DIVYA CHRISTOPHER

Module V - Pointers & Files

A pointer is a variable that represents the location (rather than the value) of a data item, such as a variable or an array element.

Pointers provide a way to return multiple data items from a function via function arguments. Pointers also permit references to other functions to be specified as arguments to a given function.

Every stored data item occupies one or more memory cells. Number of memory cells required depends on the type of a data item.

Eg:- A single character will be stored in 1 byte (8 bits) of memory.

An integer requires 2 bytes,

floating point requires 4 bytes,

double precision requires 8 bytes.

Suppose v is a variable that stores a data item, compiler will automatically assign memory cells for this data item. Data item can then be accessed later, if we Know the address location of the first memory cell.

Syntax of pointer variable declaration

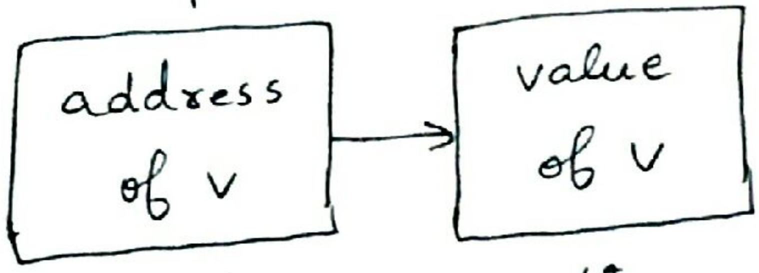
```
data-type *pointer-var-name;
```

An asterisk(*) symbol should precede every pointer variable name.
Address of v's memory location is represented as &v , where & is a unary operator called address operator.

```
int *pv;
```

Eg:- pv= &v;

Here, address of v is assigned to the pointer variable pv.



pv v

*pv and v represent the same data item.

Eg:- pv = &v ;//assign v's address to pv

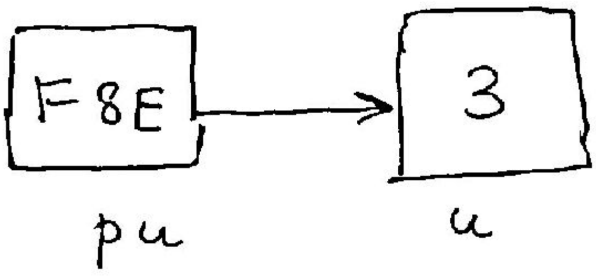
v= *pv ; //v &u holds same value.
u = *pv;

According to declaration above, pv is a pointer variables that stores address of v, not value of v.

The data item represented by v, can be accessed using indirection operator (*), applied on a pointer variable.

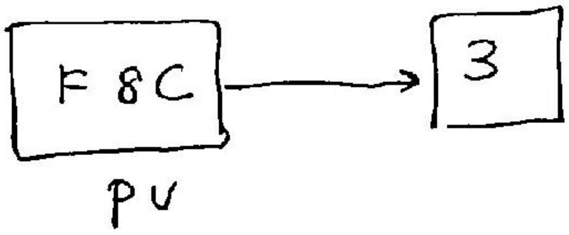
Eg:-
```
#include <stdio.h>
int main()
{
 int u=3;
 int v;
 int *pu;
 int *pv;
 pu= &u;
 v= *pu;
 pv= &v;
printf("u=%d &u=%X pu=%X *pu=%d",
        u, &u, pu, *pu);
printf("v=%d &v=%X pv=%X *pv=%d",
        v, &v, pv, *pv);
```
pu holding address of u is a pointer variable that points to address of u.

pv holding address of v is a pointer variable that points to address of v.



Address of u is F8E

pu= &u;



Address of v is F8 C

pv= &v;

Address operator is applied on variable representing data item.

The & operator cannot act upon arithmetic expressions like 2*(u+v).

The indirection operator (*) can only act upon pointer variables.

```
u = *pu;
v = *pv;
```

Eg:-
```
int main ()
{
int u1, u2, v = 3, *pv;
u1 = 2 * (v + 5);
pv = &v;
u2 = 2 * ( *pv + 5);
printf ("u1= %d , u2= %d", u1, u2);
}
```

O/P

```
u1 = 16 u2 = 16
```

Eg-
```
#include <stdio.h>
#define NULL 0
int main()
 {
 int  v = 3;

 int  *pv;

  pv = &v;

  printf ("\n*pv = %d  v = %d", *pv, v);
    int *pv = NULL;
  *pv = 0;  (or) //An indirect reference can
     //also appear on the L.H.S of
     // the assignment statement.

  printf ("\n *pv = %d  v = %d", *pv, v);

 }
```

O/P
```
*pv = 3   v = 3

*pv = 0   v = 0
```

*pv = 0;
Value at pv is assigned with a Null value (0).

Pointer variables can point to numeric variables, character variables, arrays, functions or other pointer variables.

- A pointer variable can be assigned with the address of an ordinary variable (pv=&v)
- A pointer variable can be assigned with the value of another pointer variable (pv=px).
- A pointer variable can be assigned with a null value. (pv=0)

Passing pointers to a function

Pointers are passed to a function as arguments. This allows data items within the calling portion of the program to be accessed by the function and then returned to the calling portion of the program after alteration.

