

UNIT V

FILE PROCESSING : Files – Types of file processing: Sequential access, Random access – Sequential access file - Random access file - Command line arguments.

FILES

Most of the programs we have seen so far are **transient** in the sense that they run for a short time and produce some output, but when they end, their data disappears. If you run the program again, it starts with a clean slate.

Other programs are **persistent**: they run for a long time, they keep at least some of their data in permanent storage, and if they shut down and restart, they pick up where they left off.

One of the simplest ways for programs to maintain their data is by reading and writing text files. An alternative is to store the state of the program in a database. In this section, we will discuss about files which are very important for storing information permanently.

What is a File?

Abstractly, a file is a collection of bytes stored on a secondary storage device, which is generally a disk of some kind. The collection of bytes may be interpreted, for example, as characters, words, lines, paragraphs and pages from a textual document; fields and records belonging to a database; or pixels from a graphical image. The meaning attached to a particular file is determined entirely by the data structures and operations used by a program to process the file. A file is simply a machine decipherable storage media where programs and data are stored for machine usage. Essentially there are two kinds of files that programmers deal with text files and binary files. These two classes of files will be discussed in the following sections.

- **ASCII Text files**
- **Binary files**

ASCII Text files

A text file can be a stream of characters that a computer can process sequentially. It is not only processed sequentially but only in forward direction. For this reason a text file is usually opened for only one kind of operation (reading, writing, or appending) at any given time. Similarly, since text files only process characters, they can only read or write data one character at a time. (In C Programming Language, Functions are provided that deal with lines of text, but these still essentially process data one character at a time.) A text stream in C is a special kind of file. Depending on the requirements of the operating system, newline characters may be converted to or from carriage-return/linefeed combinations depending on whether data is being written to, or read from, the file. Other character conversions may also occur to satisfy the storage requirements of the operating system. These translations occur transparently and they occur because the programmer has signaled the intention to process a text file.

Binary files

A binary file is no different to a text file. It is a collection of bytes. In C Programming Language a byte and a character are equivalent. Hence a binary file is also referred to as a character stream, but there are two essential differences.

1. No special processing of the data occurs and each byte of data is transferred to or from the disk unprocessed.
2. C Programming Language places no constructs on the file, and it may be read from, or written to, in any manner chosen by the programmer.

Binary files can be either processed sequentially or, depending on the needs of the application, they can be processed using random access techniques. In C Programming Language, processing a file using random access techniques involves moving the current file position to an appropriate place in the file before reading or writing data. This indicates a second characteristic of binary files. They are generally processed using read and writes operations simultaneously. For example, a database file will be created and processed as a binary file. A record update operation will involve locating the appropriate record, reading the record into memory, modifying it in some way, and finally writing the record back to disk at its appropriate location in the file. These kinds of operations are common to many binary files, but are rarely found in applications that process text files.

File Operations

One can perform following operations regarding files:

1. Creating a new file
2. Opening an existing file
3. Reading from and writing information to a file
4. Closing a file

Working with file

In order to work with file, you need to declare a **pointer of type file**. This declaration is needed for communication between file and program.

```
FILE *fptr;
```

Opening a file

File can be opened with the help of **fopen()** function available in **stdio.h** library file of C

The **syntax** for opening a file is

```
FILE *fptr = fopen("fileName","mode")
```

Here, **fileName** is a string literal, it refers the name of the file to be open with its extension, if the file to be open is present in some other directory, then have to specify full path. **mode** can have one of the following option.

Table: File Opening Modes

File Mode	Meaning of Mode
-----------	-----------------

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.
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.

Example:

```
fptr=fopen("C:\\\\TURBOC3\\\\program.txt","w");
```

Closing a File

File should be closed after its usage. File can be closed using **fclose()** library function

The **syntax** for closing a file is

```
fclose(file_pointer);
```

Here, file_pointer is the one which was created when the file was opened using fopen() function.

Example:

```
fclose(fptr);
```

Text File Input Output (I/O)

In order to manipulate files we have to learn about File I/O i.e. how to write data into a file and how to read data from a file. To read and write file we use the functions **fprintf()** and **fscanf()**.

Functions `fprintf()` and `fscanf()` are the file version of `printf()` and `scanf()`. The only difference while using `fprintf()` and `fscanf()` is that, the first argument is a pointer to `FILE`.

