

Unit 10

Files I/O in C

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 few commands in C.
- You can easily move your data from one computer to another without any changes.

Types of Files

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

1. Text files

- Text files are the normal .txt files that you can easily create using Notepad or any simple text editors.
- 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 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 higher amount of data, are not readable easily and provides a better security than text files.

File Operations

- In C, you can perform four major operations on the file, either text or binary:
- Creating a new file
- Opening an existing file
- Reading from and writing information to a file
- Closing 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 program.
- `FILE *fptr;`

Opening a file - for creation and edit

- Opening a file is performed using the library function in the "**stdio.h**" header file: `fopen()`.
- The syntax for opening a file in standard I/O is:
 - `ptr = fopen("fileopen","mode")`
- For 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.

Opening Modes in Standard I/O

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. i.e, Data is added to end of file.	If the file does not exist, it will be created.
ab	Open for append in binary mode. i.e, Data is added to end of file.	If the file does not exist, it will be created.

File Mode	Meaning of Mode	During Inexistence of file
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 library function `fclose()`.
- `fclose(fptr);` // `fptr` is the file pointer associated with file to be closed.

Functions for file handling

No.	Function	Description
1	fopen()	opens new or existing file
2	fclose()	closes the file
3	fprintf()	write formatted data into the file
4	fscanf()	reads formatted data from the file
5	fputc()	writes a character into the file
6	fgetc()	reads a character from file
7	fputs()	Writes a string to a file
8	fgets()	Reads a string from a file
9	putw()	writes an integer to file
10	getw()	reads an integer from file
11	fread()	It is used to read a block at a time
12	fwrite()	It is used to write a block at a time

No.	Function	Description
9	fseek()	sets the file pointer to given position
12	ftell()	returns current position
13	rewind()	sets the file pointer to the beginning of the file
14	feof()	Returns true if end of file is reached
15	ferror	Returns true if error has occurred
16	fflush()	Flushes a file
17	remove()	Erases a file

Writing a character in a file

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

```
void main()
```

```
{
```

```
    FILE *fp;
```

```
    char c;
```

```
    printf("enter a character");
```

```
    scanf(" %c",&c);
```

```
    fp = fopen("sample.txt", "w");
```

```
    if(fp==NULL){
```

```
        printf("Error while opening file");
```

```
        exit(0);
```

```
    }
```

```
    fputc(c,fp);
```

```
    fclose(fp);}
```

Read a character from a file

```
#include<stdio.h>
void main()
{
    FILE *fp;
    char c;
    fp = fopen("sample.txt", "r");
    printf("\ncontents of files are\n");
    c=fgetc(fp);
    printf("%c",c);
    fclose(fp);
}

while(c!=EOF)
{
    c=fgetc(fp);
    printf("%c",c);
}
```


Writing a string in file

```
#include<stdio.h>
void main()
{
    FILE *fp;
    int result;
    char c[100];
    printf("enter a line");
    gets(c);
    fp = fopen("source.txt", "w");
    fputs(c,fp);
    fclose(fp);
}
```

Copying from source to destination

```
#include<stdio.h>
void main()
{
    FILE *fp1,*fp2;
    char c;
    fp1 = fopen("source.txt", "r");
    fp2 = fopen("dest.txt", "w");
    if(fp1==NULL || fp2==NULL)
    {
        printf("Error while opening file");
        exit(0);
    }
```

```
    printf("\ncopying content of files\n");  
while((c=fgetc(fp1))!=EOF)  
{  
    fputc(c,fp2);  
}  
fclose(fp1);  
fclose(fp2);  
}
```

Read String from file

```
#include<stdio.h>
void main()
{
    FILE *fp;
    int result;
    char c[100];
    fp = fopen("dest.txt", "r");
    fgets(c,100,fp);

    printf("%s",c);
    fclose(fp);
}
```