When an argument is passed by value, data item is copied to the function.

When an argument is passed by reference, address of data item is passed to the function. Contents of this address can be accessed within the function or calling routine itself. Function Formal arguments are preceded by *. Actual Function arguments are preceded by &. operator.

Eg:-
```
#include <stdio.h>
      void func1(int u, int v);
            // function prototype 1
      void func2(int *pu, int *pv);
            //function prototype 2
int main()
{
 int u=1, V=3;
 printf("Before calling func1:\n");
  printf("u=%d \t v= %d \t \n", u, V);
   func1 (u, v);

 printf ("After calling func1 :\n");
  printf("u=%d\t v=%d \t\n", u, V);
printf( "Before calling func2:\n");
 printf("u= %d \t v= %d \t\n", u, v);
 func2 (&u, &v);
 printf("After calling func2:\n");
 printf ("u= %d \t %d \t\n",u,v);
  }
   void func1 (int u, int v)
    {
       u = 5 ;
       v = 7 ;
       printf ("within func1:");
       printf("u= %d \t v = %d \t\n",u,v);

    }
   void func2(int * pu, int *pv)
     {
       *pu = 9;
       *pv = 10;
   printf("within func2:");
   printf(" *pu = %d\t *pv= %d\t\n", *pu, *pv);
    }
```

Output

```
Before calling func1:
u = 1   v = 3
within func1:
u = 5   v = 7
After calling func1:
u = 1   v = 3
Before calling func2:
u = 1   v = 3
within func2:
u = 9   v = 10
After calling func2
9   10
```

The six printf statements illustrates the values of u, v, *pu and *pv within functions, main(), func1() and func2().

This program contains 2 functions, func1 and func2. func1 receives 2 integer variables as arguments. These variables are assigned with values 1 and 3 respectively.

Values are changed to 5, 7 within func1(). The new values are not recognized in main, because the arguments were passed by values, and any changes made to arguments are local to the function in which changes occur.

The second function func2() receives 2 pointers to integer variables as arguments. Arguments are identified as pointers by indirection operator. within func2(), contents of pointer addresses are re-assigned with the values 9, 10. Since the addresses are recognized within main and func2(), reassigned values are recognized inside main, after call to func2().

Pointers and 1-D array

Array name is a pointer to the first element in an array.

If x is a 1-D array,

&x[0] represents the address of the first array element.

&x[1] represents the address of the second array element.

&x[i] represents the address of ith array element of x

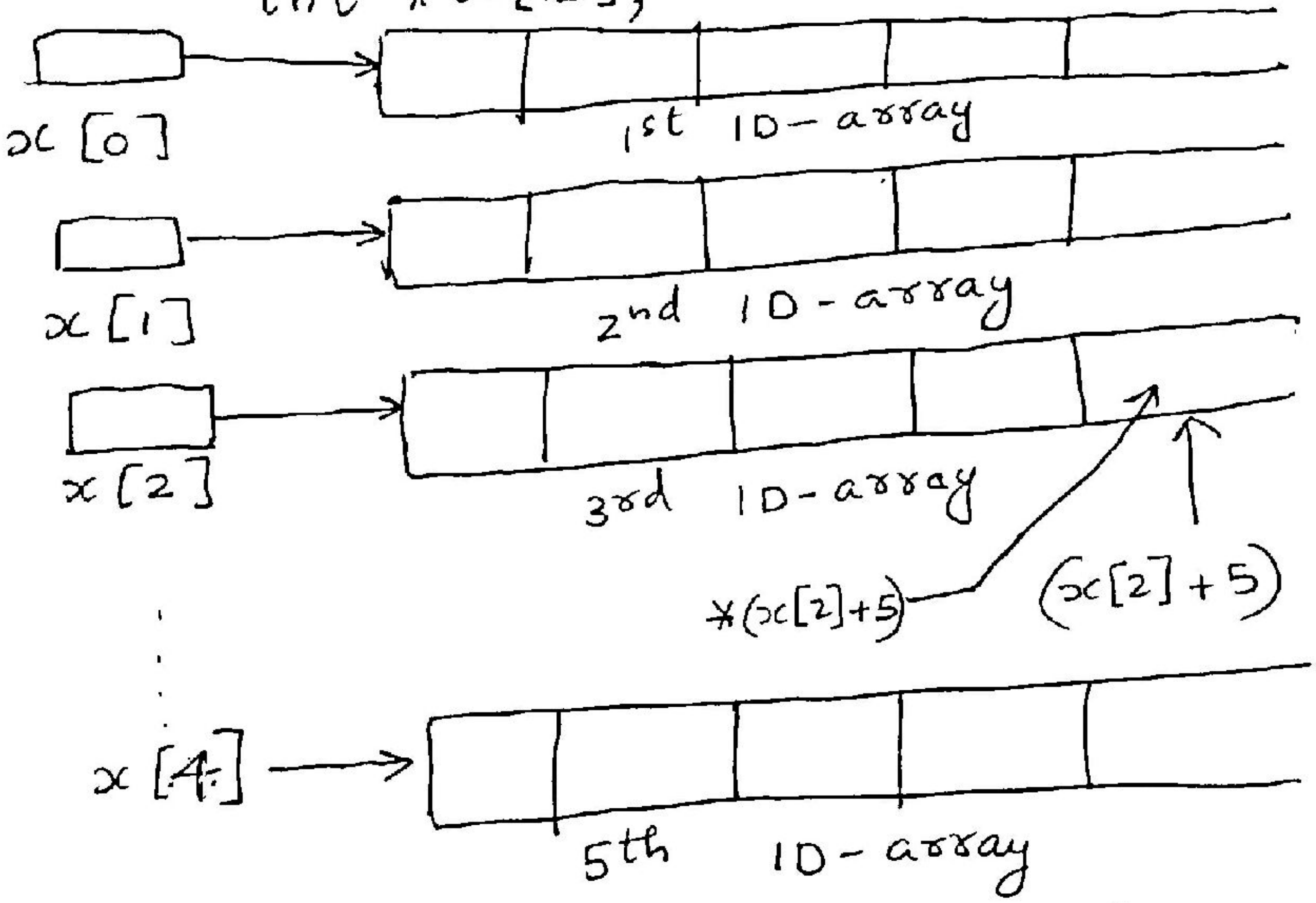
& x[1] is (x+1)

