Module Code: CSE401

Session 07: Data Files

Session Speaker:

Prema Monish
premamonish.tlll@msruas.ac.in

Session Objective

- At the end of this lecture, student will be able to
  - use standard file operations

Session Topic

- File I/O in C
- Operations on files
- Modes of operation

Files

- Many real-life problems involve large volumes of data and in such situations, the console oriented I/O operations pose two major problems
  1. It becomes cumbersome and 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.
  3. So the data given to the program for one execution may not be available for the second execution.
- Concept of files to store data
  - File is a collection of data stored on the secondary storage device with a name called file name.
  - The data stored within the file can be read, modified and deleted through program using different File I/O functions provided by the language.

Files

- According to the data stored in the file, files are classified into
  1. Text files-data is stored as a stream of characters that can be processed sequentially.eg:5689 is stored as ‘5’ ‘6’ ‘8’ ‘9’ as 4 characters so 4bytes are required to store in memory.
  2. Binary files-data is stored on the disk in the same way as it is represented in the computer memory.eg:5689 is stored in binary format so 2 bytes to store in memory.
- **FILE -** Data type (a structure) to store file information permanently

File Operations

- C language provides number of pre-defined functions to perform file I/O. Most of these functions are defined with in the header file "stdio.h"

Functions	purpose
fopen()	To create a new file or to open an existed file
fclose()	To close the opened file
fgetc()	To read a character from the file
fputc()	To write a character on to the file
fprintf()	To write formatted output on to the file
fscanf()	To read formatted data from the file
feof()	To find end of file
fread()	To read data from binary file
fwrite()	To write data from binary file

fopen() Function

- It is the function used to create a new file or to open an existed file.

```
FILE *fopen(const char *path, const char *mode )
{
.....
.
.....
}
```

- Opens a file and returns FILE pointer.
- Path is the path to the file.
- Mode can be combinations of "r", "w" or "a". Other possibilities include "+".
  - E.g. FILE *fp=fopen("igate","w"); /* Creating a file in write mode */

File Opening Modes

File Mode	Description
r	Open a text file for reading
w	Create a text file for writing, if it exists, it is overwritten
a	Open a text file and append text to the end of the file
r+	Open a text file both in reading and writing mode
w+	Create a text file for writing and reading mode
rb	Open a binary file for reading
wb	Create a binary file for writing, if it exists, it is overwritten
ab	Open a binary file and append data to the end of the file
r+b	Open a binary file both in reading and writing mode
w+b	create a binary file both in reading and writing mode

fclose()

- Closes a file that was already opened in the program

```
int fclose(FILE*FP)
{
……..
……..
}
```

Opening and Closing a File

- if we want to check whether a file is already existed with the name before creating a new file, then first we try to read the file with the projected name.

```
#include<stdio.h>
int main()
{
 FILE *fp;
 fp=fopen("igate","r");
 if(fp==NULL)
  printf("File is not existed");
 else
  printf("File is existed");
 return 0;
}
```

fputc()

```
int fputc(char ch,FILE *FP)
{
.......
.....
}
```

- It accepts two arguments that are the character that we want to write on to the file and a file pointer.
- It returns a character that is written last time on to the file, returns -1 if any thing goes wrong in writing on to the file

fgetc()

- This function accepts the file pointer as argument and returns ASCII value of last fetched (accessed) character from the file.
- It returns -1 or EOF on reaching the end of file.

```
int fgetc(FILE *FP)
{
.......
.....
}
```

Read and write operation

```
#include<stdio.h>
int main()
{
      FILE *p;
      char ch;
      p=fopen("igate","w"); /* opeing file */
      while(1)
      {
            ch=getchar();     /* reading from Console Input */
            if(ch==-1)         /* checking end of file */
              break;
            fputc(ch,p);       /* writing on to file */
      }
     while(1)
      {
             ch=fgetc(p);           /* reading character from the file */
             if(ch==-1)              /* checking end of file */
               break;
             printf("%c",ch);    /* printing character on to the console output */
          }
      fclose(p);             /* closing the file */
       return 0;
}
```

fprintf()

- Similar to printf for file I/O
- write set of data values to file

```
int fprintf(FILE *stream, const char *format,list…)
{
…….
…..
}
```

fscanf()

- Similar to scanf for file I/O
- Read set of data values to file

```
int fscanf(FILE *stream, const char *format,List…)
{
…….
…..
}
```

fprintf and fscanf

```
int main(int argc, char** argv)
{
    FILE *fp;
    int day;
    char name[10];
    fp=fopen("reads.txt","w");
    printf("\nEnter data");
    scanf("%d%s",&day,&name);
   fprintf(fp,"%d%s",day,name);
    fclose(fp);
}
```

```
int main(int argc, char** argv)
{
FILE *fp;
int day;
char name[10];
fp=fopen("reads.txt","r");
if(fp==NULL)
    printf("\nFile does not exist");
 else{
    printf("\nDetails are");
    do{
        fscanf(fp,"%d%s",&day,&name);
      printf("%d %s",day,name);
    }while(!feof(fp));
}
    fclose(fp);
return 0;
}
```

feof()

- In case of unformatted text file we can find the end of file using either -1 or EOF. So we can read the file as long as EOF won't come.
- While reading the data from a formatted file we can't find the end of file using either EOF or -1 because here we are accessing the file record by record rather character by character.
- feof() function accepts the file pointer as argument, returns 0 (false) as long as fetching of records is successful and returns a non-zero (true) on reaching the end of file

```
int feof(FILE *fp)
{
.......
.....
}
```

fread() function

- It is the function used to read the binary data from the file and store into the object.
- It reads “n” items of “size” bytes each through the file pointer

```
int fread(void *ptr,int size,int n ,FILE *fp)
{
.......
.....
}
```

fwrite() function

- It is a function used to write an object (structure variable) on to the file in a binary format. It writes “n” objects of “size” bytes each through the file pointer.
- It accepts the address of an object, size of object, number of objects to be written and the file pointer as arguments.
- It returns the number of actually written objects onto the file and returns -1 if any thing goes wrong in writing.

```
int fwrite(void *ptr,int size,int n ,FILE *fp)
{
.......
.....
}
```

```c
#include<stdio.h>
struct address
{
 char name[20];
 char place[20];
 long int pin;
};
int main()
{
 FILE *p;
 struct address x;
 int n,i;
 p=fopen("address","wb");
 printf("How many records?");
 scanf("%d",&n);
 printf("Enter %d records:\n",n);
 for(i=1;i<=n;i++)
 {
   printf("Name:");
   scanf("%s",x.name);
   printf("Place:");
   scanf("%s",x.place);
   printf("Pin code:");
   scanf("%ld",&x.pin);
   fwrite(&x,sizeof(x),1,p);
 }
 fclose(p);
 printf("1 file created");
 return 0;
}
```

Output:
How many records?2
Enter 2 records:
Name:subbu
Place:Kavali
Pin code:524201
Name:Ramaiah
Place:Ulavapadu
Pin code:523201
1 file created

Summary

- File definition and types of file
- File standard operations
- Operation modes