UNIT- 5

FILES

INTRODUCTION:-

REASONS FOR APPROACHING FILES:-

We have been using the functions such as scanf() and printf() to read and write data. These are console-oriented I/O functions which always use the terminal (keyboard & screen) as the target place.

This works fine as long as the data is small. However many real-life problems involve large volumes of data. In such situations, the console I/O functions pose two major problems:

1. It becomes time consuming to handle large volumes of data through terminals.

2. The entire data is lost when either the program is terminated or the computer is turned-off.

It is therefore necessary to have a more flexible approach where data can be stored permanently on the disks and read whenever necessary, without destroying the data.

Files allows us to store information permanently in the disk.

FILE:- A file is a place on the disk, where a group of related data is stored.

(OR)

A file is a collection of records that can be accessed through a set of library functions.

BASIC FILE OPERATIONS:-

⇒ C-supports a number of functions to perform the basic file operations.

1. Creating a file
2. Opening a file.
3. Reading data from a file
4. Writing data to a file
5. Closing a file.

⇒ There are two ways to perform file operations in C.

1. Low-level I/O : Uses UNIX System calls
2. High-level I/O : Uses functions in C's standard I/O Library.

High-Level I/O Functions:-

Function name	Operation
fopen()	- Creates a new file for use.
	- Opens an existing file for use.
fclose()	— Closes a file which has been opened for use.
getc()	- Reads a character from a file
putc()	- Writes a character to a file
getw()	- Reads an integer from a file
putw()	- Writes an integer to a file
fprintf()	- Writes a set of data values to a file
fscanf()	- Reads a set of data values from a file
fseek()	— Sets the position to a desired position in the file.
ftell()	— Gives the current position in the file.
rewind()	- Sets the position to the beginning of the file.

2. TYPES OF FILES:-

⇒ Files are used for storage and retrieval of data by a C program.

⇒ Depending on the format in which data is stored, files are divided into two types: (i) Text files
(ii) Binary files

(i) Text files:-

→ Text file stores textual information like alphabets, digits and symbols etc.

→ Text files can be read and understood by human beings.

→ Since data is stored in memory in the binary format, the text file is first converted into the binary form before it is stored in memory.

→ A text file can store different character set such as: A to Z
a to z
1, 2, ---9
Special symbols etc.

Ex: C source code files and files with .txt extensions.

(ii) Binary Files:-

→ Any file which stores the data in the form of bytes is known as binary file.

→ A binary file stores the information in binary format i.e. 0's and 1's.

→ The use of binary files eliminates the need of data conversion from text to binary format

→ The main drawback of binary file is that the data stored in the binary file is not in a human readable form.

Ex: Every executable file generated by a C-compiler is a binary file.
All files with .exe extensions
Image files etc

DEFINING and OPENING A FILE:-

⇨ A file has to be opened before beginning of read and write operations.

⇨ If we want to store data in a file, we need to specify certain things about the file.

1. Filename: A string of characters that make up a valid file name.

   Ex: input, name.txt, file.doc, add.c etc.

2. Data structure - FILE

   All files should be declared as type FILE before they are used. FILE is a defined data type.

3. Purpose: Purpose of opening a file.

Syntax:
```
FILE *fp;
fp= fopen("filename", "mode");
```

where, FILE → structure defined in I/o Library
fp → is a pointer to the data type FILE
filename → valid identifier
mode → Specifies the purpose of opening the file.

⇨ fopen function opens the specified file in specified mode.

⇨ Mode can be any one of the following:

$$\begin{tikzcd}[column sep=small]
 & & & & & \text{mode} \arrow[dll] \arrow[drr] & & & & & & \\
 & & & \text{Text ~~files~~ modes} \arrow[dlll] \arrow[dll] \arrow[dl] \arrow[d] \arrow[dr] \arrow[drr] & & & & \text{Binary modes} \arrow[dl] \arrow[d] \arrow[dr] \arrow[drr] \arrow[drrr] \arrow[drrrr] & & & & \\
w & r & a & w+ & r+ & a+ & wb & rb & ab & w+b & r+b & a+b
\end{tikzcd}$$

TEXT MODES:-

1. w (Write): Write only

→ When the mode is "w", a file with the specified file name is created if the file doesn't exist.