& x[2] is (x+2)

& x[i] is (x+i)

Syntax of Array of Pointers declaration - (1-D)

```
int *x[5];
```



x[0] points to the beginning of first row, x[1] points to the beginning of second row, and so on.

Eg:-
```
#include <stdio.h>
  int main()
  {
  static int x[6]={10,11,12,13,14,15};
  int i;
  for (i=0; i<=5 ; i++)
  {
 printf ("\n i=%d x[i]=%d \t");
 printf ("\n *(x+i)=%d \t\n", i, x[i],
                                  *(x+i) );
 printf(" &x[i] = %X \t (x+i) = %X \t",
                     &x[i], (x+i));
   }
 }
```

output

```
i=0  x[i]=10  *(x+i) = 10  &x[i] = 70  (x+i)=70
i=1  x[i]=11  *(x+i)=11   &x[i]= 72   (x+i)=72
i=2  x[i]=12  *(x+i)=12   &x[i]= 74   (x+i)=74
i=3  x[i]=13  *(x+i) =13  &x[i]= 76   (x+i)= 76
i=4  x[i]=14  *(x+i) =14  &x[i]= 78   (x+i)= 78
i=5  x[i]=15  *(x+i) =15  &x[i]= 7A   (x+i)=7
```

when assigning value to an array element x[i], left hand side of assignment statement may be written as either x[i] or *(x+i).

```
Eg :- #include<stdio.h>
      int main()
      {
      int line[80], int *p1;
        line[2] = line[1];
        line[2] = *(line+1);

         *(line+2) = line[i];

        *(line+2) = *(line+1);

         p1 = &line[1];   } The last 2
                          } assignment
      }  p1 = line + 1;   } statements
```

assigns the address of and array element to p1.

Each of the first 4 assignment statements assigns the value of second array element line[1] to 3rd array element line[2].

Eg:-

```
#include <stdio.h>
char *x = "LMCST";
int main()
{
char *y = "LOURDES MATHA COLLEGE";
printf("%s \n", x);

printf("%s\n", y);
}
```

Expansion of pointer declarations

int *p	p is a pointer to an integer
int *p[10]	p is a 10-element array of pointers to integer
int (*p) [10]	p is a pointer to a 10-element array

int *p(void);   p is a function that returns a pointer to an integer quantity.

int p(char *a); p is a function that accepts an argument that is a pointer to a character and returns a pointer to an integer quantity.

Operations on Pointers

If px is a pointer variable that holds the address of the variable x, then an integer value can be added or subtracted to and from a pointer variable to point to memory location of a variable, which is apart from the 1st element.

Eg - ++px, --px

(px+3)

(px+i)

(px-i)

Program

```
#include <stdio.h>
int main()
{
 int *px;
 int i=1;
 float f=0.3;
 double d=0.005;
 char c='*';
 px=&i;
printf("Values i=%d f=%f d=%f c=%c",
                                    i,f,d,c);
printf("Addresses &i=%X &f=%X
       &d=%X &c=%X", &i, &f, &d, &c);
```

```
printf ("Pointer values: px = %X  px+1 = %X
    px+2 = %X   px+3 = %X ", px, px+1,
                         px+2, px+3);
```

<u>O/p</u>

```
Values:  i = 1   f = 0.3   d = 0.005   c = *

Addresses:  &i = 117E  &f = 1180  &d = 1186
                                  &c = 118E

Pointer values  px 117E  px+1 = 1180  px+2 = 1182
                                  px+3 = 1184
```

First line displays the values of variables, second line displays addresses. Integer value requires 2 bytes (117E & 117F). Floating point value requires 6 bytes (1180 to 1185) though only 4 bytes are used for this purpose.

8 bytes are required for double. (addresses 1186 through 118D).

Finally, character represented by c begins with address 118E.

