Chapter XIV

File Handling

Many applications require that information be written to or read from an auxiliary storage device. Such information is stored on the device in the form of a data file. Thus, data files allow us to store information permanently, and to access and alter that information whenever necessary.

In c, an extensive set of library function 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.

When working with a data file, the first step is to establish a buffer area, where information is temporarily stored while being transferred between the computer memory and the data file. This buffer area allows information to be read from or written to a data file more rapidly than would otherwise be possible. The buffer area is established by writing

FILE * fp;

Where FILE is a special structure type that establishes the buffer area, and fp is a pointer variable that indicates the beginning of the buffer area. The structure type FILE is defined in the file stdio.h.

A data file must then be opened before it can be created or processed. This associates the filename with the buffer area. It also specifies how the data file will be utilized, i.e. as a read-only file, a write-only file, or a read/write file, in which both operations are permitted. The library function fopen is used to open a file. This function is typically written as

fp = f open(file-name, file-type);

Where file-name & file-type are strings that represent the name of the data-file and the manner in which the data file will be utilized. The file-type must be one of the strings mentioned below:

Mode	if the file exists	if the file does not exist
"r"	opens the file for reading	Error
"w"	opens a new file for writing	Creates a new file
"a"	opens the file for appending	Creates a new file
"r+"	opens the file for reading & writing	Error
"w+"	opens a new file reading & writing	creates a new file
"a+"	opens the file for reading & appending	Creates a new file

The fopen function returns a pointer to the beginning of the buffer are associated with the file. A NULL value is returned if the file cannot be opened. Finally, a data file must be closed at the end of the program. This can be accomplished with the library function f close. The syntax is

fclose(fp);

The value returned by the fopen function can be used to generate an error message if a data file cannot be opened.

1. stdin, stdout, stderr

Whenever a C program is executed, three "files" are automatically opened by the system for use by the program. These files are identified by the file pointer stdin, stdout, stderr which are defined in stdio.h. The file pointer stdin' identifies the standard input of the program, and it defaults to the terminal.

All standard I/0 functions that perform input and do not take a file pointer as an argument get their input from stdin'.

Example:

The scanf function reads its input from stdin, and a call to this function is equivalent to a call to the f scanf function with stdin'as the first argument. So,

```
fscanf (stdin, "%d", &i);
```

will read in the next integer value from the standard input. stout refers to the standard output which also normally defaults to the terminal. So, printf("hello\n"); can be replaced by an equivalent call to the fprintf function with stout as the first argument:

```
fprintf (stdout, "hello\n");
```

Note: The function putchar (c) is equivalent to putc (c, stdout)

The function getchar () is equivalent to getc (stdin)

The file pointer "stderr" identifies the standard error file. This is where the error messages produced by the system are written, and it also defaults to the terminal. stderr exists so that error messages can be log get to a device or file other than that where the normal output is written. This is particularly desirable whenever the program's output is redirected to a file.

Example:

```
if((fp  = fopen(file,  "r"))  == NULL)
{
    fprintf(stderr, "can't open file  \n");
    ----------
    ----------
 }
```

2. Character Input / Output

   i) getc : The function get c expects a single argument, a pointer to the file to be read; get c then will return the next character in the specified file or E0F if the end of the file is reached or if there is an error.
   ii) putc : The function put c expects two arguments, a character to write and a pointer to the file to write.

usage :

```
c = getc (fp, source);
putc (c, fpsink);
```

3. The exit function

The function call

```
exit(a);
```

has the effect of termination (exiting from) the current program. Any open files will be automatically closed by the system.

Example:

```
#include<stdio.h>
main ()
{
   FILE *fp;
   if ((fp  =  fopen(fil,  "r"))  == NULL)
   {
     fprintf(stderr, "can't open file \n");
     exit(1);
    }
 }
```

To detect an end-of-file condition, we have the library function feof available. The function returns a non-zero value (TRUE) if an end-of-file condition has been detected and a value zero(FALSE) if an end-of-file is not detected.