`fprintf()` - Writing to file

Example:

```
#include <stdio.h>
void main()
{
    int n;
    FILE *fptr;
    fptr=fopen("C:\\TURBOC3\\sample.txt","w");
    if(fptr==NULL){
        printf("Cannot Open File!");
        exit(1);
    }
    printf("Enter a Number: ");
    scanf("%d",&n);
    fprintf(fptr,"%d",n);
    fclose(fptr);
}
```

This program reads a number from user and stores it in file. After you compile and run this program, you can see a text file `sample.txt` created in `C:\\TURBOC3\\` path of your computer. When you open that file, you can see the integer you entered. Similarly, `fscanf()` can be used to read data from file.

`fscanf()` - Reading from file

```
#include <stdio.h>
void main()
{
    int n;
    FILE *fptr;
    if ((fptr=fopen("C:\\TURBOC3\\sample.txt","r"))==NULL){
        printf("Cannot Open File ");
        exit(1);
    }
    fscanf(fptr,"%d",&n);
    printf("Value in file is=%d",n);
    fclose(fptr);
}
```

```
}
```

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

Other unformatted I/O functions like `fgetc()`, `fputc()`, etc.. are used to read and write data in files.

fputc(), fputs()

The function **fputc()** writes the character value of the argument `c` to the output stream referenced by `fp`. It returns the written character on success otherwise **EOF** if there is an error.

```
int fputc( int c, FILE *fp );
```

The function **fputs()** writes the string `s` to the output stream referenced by `fp`. It returns a non-negative value on success, otherwise **EOF** is returned in case of any error.

```
int fputs( const char *s, FILE *fp );
```

Example:

```
#include <stdio.h>
void main() {
    FILE *fptr;
    fptr = fopen("C:\\TURBOC3\\sample.txt", "w+");
    fputs("Printing Using fputs", fptr);
    fclose(fptr);
}
```

When the above program run, it creates a new file `sample.txt` in `C:\\TURBOC3\\` directory and writes the string "Printing Using fputs" to that file.

fgetc(), fgets()

The **fgetc()** function reads a character from the input file referenced by `fp`. The return value is the character read, or in case of any error, it returns **EOF**.

```
int fgetc( FILE *fp );
```

The functions **fgets()** reads up to `n-1` characters from the input stream referenced by `fp`. It copies the read string into the buffer **buf**, appending a **null** character to terminate the string.

If this function encounters a newline character '\n' or the end of the file EOF before they have read the maximum number of characters, then it returns only the characters read up to that point including the new line character.

```
char *fgets( char *buf, int n, FILE *fp );
```

Example:

```
#include <stdio.h>
void main()
{
    FILE *fptr;
    char buff[255];
    fptr = fopen("C:\\TURBOC3\\sample.txt", "r");
    fgets(buff, 255, fptr);
    printf("%s\n", buff );
    fclose(fptr);
}
```

When the above program run, it reads the file created in the previous program and produces the following result.

Printing Using fputs**Binary File I/O Functions**

If a large amount of numerical data is to be stored, text mode will be insufficient. In such case binary file is used.

Working of binary files is similar to text files with few differences in opening modes, reading from file and writing to file.

Opening modes of binary files are rb, rb+, wb, wb+,ab and ab+. The only difference between opening modes of text and binary files is that, b is appended to indicate that, it is binary file.

fread() and fwrite() - To read and write 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.

fwrite()

Function fwrite() 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.

Syntax:

```
fwrite(address_data,size_data,numbers_data,pointer_to_file);
```

Example :

```
#include <stdio.h>
struct marks
{
    int m1, m2,m3,m4,m5;
};
void main()
{
    int n;
    struct marks m;
    FILE *fptr;
    if ((fptr = fopen("C:\\TURBOC3\\mark.bin","wb")) == NULL){
        printf("File Cannot Open!");
        exit(1);
    }
    printf("Enter 5 Students Marks\\n");
    for(n = 1; n <= 5; ++n)
    {
        printf("Enter English Mark of Student %d : ", n);
        scanf("%d",&m.m1);
        printf("Enter Math's Mark of Student %d : ", n);
        scanf("%d",&m.m2);
        printf("Enter Physics Mark of Student %d : ", n);
        scanf("%d",&m.m3);
        printf("Enter Chemistry Mark of Student %d : ", n);
        scanf("%d",&m.m4);
        printf("Enter Python Mark of Student %d : ", n);
        scanf("%d",&m.m5);
        fwrite(&m, sizeof(struct marks), 1, fptr);
    }
    fclose(fptr);
}
```