One pointer variable can be subtracted from another pointer variable, provided both the variables point to elements of same array. The resulting value indicates the number of words or bytes separating the corresponding array elements.

```
#include <stdio.h>
int main()
 {
 int *px,*py;
  static int a[6]= {1,2,3,4,5,6};

  px= &a[0];

  py = &a[5] ;
  printf ("px= %X py=%X",px, py);

  printf ("py - px = %X", py-px);
```

o/p

```
px = 52   py = 5C

py - px = 5
```

Pointer variables can be compared, provided both the variables are of the same datatype.

Comparesons can test for either equality or inequality.

Suppose px and py are pointer variables that points to elements of same array, then the following comparisons can be done:-

```
(px < py)
(px >= py)
(px == py)
(px != py)
(px == NULL)
```

```
if(px<py)
printf("px<py");
else
printf("px>=py");
```

Permissible operations of Pointers

(a) A pointer variable can be assigned the address of an ordinary variable. (pv = &v)

(b) A pointer variable can be assigned the value of another pointer variable. (pv = px)

(c) An integer quantity can be added or subtracted from a pointer variable (pv+3)

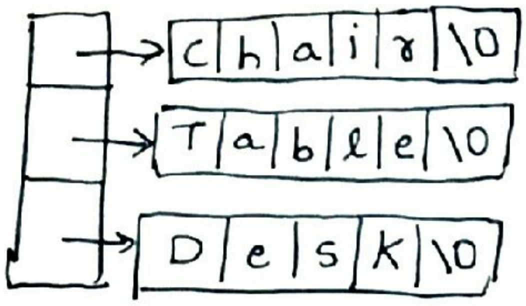
(d) Pointer variable can be assigned a NULL value. (px = NULL)

(e) A Pointer variable can be subtracted from another pointer variable, provided both pointers point to elements of the same array.

(f) Two pointer variables can be compared, provided both pointers points to object of same data type.

Eg:- Array of pointers to Strings

```
char * item[] = { "Chair",
                  "Table",
                  "Desk",
                };
```



Each element of item points to string.

Qn) Write a C++ program to find the largest element in an array using return pointer?

```
#include<stdio.h>
int * large (int *x);
int main ()
{
int a[5], *c, i=0;
printf("Enter 5 integers");
for (i=0; i<5; i++)
scanf("%d", &a[i]);
c= large (a);
printf(" Bigger value= %d", *c);
 }

int *large(int *x)
    {
     int larg =*x;
     for(int i= 1; i<5; i++)
      {
     if(x[i] >larg)
      larg =x[i];
      }
     return (&large);
    }
```

Qn) Write a C program to sum an array using pointers?

```
#include <stdio.h>
int main()
{
int *p;
int a[10], i, sum = 0;
printf("Enter the 4 array
                    elements");
for(i=0 ; i<4 ; i++)
  scanf ("%d", &a[i]);
  p=a;
for(i=0; i<4 ; i++)
  sum=sum +*(p+i);
printf("Sum of array using
        pointers = %d ", sum);
}
```

Qn) write a C program to find the smallest number using returning pointer?

```
#include <stdio.h>
int *small (int &x, int &y);
int main ()
  {
  int a,b,*c;
printf("Enter 2 integers");
 scanf("%d%d", &a, &b);
 c = small (a,b);
 printf ("Smaller value =%d", *c);
  }
  int * small (int &x, int &y)
   {
    if(x <y)
       return (&x);
    else
       return (&y);
   }
```

Qn) Write a C program to sort an array using Pointers?

```
#include <stdio.h>
int main()
{
 int i, j, a[100], n, temp;
 int *p;
 printf("Enter the limit");
 scanf("%d", &n);
 printf("Enter the array");
 for(i=0; i<n; i++)
   scanf("%d", &a[i]);
   p=a;
 for(i=0; i<n; i++)
 {
 for (j=i+1; j<n; j++)
 {
```

```
if (*(p+i) > *(p+j))
  {
   temp= *(p+i);
     *(p+i)= *(p+j);
     *(p+j)= temp;
    }
  }
 }
   printf ("Sorted array\n";
     for (i= 0; i<n; i++)
       printf ( "%d \n", a[i]);
       }
```

Qn) Write a C program to sort a character and floating point array by passing array of pointers to function?

```
#include <stdio.h>
void sort1(char *a, int n);
void sort2 (float *b, int n);
void main()
{
int n;
 char a[10], *p1;
 float b[10], *p2;
printf("Enter the limit");
 scanf("%d", &n);
   p1 =&a[0];
   p2 = &b[0];
printf("Enter character array");
for(i=0;i<n; i++)
 scanf("%c", &a[i]);
printf ("Enter floating-point array");
 for (i=0; i<n; i++)
  scanf ("%f", &b[i]);
   sort1 (p1, n);
   sort2 (p2, n);
 }
      void sort1 (char *a ,int n)
       {
       int i,j;
       char temp;
      for (i=0 ; i<n; i++)
        {
      for (j= i+1 ; j<n ;j++)
      {
      if (*(a+i) > *(a+j ))
          {
          temp= *(a+i);
        *(a+i) = *(a+j);
         *(a+j) = temp;
        }
      }
     }
       for (i=0; i<n;  i++)
        printf ("%c\t ", a[i]);
     }
```

```
void sort2(float *b, int n)
   {
    int i,j;
    float temp;
  for(i=0; i<n; i++)
   {
  for(j=i+1; j<n; j++)
    {
     if (*(b+i) > *(b+j))
       {
        temp = *(b+i);
        *(b+i)= *(b+j);
        *(b+j) = temp;
       }
    }
  }
  for(i=0; i<n; i++)
   {
   printf("%f\t", b[i]);
   }
}
```

