UNIT -5Syllabus:
Introduction using Files in c, Declaration of File pointer, Opening a file, Closing and Flushing files, working with Text Files, Character Input and Output, End of File (EOF). Working with Binary Files, Direct File Input and Output, Sequential Versus Random File Access, Files of Records, working with Files of Records, Random Access to Files of Records, Other File Management Functions, Deleting a File Renaming a File. Low- Level I/O, Creating the header file using in the C program, Working with C Graphics functions.

1. Introduction to Files in C:

A File is a collection of data stored in the secondary memory. So far data was entered into the programs through the keyboard. So Files are used for storing information that can be processed by the programs. Files are not only used for storing the data, programs are also stored in files. In order to use files, we have to learn file input and output operations. That is, how data is read and how to write into a file.

A Stream is the important concept in C. The Stream is a common, logical interface to the various devices that comprise the computer. So a Stream is a logical interface to a file. There are two types of Streams, Text Stream and Binary Stream. A Text File can be thought of as a stream of characters that can be processed sequentially. It can only be processed in the forward direction.

2. Using Files in C:

To use a file four essential actions should be carried out. These are,

a. Declare a file pointer variable.
b. Open a file using the fopen() function.
c. Process the file using suitable functions.
d. Close the file using the fclose() and fflush() functions.

2.1. Declaration of file pointer:

A pointer variable is used to points a structure FILE. The members of the FILE structure are used by the program in various file access operation, but programmers do not need to concerned about them.

```
FILE *file_pointer_name;
Eg : FILE *fp;
```

2.2. Opening a file

To open a file using the fopen() function. Its syntax is,

```
FILE *fopen(const char *fname,const char* mode);
```

const char *fname represents the file name. The file name is like “D:\\501\example.txt”.

Here D: is a drive, 501 is the directory, example.txt is a file name.

const char *mode represents the mode of the file. It has the following values.

Mode	Description
r	Opens a text file in reading mode
w	Opens or create a text file in writing mode
a	Opens a text file in append mode
r+	Opens a text file in both reading and writing mode
w+	Opens a text file in both reading and writing mode
a+	Opens a binary file in reading mode
rb	Opens a binary file in reading mode
wb	Opens or create a binary file in writing mode
Ab	Opens a binary file in append mode
rb+	Opens a binary file in both reading and writing mode
wb+	Opens a binary file in both reading and writing mode
ab+	Opens a binary file in both reading and writing mode

Difference between write mode and append mode is,

Write (w) mode and Append (a) mode, while opening a file are almost the same. Both are used to write in a file. In both the modes, new file is created if it doesn't exists already.

The only difference they have is, when you open a file in the write mode, the file is reset, resulting in deletion of any data already present in the file. While in append mode this will not happen. Append mode is used to append or add data to the existing data of file(if any). Hence, when you open a file in Append(a) mode, the cursor is positioned at the end of the present data in the file.

2.3. Closing and Flushing Files

The file must be closed using the fclose() function. Its prototype is ,

*int fclose(FILE *fp);*

The argument fp is the FILE pinter associated with the stream. Fclose() returns 0 on success and -1 on error.

fflush() used when a file's buffer is to be written to disk while still using the file. Use of fflushall() to flush the buffers of all open streams. The prototype of these two functions are,

int flushall(void);

*int fflush(FILE *fp);*

Examples:

/*   Program to check whether the file exist or not */

```
void main()
{
        FILE *fp;
        char *x;
        clrscr();
        printf("\n enter the file name : ");
         gets(x);
        fp=fopen(x,"r");
        if(fp==NULL)
        {
                printf("The file  ---%s---  is not found in the present directory",x);
        }
        else
         {
                printf("The file ---%s--- is found in the present directory",x);
         }
          fclose(fp);
        getch();
}
```

3. Character input / output:

Two functions are used to read the character from the file and write it into another file. These functions are getc() & putc() functions.

Its prototype is,

*int getc(FILE *file_pointer);*

*putc(char const_char, FILE *file_pointer);*

3.1Detecting the end of a file:

When reading from a text-mode file character by character, one can look for the end-of-file character. The symbolic constant EOF is defined in stdio.h as -1, a value never used by a real character.

Eg: while((c=getc(fp)!=EOF). This is the general representation of the End Of the File.