Output:

```
Enter 5 Students Marks
Enter English Mark of Student 1 : 98
Enter Math's Mark of Student 1 : 89
Enter Physics Mark of Student 1 : 78
Enter Chemistry Mark of Student 1 : 98
Enter Python Mark of Student 1 : 70
Enter English Mark of Student 2 : 60
Enter Math's Mark of Student 2 : 89
Enter Physics Mark of Student 2 : 98
Enter Chemistry Mark of Student 2 : 76
Enter Python Mark of Student 2 : 98
Enter English Mark of Student 3 : 89
Enter Math's Mark of Student 3 : 78
Enter Physics Mark of Student 3 : 96
Enter Chemistry Mark of Student 3 : 56
Enter Python Mark of Student 3 : 78
Enter English Mark of Student 4 : 78
Enter Math's Mark of Student 4 : 96
Enter Physics Mark of Student 4 : 87
Enter Chemistry Mark of Student 4 : 96
Enter Python Mark of Student 4 : 97
Enter English Mark of Student 5 : 86
Enter Math's Mark of Student 5 : 79
Enter Physics Mark of Student 5 : 76
Enter Chemistry Mark of Student 5 : 88_
```

In this program, you create a new file mark.bin in C:\\TURBOC3\\ directory.

We declare a structure marks with five integers – m1, m2, m3, m4 and m5, and define it in the main function as m.

Now, inside the for loop, we read marks of 5 subjects and store the value into the file using fwrite.

The first parameter takes the address of m and the second parameter takes the size of the structure marks.

Since, we're only inserting one instance of m, the third parameter is 1. And, the last parameter *fptr points to the file we're storing the data.

Finally, we close the file.

fread()

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

Syntax:

```
fread(address_data,size_data,numbers_data,pointer_to_file);
```

Example:

```
#include <stdio.h>
struct marks
{
    int m1, m2,m3,m4,m5;
};
void main()
{
    int n;
```



```

struct marks m;
FILE *fptr;
if ((fptr = fopen("C:\\TURBOC3\\mark.bin","rb")) == NULL){
    printf("Cannot Open File !");
    exit(1);
}
printf("Marks are\n");
for(n = 1; n <= 5; ++n)
{
    fread(&m, sizeof(struct marks), 1, fptr);
    printf("Student %d Marks : English: %d\t Maths : %d\t Physics: %d\t Chemistry : %d\t\n", n, m.m1, m.m2, m.m3, m.m4, m.m5);
}
fclose(fptr);
}

```

Output:

```

Marks are
Student 1 Marks : English: 98    Maths : 89    Physics: 78    Chemistry : 98
Python: 70
Student 2 Marks : English: 60    Maths : 89    Physics: 98    Chemistry : 76
Python: 98
Student 3 Marks : English: 89    Maths : 78    Physics: 96    Chemistry : 56
Python: 78
Student 4 Marks : English: 78    Maths : 96    Physics: 87    Chemistry : 96
Python: 97
Student 5 Marks : English: 86    Maths : 79    Physics: 76    Chemistry : 88
Python: 78
-

```

In this program, you read the same file mark.bin in C:\\TURBOC3\\ directory and loop through the records one by one.

In simple terms, you read one marks record of marks size from the file pointed by *fptr into the structure m.

You'll get the same records you inserted in previous Example.

SEQUENTIAL AND RANDOM ACCESS

In computer programming, the two main types of file access are allowed they are:

