forbeginners.c tutorials.out

CSE1062 | CCS1063 ‘Practicals’ {

[Fundamentals of Computer Programming]

< Tutorial Session 12 - File Handling >

}

C File Handling

A file is a container in computer storage devices used for storing data.

Why files are needed?

* When a program is terminated, the entire data is lost. Storing in a file will preserve your data even if the program terminates.
* If you have to enter a large number of data, it will take a lot of time to enter them all.
* However, if you have a file containing all the data, you can easily access the contents of the file using a few commands in C.
* You can easily move your data from one computer to another without any changes.

C File Handling...

A collection of data or information that are stored on a computer known as file.

A file is a collection of bytes stores on a secondary storage device.

There are four different type of files.

1. Data files
2. Text files
3. Program files
4. Directory files

C File Handling...

Different types of file store different types of information.

A file has a beginning and an end.

We need a marker to mark the current position of the file from the beginning(in terms if bytes) while reading and write operation, takes place on a file.

C File Handling...

Initially the marker is at the beginning of the file.

We can move the marker to any other position in the file.

The new current position can be specified as an offset from the beginning the file.

Types of Files

When dealing with files, there are two types of files you should know about:

1. Text files
2. Binary files

https://www.programiz.com/c-programming/c-file-input-output

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.

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 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

Working with files

When working with files, you need to declare a pointer of type file. This declaration is needed for communication between the file and the program.

```
FILE *fptr;
```

Opening a file - for creation and edit

Opening a file is performed using the fopen() function defined in the stdio.h header file.

The syntax for opening a file in standard I/O is:

```
ptr = fopen("fileopen","mode");
```

ex:-

```
fopen("E:\\cprogram\\newprogram.txt","w");
fopen("E:\\cprogram\\oldprogram.bin","rb");
```

Opening Modes in Standard I/O

Mode	Meaning of Mode	During Inexistence of file
`r`	Open for reading.	If the file does not exist, `fopen()` returns NULL.
`rb`	Open for reading in binary mode.	If the file does not exist, `fopen()` returns NULL.
`w`	Open for writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
`wb`	Open for writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.

Opening Modes in Standard I/O...

Mode	Meaning of Mode	During Inexistence of file
`a`	Open for append. Data is added to the end of the file.	If the file does not exist, it will be created.
`ab`	Open for append in binary mode. Data is added to the end of the file.	If the file does not exist, it will be created.
`r+`	Open for both reading and writing.	If the file does not exist, `fopen()` returns NULL.
`rb+`	Open for both reading and writing in binary mode.	If the file does not exist, `fopen()` returns NULL.

https://www.programiz.com/c-programming/c-file-input-output

Opening Modes in Standard I/O...

Mode	Meaning of Mode	During Inexistence of file
`w+`	Open for both reading and writing.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
`wb+`	Open for both reading and writing in binary mode.	If the file exists, its contents are overwritten. If the file does not exist, it will be created.
`a+`	Open for both reading and appending.	If the file does not exist, it will be created.
`ab+`	Open for both reading and appending in binary mode.	If the file does not exist, it will be created.

Closing a File

The file (both text and binary) should be closed after reading/writing.

Closing a file is performed using the fclose() function.

```
fclose(fptr);
```

Here, fptr is a file pointer associated with the file to be closed.

Reading and writing to a text file

For reading and writing to a text file, we use the functions fprintf() and fscanf().

They are just the file versions of printf() and scanf(). The only difference is that fprintf() and fscanf() expects a pointer to the structure FILE.

Example 1: Write to a text file

This program takes a number from the user and stores in the file program.txt.

After you compile and run this program, you can see a text file program.txt created in C drive of your computer. When you open the file, you can see the integer you entered.

```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;
 }
```

https://www.programiz.com/c-programming/c-file-input-output

Example 2: Read from a text file

This program reads the integer present in the program.txt file and prints it onto the screen.

If you successfully created the file from Example 1, running this program will get you the integer you entered.

Other functions like fgetchar(), fputc() etc. can be used in a similar way.

```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 of n=%d", num);
   fclose(fptr);

   return 0;
  }
```

https://www.programiz.com/c-programming/c-file-input-output

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);
```

Example 3:

Write to a binary file using fwrite()

```c
#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 from a binary file

Function fread() also take 4 arguments similar to the fwrite() function as above.

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

Example 4:

Read from a binary file using fread()

```c
  #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;
  }
```

Getting data using fseek()

If you have many records inside a file and need to access a record at a specific position, you need to loop through all the records before it to get the record.

This will waste a lot of memory and operation time. An easier way to get to the required data can be achieved using fseek().

As the name suggests, fseek() seeks the cursor to the given record in the file.

```
fseek(FILE * stream, long int offset, int whence);
```

Different whence in fseek()

Whence	Meaning
SEEK_SET	Starts the offset from the beginning of the file.
SEEK_END	Starts the offset from the end of the file.
SEEK_CUR	Starts the offset from the current location of the cursor in the file.

Example 5: fseek()

This program will start reading the records from the file program.bin in the reverse order (last to first) and prints it.

```c
#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);
   }
// Moves the cursor to the end of the file
   fseek(fptr, -sizeof(struct threeNum), SEEK_END);

   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);
      fseek(fptr, -2*sizeof(struct threeNum),
SEEK_CUR);
   }
   fclose(fptr);
   return 0;
  }
```

```
Thanks; {
    ‘Do you have any questions?’

        < bgamage@sjp.ac.lk >

}
```