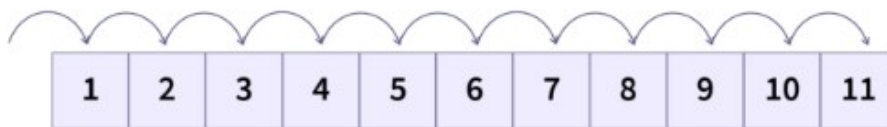


5.7 RANDOM ACCESSING FILES IN C LANGUAGE

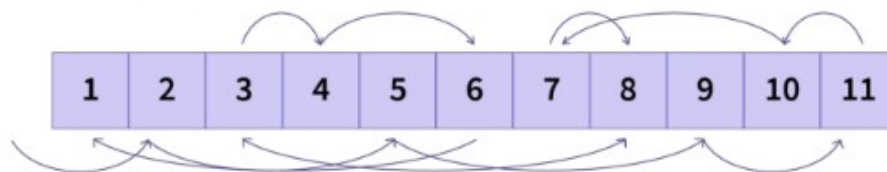
Random file access in C, enabling direct data reading or writing without processing all preceding data. Distinct from sequential access, it offers enhanced flexibility for data manipulation. Random access, ideal for large files, involves functions like `ftell()`, `fseek()`, and `rewind()`. This method, akin to choosing a song on a CD, requires more coding but offers superior efficiency and flexibility in file handling.

Sequential Access -



Access Order : 1 2 3 4 5 6 7 8 9 10 11

Random Access -



Access Order : 2 5 9 11 10 7 8 3 4 6 1

How to use the ftell() function in C

Highlights:

1. **ftell()** is used to find the position of the file pointer from the starting of the file.
2. Its syntax is as follows:

```
ftell(FILE *fp)
```



In C, the function **ftell()** is used to determine the file pointer's location relative to the file's beginning. **ftell()** has the following syntax:

```
pos = ftell(FILE *fp);
```



Where fp is a file pointer and pos holds the current position i.e., total bytes read (or written). For Example: If a file has 20 bytes of data and if the ftell() function returns 5 it means that 5 bytes have already been read (or written). Consider the below program to understand the ftell() function:

First, let us consider a file - Scaler.txt which contains the following data:

Scaler is amazing

```
#include<stdio.h>

int main()
{
    FILE *fp;
    fp=fopen("scaler.txt","r");
    if(!fp)
    {
        printf("Error: File cannot be opened\n") ;
        return 0;
    }
}
```



```
//Since the file pointer points to the starting of the file, ftell
printf("Position pointer in the beginning : %ld\n",ftell(fp));

char ch;
while(fread(&ch,sizeof(ch),1,fp)==1)
{
    //Here, we traverse the entire file and print its contents unt
    printf("%c",ch);
}

printf("\nSize of file in bytes is : %ld\n",ftell(fp));
fclose(fp);
return 0;
}
```

Output:

Position pointer in the beginning : 0

Scaler is amazing

Size of file in bytes is : 17

We can observe that in the beginning, ftell returns 0 as the pointer points to the beginning and after traversing completely we print each character of the file till the end, and now ftell returns 17 as it is the size of the file.

How to use the `rewind()` function in C

Highlights:

1. `rewind()` is used to move the file pointer to the beginning of the file.
2. Its syntax is as follows:

```
rewind(FILE *fp);
```

The file pointer is moved to the beginning of the file using this function. It comes in handy when we need to update a file. The following is the syntax:

```
rewind(FILE *fp);
```

Here, **fp** is a file pointer of type FILE. Consider the following program to understand the **rewind()** function:

```
#include<stdio.h>

int main()
{
    FILE *fp;
    fp = fopen("scaler.txt","r");
    if(!fp)
    {
        printf("Error in opening file\n");
        return 0;
    }
    //Initially, the file pointer points to the starting of the file.
```



```

printf("Position of the pointer : %ld\n",ftell(fp));

char ch;
while(fread(&ch,sizeof(ch),1,fp)==1)
{
    //Here, we traverse the entire file and print its contents unt
    printf("%c",ch);
}
printf("Position of the pointer : %ld\n",ftell(fp));

//Below, rewind() will bring it back to its original position.
rewind(fp);
printf("Position of the pointer : %ld\n",ftell(fp));

fclose(fp);
return 0;

```

}

Output:

```

Position of the pointer : 0
Scaler is amazing
Position of the pointer : 17
Position of the pointer : 0

```

We can observe that firstly when ftell is called, it returns 0 as the position of the pointer is at the beginning, and then after traversing the file, when ftell is called, 17 is returned, which is the size of the file. Now when rewind(fp) is called, the pointer will move to its original position, which is 0. So last ftell returns 0.

How to use the fseek() function in C

Highlights:

1. The **fseek()** function moves the file position to the desired location.
2. Its syntax is:

```
int fseek(FILE *fp, long displacement, int origin);
```



To shift the file position to a specified place, use the **fseek()** function.

Syntax:

```
int fseek(FILE *fp, long displacement, int origin);
```



The various components are as follows:

- **fp** – file pointer.
- **displacement** - represents the number of bytes skipped backwards or forwards from the third argument's location. It's a long integer that can be either positive or negative.
- **origin** – It's the location relative to the displacement. It accepts one of the three values listed below.