→ If the file already exist, it will delete the contents of the file and file is opened as a new file.

Ex: FILE *fp;
→ mode
fp= fopen("data.txt", "w");
↓ File name

2. r(read) : Opens the file for reading only

⇒ When the mode is "r", fopen function opens a pre-existing file for reading. If the file doesn't exist, then the compiler returns NULL to the file pointer.

Ex:
```
fp=fopen("data.txt", "r");
 if(fp==NULL)
  printf("File doesn't exist");
```

3. a (append): opens a file for appending data

⇒ When the mode is "a", fopen function opens a pre-existing file in append mode without erasing its contents and append data at the end of the file.

⇒ If the file doesn't exist the mode of "a" is same as "w".

Ex: `fp = fopen("data.txt", "a");`

4. w+ (write+read):-

The file is opened for both reading and writing.

Ex: `fp=fopen("data.txt", "w+");`

5. r+ (read+write):-

This mode opens the file for both reading and writing.

Ex: `fp=fopen("data.txt", "r+");`

6. a+ (append+read):-

When the mode is "a+", the file can be read and data can be appended at the end of the file.

Ex: `fp = fopen("data.txt", "a+");`

CLOSING A FILE:-

→ A file must be closed as soon as all operations on it have been completed.

→ Closing a file prevents any accidental misuse of the file. A file must be closed when we want to reopen the same file in different mode.

→ The function fclose is used to close a file.

Syntax: `fclose(file-pointer);`

Ex: FILE *fp1, *fp2;

fp1 = fopen("input.c", "w");

fp2 = fopen("output.c", "r");

- - - - - - - - - - - - - -

- - - - - - - - - - - - - -

fclose(fp1); → close the file associated with file pointer fp1.

fclose(fp2); → close the file associated with file pointer fp2.

⇒ Once a file is closed, its file pointer can be reused for another file.

⇒ To close more than one file at a time fcloseall() function is used.

Syntax: $\boxed{\text{fcloseall();}}$

INPUT/OUTPUT OPERATIONS ON FILES:-

Once a file is opened, reading (or) writing operations are accomplished using the standard I/O functions.

1. getc() and putc() functions:-

getc():- getc() function is used to read a character from a file that has been opened in read mode.

Syntax: ch = getc(fp); → File pointer

↓ Character variable

→ getc() function reads a character from the file whose file pointer is fp and moves the file pointer to the next location immediately.

→ getc() function returns EOF, if the end of file is reached. Therefore reading should be terminated when EOF is encountered.

Ex:
```
fp = fopen("sample.txt", "r");
char ch;
while ((ch=getc(fp)) != EOF) /* Reads characters from the file
                                  until EOF is reached */
printf("%c", ch);
```

2. putc() function:-

putc() function is used to write a single character to a file.

Syntax: $\boxed{\text{putc(c, fp);}}$

where c - is a character variable

fp - is a file pointer

→ putc() function writes the character contained in the character variable 'c' to the file associated with FILE pointer fp.

→ File pointer moves by one character position for every operation of getc or putc.

NOTE:- The end-of-the file is indicated by entering an EOF character, which is Control-Z in a new line.

Example program:- Write a program to read data from the keyboard, write it to a file. Again read same data from the file, and display it on the screen.

```
#include<stdio.h>
void main()
{
    FILE *fp;
    char c;
    clrscr();
    fp=fopen("sample.txt", "w"); /* opening file in write mode */
    while((c=getchar())!=EOF)   /* Get a character from keyboard */
    {
        putc(c,fp);  /* Write a character to sample.txt file */
    }
    fclose(fp);
    fp=fopen("sample.txt", "r"); /* opening file in read mode */
    while((c=getc(fp))!=EOF)  /* Read a character from the file */
    {
        printf("%c", c);   /* Display a character on screen */
    }
    fclose(fp);  /* closing the file sample.txt */
   getch();
}
```

putw and getw Functions:- The getw and putw are integer-oriented functions.

1. **putw:-** putw function is used to write an integer value to a file which is pointed by file pointer. fp.