- Sequential
- Random access

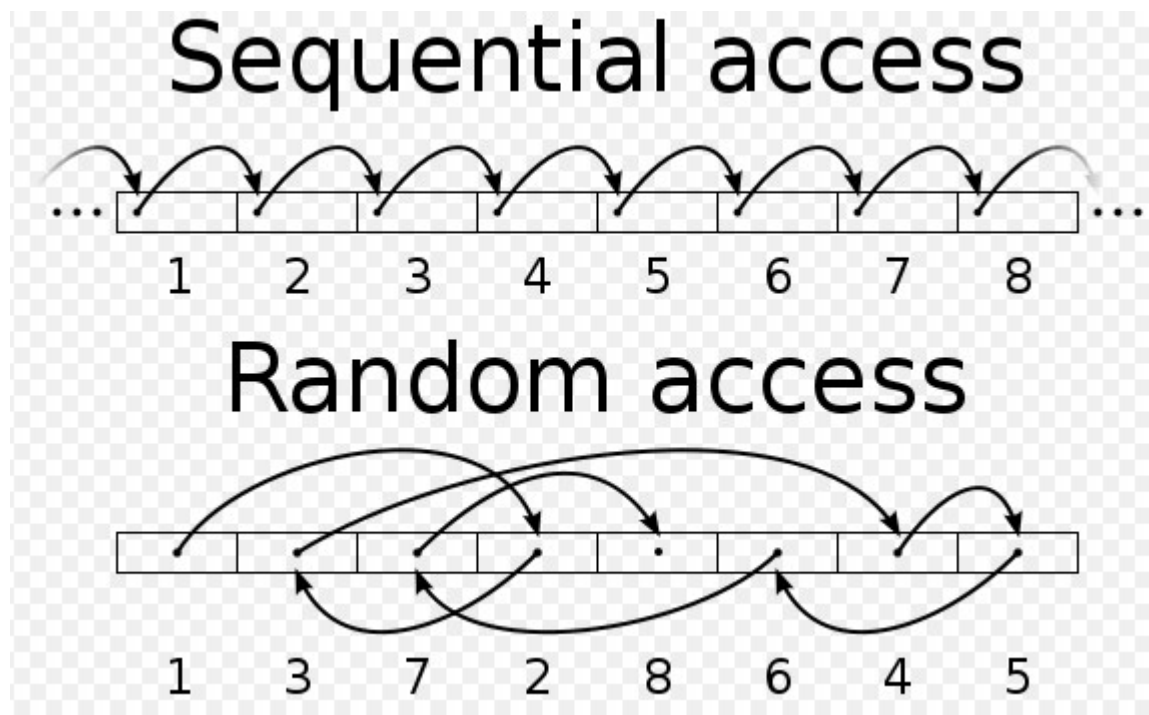


Figure : File Access Types

Sequential Access Files: are generally used in cases where the program processes the data in a sequential fashion – i.e. counting words in a text file. All the programs explained before use the sequential access of file.

Random Access Files: There are situations at which the records in the file need to be accessed randomly. If we want to access a particular record randomly, C provides these functions for random access file processing.

- `fseek()`
- `ftell()`
- `rewind()`

`fseek()`:

This function is used for seeking the pointer position in the file at the specified byte.

Syntax:

```
fseek( file pointer, displacement, pointer position);
```

Where

file pointer-----It is the pointer which points to the file.

displacement-----It is positive or negative. This is the number of bytes which are skipped backward (if negative) or forward (if positive) from the current position. This is attached with L because this is a long integer.

Pointer position:

This sets the pointer position in the file.

<i>Value</i>	<i>pointer position</i>
0	Beginning of file.
1	Current position
2	End of file

Example:

1) `fseek(p,10L,0)`

0 means pointer position is on beginning of the file, from this statement pointer position is skipped 10 bytes from the beginning of the file.

2) `fseek(p,5L,1)`

1 means current position of the pointer position. From this statement pointer position is skipped 5 bytes forward from the current position.

3) `fseek(p,-5L,1)`

From this statement pointer position is skipped 5 bytes backward from the current position.

ftell()

This function returns the value of the current pointer position in the file. The value is count from the beginning of the file.

Syntax:

```
ftell(fptr);
```

Where fptr is a file pointer.

rewind()

This function is used to move the file pointer to the beginning of the given file.

Syntax:

```
rewind( fptr);
```

Where fptr is a file pointer.

Example

Program to read last 'n' characters of the file using appropriate file functions (Here we use `fseek()` and `fgetc()`).

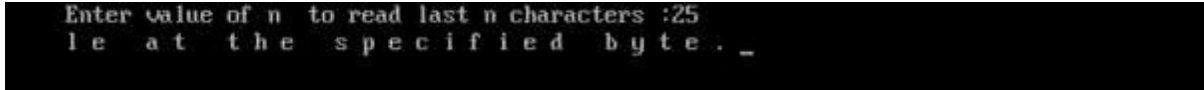
```
#include<stdio.h>
#include<conio.h>
void main()
{
    FILE *fp;
    char ch;
    int n;
    clrscr();
```

```
fp=fopen("C:\\TURBOC3\\fseek.txt", "r");
if(fp==NULL)
    printf("file cannot be opened");
else
{
    printf("Enter value of n to read last n characters :");
    scanf("%d",&n);
    fseek(fp,-n,2);
    while((ch=fgetc(fp))!=EOF)
    {
        printf("%c ",ch);
    }
}
fclose(fp);
getch();
}
```

Output:

fseek.txt - file content is

This function is used for seeking the pointer position in the file at the specified byte.



```
Enter value of n to read last n characters :25
le at the specified byte . _
```

