Files in 'C'

Developed by www.futureforyou.net & www.eshikshak.co.in

Introduction

- A collection of data or information that are stored on a computer known as file
- A file is a collection of bytes stored on a secondary storage device.
- There are four different types of file
  - Data files
  - Text files
  - Program files
  - Directory files
- Different types of file store different types of information

Introduction

- 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 of bytes) while reading and write operation, takes place on a file.
- 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.

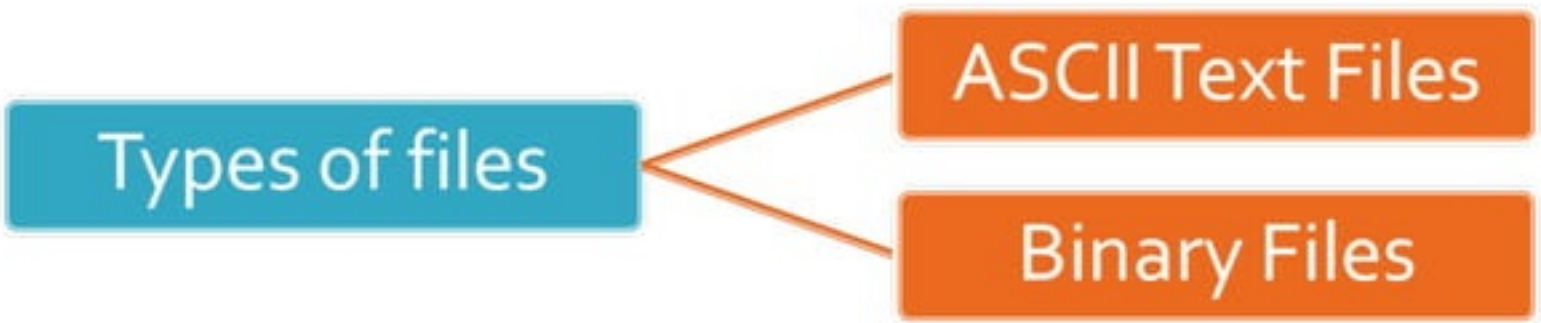
File streams

- A stream refers to the flow of data (in bytes) from one place to another (from program to file or vice-versa).
- There are two types of streams
  - Text Stream
    - It consists of sequence of characters
    - Each line of characters in the stream may be terminated by a newline character.
    - Text streams are used for textual data, which has a consistent appearance from one environment to another or from one machine to another

File streams

- Binary Stream
  - It is a series of bytes.
  - Binary streams are primarily used for non-textual data, which is required to keep exact contents of the file.

Types of files



ASCII Text File

- A text file can be a stream of characters that a computer can process sequentially.
- It is processed only in forward direction.
- It is opened for one kind of operation (reading, writing, or appending) at any give time.
- We can read only one character at a time from a text file.

Binary Files

- A binary file is a file consisting of collection of bytes.
- A binary file is also referred to as a character stream

Usage of file management functions

fopen()	• Creates a new file for use. • Opens new existing file for use
fclose()	• Closes a file which has been opened for use.
getc()	• Reads a character from a file.
puts()	• Writes a character to a file.
fprintf()	• Writes a set of data values to a file
fscanf()	• Reads a set of data values from a file

Usage of file management functions

getw()	• Reads a integer from a file.
putw()	• Writes an integer to the file.
fseek()	• Sets the position to a desired point in the files
ftell()	• Gives the current position in the file
rewind()	• Sets the position to the beginning of the file

Naming a file

- A file is identified by its name.
- This name is divided into two parts
  - File Name
    - It consists of alphabets and digits.
    - Special characters are also supported, but it depends on the operating system we use.
  - Extension
    - It describes the file type

Opening a file

- Before opening a file, we need to declare a file pointer. A file pointer is a pointer variable of type FILE which is defined in the "stdio.h" header file.
  - Syntax
    - FILE *fptr;
- A file pointer has the complete information about file being opened and processed such as:
  - Name of file, mode it is opened in, starting buffer address, a character pointer that points to the character being read.