Syntax: $\boxed{\text{putw(x, fp);}}$

where x is a integer variable.

2. **getw:-** getw() function is used to read an integer value from a file.

Syntax:

```
int x;
x=getw(fp);
```

getw() function reads an integer value from a file whose file pointer is fp

⇒ getw() and putw() functions are useful when we deal with integer data.

Example program:-

⇒ Write a program to write integers to a file and again read the integers from the same file and display it on the screen.

```
#include<stdio.h>
void main()
{
   FILE *fp;
   int x;
   clrscr();
   fp=fopen("numbers.txt", "w");
   printf(" enter integer numbers:");
   while(1)
   {
     scanf("%d",&x);
     if(x==0)  /* if the entered number is '0', it stops */
     break;
     else
     putw(x,fp); /* writing integer number to the file number.txt */
   }
   fclose(fp);
   fp=fopen("number.txt", "r");
   while((x=getw(fp))!=EOF)  /* Reading integer values from the file */
    printf(" %d", x);
   getch();
}
```

3. fprintf and fscanf functions:-

fprintf and fscanf functions are used to handle group of mixed data items simultaneously.

1. fprintf():-

fprintf() function is used for writing characters, strings, integers, floating point values etc to a file.

Syntax: $\boxed{\texttt{fprintf (fp, "control strings", variable list);}}$

Where fp → fp is a file pointer associated with a file that has been opened for writing.

Control string → Specifies the format specifications for the variables in the variable list.

Variable-list → The variable list may include variables, constants and strings.

Example:- `fprintf( fp, "%s %d %f", name, age, 7.5);`

The above function writes a string 'name', an integer age and a floating point number 7.5 to the file associated with file pointer fp.

2. fscanf():-

fscanf() function is used to read mixed data items from a file.

Syntax: $\boxed{\texttt{fscanf(fp, "control-strings", list);}}$

The above statement reads the items in the list from the file specified by fp, according to the specifications contained in the control string.

Example: `fscanf( fp, "%s %d", &name, &quantity);`

⇒ fscanf function returns number of items successfully read. When end of file is reached, it returns EOF

Example program:-

⇒ Write a program to write data into the text file and the same using fprintf and fscanf functions.

```
#include<stdio.h>
void main()
{
    FILE *fp;
    char name[20];
    int age;
    Float height;
    fp=fopen("data.txt", "w");
    printf(" Enter name, age and height :");
    Scanf("%s %d %f", &name, &age, &height);
    fprintf(fp, "%s %d %f", name, age, height); /* Writing mixed data items
                                                   to a file */

    Fclose(fp);
    fp=fopen("data.txt", "r");
    fscanf (fp, " %s %d %f", &name, &age, &height); /* Reading data items
                                                       from a file */
    printf( " Name is: %s", name);
    printf("\n Age: %d", age);
    printf("\n Height : %f", height);
    fclose(fp);
    getch();
}
```

Input:- Enter name, age and height : puja 25 5.5

Output:
```
Name is: puja
 Age: 25
 Height: 5.5
```

ERROR HANDLING DURING I/O OPERATIONS:-

⇒ It is possible that an error may occur during I/O operations on a file.

⇒ Typical error situations include:

1. Trying to read beyond the end-of-the file mark.
2. Device overflow.
3. Trying to use a file that has not been opened.
4. Trying to perform an operation on a file, when the file is opened for another type of operation.
5. Opening a file with an invalid filename.

⇒ If we fail to check these errors, a program may behave abnormally when an error occurs. An unchecked error may result in termination of the program or incorrect output.

Therefore it is necessary to check these errors during I/O operations on a file.

C-language supports two library functions to detect I/O errors in the files.

1. feof()
2. ferror()

1. feof():-

The feof function is used to test for an end of file condition.

Syntax: $\boxed{\text{feof(fp);}}$

→ feof function takes FILE pointer fp as its only argument

→ feof() function returns non-zero integer value if the file pointer is at the end of the file. Otherwise it returns zero.

Ex:
```
if( feof(fp) != 0)
    printf("End of data (or) file");
```

would display the message "End of data" on reaching the end of file condition.