Format: feof(file-pointer)

Example:

```
/*program to read a file and print it */
#include<file.h>
#include<stdio.h>
 main ( )
{
    char   c;
    FILE * fp;
    fp = f open ("a.inp", "r");
    while((c  = getc (fp))  ! = E0F)
          putchar(c);
    fclose(fp);
 }
```

Example:

/* program to count the number of bytes in a source file. The program prompts the user for a filename and then Concatenates the .c extension to this name */

```
#include<stdio.h>
#include<string.h>
main ()
{
      FILE   * fp;
      static char extension[ ]  =  ".c";
      char fil  [25];
      int count;
      printf(""filename: ");
      scanf("%s",  fil );
      strcat(fil,  extension );
      fp = fopen(fil,  "r");
      for (count = 0; getc(fp) != E0F; ++count)
            ;
      printf("byte size: %d\n", count);
      fclose(fp);
 }
```

4. String Input-Output

i) fgets

The function gets expects three arguments: the address of an array to store a character string, the max. no of characters to store, and a pointer to the file to read. If MAX is the specified max. Number of characters to store, fgets will read characters from the file into the array until

- MAX-1 characters have been read
- All characters up to MAX including the next newline character have been read
- The end-of file is reached

if fgets reads a newline, the newline will be stored in the array. If at least one character was read, fgets will add the null terminator '0' to the end of the string. fgets never stores more than MAX characters (including newline and '\0'). If no characters were stored or an error occurs, gets will return NULL; otherwise, fgets will return the address of the array.

Format: fgets(buffer, n, fil-pointer)

ii) fputs

The function fputs, writes to a file. If expects two arguments: the address of a null - terminated character string and a pointer to a file; fputs simply copies the string to the specified file. It does not add a newline to the end of the string. fputs copies the terminating null character to the respective file. Both return the last character written or, in case of error, E0F.

Format: fputs(buffer, file-pointer)

5. Other Input-Output Function

i) fread

The function fread is more specialized than fgets.It expects four arguments:

fread (receiving-array, input size, input count, file-ptr) The argument receiving-array must be large enough to load input-count strings, each of size input-size from the file pointer to by file-ptr. The function fread returns the number of strings it successfully reads into the receiving array.

ii) fwrite

The function fwrite is the inverse of fread. It, too, expects four arguments:

fwrite(array, output-size, oupput-count, file-ptr) The function tries to write output-count strings, each of size output-size, from array into the file printer to by file-ptr.The function fwrite returns the number of strings successfully written.

iii) fprintf and fscanf

These functions are provided to perform the analogous operations of the printf and scanf functions on a file. Those functions take an additional argument, which is the file pointer that identifies the file to which the data is to be written or

from which the data is to be read. The syntax is **fprintf(file-pointer, format string, argument-int);**
Writes the specified arguments to the file identified by file-pointer, according to the format specified by the format string. the number of characters written is returned.

fscanf(file-pointer, format string, arglist);

Data item are read from the file identified by file-pointer, according to the format specified by the format string. The values that are read are stored into the arguments specified, each of which must be a pointer. The fscanf function returns the number of items successfully nead and assigned or the value E0F if end of file is reaches before the first item is read.
Example:

```
/* to count the no. of lines in a file */
#include<stdio.h>
#include<file.h>
main ()
{
      char fn[20], line[80];
      FILE *fpi;
      int numLines  =  0
      printf("file name : ");
      gets(fn);
     if((fpi = fopen(fn, “r”)) = = NULL)
          {
               printf("%s file not found\n", fn);
               exit(1);
          }
      while((fgets (line, 80, fpi)) ! = NULL)
      numLines++;
      printf(Ïnput lines = %d", numLines);
}
```

Example: program to find net pay of an employee; Input given is employee no., name, basic pay.

np = basic pay = da = har -pf
da = 10% of bp subject to a maximum of 170
hra = 12% of (bp + da)
pf = 8.33% of bp
Output to be printed is empNum, name, basic pay & net pay.

```
/* program without using a structure*/
#include<stdio.h>
#include<string.h>
main ()
{
          FILE * fp1, *fb2;
          float bp, da, hra, bf, np;
          Char name [50];
          int empNum;
         if((fp1 = fopen(ïnput.dat", "r"))  == NULL)
         {
              printf("cannot open input file");
              exit(1);
         }
         if((fp2 = fopen(öutput.dat", "w"))  ==  NULL)
         {
               printf("cannot open output file");
```

```
            exit(2);
        }
    while((fscanf(fp1, "%d%s%f", &empNum, name, &bp))  !=  E0F)
    {
        da =  (10 / 100) * bp;
        if(da > 170)
        da = 170;
        hra = (ba + da)  * 12 / 100;
        pf = 8.33  *  bp / 100;
        np =  bp + da + hra  - pf;
        fprintf(fp2, "%d%s%f%f", empNum, name, bp, np);
    }
    fclose(fp1)
    fclose(fp2)
}
```

