Data File Handling

Unit 11

Introduction

Data stored on a medium, such as disk, are called a file. A computer file is analogous to the customary office file, which stores related information according to title or some other convenient label. In many programming situations, it is convenient to access a file than to enter a succession of individual data items from the keyboard.
Many applications require that information be written to or read from an auxiliary memory device in the firm of a data files. Thus, data files allow us to store information permanently, and to access and after that information whenever necessary.

End-Of-File:
It is important to understand that EOF is not a character. It is actually an integer value sent to the program by the operating system and is defined in a header file stdio.h to have a value of 1. No character with this value is stored in a file in a disk. While creating a file, when the operating system finds the last character to the file has been sent, it transmits the EOF signal. Both in text as well as in binary mode, the system keeps track of the total length of the file and will signal an EOF when this length has been reached.

Types of Files

1. Text files

Text files are the normal .txt files. You can easily create text files using any simple text editors such as Notepad.

When you open those files, you'll see all the contents within the file as plain text. You can easily edit or delete the contents.

They take minimum effort to maintain, are easily readable, and provide the least security and takes bigger storage space.

Types of Files

2. Binary files

Binary files are mostly the .bin files in your computer.

Instead of storing data in plain text, they store it in the binary form (0's and 1's).

They can hold a higher amount of data, are not readable easily, and provides better security than text files.

File

A file represents a sequence of bytes on the disk where a group of related data is stored.

File is created for permanent storage of data. It is a ready made structure.In C language, we use a structure pointer of file type to declare a file.

FILE *fp ;

File Operations

In C, you can perform four major operations on files, either text or binary:

1. Creating a new file
2. Opening an existing file
3. Closing a file
4. Reading from and writing information to a file

Standard I/O

It is the most common way of performing I/O in C programs. It provides four different ways of reading and writing data.

Character I/O

If the data can be read or written one character at a time, it is called character I/O.

Function with character I/O are:

fgetc(): it is used to read a character from a file. Its syntax is:

char_variable = fgetc(file_ptr_variable);

fputc(): it is used to write a character to a file. Its syntax is

fputc(char_variable, file_ptr_variable);

String I/O

If data can be written or read as strings, then it is String I/O.

Function with string I/O are:

fgets(): it is used to read a string from file. Its syntax is:

fgets(string_variable, int_value, file_ptr_variable);

Here, int_variable is the number of character in a string to be read including a null character.

fputs(): it is used to write a string to a file. Its syntax is:

fputs(string_variable, file_ptr_variable);

Formatted I/O

Like printf() and scanf() functions data can be read or written to a disk using functions like:

fscanf(): this function is formatted input function to read some integers, float, char or strings from a file. Its syntax is:

fscanf(file_ptr_variable, “control-specifier”, &list_variable);

fprintf(): this function is formatted output function to write some integers, float, char, or strings to a file. Its syntax is:

fprintf(file_ptr_variable, “control-specifier”, list_variable);

Record I/O

Record I/O writes numeric values to files in binary format so that integers are stored in 2 bytes, long integers are stored in 4 bytes and single precision floating point numbers are stored in 4 bytes.

This record I/O permits reading and writing and is not limited to character or strings or some numeric values but also arrays, structures or array of structures are stored in file using record I/O.

Record I/O

Functions involved in record I/O are:

fwrite(): it is used for record output. Its syntax is:

fwrite(&ptr, sizeof_array_or_structure, no._of_structure_or_array, fptr);

where ptr is the address of the array or structure to be written.

Size_of_array_or_structure is the integer value that shows the size of the structure or array which is being read or written

No._of_structure_or_array is the integer value that indicates the number of arrays or structures to be written or read.

Fptr is the file pointer of a file opened in binary mode.

Record I/O

Functions involved in record I/O are:

fread(): it is used for record input. Its syntax is:

fread(&ptr, sizeof_array_or_structure, no._of_structure_or_array, fptr);

where ptr is the address of the array or structure to be written.

Size_of_array_or_structure is the integer value that shows the size of the structure or array which is being read or written

No._of_structure_or_array is the integer value that indicates the number of arrays or structures to be written or read.

Fptr is the file pointer of a file opened in binary mode.

Opening a file in C

Before a program can write to a file or read from a file, the program must open it. While working with file, we need buffer area where information is stored temporarily. The buffer area is written as

```
FILE *ptr_variable;
```

Here, FILE is a special structure declared in stdio.h. The ptr_variable is a pointer that stores the beginning address of the buffer area allocated after a file has been opened. Using the fopen() function, a data file is opened.

The fopen() function is used to create a new file or to open an existing file.

∗fp = FILE ∗ fopen (const char ∗ file name, const char ∗mode) ;

fp=fopen (” text.txt ” , ”w”) ; // opens a file text.txt in writing mode

Here filename is the name of the file to be opened and mode specifies the purpose of opening the file. *fp is the FILE pointer (FILE *fp), which will hold the reference to the opened(or created) file.

Closing a file

The fclose() function is used to close an already opened file.

int fclose (FILE *fp) ;

Here fclose() function closes the file and returns zero on success, or EOF if there is an error in closing the file. This EOF is a constant defined in the header file stdio.h.