FILES

Many applications require information be to be read from or written to an auxiliary memory device. Such information is stored on the memory device in the form of a *data file*. Thus, data files allow us to store the information permanently, and to access and alter that information whenever necessary.
In C, an extensive set of library functions is available for creating and processing data files. Unlike other Programming languages, C does not distinguish between sequential and direct access (random access) data Files. However, there are two different types of data files, called *stream-oriented* or *standard* data files, and *system-oriented* or *low-level* data files. Stream-oriented data files are easier to work with and are more commonly used. Stream-oriented data files are subdivided into text files and unformatted data files. *Text* files that are *Sequential files* consist of consecutive characters that are interpreted as individual data items using scanf and printf functions where as *unformatted* data files, organizes data into blocks which contains contiguous bytes of information involving complex data structures, like arrays and structures.

OPENING AND CLOSING A DATA FILE

When working with a stream-oriented data file, data is read and modified sequentially. The first step is to establish a buffer area, where information has to be temporarily stored while being transferred between the computer's memory and the data file. This buffer area allows information to be read from or written to the data file.
The buffer area is established by writing
FILE *ptvar;
where FILE (uppercase letters required) is a special structure type that establishes the buffer area, and ptvar is a pointer variable that indicates the beginning of the buffer area. The structure type FILE is defined within the system include file, stdio.h. The pointer ptvar is often referred to as a stream pointer.
A data file must be opened before it can be created or processed. This associates the file name with the buffer area (i.e., stream). It also specifies how the data file should be utilized, whether as a read-only file, a write-only file, or as a read-write file.

The library function fopen is used to open a file which is written as,
ptvar = fopen(file-name, file-type);
where file-name and file- type are strings that represent the name of the data file and file-type denotes the manner in which the data file will be utilized.
e.g fp=fopen("data.txt","r"); // open the file data.txt in read mode
The fopen function returns a pointer to the beginning of the buffer area associated with the file. A NULL value is returned if the file cannot be opened as, for example, when an existing data file cannot be found.
Finally, a data file must be closed at the end of the program. This can be accomplished with the library function fclose. The syntax is:

fclose (ptvar);

Table 12-1 File-Type Specifications

File-Type	***Meaning***
"r"	Open an existing file for reading only.
"w"	Open a new file for writing only. If a file with the specified ***file-name*** currently exists, it will be destroyed and a new file created in its place.
"a"	Open an existing file for appending (i.e., for adding new information at the end of the file). A new file will be created if the file with the specified ***file-name*** does not exist.
"r+"	Open an existing file for both reading and writing.
"w+"	Open a new file for both reading and writing. If a file with the specified *file-name* currently exists, it will be destroyed and a new file created in its place.
"a+"	Open an existing file for both reading and appending. A new file will be created if the file with the specified *file-name* does not exist.

```
/* read a line of lowercase text and store in uppercase within a data file */
#include <stdio.h>
#include <ctype.h>
int main( )
{
FILE * fpt ; / * define a pointer to predefined structure type FILE */
char c;
/* open a new data file for writing only*/
fpt = fopen("sample.dat", "w");

/* read each character and write its uppercase equivalent to the data file */
do
{
putc(toupper(c = getchar()), fpt ) ;
while (c != '\n') ;
/ * close the data file */
fclose (fpt) ;
}
```

After the program has been executed, the data file sample. dat will contain an uppercase equivalent of the line of text entered into the computer from the keyboard.

For example, if the original line of text had been
We, the people of our country India

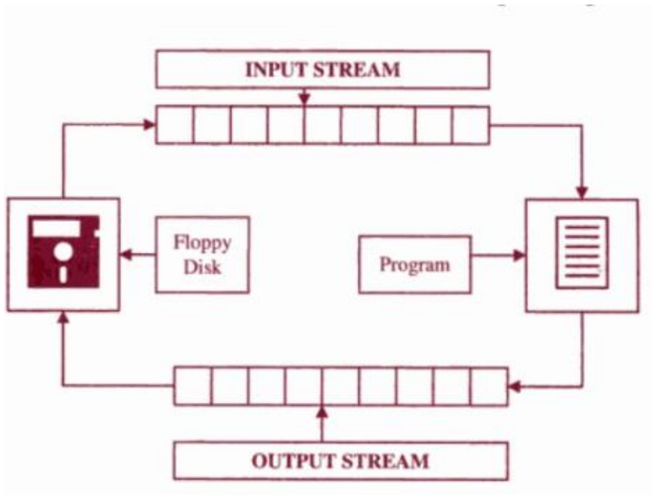
After processing, the data file would contain the text
WE, THE PEOPLE OF OUR COUNTRY INDIA

UNFORMATTED DATA FILES (Random access Files)
In Random access files, data is read and modified randomly To read and write blocks of contiguous data representing a structure or an array, we use the library functions fread() and fwrite().These functions are often referred to as unformatted read and write functions. Similarly, data files of this type are often referred to as unformatted data files.