Example: Program to read name, empCode, rate, allowance, hours-
working.
pay of an employee = hours- working x rate
If pay <= 1500 tax = 0 otherwise,
tax = (pay -1500) * 10/100
net pay = pay - tax + allowance

print in the output- name, empCode, pay tax, allowance and net-pay.

```
#include<stdio.h>
#include<string.h>
#include<file.h>
main ()
{
    FILE * fpIn, *fpOut;
    struct empRec
    {
        char name[50];
        int empCode;
        float rate;
        float allowance;
        float hrwkd;
    } emp;
    struct outRec
    {
        char name [50];
        int  empCode;
        float grosspay;
        float tax;
        float allow;
        fload npay;
    }  emp0;
    if((fpIn  =  fopen(“inp.fil", "r"))  == NULL)
    {
        printf("cannot open input file");
        exit(1);
    }
```

```
        if((fpout = fopen("out.fil", "w")) == NULL)
        {
                printf("cannot open output file");
                exit(2);
        }
        while(fscanf(fpIn, "%s%d%f%f%f", emp.name, &emp.empCode, &emp.rate,
                        &emp.allowance &emp.hrwkd) ! = E0F)
        {
                        Emp0.grasspay = emp.hewkd * emp.rate;
                        if(emp0.grosspay > 1500)
                        emp0.tax = (emp0.grosspay - 1500) * 10/ 100;
                        else
                                emp0.tax =0;
                    emp0.allow = emp.allowance;
                    emp0.pay = emp0.grosspay - emp0.tax +emp0. allow;
                    strcpy(emp0.name, emp.name);
                    emp0.empcode = emp.empCode;
                    fprintf(fpout, "%s%d%f%f%f%f\n", emp0,name,emp0.empCode,
                                                        emp0.glosspay, emp0.tax, emp0.allow, emp0.`npay);
        }
        fclose(fpIn);
        fclose(fpout);
}
```

Example

```
      /* to write data into a binary file using fwrite */
        #include<stdio.h>
        struct Student
        {
              char name[21];
              int rNum;
         };
         main ()
         {
                FILE * fp;
                struct Student v;
                int i, num;
                if((fp = fopen("binfile",wb")) == NULL)
                {
                      printf("trouble writing binfile");
                      exit(1);
                }
                printf("no. of records: ");
                scanf("%d%*c", &num);
                for(i = 0; i < num ; i++)
                {
                       printf("name: ");
                       gets(v. name);
                       printf("roll no.: ");
                       scanf("%d*c",&v.rNum);
                       fwrite(&v, sizeof(v), 1, fp);
                }
```

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

Input : 3
ABCD
100
PQRS
101
XYZ
102

Example :

```
/* to read a binary file created by write using fread */
#include<stdio.h>
struct Student
{
      char name[21];
      int rNum;
};
int nRecs (FILE * fp, int rexSize);
main()
{
        FILE *fp;
        struct  Student  v;
        if((fp  =  fopen("binfile","rb"))  = =  NULL)
        {
              printf("trouble  reading binfile");
              exit(1);
        }
        printf("\n file contains %d recs.\n", nRecs(fp, sizeof(struct student)));
        printf("name \t roll no. \n");printf(" ---- \t  --------  \n");
        while(fread(&v, sizeof(v), 1, fp)  = = 1)
              printf("%s \t %d \n", v.name , v.rNum);
        fclose(fp);
  }
  int nRecs (FILE *fp, int recSize)
  {
        int ret;
        ong sv;
        sv = ftell(fb);
        ret = fseek(fp, 0l, 2) ? 0 : ftell(fp) / recSize;
        fseek(fp, sv, 0);
        return ret;
  }
```

Output:

name	roll no
ABCD	100
PQRS	101
xyz	102

6. In-Memory Format Conversion Functions

The functions sprintf and sscanf are provided for performing data conversion in memory. These functions are analogous to the fprintf and fscanf functions except a character string replaces the file pointer as the first argument.

i) Sprintf (buffer, format string, agilest)

The specified arguments are converted according to the format specified by the format string and are placed into the character array pointer to by buffer. A null character is automatically placed at the end of the string inside buffer. The number of character placed into buffer

Example:

springf(text, "%d + %d", 20, 50);

will place the character string "20 + 50" into text.

ii) ii) sscanf(buffer, format ageist)

The values specified by the format string are "read" from the buffer and stored into the corresponding pointer argument. The number of items successfully assigned is returned by this function.