File Opening Modes

Modes	Description
r	This opens an existing text file for reading only. If the file exists, it loads into memory and sets up a pointer which points to the first character in it. If the file does not exist, it returns NULL
w	This opens an existing text file for writing only. If the file already exist, the contents are overwritten (content is deleted first and then written). If the file does not exist, a new file is created.
a	This opens an existing text file for appending (adding new information at the end of the file). If the file exist, it loads into memory and sets up a pointer that point to the last character in it. If the file does not exist, a new file is created.
r+	It opens a existing text file for reading and writing. Same as r mode.
w+	It opens a existing text file for reading and writing. Same as w mode.
a+	It opens a existing text file for both reading and appending. Same as a mode.

File Opening Modes

Modes	Description
rb	This opens an existing binary file for reading only. If the file exists, it loads into memory and sets up a pointer which points to the first character in it. If the file does not exist, it returns NULL
wb	This opens an existing binary file for writing only. If the file already exist, the contents are overwritten (content is deleted first and then written). If the file does not exist, a new file is created.
ab	This opens an existing binary file for appending (adding new information at the end of the file). If the file exist, it loads into memory and sets up a pointer that point to the last character in it. If the file does not exist, a new file is created.
rb+ or (r+b)	It opens a binary file for read/write
wb+ or (w+b)	It creates a binary file for read/write.
ab+ or (a+b)	It appends a binary file for read/ write.

Writing a character in a file

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

    f=fopen("abc.txt","w");
    if(f==NULL){
        printf("File not found");
    }
    printf("Enter a character:");
    scanf("%c",&ch);
    fputc(ch,f);
    fclose(f);
   return 0;
}
```

Reading a character in a file

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

    f=fopen("abc.txt","r");
    if(f==NULL){
        printf("File not found");
    }

    ch=fgetc(f);
    printf("The character read from the file is %c",ch);
    fclose(f);
   return 0;
}
```

Writing a string to a file

```
#include<stdio.h>
int main(){
    FILE *f;
    char ch[30];

    f=fopen("abc.txt","w");
    if(f==NULL){
        printf("File not found");
    }
    printf("Enter a character:");
    scanf("%s",ch);
    fputs(ch,f);
    fclose(f);
   return 0;
}
```

Reading a string from a file

```
#include<stdio.h>
int main(){
    FILE *f;
    char ch[30];

    f=fopen("abc.txt","r");
    if(f==NULL){
        printf("File not found");
    }
    fgets(ch,5,f); // 5 represents the no. of characters in string including null character
    printf("The string read from the file is %s",ch);
    fclose(f);
   return 0;
}
```

Writing using fprintf()

```
#include <stdio.h>

void main()

{

FILE *f;

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

fprintf(f, "Reading data from a file is a
common feature of file handling.\n");

printf ("Data Successfully Written to the
file!");

fclose(f);

}
```

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
   int num;
   FILE *fptr;

   // use appropriate location if you are using MacOS or Linux
   fptr = fopen("C:\\program.txt","w");

   if(fptr == NULL)
   {
      printf("Error!");
      exit(1);
   }

   printf("Enter num: ");
   scanf("%d",&num);

   fprintf(fptr,"%d",num);
   fclose(fptr);

   return 0;
  }
```

Reading using fscanf()

```
#include <stdio.h>

void main() {

FILE *f;

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

fprintf(f, "Reading data from a file is a common feature of file handling.\n");

fclose(f);

char arr[50];

f = fopen("file.txt", "r");

while(fscanf(f, "%s", arr)!=EOF)

{

printf("%s ", arr);

}

fclose(f);

 }
```

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
   int num;
   FILE *fptr;

   if ((fptr = fopen("C:\\program.txt","r")) == NULL){
       printf("Error! opening file");

       // Program exits if the file pointer returns NULL.
       exit(1);
   }

   fscanf(fptr,"%d", &num);

   printf("Value read from file is n=%d", num);
   fclose(fptr);

   return 0;
}
```

```
#include<stdio.h>
struct emp {
char name[10] ;
int age;
};
void main ( ) {
    struct emp e;
    FILE *p;
    p =fopen("new1.txt","a");
    printf("Enter Name and Age:") ;
    scanf("%s %d" ,e.name , &e.age);
    fprintf( p,"%s %d",e.name,e.age) ;
    fclose(p);
}
```

You can also write structures data into a file using fprintf() functions

Reading and writing to a binary file

Functions `fread()` and `fwrite()` are used for reading from and writing to a file on the disk respectively in case of binary files.