2. ferror():-

→ This function is used for detecting any error that might occur during read/write operations on a file.

→ It takes FILE pointer as its argument.

⇒ Returns non-zero integer if an error has been detected.
Returns zero otherwise.

Syntax: $\boxed{\text{ferror(fp)}}$

Ex:
```
if(ferror(fp)!=0)
    printf("An error has occured");
```

⇒ When a file is opened using fopen(), the file pointer is returned. If the file can't be opened for some reason, then the function returns a null pointer. This can be used to test whether the file is opened or not.

Ex:
```
FILE *fp;
fp=fopen("sample.txt", "r");
if(fp==NULL)
printf("File can't be opened");
```

Example program:-

⇒ Write a program to detect errors that has occured during I/O operations.

```
#include<stdio.h>
void main()
{
    FILE *fp;
    char c;
    fp=fopen("sample.txt", "w");
    if(fp==NULL)
    {
      printf("Can't open a file");
      exit(0);
    }
    while(!feof(fp))  /* checking for end of file condition */
    {
        c=getc(fp);
        if(ferror(fp)!=0)  /* checking error using ferror() */
        {
          printf("\n An error has occured");
          exit(0);
        }
        printf("%c", c);
    }
    fclose(fp);
}
```

RANDOM ACCESS TO FILES:-

⇒ Using Random access functions we can access only a particular part of a file.

⇒ There are 3 random access functions

1. fseek()
2. ftell()
3. rewind()

1. fseek():-

The fseek function is used to move file pointer to a desired location within the file.

Syntax: $\boxed{\text{fseek(Filepointer, offset, position);}}$

where,

→ file pointer points to the specified file

→ offset is a number or variable of type long.

offset specifies the number of positions to be moved from the location specified by position.

offset may be positive means move in forward direction (or) it may be negative → move in backward direction.

Position:- position indicates the current position of the file pointer. position may take one of the following 3 values.

Value	Meaning
0	- Beginning of the file
1	- Current position
2	- End of the file

Examples:-

1. fseek(fp,0,0) - Go to the beginning
2. fseek(fp,0,1) - Stay at the current position
3. fseek(fp,0,2) - Go to the end of the file.
4. fseek(fp,m,0) - fp is moved to (m+1)th byte in the file.
5. fseek(fp,m,1) - moves fp in forward direction by m bytes from the current position.
6. fseek(fp,-m,1) - Go backward by m bytes from the current position
7. fseek(fp,-m,2) - Go backward by m bytes from the end.

⇨ fseek function returns zero if the operation is successful.
returns -1 if there is any error in moving file pointer.

<u>Example program</u>:-

⇨ Write a program to read last n characters in a file using fseek function.

```
#include<stdio.h>
void main()
{
    FILE *fp;
    char ch;
    int n;
    clrscr();
    printf(" Enter number of characters to be read:");
    scanf("%d",&n);
    fp=fopen("sample.txt", "r");
    if(fp==NULL)
    {
        printf(" File cann't be opened");
        exit(0);
    }
     fseek(fp,-n,2);
     while((ch=getc(fp))!=EOF)
     {
        printf("%c", ch);
     }
     fclose(fp);
    getch();
}
```

<u>Example program</u>:-

⇨ Write a program to read the text after skipping n characters from the beginning of the file.

HINT: fseek(fp, n, 0)

3. ftell():-

⇒ ftell() function returns the current position of the file pointer in a file.

Syntax:
```
int n;
n=ftell(fp);
```

⇒ ~~This function is useful in saving the current position of a file, which~~ can be used later in the program.

Ex
```
fseek(fp,2,0)
n=ftell(fp);
⇒n=2
```

Example program:-

⇒ Write a program to print the current position of a file pointer using ftell() function

```
#include<stdio.h>
#include<conio.h>
void main()
{
    FILE *fp;
    Clrscr();
    fp=fopen("sample.txt","w");
    if(fp==NULL)
    {
        printf("File cann't be opened");
        exit(0);
    }
    printf("At the beginning file pointer position is: %d",ftell(fp));
    fseek(fp,5,0);
    printf("\n After moving file pointer position is: %d", ftell(fp));
    fclose(fp);
    getch();
}
```

Output:
```
At the beginning file pointer position is : 0
After moving file pointer position is : 5
```

3. rewind:-

The rewind() function resets the position of file pointer to the beginning of the file.