Constant	Value	Position
SEEK_SET	0	Beginning of file
SEEK_CURRENT	1	Current position
SEEK_END	2	End of file

Here is the list of common operations that we can perform using the **fseek()** function.

Here is the list of common operations that we can perform using the **fseek()** function.

Operation	Description
<i>fseek(fp, 0, 0)</i>	This takes us to the beginning of the file.
<i>fseek(fp, 0, 2)</i>	This takes us to the end of the file.
<i>fseek(fp, N, 0)</i>	This takes us to (N + 1)th bytes in the file.
<i>fseek(fp, N, 1)</i>	This takes us N bytes forward from the current position in the file.

<code>fseek(fp, -N, 1)</code>	This takes us N bytes backward from the current position in the file.
<code>fseek(fp, -N, 2)</code>	This takes us N bytes backward from the end position in the file.

Let us see the below program to understand the `fseek()` function:

```
#include<stdio.h>

int main()
{
    FILE *fp;
    fp = fopen("scaler.txt","r");
    if(!fp)
    {
        printf("Error: File cannot be opened\n");
        return 0;
    }
    //Move forward 6 bytes, thus we won't be seeing the first 6 bytes
    fseek(fp, 6, 0);
    char ch;
    while(fread(&ch,sizeof(ch),1,fp)==1)
    {
        //Here, we traverse the entire file and print its contents unt
        printf("%c",ch);
    }

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

Output:

is amazing

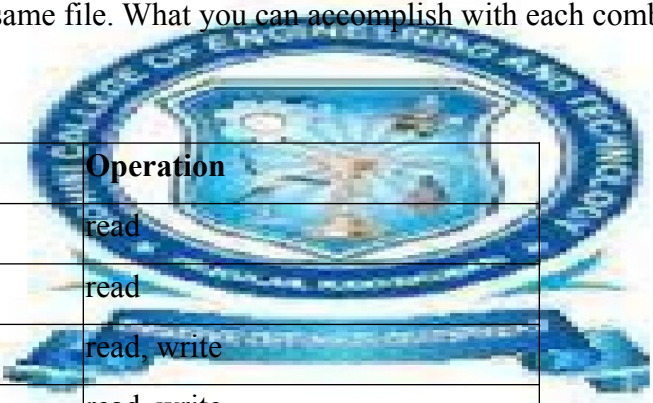
We can observe that when `fseek(fp,6,0)` the pointer moves to the 7th byte in the file, or we can say 6 bytes forward from the beginning. So when we traverse the file from that position, we receive output as is amazing.

File Mode Combinations

Highlights:

File Mode combinations allow us to accomplish reading and writing operations simultaneously.

In general, you can only read from or write to a text file, not simultaneously. A binary file allows you to read and write to the same file. What you can accomplish with each combination is shown in the table below:



Combination	Type of File	Operation
r	text	read
rb+	binary	read
r+	text	read, write
r+b	binary	read, write
rb+	binary	read, write
w	text	write, create, truncate
wb	binary	write, create, truncate
w+	text	read, write, create, truncate
w+b	binary	read, write, create, truncate
wb+	binary	read, write, create, truncate
a	text	write, create
ab	binary	write, create
a+	text	read, write, create
a+b	binary	write, create

Combination	Type of File	Operation
ab+	binary	write, create

Creating a Random-Access File

Functions like `fopen()` can be used to create files if they do not exist.

Functions like `fopen()` can be used to create files if they do not exist. This can be seen in the example below:

```
#include<stdio.>
intmain()
{
    char ch;
    //file pointer
    FILE *fp;
    // open and creates file in write mode if it does not exist.
    fp = fopen("char", "w");
    if (fptr != NULL)
    {
        printf("File created successfully!\n");
    }
    else
    {
        printf("Failed to create the file.\n");
        return 0;
    }
    fclose(fp)
    return 0;
}
```

Writing Data Randomly to a Random-Access File

The program writes data to the file "student.txt". It stores data at precise points in the file using a mix of fseek() and fwrite(). The file position pointer is set to a given place in the file by fseek(), and then the data is written by fwrite(). Let us see the code below:

```
#include<stdio.h>

// Student structure definition
struct Student {
    char name[20]; // student name
    int roll_number; // roll number
};

int main()
{
    FILE *fp; // file pointer
    // The below line creates a student object with default values
    struct Student s = {"", 0};
    // fopen opens the file, and exits if file cannot be opened
    if (!(fp = fopen( "student.txt", "r+" )))
    {
        printf("File cannot be opened.");
        return 0;
    }

    // The user will enter information which will be copied to the file
    while(1)
    {
        // require the user to specify roll number
        printf("Enter roll number from (1 to 100) , -1 to end input : ");
        scanf("%d",&s.roll_number);
        if(s.roll_number == -1)
            break;
```

```
// require the user to specify name
printf("Enter name : ");
scanf("%s",s.name);
fseek(fp,(s.roll_number-1)*sizeof(s),0);
fwrite(&s, sizeof(s), 1, fp);
    }
fclose(fp); //fclose closes the file
    return 0;
}
```

Output:

```
Enter roll number from (1 to 100) , -1 to end input : 1
Enter name : Scaler
Enter roll number from (1 to 100) , -1 to end input : 10
Enter name :Aaradhya
Enter roll number from (1 to 100) , -1 to end input : -1
```