Examples:

/* Read a string into a text file */

```
void main()
{
        FILE *fp;
        char text[80];
        int i;
        clrscr();
        fp=fopen("Sample.txt","w");
        printf("\n Enter the text : ");
        gets(text);
        for(i=0;i<strlen(text);i++)       // Read a stream of charcters
        putc(text[i],fp);
        if(fp!=NULL)
                printf("\n The string copied into a text file ");
        fclose(fp);
        getch();
}
```

4. Working with Text files :

C provides various functions to working with text files. Four functions can be used to read text files and four functions that can be used to write text files into a disk.

These are,

- fscanf() & fprintf()
- fgets() & fputs()
- fgetc() & fputc()
- fread() & fwrite()

The prototypes of the above functions are,

1. int fscanf(FILE *stream, const char *format,list);
2. int getc(FILE *stream);
3. char *fgets(char *str,int n,FILE *fp);
4. int fread(void *str,size_t size,size_t num, FILE *stream);
5. int fprintf(FILE *stream, const char *format,list);
6. int fputc(int c, FILE *stream);
7. int fputs(const char *str,FILE *stream);
8. int fwrite(const void *str,size_t size, size_t count,FILE *stream);

Example programs on text files:

/* Read a string into a text file using fputs() function */

```
void main()
```

```
{
        FILE *fp;
        char text[80];
        int i;
        clrscr();
        fp=fopen("Sample.txt","w");
        printf("\n Enter the text : ");
        gets(text);
        fputs(text,fp);                         //write a string into a file
        if(fp!=NULL)
                printf("\n The string copied into a text file ");
        fclose(fp);
        getch();
}
```

/* Program to copy from one file to another file */

```
void main()
{
        FILE *fp,*fp1;
        int ch;
        clrscr();
        fp=fopen("ex_file4.c","r");
        fp1=fopen("ex_file2.c","w");
        if(fp==NULL)
                printf("File is not available");
        ch=getc(fp);
        while(ch!=EOF)
        {
                putc(ch,fp1);
                ch=getc(fp);
        }
        fclose(fp);
        fclose(fp1);
        getch();
}
```

/* Program to display the content of the file */

```
void main()
{
            FILE *fp;
            int ch;
            clrscr();
            fp=fopen("ex_file4.c","r");
            if(fp==NULL)
                        printf("File is not available");
            ch=getc(fp);
            while(ch!=EOF)
            {
                        printf("%c",ch);
                        ch=getc(fp);
            }
            fclose(fp);
            getch();
}
```

/* read and write the content of the file using fprintf() and fscanf() functions */

```
void main()
{
    FILE *fptr;
    char name[20];
    int age;
    float salary;
    clrscr();
    /*  open for writing */
 fptr = fopen("abc.txt", "w");
    if (fptr == NULL)
    {
            printf("File does not exists \n");
            return;
    }
    printf("Enter the name \n");
    scanf("%s", name);
```

```
   fprintf(fptr, "Name  = %s\n", name);
   printf("Enter the age\n");
   scanf("%d", &age);
   fprintf(fptr, "Age  = %d\n", age);
   printf("Enter the salary\n");
   scanf("%f", &salary);
   fprintf(fptr, "Salary  = %f\n", salary);
   getch();
   fclose(fptr);
}
```

5. Working with binary files

A Binary file is similar to the text file, but it contains only large numerical data. The Opening modes are mentioned in the table for opening modes above.

Following steps are sued for copying a binary file into another are as follows,

1. open the source file for reading in binary mode.
2. Open the destination file for writing in binary mode.
3. Read a character from the source file. Remember, when a file is first opened, the pointer is at the start of the file, so there is no need to position the file pointer explicitly.
4. If the function feof() indicates that the end of thesource file has been reached, then close both files and return to the calling program.
5. If end-of- file has not been reached, write the character to the destination file, and then go to step 3.