Opening a file

- To perform any operation (read or write) on a file, the file must be brought into memory from the storage device (hard disk).
- This process of bringing the copy of the file from disk (secondary storage) to memory (main storage) is called opening the file.

Mode to use file

Mode	Meaning
r	▪ Open a text file for reading only. If the file doesn't exist, it returns null.
w	▪ Opens a file for writing only. ▪ If file exists, than all the contents of that file are destroyed and new fresh blank file is copied on the disk and memory with same name ▪ If file dosen't exists, a new blank file is created and opened for writing. ▪ Returns NULL if it is unable to open the file
a	▪ Appends to the existing text file ▪ Adds data at the end of the file. ▪ If file doesn't exists then a new file is created. ▪ Returns NULL if it is unable to open the file.
rb	▪ Open a binary file for reading
wb	▪ Open a binary file for reading
ab	▪ Append to a binary file
r+	▪ Open a text file for read/write
w+	▪ Opens the existing text file or Creates a text file for read/write

Mode to use file

Mode	Meaning
a+	▪ Append or create a text file for read/write
r+b	▪ Open a binary file for read/write
w+b	▪ Create a binary file for read/write
a+b	▪ Append a binary file for read/write

Opening a file

Syntax

- FILE *fopen(char *fname, char *mode)

- fopen() function, like all the other file-system functions, uses the head file stdio.h
- The name of the file to be opened is pointed to by *fname*
- The string given as the second parameter - for mode, determines how the file should be accessed (r-read, w-write a-append).

Opening a file

```
FILE *fp;
if(fp = fopen(“myfile”,”r”)) == NULL)
{
  printf(“Error opening a file”);
     exit(1);
}
```

Reading a file

- To read contents from an existing file, we need to open that file in read mode that means "r" mode
- Algorithm to read data from a file:

1. Open the file in read mode
2. Read data from the file
3. Write the data into an output device
4. Repeat steps 3 and 4 untill the end of file occurs
5. Stop procedure

Reading a file

```
#include<stdio.h>
void main()
{
   FILE *fp;
  char ch;
   fp=fopen("clear.c","r");
  if(fp==NULL)
        print("Unable to open clear.c");
   else
   {
       do
       {
              ch = getc(fp);
              putchar(ch);
       }while(ch!=EOF);
      fclose(fp);
   }
}
```

EOF (End of File)

- Generally, a file contains a large amount of data.
- In a large file, it is difficult to detect the end of file while reading.
- In order to mark the end of a text file, a special character EOF is stored at the end.

Closing a file

Syntax

- fclose(FILE *fp);

- To close a file and dis-associate it with a stream (file pointer), use fclose() function.
- fclose() returns o if the file is closed successfully
- The **fcloseall()** closes all the files opened previously.

Reading a file

```
#include<stdio.h>
void main()
{
   FILE *fp;
  char ch;
   fp=fopen("clear.c","r");
  if(fp==NULL)
        print("Unable to open clear.c");
   else
   {
       do
       {
              ch = getc(fp);// gets the character from file
              putchar(ch);
       }while(ch!=EOF);
     fclose(fp);
   }
}
```

Appending files

- We can add contents to an existing file whenever required.
- Perform the following steps to append an existing file:
  1. Declare a file pointer
  2. Open the file in append mode
  3. Read data from the keyboard
  4. Write it into the file
  5. Repeat steps 3 and 4 according to until the user gives input
  6. Stop the process

String I/O Functions in Files

- char *fgets(char *str, int n, FILE *fptr);
  - This function reads a character from the file stream pointed by fptr and stores it in the character array 'str' until a new line character (\n) is read or end of file (EOF) is reached or n-1 characters have been read.

- fputs(const char *str, FILE *fptr);
  - This function writes data to the stream pointed to by fptr, the content of the string stored in 'str'

Reading a file

```
#include<stdio.h>
void main()
{
   FILE *fp;
  char line[280];int ch, i=0;
   fp=fopen("a.dat","a");
  if(fp==NULL)
       print("Unable to open clear.c");
   else{
       do{
                do{
                        line[i++] = getchar();
                }while(line[i-1]!='*');
                fputs(line, fp);//Writes string to the file
                i=0;
                printf("\nPress 1 to continue");
                scanf("%d",&ch);
       }while(ch==1);
     fclose(fp);
      printf("File is successfully created");
   }
}
```

Modifying files

- A file can be accessed in two ways:
  - Serial access
  - Random access
- Generally all the text files are considered to be sequential files because lines of text (also called records) are stored in a file
- The beginning of each record in a sequential file is unpredictable
- Whereas, in random access files, all the records are in same length.