Example:

i) char str[80];
gets(str);
sscanf(str, "%2d%2d%d", &dd, &mm, &yy);
Now if str = 230891 then dd = 20, mm = 08 and yy = 91.

ii) sprintf(str, "%d", i);
This function will for converting the value of i into str. It can be used for converting itoa(0).

iii) sscanf("july 16", "%s%d", mon, &day:
Will store the string "july"inside 'mon' and will assign the integer value 16 today'.

iv) if(sscanf(argv[1], "%f", &fv) ! =1)
{
fprintf(stderr, "bad no: %s\n", argv[1]);
exit(1);
}

will convert the first argument to a floating point number, and will check the value returned by sscanf to sec if a number was successfully read from argv[1].

7. Moving Around In A File

The functions fseek, ftell, and rewind can be used to determine or change the location of the file position marker.
The header of fseek can be writer as

```
int fseek(file-pointer, offset, base-position)
FILE *file-pointer;
long fset;
int base-position;
```

The function seek resets the file position marker in the file specified by file-pointer to offset bytes from the beginning of the file (base-position=0), from the current location of the file position marker (base-position=1), or from the end of the file (base-position=2). The function fseek will return 0 if they seek is successful, and E0F otherwise.

i) fseek (file-pointer, 0L 0;
sets the file position marker zero bytes beyond the first bytr, i.e. to the first byte.

ii) fseek(file-pointer, 0L, 2);
sets the file-position marker zero bytes from the end of file, i.e., to the end of file.

iii) fseek(file-pointer, -1L, 2);
sets the file position marker to the last byte in the file.

iv) fseek(file-pointer, 0L, 1)

does not change the file position marker.

The header of rewind can be written as

```
Void rewind (file-pointer)
FILE *file-pointer;
```

The function rewind resets the file position marker in the file specified by the file-pointer to the beginning of the file.
Executing

```
rewind(file-pointer);
```

has the same effect as executing

```
fseek(file-pointer,  0L,  0);
```

The header of ftell can be writer as

```
Long ftell(file-pointer)
FILE *file-pointer;
```

The function fell returns the location of the file position marker in the file specified by fill-pointer.

Exercise # 14

Part A

Q 1: Fill in the blanks

a) If a disk file opened inn read in read mode is non-existent, then "fopen" returns ----------.

b) The function "fclose" is used to ----------.

Part B

Q 1 : Differentiate between the following :

a) Scanf and fscanf

b) printf and fprintf

Q 2 : Explain the format and purpose of the following function :

a)	fprintf	d)	fclose
b)	fscanf	e)	sprintf
c)	getc	f)	sscanf

Q 3 : Suppose that the file test.data contains

```
ABC
```

What is printed?

```
#include<stdio.h>
main ()
{
    FILE    *gp;
    long    pos;
      fp =  fopen("TEST.DAT". "r+");
      putchar(getc(fp));
      pos =  ftell(fp);
      putchar(getc(fp));
      fseek(fp,pos,  0);
      putchar(getc(fp));
      fseek(fp,  pos,  0);
      putc =  ('Q',  fp);
      fseek(fp,pos,  0);
      putchar(getc(fp));
      fseek(fp,  0L,  0 );
      putchar(getc(fp));
}
```

Q 4 : Suppose that the file data.dat is

ABCDEFGHIJKLMNOPQRSTUVWXYZ

What is printed?

```
#include<stdio.h>
main ()
{
        char     l;
        FILE      *fp;
        fp = fopen("data.dat",  "r" );
        fseek(fp, 5L, 0);
        l = getc(fp);
        printf("%c\n",  l);
        fseek(fp, 0L, 2);
        printf(" E0F \n");
        else
        printf("%c\n", l );
        fseek(fp,  -5L, 2 );
         l = getc(fp );
        printf("%c",  l );
}
```

Chapter-XV

Command-line Arguments

The function main() can take arguments from a command line. Thus a user can write his own command by using their command line arguments.