Example:

/* program to copy the binary file from another file   */

```
void main()
{
        FILE *fp,*fp1;
        char *s,*s1;
        int ch;
        clrscr();
        printf("\n enter the source file : ");
        gets(s);
        printf("\n enter the destination file : ");
        gets(s1);
        fp=fopen(s,"rb");
        fp1=fopen(s1,"wb");
        ch=fgetc(fp);
        while(!feof(fp))
        {
                fputc(ch,fp1);
                ch=fgetc(fp);
        }
        fclose(fp);
        fclose(fp1);
        getch();
}
```

6. Direct File Input and Output:

Direct input and output is only with binary-mode files. With direct output, blocks of data are written from the memory to disk. Direct input reverses the process. A block of data is read from a disk file into memory.

fread() and fwrite() functions are used to read and write the binary files on direct I/O.

example:

```
        const char *mytext = "The quick brown fox jumps over the lazy dog";
        FILE *bfp= fopen("test.txt", "wb");
        if (bfp)
         {
                fwrite(mytext, sizeof(char), strlen(mytext), bfp) ;
                fclose(bfp) ;
        }
```

6.1. Sequential versus Random File Access

Every open file has an associated file position indicator, which describes where read and write operations take place in the file. The position is always specified in bytes from the beginning of the file. When a new file is opened, the position indicator is always at the beginning of the file, i.e., at position 0. Because the file is new and has a length of 0, there is no other location to indicate. When an existing file is opened, the position indicator is at the end of the file if the file was opened in the append mode, or at the beginning of the file if the file was opened in any other mode.

Sequential files are generally used in cases where the program processes the data in a sequential fashion – i.e. counting words in a text file – although in some cases, random access can be feigned by moving backwards and forwards over a sequential file.

True random access file handling, however, only accesses the file at the point at which the data should be read or written, rather than having to process it sequentially. A hybrid approach is also possible whereby a part of the file is used for sequential access to locate something in the random access portion of the file, in much the same way that a File Allocation Table (FAT) works.

7. File of Records:

Most C program files may be binary files, which can logically be divided into fixed-length records. Each record will consist of data that conforms to a previously defined structure. In C, This structure can be formed using a struct data type. The records are written into disk sequentially. This happens because as each record is written to disk, the file position indicator is moved to the bye immediately after the last byte in the record just written. Binary files can be written sequentially to the disk or in a random access manner.

7.1. Working with files of records:

Example:

/* Program to append , modify and display the records using files of records */

```
struct item
{
        int itemcode;
        char name[30];
        double price;
};
void append();
void modify();
void dispall();
void dele();
int main()
{
        int ch;
        struct item it;
        FILE *fp;
        fp=fopen(“item.dat”,”w”);
        if(fp==NULL)
                printf(“\n Error in opening the file … ”);
        printf(“\n enter the ITEM code :”);
        scanf(“%d”,&it.itemcode);
        printf("\n Enter the ITEM name : ");
        scanf("%s",&it.name);
        printf("\n Enter the price : ");
        scanf("%lf",&it.price);
        fprintf(fp,"%d \t %s \t %lf \n",it.itemcode,it.name,it.price);
        fprintf(fp,"%d",0);
        fclose(fp);
        while(1)
        {
                printf("\n 1. Append records …");
                printf("\n 2. Display all the records …");
                printf("\n 3. Edit the records …");
                printf("\n 4. Delete the records …");
                printf("\n 5. Exit  …");
                printf("\n Enter your choice …");
                scanf("%d",&ch);
        }
        switch(ch)
        {
                case 1: append(); break;
                case 2: dispall(); break;
                case 3: modify(); break;
                case 4: dele(); break;
                case 5: exit(0); break;
        }
}
void append()
{
        FILE *fp;
        struct item it;
        fp=fopen("item.dat","r");
        if(fp==NULL)
                printf("\n Error in opening  a file …");
        printf("\n enter the ITEM code :");
        scanf("%d",&it.itemcode);
        printf("\n Enter the ITEM name : ");
        scanf(“%s”,&it.name);
        printf(“\n Enter the price : ”);
        scanf(“%lf”,&it.price);
        fprintf(fp,”%d \t %s \t %lf \n”,it.itemcode,it.name,it.price);
        fprintf(fp,”%d”,0);
        fclose(fp);
}
void dispall()
{
        FILE *fp;
        struct item it;
        fp=fopen(“item.dat”,”r”);
        if(fp==NULL)
                printf(“\n Error in opening a file …..”);
        while(1)
        {
                fscanf(fp,”%d”,&it.itemcode);
                if(it.itemcode==0)
                        break;
                fscanf(fp,”%s”,it.name);
                fscanf(fp,”%lf”,&it.price);
                fprintf(fp,”%d \t %s \t %lf \n”,it.itemcode,it.name,it.price);
        }
        fclose(fp);
}
void modify()
{
        FILE *fp,*fptr;
        struct item it;
        int icd,found=0;
        fp=fopen(“item.dat”,”r”);
        if(fp==NULL)
                printf(“\n Error in opening a file ”);
        fptr=fopen(“temp.dat”,”w”);
        if(fptr==NULL)
                printf(“\n Error in opening a file ”);
        printf("\n Enter the Item code to edit");
        scanf("%d",&icd);
        while(1)
        {
                fscanf(fp,"%d",&it.itemcode);
                if(it.itemcode==0)
                        break;
                if(it.itemcode==icd)
                {
                        found=1;
                        fscanf(fp,"%s",it.name);
                        fscanf(fp,"%lf",&it.price);
                        printf("\n Existing record is ………. \n");
                        fprintf(fp,"%d \t %s \t %lf \n",it.itemcode,it.name,it.price);
                        printf("\n Enter the new ITEM name : ");
                        scanf("%s",&it.name);
                        printf("\n Enter the new item price : ");
                        scanf("%lf",&it.price);
                        fprintf(fp,"%d \t %s \t %lf \n",it.itemcode,it.name,it.price);
                }
                else
                {
                        fscanf(fp,"%s",it.name);
                        fscanf(fp,"%lf",&it.price);
                        fprintf(fp,"%d \t %s \t %lf \n",it.itemcode,it.name,it.price);
                }
        }
        fclose(fptr);
        fclose(fp);
}
void dele()
{
        FILE *fp,*fptr;
        struct item it;
        int icd,found=0;
        fp=fopen("item.dat","r");
        if(fp==NULL)
                printf("\n Error in opening a file ");
        fptr=fopen("temp.dat","w");
        if(fptr==NULL)
                printf("\n Error in opening a file ");
        printf("\n Enter the Item code to delete");
        scanf("%d",&icd);
        while(1)
        {
                fscanf(fp,"%d",&it.itemcode);
                if(it.itemcode==0)
                        break;
                if(it.itemcode==icd)
                {
                        found=1;
                        fscanf(fp,"%s",it.name);
                        fscanf(fp,"%lf",&it.price);
                }
                else
                {
                        fscanf(fp,"%s",it.name);
                        fscanf(fp,"%lf",&it.price);
                        fprintf(fp,"%d \t %s \t %lf \n",it.itemcode,it.name,it.price);
                }
        }
        fclose(fptr);
        fclose(fp);
}
```