Each of these functions requires four arguments: a pointer to the data block, the size of the data block, the number of data blocks being transferred, and the stream pointer.

Thus, a typical fwrite function is written as
`fwrite(&customer, sizeof(record), 1, fpt) ;`
where customer is a structure variable of type record, and fpt is the stream pointer associated with a data file that has been opened for output.

Stream means reading and writing of data. Streams allow users to access files efficiently. A stream is a file or physical device like keyboard, printer and monitor.
Figure below shows the input and output streams. The input stream brings data to the program and output stream collects data from the program. In this way, Input stream extracts data from the file and transfers it to the program while the output stream stores the data into the file provided by the program.



The FILE object uses all these devices. The FILE object contains all the information about stream like current position, pointer to any buffer, EOF (end of file).
Other file functions are listed below:

Function	**Operation**
fgetc() or getc()	Reads a character from current pointer position and advances the pointer to next character
fprintf()	Writes all types of data values to the file
fscanf()	Reads all types of data values from the file
fputc() or putc()	Writes character one by one to file
gets()	Reads string from file
puts()	Writes string to file
putw()	Writes an integer to the file
getw()	Reads an integer from the file
fread()	Reads structured data written by fwrite() function
fwrite()	Writes block of structured data to the file
fseek()	Sets the pointer position anywhere in the file
feof()	Detects the End of File
ftell()	Returns the current pointer position
rewind()	Sets the record pointer at the beginning of the file

Opening File in Append mode E.g.

```
#include<stdio.h>
#include<process.h>
{ FILE *fp;
char c;
printf("Contents of file before appending \n");
fp=fopen("data.txt","r");
while(!feof(fp))
{
c=fgetc(fp);
printf("%c",c);
}
fp=fopen("data.txt","a");
if(fp==NULL)
{
printf("File cannot be appended");
exit(1);
}
printf("Enter string to append");
while(c!='.')
{
c=getche();
fputc(c,fp);
}
fclose(fp);
printf("\n Contents of File after appending");
fp=fopen("data.txt","r");
while(!feof(fp))
{
c=fgetc(fp);
printf("%c",c);
}
}
```

```
Contents of file before appending LMCST.
Enter string to append LOURDES MATHA COLLEGE OF SCIENCE AND TECHNOLOGY
Contents of File after appending LMCST. LOURDES MATHA COLLEGE OF SCIENCE AND
TECHNOLOGY
```

Create a text file and perform the following: a) write data to the file b) read the data in a given file and display the file contents on console c) append new data and display on console.

Qn.) Write a C program to enter name and age to text file, use w+ file mode.

```
#include<stdio.h>
int main() {
FILE *fp;
char text[15];
int age,rollno;
float cgpa;
fp=fopen("Text.txt","w+");
printf("Enter Name and \t Age \n");
scanf("%s %d", text, &age);
fprintf(fp, "%s %d", text, age);
printf("Name \t Age \n");
fscanf(fp, "%s %d", text, &age);
printf("%s\t %d\n", text, age);
fp=fopen("Text.txt","a");
printf("Enter Roll no and CGPA \n");
scanf("%d %f", &rollno, &cgpa);
fprintf(fp, "%d %f", rollno, cgpa);
printf("Name \t Age \t Rollno \t CGPA\t\n ");
fscanf(fp, "%s %d %d %f", text, &age, &rollno , &cgpa);
printf("%s\t %d\t %d \t %f \n", text, age, rollno, cgpa);
fclose(fp);
}
```

Output

```
Enter Name and Age
Rini   18
```

```
Name Age
Rini   18

Enter Roll no and CGPA
33     85

Name Age Rollno CGPA
Rini   18     33        85
```

READ A LINE OF TEXT FROM A FILE

```
#include <stdio.h>
#include <stdlib.h> // For exit() function
int main() {
    char c[1000];
    FILE *fptr;
    if ((fptr = fopen("program.txt", "r")) == NULL) {
        printf("Error! opening file");
        exit(1);
    }

    // reads text until newline is encountered
    fscanf(fptr, "%[^\n]", c);
    printf("Data from the file:\n%s", c);
    fclose(fptr);

    return 0;
}
```

```
Data from the file:
LOURDES MATHA COLLEGE OF SCIENCE AND TECHNOLOGY
```

