a file :-

⇒ A file must be opened at the begining of program and after declaring the file pointer Variable.

Syntax :- pointer variable = fOpen ("File name", "mode");

Ex :-
```
FILE *P1;
P1 = fopen ("Data", "r");
```

Closing a file :-

⇒ A file must be closed when no more Input/output is to be performed on it.

Syntax :- fclose (file pointer variable);

Example :-
```
FILE *P1;
P1 = fopen ("Data", "r");
- - - -
= = = =
- - - -
fclose (P1);
```

⇒ Here P1 file will be closed by using a file Closing command.

Input / Output Operations on a file :-

⇒ When a file is opened it must be ~~read~~ reading the data and writing the data from files. So the Standard library provides many functions for I/O operations on a file.

They are :-

Function Name operations.

Character I/O Functions

getc() (or) fgetc() → Reads a character from a file
putc() (or) fputc() → writes a character to a file.

String I/O Functions

fgets() → Reads a string from a file.
fputs() → write a string to a file

word I/o Functions:

getw() → Reads an integer from a file
putw() → writes an integer to a file.

Formatted I/O functions :-

fscanf() → Reads a set of data values from a file
fprintf() → writes a set of data values to a file

Other Functions

fseek() → Moves the character pointer to a desired location in the file

ftell() → Returns the current position of the character pointer in the file.

rewind() → Moves the character pointer to the beginning of the file.

Random Access to files :-

⇒ When we are interested in accessing only a particular part in a file and not to read the other parts, that can be done by using of the functions. They are "Fseek", "Ftell", and "rewind".

⇒ These are Random Access Functions to files.

fseek():- It is used to move the file position to a desired location within the file.

Syntax:- fseek (fileptr, offset, position)

=> File ptr is a pointer to the file concerned.

=> offset is a number or variable of type long.

=> position is a integer number.

ftell():- It is used to takes a file pointer and return a number of type long, that corresponds to the current position.

Syntax:- file current position name = ftell (pointer variable);

Ex:- n = ftell (fp);

Rewind:- It's used to takes a file pointer and resets the position to the start of the file.

Syntax:- rewind (pointer variable);

Ex:- rewind (fp);

PRE - PROCESSOR DIRECTIVES

Need of preprocessor directives:-

1. programs easier to develop.
2. Easier to read.
3. Easier to modify.
4. Customize the c language.
5. Easier to transport to a different computer system.

Explanations

=> The processor is a program that processes the source program before it is extend on to ~~comp~~ Compiler.

=> It is related to file inclusion directive category.

=> They are two types of preprocessor directives.

(i) #include (ii) #define

(i) #include :-

⇒ The directives tells to preprocessor to read and include the program with the system library.

Ex:- #include <stdio.h>
math.h
conio.h
ctype.h etc.

⇒ The most commonly where the used the header file that is stdio.h. It is standard input/output function for all programs.

(ii) #define :-

⇒ The directives is useful to declare the constants names and values.

⇒ Those we use throughout program.

⇒ This ~~we~~ directive is related to macro substitution directive category.

Syntax :- #define Constant name constant value;

Ex:- #define PI 3.141592(97)