It takes file pointer as its argument.

Syntax: $\boxed{\text{rewind(fp);}}$

Ex:
```
fp=fopen("sample.txt", "r");
rewind (fp);
n=ftell (fp);
```
The value of n=0 because the file pointer position has been set to to the beginning of the file by rewind.

Ex2:
```
fp=fopen("sample", "w");
fseek (fp,5,0);
n=ftell(fp);  ⇒ Now n=5 because fseek() moves fp by 5 bytes
rewind (fp);
n=ftell(fp)   ⇒ Now n=0 because rewind function resets the fp
                 position to the start of the file.
```

⇒ ~~Write a program to copy~~

APPLICATIONS OF COMMAND LINE ARGUMENTS:-

⇒ The key application of command line arguments is run-time specification of data i.e. the programmer can specify the required data during runtime.

⇒ For example consider a situation, where you are required to copy the contents of onefile into another.

As long as the source and target files remains the same, we can specify the names of the files statically within the program code.

But, if we want the program ~~code~~ to copy the contents of any file to any other file, then we must use the concept of command line arguments.

The command line arguments allow the user to specify the filenames dynamically at run time.

⇒ We can use the concept of command line arguments whenever data is required to be specified dynamically.

Example:-

⇒ Write a program to copy the contents of one file into another file using the command line arguments.

```
/* program for copying contents of one file into another */
#include<stdio.h>
#include <conio.h>
void main ( int argc, char *argv[])
{
    FILE *fp1, *fp2;
    char ch;
    clrscr();
    if(argc != 3)
    {
        printf("Incorrect arguments");
        exit(0);
    }
    fp1 = fopen(argv[1], "r");   /* opening source file in read mode */
    fp2 = fopen(argv[2], "w");   /* opening target file in write mode */
```

⇒ Write a program to read the existing filename and display its contents on the monitor.

```
#include<stdio.h>
void main(int argc, char *argv[])
{
        FILE *fp;
        char c;
        clrscr();
        if(argc!=2)
        {
          printf("Incorrect arguments");
          exit(0);
        }
       fp=fopen(argv[1], "r");
       if(fp==NULL)
       {
         printf(" File can't be opened");
         exit(0);
       }
       while((c=getc(fp))!=EOF)
       {
          printf("%c", c);
       }
       fclose(fp);
      getch();
}
```

Explanation:-

In the above program existing filename is passed from the command line argument. The file is opened in read mode.

getc() function reads a character from the file. The printf() functi displays it on the screen.

To Run the above program:-

1. Save the program as display.c and make it executable file.
2. Switch to command prompt and type the following:

```
C:\> display file1.c
```

Example program:-

⇒ Write a program to display the number of command line arguments and their names.

```
#include<stdio.h>
void main(int argc, char *argv[3])
{
    int i;
    clrscr();
    printf(" Number of arguments : %d", argc);
    printf(" Names of arguments:");
    for(i=0; i<argc; i++)
    printf(" %s", argv[i]);
   getch();
}
```

⇒ To run the above program

1. Save and compile the program. Make it as executable file.
2. Switch to command prompt and type the following:

```
C:\> sample   Hello World
```

Output:

```
Number of arguments:3
Names of arguments : sample
                     Hello
                     World.
```

Command-Line arguments:-

The parameters supplied to a program when the program is executed is known as "command-line arguments".

⇒ We know that every C program should have a main function.

→ Like other functions main function can also take arguments.

⇒ The main() can take 2 arguments: 1. argc
2. argv

Syntax:
```
main(int argc, char *argv[])
{
    -------
    ----
}
```

1. argc: argument counter

The argc counts the number of arguments passed to the program from the command line.

2. argv: argument vector

It is a pointer to an array of strings which contains the names of arguments.

→ The size of argv is equal to the ~~siz~~ value of argc.

⇒ Information contained in the commandline is passed onto the program through these arguments.

⇒ To give command line arguments, the user has to switch to command promp.

