Andhra Pradesh State Skill Development Corporation

Skill AP
APSSDC

Programming in C

FILES

FILES

File:

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

Purpose of files in C language:

- 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.

Types of Files:

1. Text files
2. Binary files

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 take up bigger storage space.

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 provide 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");
```

Example:

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

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

- Let's suppose the file newprogram.txt doesn't exist in the location E:\cprogram. The first function creates a new file named newprogram.txt and opens it for writing as per the mode 'w'.
  The writing mode allows you to create and edit (overwrite) the contents of the file.

- Now let's suppose the second binary file oldprogram.bin exists in the location E:\cprogram. The second function opens the existing file for reading in binary mode 'rb'.
  The reading mode only allows you to read the file, you cannot write into the file.

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.
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.
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 fprint() and fscanf() expects a pointer to the structure FILE.

Example for write to a text file

```
#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;
}
```

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.

Example 2: Read from a text file

```
#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;
}
```

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.

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.

Syntax of seek():

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

The first parameter stream is the pointer to the file. The second parameter is the position of the record to be found, and the third parameter specifies the location where the offset starts.