Modifying files

- To modify the content of a file, open the file with read and write mode ("r+" or "w+" or "a+")
- Generally "r+" mode is used for both reading and writing operation. The procedure is as follows:
  1. Initialize a pointer variable
  2. Open the file in read and write mode
  3. Read data from file and Print it
  4. Move the file pointer to the place where we have the data to be modified and re-write the new data in that place.
  5. Repeat steps 3 and 4 till the end of file reaches.
  6. Stop the Process

Modifying a file

```
#include<stdio.h>
struct stock
{
   int itid, qty;
   char n[100];
   float rate;
}it;

void main()
{
   FILE *fp; int ch; int r = 0;
   fp = fopen("item.c", "r+");
   if(fp==NULL)
   {
        printf("Unable to open item.c");
   }
}
```

Modifying a file

```
else{
   do{
          fread(&it, sizeof(it),1,fp);
          printf("\n%d %s %d %f", it.itid, it.n, it.qty, it.rate);
          printf("\n Press 1 to change it?");
          scanf("%d",&ch);

          if(ch==1)
          {
                 printf("\n Enter Itemid ItemName Quantity & Price");
                 scanf("%d%s%d%f", &it.itid, it.n, &it.qty, &it.rate);
                 fseek(fp, r*sizeof(it), 0);
                 fwrite(&it, sizeof(it),1,fp);
          }r++;
   }while(!feof(fp));
   fclose(fp);
}
}
```

String I/O functions in Files

fseek(FILE *fptr, long offset, int reference)

- Moves the pointer from one record to another.
- The first argument is the file pointer
- The second argument tells the compiler how many bytes (offset) the pointer should be moved from a particular position.
- The third argument is the reference from where the pointer should be moved n bytes (specified in the offset).