8. Random Access to files of records

For random access to files of record, the following functions are used.

1. fseek() -- by using fseek(), one can set the position indicator anywhere in the file. It's prototype is,

```
int fseek(FILE *fp,long offset,int origin);
```

It has the origins as, SEEK_SET, SEEK_CUR,SEEK_END

2. ftell() --- To determine the value of a file's position indicator, use ftell(). The prototype is,

```
long ftell(FILE *fp);
```

3. rewind() ------ To set the position indicator to the beginning of the file, use the function rewind(). Its prototype is,

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

Example:

/* program to copy the content from one file to another file using fseek() function. */

```
void main()
{
   /*
   File_1.txt is the file with content and,
   File_2.txt is the file in which content of File_1
   will be copied.
   */
   FILE *fp1, *fp2;
   char ch;

   int pos;
   clrscr();
   if ((fp1 = fopen("File_1.txt","r")) == NULL)
   {
         printf("\nFile cannot be opened");
         return;
   }
   else
   {
         printf("\nFile opened for copy...\n ");
   }
   fp2 = fopen("File_2.txt", "w");
   fseek(fp1, 0L, SEEK_END); // file pointer at end of file
   pos = ftell(fp1);
   fseek(fp1, 0L, SEEK_SET); // file pointer set at start
while (pos--)
   {
         ch = fgetc(fp1);  // copying file character by character
```

```
            fputc(ch, fp2);
    }
    getch();
    fcloseall();
}
```

9. Other File Management functions

The copy and delete operations are also associated with file management. Though one could write programs for them, the C standard library contains functions for deleting and renaming files.

9.1. Deleting a file

The library function remove() is used to delete a file. Its prototype is,

```
            int remove(const char *filename);
```

9.2. Renameing a file :

The rename() function changes the name of an existing disk file. Its prototype is,

```
            int rename(const char *oldname, const char *newname);
```

Example:

```
int main()
{
            char file[80];
            /* prompt for filename to delete */
            printf("File to delete :");
            gets(file);
            /* delete the file */
            if(remove(file)==0)
                        printf("removed %s",file);
            else
                        printf("file cannot be removed");
}
```

10. Low – Level I/O

This form of I/O is unbuffered. That is, each read or write request results in accessing the disk directly to fetch/put a specific number of bytes. There are now formatting facilities. Instead of file pointers, we use low-level file handles or file descriptors, which give a unique integer number to identify each file.