Writing to a binary file

To write into a binary file, you need to use the `fwrite()` function. The functions take four arguments:

1. address of data to be written in the disk
2. size of data to be written in the disk
3. number of such type of data
4. pointer to the file where you want to write.

```
fwrite(addressData, sizeData, numbersData, pointerToFile);
```

Writing a binary file

```
#include <stdio.h>
#include <stdlib.h>

struct threeNum
{
   int n1, n2, n3;
};

int main()
{
   int n;
   struct threeNum num;
   FILE *fptr;

   if ((fptr = fopen("C:\\program.bin","wb")) == NULL){
       printf("Error! opening file");

       // Program exits if the file pointer returns NULL.
       exit(1);
   }

   for(n = 1; n < 5; ++n)
   {
      num.n1 = n;
      num.n2 = 5*n;
      num.n3 = 5*n + 1;
      fwrite(&num, sizeof(struct threeNum), 1, fptr);
   }
   fclose(fptr);

   return 0;
}
```

Reading a binary file

```
#include <stdio.h>
#include <stdlib.h>

struct threeNum
{
   int n1, n2, n3;
};

int main()
{
   int n;
   struct threeNum num;
   FILE *fptr;

   if ((fptr = fopen("C:\\program.bin","rb")) == NULL){
       printf("Error! opening file");

       // Program exits if the file pointer returns NULL.
       exit(1);
   }

for(n = 1; n < 5; ++n)
   {
      fread(&num, sizeof(struct threeNum), 1,
fptr);
      printf("n1: %d\tn2: %d\tn3: %d\n", num.n1,
num.n2, num.n3);
   }
   fclose(fptr);

   return 0;
}
```

Some Problems

1. C program to read name and marks of n number of students and store them in a file.

2. C program to read name and marks of n number of students from and store them in a file. If the file previously exits, add the information to the file.

3. C program to write all the members of an array of structures to a file using fwrite(). Read the array from the file and display on the screen.

```
// Question 1.
#include <stdio.h>
#include<stdlib.h>
int main()
{
   char name[50];
   int marks, i, num;

   printf("Enter number of students: ");
   scanf("%d", &num);

   FILE *fptr;
   fptr = (fopen("C:\\student.txt", "w"));
   if(fptr == NULL)
   {
       printf("Error!");
       exit(1);
   }
for(i = 0; i < num; ++i)
   {
      printf("For student%d\nEnter name: ", i+1);
      scanf("%s", name);

      printf("Enter marks: ");
      scanf("%d", &marks);

      fprintf(fptr,"\nName: %s \nMarks=%d \n", name, marks);
   }

   fclose(fptr);
}
```

```c
// Question 2.
#include <stdio.h>
#include<stdlib.h>
int main()
{
   char name[50];
   int marks, i, num;

   printf("Enter number of students: ");
   scanf("%d", &num);

   FILE *fptr;
   fptr = (fopen("C:\\student.txt", "a"));
   if(fptr == NULL)
   {
       printf("Error!");
       exit(1);
   }

for(i = 0; i < num; ++i)
   {
      printf("For student%d\nEnter name: ", i+1);
      scanf("%s", name);

      printf("Enter marks: ");
      scanf("%d", &marks);

      fprintf(fptr,"\nName: %s \nMarks=%d \n", name, marks);
   }

   fclose(fptr);
   return 0;
}
```

```c
// Question 3.
#include <stdio.h>
struct student
{
   char name[50];
   int height;
};
int main(){
    struct student stud1[5], stud2[5];
    FILE *fptr;
    int i;

    fptr = fopen("file.txt","wb");
    for(i = 0; i < 3; ++i)
    {
        printf("Enter name: ");
        scanf("%s",stud1[i].name);

        printf("Enter height: ");
        scanf("%d", &stud1[i].height);
    }

    fwrite(stud1, sizeof(stud1), 1, fptr);
    fclose(fptr);

    fptr = fopen("file.txt", "rb");
    fread(stud2, sizeof(stud2), 1, fptr);
    for(i = 0; i < 3; ++i)
    {
        printf("\nName: %s\tHeight: %d\n", stud2[i].name,
stud2[i].height);
    }
    fclose(fptr);
}
```

Some functions used in file handling

- fseek(): it sets the pointer associated with a stream/file handle to a new position. This function is used to move the file pointer to different positions.

  The syntax is:

  fseek(fptr, offset, mode);

  where, fptr is the file pointer

  Offset is the integer that specifies the number of bytes by which file pointer is moved

  mode specifies from which position the offset is measured. Its value may be 0,1, or 2 which represents beginning of the file (SEEK_SET), current position of the file (SEEK_CUR), and end of the file (SEEK_END) respectively.

Some functions used in file handling

Examples:

fseek(fptr, 0, SEEK_SET) - It moves file pointer at the beginning of the file

fseek(fptr, 0, SEEK_END) - It moves file pointer at the end of the file

fseek(fptr, 10, SEEK_SET) - It moves file pointer at 10 bytes right from the beginning of the file.

fseek(fptr, -2, SEEK_CUR) - It moves file pointer at 2 bytes left from the current position

Some functions used in file handling

- rewind(): this function positions the file pointer in the beginning of the file. The use of the function rewind() is equivalent of using fseek(fptr, 0, SEEK_SET)

- ftell(): This function determines the current location of the file pointer within the file. It returns an integer value and also can be used to get file size, if file pointer is located at the end of the file. The syntax of the function is:

  ftell(fptr); where fptr is the file pointer for the currently opened file.