The function main () can have two arguments. These arguments are specified as :

```
main (int argc, char *argv[])
```

Where argv is an array of pointers to strings, i.e., an array of strings and argc is an integer variable whose value is equal to the number of parameters passed.

The parameters are passed on the command line as follows:

```
prog-name par1 par2 ..... par3
```

The individual parameters must be separated from one another either by blank spaces or by tabs.

The program name will be stored as the first item in argv, followed by each of the parameters. Hence, if the program name is followed by n parameters, there will be (n + 1) entries in argv, ranging from argv[0] to argv[n]. Moreover, argc will be assigned the value (n + 1).

Example:

```
$ echo hello hi
```

Now, echo is stored in argv[0], hello as argv[1] and hi as argv[2]. It has a count of 3 arguments, i.e., argc = 3.

Example:

```
#include<stdio.h>
main(int argc, char *argv[ ] )
{
    int   i;
    printf(ärgc = %d\n", argc);
     for(i   =  0;  i  <  argc; i++)
        printf(ärgv[%d) = %s\n", i, argv[i]);
}
```

Example:

```
$ echon 3hello
/* to display hello3 times */
#include<stdio.h>

main(int argc, char *argv[])
    {
            int  cnt;
            cnt = atoi(argv[1]);
        while(cnt)
        {
            printf("%s\n", argv[2]);
            cnt--;
        }
    }
```

Example:

```
#include<stdio.h>
int cub (int), sq(int);
main(int argc,char *argv[ ])
{
        int (*a) (int);
        if(! (strcmp(argv[1], "square")))
            a = sq;
        else
```

```
                a = cub;
        printf("%s of %d is : %d\n", argv[1], atoi(argv[2]), (*a)(atoi(argv[2])));
    }
    int cub(int i)
    {
        return  i * i * i;
    }
        int sq(int i)
    {
        return i * i;
    }
```

Example:

```
    /* to display a range of lines from a file*/

  #include<stdio.h>
     #include<stdio.h>
  #define  SIZ 132
    main(int args, char *argv[ ]
    {
        FILE *fp;
        int x, y, c, j, i;
        char buff[SIZ];
        if(argc  !  4)
        {
            printf(üsage:   comm. fileName spos epos\n");
            exit(1);
        }
        x = atoi(argv[2]);
        y = atoi(argv[3]);
        if (x > y)
        {
            printf("range not proper\n");
            exit(2);
        }
        }
        for(i = 0; i < x; ++i)
            fgets(buff, SIZ, fp);
        for(; i <= y; ++i)
        {
            fgets(buff,  SIZ, fp);
            puts(buff);
        }
        fclose(fp);
    }
```

Example:

```
    /* to display all the line from a file containing a particular string*/

    #include<stdio.h>
    #define MAX 132
    main(int argc, char *argv[ ])
    {
        char buff[MAX];
        FILE *fp;
        int i, j,  fl, ln;
        i  =  0;
        fl =  0;
        if(argc  ! = 3)
```

```
        {
                printf(usage: comm. filename str\n");
                exit(1);
        }
        ln = strlen(argv[2]);
        while(! feof(fp))
        {
                fgets(buff, MAX, fp);
                for(j = 0, i = 0; buff[i] && argv[1][j]; ++)
                {
                        if(buff[i] = = argv[1][j]
                        {
                                fi = 1;
                                j  ++:
                        }
                        else
                        {
                                i  =  0;
                                fl  =  0;
                        }
                }
                if(fl)
                        printf("%s\n", buff);
        }
        fclose(fp);
}
```

Exercise # 15

Part A

1. Write a program which will perform mathematical operations on the command line.
   e.g. $ calc 5 + 2 should return 7
2. Modify the above program to have a number of argument.
   e.g. $ rcalc * 2 3 5 should return 30
3. Write a program which appends the contents of one file into another file.
4. Write a program using command line arguments to display the last "n" lines from a file. If "n" is not specified, it should display the last 10 lines.
5. Write a program that displays the contents of a file 20 lines at a time. At the end of each 20 lines, have the program wait for a character to be entered from the terminal. If the character is the letter q, then the program should stop the display of the file, otherwise continue.