To open a file the following function is used.

```
int open(char *filename, int flag, int perms);
```

To create a file the following function is used.

```
creat(char *filename, int perms);
```

To close the file the following function is used.

```
int close(int handle);
```

11. Creating the header files in C

Creating the header files using the following steps,

Step 1: Create the function, using function header and function body.

Example:

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

Step 2: Save the above function using [.h] extension. and compile the code.

Example:

above function can be save as myhead.h.

Step 3: Write the main program, include our own header file in the program. Now main function automatically includes the function available in the header file.

Example:

```
#include<stdio.h>
#include ”myhead.h”
void main()
{
        int num1=10,num2=20,num3;
`       num3=add(num1,num2);
        printf(“\n Sum of two numbers is  : %d”,num3);
}
```

Note : When running the program, header file and the program must be present in the same folder.

12. Working with C Graphics:

In a C program, first step is to initialize the graphics drivers on the computer. This is done using the initgraph method provided in graphics.h library. In the next few pages we will discuss graphics.h library in details. Important functions in graphic.h library will be discussed in details and samples programs will be provided to show the power of C programming language.

We will restrict our discussion on Graphics in C Language to 16 bit C programming and MS DOS environment and 640×480 VGA monitor.

12.1 Graphics mode Initialization

First of all we have to call the initgraph function that will initialize the graphics mode on the computer. initigraph has the following prototype.

```
void initgraph(int far *graphdriver, int far *graphmode, char far *pathtodriver);
```

Initgraph initializes the graphics system by loading the graphics driver from disk (or validating a registered driver) then putting the system into graphics mode. Initgraph also resets all graphics settings (color, palette, current position, viewport, etc.) to their defaults, then resets graphresult to 0.

graphdriver

Integer that specifies the graphics driver to be used. You can give graphdriver a value using a constant of the graphics_drivers enumeration type whcih is listed in graphics.h. Normally we use value as “0” (requests auto-detect). Other values are 1 to 10 and description of each enumeration type.

graphmode

Integer that specifies the initial graphics mode (unless *graphdriver = DETECT). If *graphdriver = DETECT, initgraph sets *graphmode to the highest resolution available for the detected driver. You can give *graphmode a value using a constant of the graphics_modes enumeration type and description of each enumeration type.

pathtodriver

Specifies the directory path where initgraph looks for graphics drivers (*.BGI) first.

1. If they’re not there, initgraph looks in the current directory.
2. If pathtodriver is null, the driver files must be in the current directory.

graphdriver and graphmode must be set to valid graphics_drivers and graphics_mode values or you’ll get unpredictable results. (The exception is graphdriver = DETECT.)

After a call to initgraph, *graphdriver is set to the current graphics driver, and *graphmode is set to the current graphics mode. You can tell initgraph to use a particular graphics driver and mode, or to auto detect the attached video adapter at run time and pick the corresponding driver. If you tell initgraph to auto detect, it calls detectgraph to select a graphics driver and mode.

Normally, initgraph loads a graphics driver by allocating memory for the driver (through _graphgetmem), then loading the appropriate .BGI file from disk. As an alternative to this dynamic loading scheme, you can link a graphics driver file (or several of them) directly into your executable program file.

Example:

/* Write a C program to draw a rectangle, circle, line, and ellipse and display text on the screen using graphics */

```
#include<graphics.h>
#include<conio.h>
#include<stdio.h>
main()
{
   int gd = DETECT,gm,left=100,top=100,right=200,bottom=200,x= 300,y=150,radius=50;

   initgraph(&gd, &gm, "C:\\TC\\BGI");

   rectangle(left, top, right, bottom);
   circle(x, y, radius);
   bar(left + 300, top, right + 300, bottom);
   line(left - 10, top + 150, left + 410, top + 150);
   ellipse(x, y + 200, 0, 360, 100, 50);
   outtextxy(left + 100, top + 325, "My First C Graphics Program");

   getch();
   closegraph();
   return 0;
}
```