Read and write formatted data

```
#include<stdio.h>

void main()
{
    FILE *fp;
    int x,a=10;
    float y,b=5.5;
    char z,c='M';
    fp = fopen("form.txt", "w");
    fprintf(fp,"%d %f %c\n",a,b,c);
    fclose(fp);

    fp=fopen("form.txt", "r");
    fscanf(fp,"%d %f %c\n",&x,&y,&z);
    printf("\n%d \n%f\n%c",x,y,z);}
```

Write and Read int into file

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
void main()
{
    FILE *fp;
    int x,y;
    fp = fopen("integ.txt", "a");
    printf("enter a number");
    scanf("%d",&x);
    putw(x,fp);
    fclose(fp);
```

```
fp = fopen("integ.txt", "r");
    y=getw(fp);
    while(y!=-1)
    {
        printf("%d",y);
        y=getw(fp);

    }

}
```

Write a block into a file

```
#include<stdio.h>

struct student
{
    int roll;
    char name[50];
};

void main()
{
    FILE *fp;
    struct student s1;
```



```
printf("enter your roll no");  
scanf("%d",&s1.roll);  
printf("enter your name");  
fflush(stdin);  
gets(s1.name);
```

```
fp=fopen("a.txt","w");  
fwrite(&s1,sizeof(s1),1,fp);  
fclose(fp);
```

```
}
```

Read a block from file

```
#include<stdio.h>

struct student
{
    int roll;
    char name[50];
};

void main()
{
    FILE *fp;
    struct student s1;
```

```
fp=fopen("a.txt","r");  
fread(&s1,sizeof(s1),1,fp);
```

```
printf("\nroll no is %d",s1.roll);  
printf("\n name is %s",s1.name);
```

```
fclose(fp);
```

```
}
```

Writing a Block into a file

```
#include<stdio.h>

struct employee
{
    char name[20];
    int age;
    float salary;
};

void main()
{
    struct employee emp;
    FILE *fp;
    char c;
    fp = fopen("student.txt", "w");
```

```
do
{
    printf("\nenter name");
    fflush(stdin);
    gets(emp.name);
    printf("\nenter age");
    scanf("%d",&emp.age);
    printf("\nenter salary");
    scanf("%f",&emp.salary);
    fwrite(&emp,sizeof(emp),1,fp);
    printf("Want to add another record (y/n)");
    scanf(" %c",&c);
}while(c=='y');
fclose(fp);
}
```

Reading a block from a file

```
#include<stdio.h>
struct employee
{
    char name[20];
    int age;
    float salary;
};
void main()
{
    struct employee emp;
    FILE *fp;
```

```
fp = fopen("student.txt", "r");
while(fread(&emp,sizeof(emp),1,fp))
{
    printf("\nname is %s\nage is %d\nsalary
is%f",emp.name,emp.age,emp.salary);
    printf("\n*****");
}
fclose(fp);
}
```

Program to create a file named “university.dat”.
Write a program to keep the records of N colleges under Tribhuwan University. These records contain name, location and no_of_faculties of the college and display the names of colleges in kathmandu location


```
#include<stdio.h>
struct college
{
    char name[20];
    char location[20];
    int faculty_no;
};
void main()
{
    struct college col;
    FILE *fp;
    char c;
    fp = fopen("university.dat", "wb+");
```

```
do
{
    printf("\nenter college name");
    fflush(stdin);
    gets(col.name);
    printf("\nenter college location");
    fflush(stdin);
    gets(col.location);
    printf("\nenter no of faculties");
    scanf("%d",&col.faculty_no);
    fwrite(&col,sizeof(col),1,fp);
    printf("Want to add another record (y/n)");
    scanf(" %c",&c);
}while(c=='y');
rewind(fp);
```

```
fread(&col,sizeof(col),1,fp);
printf("name of colleges in kathmandu");
while(!feof(fp))
{
    if(strcmp(col.location,"kathmandu")==0)
    {
        printf("\ncollege name is %s",col.name);
        printf("\n*****");
    }
    fread(&col,sizeof(col),1,fp);
}
fclose(fp);
}
```

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 function `fwrite()`. The function takes four arguments: Address of data to be written in disk, Size of data to be written in disk, number of such type of data and pointer to the file where you want to write.
- `fwrite(address_data,size_data,numbers_data,pointer_to_file);`

Example 3: Write to a binary file using fwrite()

```
#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");
    }
```

```
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 `fwrite()` function as above.
- `fread(address_data,size_data,numbers_data,pointer_to_file);`

Example 4: Read from a binary file using fread()

```
#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");
    }
```



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

Syntax of fseek()

- `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.