Write a C program to count the no of lines, words, characters, spaces in a text file?

```
#include <stdio.h>
void main()
{
    int line_cnt = 0, sp_cnt=0,ch_cnt=0,wrd_cnt=1;
    FILE *fp;
    char ch;
    fp = fopen("txt1.txt", "r");
    if(fp==NULL)
            printf("\nError: file not found");
    printf("\nFile Contents are:");
    printf("\n-------------------\n");
    ch = fgetc(fp);
    while (ch != EOF)
    {
            printf("%c",ch);
            if (ch  ==  '\n')
           line_cnt++;
            if (ch  ==  ' ')
           sp_cnt++;
            if (ch == ' ' || ch == '\t' || ch == '\n' || ch == '\0')
                wrd_cnt++;
            //read next character
            ch = fgetc(fp);
            ch_cnt++;
    }
    printf("\n----------------------------------------------\n");
    printf("Total number of lines                 : %d\n", line_cnt+1);
    printf("Total number of words                           : %d\n", wrd_cnt);
    printf("Total number of characters :   \t%d\n", ch_cnt - sp_cnt);
    printf("----------------------------------------------\n");
}
```

```
 C programming
 Java programming
 C++
 C#
 VB.NET
 ASP.NET
 Total number of lines               : 6
 Total number of words               : 8
 Total number of characters          : 50
```

```
/* C Program to Append the Content of File at the end of Another */
#include <stdio.h>
#include <stdlib.h>
int main()
{
    FILE *fsring1, *fsring2, *fmerge;
    char ch, file1[20], file2[20], file3[20];
    printf("Enter name of first file ");
    gets(file1);
    printf("Enter name of second file ");
    gets(file2);
    printf("Enter name to store merged file ");
    gets(file3);
    fsring1 = fopen(file1, "r");
    fsring2 = fopen(file2, "r");
    fmerge = fopen(file3, "w");
    while ((ch = fgetc(fsring1)) != EOF)
        fputc(ch, fmerge);
    while ((ch = fgetc(fsring2) ) != EOF)
        fputc(ch, fmerge);
    printf("Two files merged  %s successfully.\n", file3);
    fclose(fsring1);
    fclose(fsring2);
    fclose(fmerge);
    return 0;
```

```
 }
```

Output

```
gcc pgm.c
$ ./a.out
Enter name of first file a.txt
Enter name of second file b.txt
Enter name to store merged file merge.txt
Two files merged merge.txt successfully.
```

fwrite() function

The fwrite() function is used to write records (sequence of bytes) to the file. A record may be an array or a structure. The Syntax of fwrite() function is as shown below:

```
fwrite( ptr, int size, int n, FILE *fp );
```

The fwrite() function takes four arguments.
ptr : ptr is the reference of an array or a structure stored in memory.
size : size is the total number of bytes to be written.
n : n is number of times a record will be written.
FILE* : FILE* is a file where the records will be written in binary mode.
The fread() function is used to read bytes form the file.

Syntax of fread() function

```
fread( ptr, int size, int n, FILE *fp );
```

The fread() function takes four arguments.
ptr : ptr is the reference of an array or a structure where data will be stored after reading.
size : size is the total number of bytes to be read from file.
n : n is number of times a record will be read.
FILE* : FILE* is a file where the records will be read.

```
// C program for writing a block of data involving structures to file
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct person
{
   int id;
   char fname[20];
   char lname[20];
};

int main ()
{
   FILE *outfile;

   // open file for writing
   outfile = fopen ("person.dat", "w");
   if (outfile == NULL)
   {
      fprintf(stderr, "\nError opening file\n");
```

```
        exit(1);
    }

    struct person input1 = {1, "royce", "anto"};
    struct person input2 = {2, "rinil", "jose"};
      //input1 and inut2 are structure variables
    // write struct to file
    fwrite (&input1, sizeof(struct person), 1, outfile);
    fwrite (&input2, sizeof(struct person), 1, outfile);

    if(fwrite != 0)
        printf("contents to file written successfully !\n");
    else
        printf("error writing file !\n");

    // close file
    fclose (outfile);
  }
// C program for reading a block of data involving structure from a file
#include <stdio.h>
#include <stdlib.h>

// struct person with 3 fields
struct person
{
    int id;
    char fname[20];
    char lname[20];
};
int main ()
{
    FILE *infile;
    struct person input;
    // Open person.dat for reading
    infile = fopen ("person.dat", "r");
    if (infile == NULL)
    {
        fprintf(stderr, "\nError opening file\n");
        exit (1);
    }
    // read file contents till end of file
        while(fread(&input, sizeof(struct person), 1, infile))
        printf ("id = %d name = %s %s\n", input.id, input.fname, input.lname);

    // close file
    fclose (infile);
}
gcc demoread.c
./a.out
id = 1   name = royce anto
id = 2   name = rinil jose
```

Write a program to write and read the information about player containing players name, age and runs. Use fread( ) and fwrite( ) functions?

```
#include<stdio.h>
#include<process.h>
struct record
{
int age;
int runs;
};
void main()
{
FILE *fp;
struct record emp;
fp=fopen(“record.dat”,”w”);
if(fp==NULL)
{
printf(“Cannot open file”);
exit(1);
}
printf(“Enter player name, age and runs scored”);
printf(“==============================”);
scanf(“%s %d %d”,emp.player, &emp.age,&emp.runs);
fwrite(&emp,sizeof(emp),1,fp);
fclose(fp);
if(fp=fopen(“record.dat”,”r”))==NULL)
{
printf(“error in opening file”);
exit(1);
}
printf(\n Record entered is\n”);
fread(&emp,sizeof(emp),1,fp);
printf(“%s %d %d”,emp.player, emp.age, emp.runs);
fclose(fp);
}
```