String I/O functions in Files

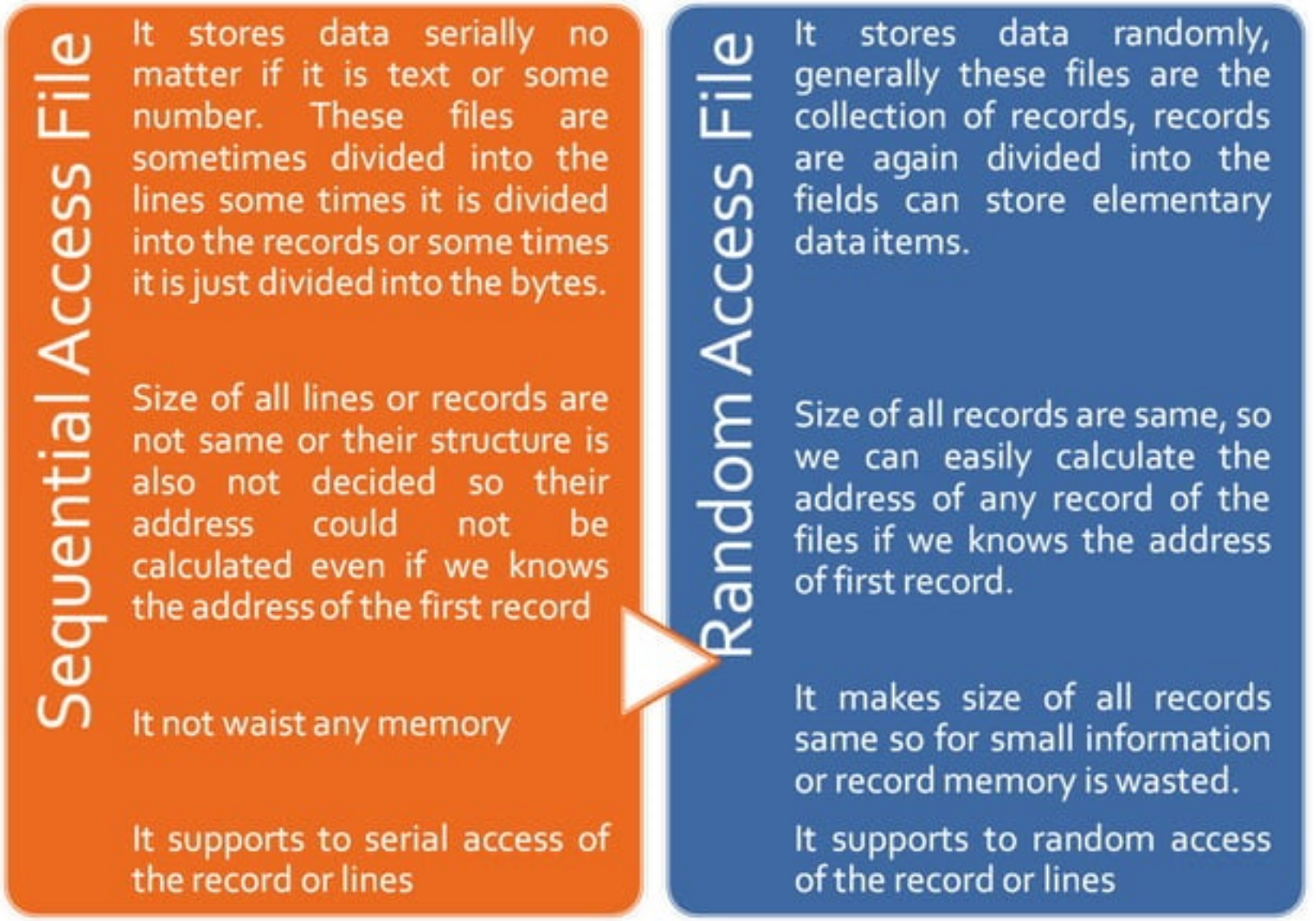
fseek(FILE *fptr, long offset, int reference)

- The third argument is the reference from where the pointer should be moved n bytes as specified in the offset.
  - SEEK_END moves the pointer from the end marker
  - SEEK_CUR moves the pointer from the current position
  - SEEK_SET moves the pointer from the beginning of the file

String I/O functions in Files

fseek(FILE *fptr, long offset, int reference)

- Example
  - fseek(fptr, size, SEEK_CUR) sets the cursor ahead from current position by size bytes
  - fseek(fptr, -size, SEEK_CUR) sets the cursor back from current position by size bytes
  - fseek(fptr, 0, SEEK_END) sets cursor to the end of the file
  - fseek(fptr, 0, SEEK_SET) sets cursor to the beginning of the file


Sequential Access File
It stores data serially no matter if it is text or some number. These files are sometimes divided into the lines some times it is divided into the records or some times it is just divided into the bytes.
Size of all lines or records are not same or their structure is also not decided so their address could not be calculated even if we knows the address of the first record
It not waist any memory
It supports to serial access of the record or lines
Random Access File
It stores data randomly, generally these files are the collection of records, records are again divided into the fields can store elementary data items.
Size of all records are same, so we can easily calculate the address of any record of the files if we knows the address of first record.
It makes size of all records same so for small information or record memory is wasted.
It supports to random access of the record or lines

Sequential Access File

It is slow because of serial access

It supports to non-contiguous allocation of the file.

It gives better performance for linked allocation of the file

It was used by early operating system

For accessing these files it will not required any extra software overhead, because these files are not having any special structure.

Example : text files, machine language files or some data base files

The fgets, fputs, fscanf, fprintf, getc, putc functions are used in the C language

Random Access File

It is fast because of random access

It supports the contiguous allocation of the file as well as linked allocation of the file

It gives poor performance for the linked allocations of the file

Now all modern operating system uses these techniques

For accessing these files it will required special software who knows the structure of that file. There are many kinds of the structure requires their own software which can knows that structure and access the data.

Example : text files, picture files images animation or data base files

The fwrite, fread, fseek functions are used in the c languages.