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.

Statement	Meaning
<code>fseek(fp,0,0)</code>	Go to the beginning
<code>fseek(fp,0,1)</code>	Stay at current position
<code>fseek(fp,0,2)</code>	Go to the end of file, past the last character of the file
<code>fseek(fp,m,0)</code>	Move to (m+1)th byte in the file
<code>fseek(fp,m,1)</code>	Go forward by m bytes
<code>fseek(fp,-m,1)</code>	Go backward by m bytes from the current position
<code>fseek(fp,-m,2)</code>	Go backward by m bytes from the end.

Example

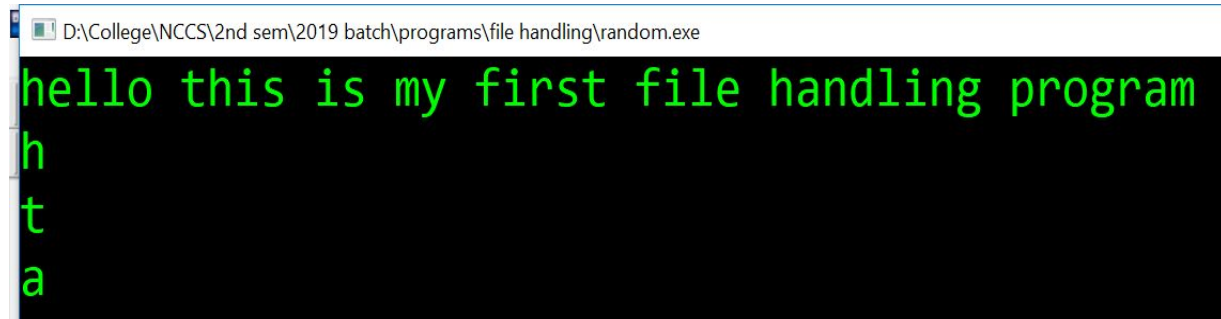
```
#include<stdio.h>
#include<conio.h>
void main()
{
    FILE *fp;
    char a,b[50];
    fp=fopen("dest.txt","r");
    fgets(b,50,fp);
    printf("%s",b);
    rewind(fp);
```

```
a=fgetc(fp);  
printf("\n%c",a);
```

```
fseek(fp,6,0);
```

```
a=fgetc(fp);  
printf("\n%c",a);
```

```
fseek(fp,-2,2);  
a=fgetc(fp);  
printf("\n%c",a);  
fclose(fp);  
}
```



The screenshot shows a Windows command prompt window with the title bar "D:\College\NCCS\2nd sem\2019 batch\programs\file handling\random.exe". The command prompt displays the output of the program in green text on a black background. The output consists of three lines: "hello this is my first file handling program", "h", and "t". The character "a" is visible on the next line, indicating the program has reached the end of the file.

```
hello this is my first file handling program  
h  
t  
a
```

```
#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");
        exit(1);
    }
```

```
// Moves the cursor to the end of the file
fseek(fp_ptr, -sizeof(struct threeNum), SEEK_END);
for(n = 1; n < 5; ++n)
{
    fread(&num, sizeof(struct threeNum), 1, fp_ptr);
    printf("n1: %d\tn2: %d\tn3: %d\n", num.n1, num.n2,
num.n3);
    fseek(fp_ptr, -2*sizeof(struct threeNum), SEEK_CUR);
}
fclose(fp_ptr);
return 0;
}
```