OUTPUT

```
Enter player name, age and runs scored
============================
Arun 25 10000

Record entered is
Arun 25 10000
```

fseek() function

It is a file function used to position file pointer on the stream. Three arguments are passed through this function:

- ➔ File pointer
- ➔ Negative or Positive number used to re-position file pointer towards backward or forward direction.
- ➔ The current position of the file pointer

Integer Value	Constant	Location in the file
0	SEEK_SET	Beginning of the file
1	SEEK_CUR	Current position of the file pointer
2	SEEK_END	End of the file

Example –

fseek(fp,10,0); // Reads file through forward direction from beginning of the file
fseek(fp,10,SEEK_CUR); // Reads file through forward direction from current position of file pointer
fseek(fp,-n, SEEK_END); // Reads file through backward direction from end of file

ftellp() function

It is a file function which returns the current position of the file pointer. It returns the pointer from the beginning of the file.

```
// C Program to demonstrate the use of fseek()
#include <stdio.h>
int main()
{
    FILE *fp;
    fp = fopen("test.txt", "r");

    // Moves file pointer to end of the file
    fseek(fp, 0, SEEK_END);

    // Printing current position of file pointer
    printf("%ld", ftell(fp));
  }
```

Output:

```
81
```

The file test.txt contains the following text:

"Someone over there is calling you. We are going for work. Take care of yourself."

```
#include <stdio.h>

int main () {
   FILE *fp;

   fp = fopen("file.txt","w+");
   fputs("This is Cpp Program", fp);

   fseek( fp, 7, SEEK_SET );
   fputs(" C Programming Language", fp);
   fclose(fp);

   return(0);
}
```

Output:

```
This is C Programming Language
```

feof() function

The function feof() is used for detecting the file pointer whether it is at the end of file or not. It returns non-zero if the file pointer is at the end of file, otherwise it returns zero.
Syntax -
int feof(FILE *stream)

```
#include <stdio.h>
int main () {
   FILE *fp;
   int c;
   fp = fopen("file.txt","r");
   if(fp == NULL) {
      perror("Error in opening file");
      return(-1);        }
   while(1) {
     c = fgetc(fp);
     if( feof(fp) ) {
         break ;
                         }
      printf("%c", c);
                  }
   fclose(fp);
}
```

Output

```
LMCST
```

C feof function returns true in case end of file is reached, otherwise it returns false.

```
#include<stdio.h>
int main()
{
FILE *f1 = NULL;
char buf[50];
f1 = fopen("infor.txt","r");
if(f1)
    {
         while(!feof(f1))
         {
             fgets(buf, sizeof(buf), f1);
             puts(buf);
         }
         fclose(f1);
   }
}
```

Output

```
Lourdes Matha College of Science and Technology
```

ferror() function

ferror() is a function used to find out error that has occurred when file read or write operation is carried out.

```
#include <stdio.h>
int main () {
  FILE *fp;
  char c;

  fp = fopen("file.txt", "w");

  c = fgetc(fp);
  if( ferror(fp) ) {
    printf("Error in reading from file : file.txt\n");
  }
  fclose(fp);
}
```

rewind() function

This function resets the file pointer to the beginning of the file stream. It also clears the error and end-of-file indicators for stream.

The syntax for the rewind function is:

```
void rewind(FILE *stream);
```

stream - The stream whose file position indicator is to be set to the beginning of the file.

```
#include<stdio.h>
#include<conio.h>
void main(){
FILE *fp;
char c;
clrscr();
fp=fopen("file.txt","r");

while((c=fgetc(fp))!=EOF){
printf("%c",c);
}

rewind(fp);//moves the file pointer at beginning of the file

while((c=fgetc(fp))!=EOF){
printf("%c",c);
}

fclose(fp);
getch();
}
```

Output:

```
this is a simple text this is a simple text
```

C PROGRAM TO copy the contents of one file to another

```
#include <stdio.h>
#include <stdlib.h> // For exit()

int main()
{
   FILE *fptr1, *fptr2;
   char filename[100], c;

   printf("Enter the filename to open for reading \n");
   scanf("%s", filename);

   // Open one file for reading
   fptr1 = fopen(filename, "r");
   if (fptr1 == NULL)
   {
      printf("Cannot open file %s \n", filename);
      exit(0);
   }

   printf("Enter the filename to open for writing \n");
   scanf("%s", filename);

   // Open another file for writing
   fptr2 = fopen(filename, "w");
   if (fptr2 == NULL)
   {
      printf("Cannot open file %s \n", filename);
      exit(0);
   }
     // Read contents from file
   c = fgetc(fptr1);
   while (c != EOF)
   {
      fputc(c, fptr2);
      printf("%c",c);
      c = fgetc(fptr1);
   }
     printf("\nContents copied to %s", filename);

   fclose(fptr1);
   fclose(fptr2);
   return 0;
}
```

Output:

```
Enter the filename to open for reading
a.txt
Enter the filename to open for writing
b.txt
HELLO WORLD
Contents copied to b.txt
```