Ex: `C:\> program File1 Hai`

→ The first argument is always an executable program followed by arguments.

For the above example argc = 3
argv[0] → program
argv[1] → File1
argv[2] → Hai

```
while((ch=getc(fp1))!=EOF) /* Reads character from source file */
{
    putc(ch,fp2);        /* Writes characters to the target file */
}

fcloseall();
fp2=fopen(argv[2],"r");
printf("After copying the content of target file is:");
while((ch=getc(fp2))!=EOF)
{
   printf("%c",ch);
}
fclose(fp2);
getch();
}
```

⇒Now make the above program as executable file and switch to command prompt and type the following:

```
c:\> copy source.txt target.txt
```

Explanation:-

In the above program source file name and target file name is suppliec as command line arguments.

The source file is opened in read mode and target file is opened in write mode.

The program reads the characters from the source file using getc() function and copies the contents to target file using putc() function.

Output:-

```
After copying the content of target file is: This is an application of command line arguments.
```

NOTE: Source.txt contains: This is an application of command line arguments.

⇒ Write a program to write integers to a text file & read it.

```
#include<stdio.h>
#include<conio.h>
void main()
{
   FILE *fp;
   int n;
   clrscr();
   fp=fopen("input.txt","w"); /* opens the file input.c in write mode */
   while(1)
   {
     scanf("%d",&n);
     if(n==0)
     break;
     else
     putw(n,fp); /* writes an integer n to the file pointed by fp */
  }
  fclose(fp);
  fp=fopen("input.txt","r"); /* opens the file in read mode */
  while((n=getw(fp))!=EOF) /* reading integers from the file */
  {
     printf("%d",n);
  }
  fclose(fp);
 getch();
}
```

Explanation:

In the above program putw() function is used to write integer to a file and getw function is used to read an integer from a file.

⇨ Write a c program to reverse the contents of a file and copies it into a new file.

```
/* Program to reverse the contents of a file and copies it into a new file *
#include<stdio.h>
#include<conio.h>
void main()
{
   FILE *fs,*ft;
   char str[100];
   int i=0;
   clrscr();
    fs=fopen("source.txt","r"); /*opening source file in read mode */
    if( fs==NULL)
    {
       printf("Source file cann't be opened");
       exit(0);
    }
    ft=fopen("target.txt","w"); /* opens the target file in write mode*/
    if(ft==NULL)
    {
       printf("Target file cann't be opened");
       exit(0);
    }
    while((ch=getc(fs))!=EOF) /* Reads the contents of source file
    {                               character by character and stores in `str'
       str[i]=ch;
       i++;
    }
     for( i=strlen(str)-2; i>=0; i--)
     {
         ~~fp~~ putc(str[i],ft); /* Writing the reverse of the file contents */
     }
     fclose(fs);
     fclose(ft);
     getch();
}
```

Explanation:-

The above program reverse the contents of the source file and copies it into a target file.

The source file is opened in read mode and target file is opened in write mode.

The program reads the contents of source file character by charac and stores it into the string str.

The program reverse the content of source file and write it into target file using putc() function.

Reversing 'n' characters using Command line

```
main(int argc, char *argv[])
{
  char ch, c[100];
  int n, i=0;
  FILE *fp;
  clrscr();
  fp= fopen(argv[1], "r+");
  if (fp==NULL)
  { printf("file doesn't exist");
      getch();
      exit();
  }
  n= atoi(argv[2]);
```

```
printf("contents of file:");
while ((ch= getc(fp)) != EOF)
{ printf("%c", ch);
    c[i]= ch;
    i++;
}
rewind(fp);
for(i=n-1; i>=0; i--)
{ ~~putc(c[i]) fp~~
} putc(c[i], fp);
fclose(fp);
printf("Contents of file after
        reversing is");
fp=fopen(argv[1], "r");
while((ch=getc(fp))!= EOF)
{ printf("%c", ch);
}
fclose(fp);
getch();
```

12/3 1, 2, 6, 8, 9, 10, 11, 15, 16 29, 42, 46, 47, 58, 60

7/3

n=3 c[0]=w 1=e 2=l

low welcome

~~15, 42, 1, 42, 1, 49, 1, 57~~