ILLUSTRATIVE PROBLEMS:

Program to Finding average of numbers stored in sequential access file

Program:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    FILE *fp;
    char ch;
    int n;
    clrscr();
    fp=fopen("C:\\TURBOC3\\fseek.txt", "r");
    if(fp==NULL)
        printf("file cannot be opened");
    else
    {
        printf("Enter value of n to read last n characters :");
        scanf("%d",&n);
        fseek(fp,-n,2);
        while((ch=fgetc(fp))!=EOF)
```

```
    {  
        printf("%c ",ch);  
    }  
}  
fclose(fp);  
getch();  
}
```

Output:

```
Enter the Numbers to be Stored in File  
Enter -1 to stop Entering  
95  
65  
78  
90  
-1  
  
Sum of the Numbers in the File is 328  
Average of the Numbers in the File is 82.000000_
```

Program to Transaction processing using random access file

Program:

```
#include<stdio.h>  
#include<conio.h>  
struct account  
{  
    int number;  
    long amount;  
    char name[20];  
};  
  
void create()  
{  
    FILE *fptr;  
    int i, n;  
    struct account acc;  
    if ((fptr = fopen("C:\\\\TURBOC3\\\\account.bin","wb")) == NULL){  
        printf("File Cannot Open!");  
        exit(1);  
    }  
    printf("Enter Total Number of Customers\\n");  
    scanf("%d",&n);  
    for(i= 1; i <= n; i++)  
    {  
        acc.number=i;
```

```
    printf("Enter Name of User %d : ", i);
    scanf("%s",acc.name);
    printf("Enter Initial Amount of User %d : ", i);
    scanf("%ld",&acc.amount);
    fwrite(&acc, sizeof(struct account), 1, fptr);
}
fclose(fptr);
}
void view()
{
    FILE *fptr;
    int n;
    struct account acc;
    if ((fptr = fopen("C:\\TURBOC3\\account.bin","rb")) == NULL){
        printf("File Cannot Open!");
        exit(1);
    }
    while(1)
    {
        printf("\nEnter Customer Number to be view : -1 to stop \n");
        scanf("%d",&n);
        if(n== -1)
            break;
        fseek(fptr,(n-1)*sizeof(struct account),0);
        fread(&acc, sizeof(struct account), 1, fptr);
        printf("Customer Number: %d\t Customer Name: %s \tBalance: %ld",
acc.number,acc.name,acc.amount);

    }
    fclose(fptr);
}
void transfer()
{
    int fromno,tono;
    long tamount;
    struct account acc,fromacc,toacc;
    FILE *fptr;
    if ((fptr = fopen("C:\\TURBOC3\\account.bin","rb+")) == NULL){
        printf("File Cannot Open!");
        exit(1);
    }

    printf("\nEnter From Account No:\n");
    scanf("%d",&fromno);
    printf("\nEnter To Account No:\n");
    scanf("%d",&tono);
    printf("\nEnter Amount to be transfer:");
```

```
scanf("%ld",&tamount);
fseek(fptr,(fromno-1)*sizeof(struct account),0);
fread(&fromacc, sizeof(struct account), 1, fptr);
fseek(fptr,(tono-1)*sizeof(struct account),0);
fread(&toacc, sizeof(struct account), 1, fptr);
fromacc.amount-=tamount;
toacc.amount+=tamount;
fseek(fptr,(fromno-1)*sizeof(struct account),0);
fwrite(&fromacc, sizeof(struct account), 1, fptr);
fseek(fptr,(tono-1)*sizeof(struct account),0);
fwrite(&toacc, sizeof(struct account), 1, fptr);
fclose(fptr);
}
void main()
{
    int ch;
    clrscr();
    while(1)
    {
        printf("\n_____MENU_____\\n");
        printf("1.Create Account\\n2.View Account Detail \\n3.Transfer Amount\\n4.Exit");
        printf("\\nEnter Your Choice: ");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:
                create();
                break;
            case 2:
                view();
                break;
            case 3:
                transfer();
                break;
            case 4:
                exit(0);
        }
    }
}
```

Output:

```
____MENU____
1.Create Account
2.View Account Detail
3.Transfer Amount
4.Exit
Enter Your Choice: 1
Enter Total Number of Customers
3
Enter Name of User 1 : Vinu
Enter Initial Amount of User 1 : 75000
Enter Name of User 2 : Ranjith
Enter Initial Amount of User 2 : 70000
Enter Name of User 3 : Ram
Enter Initial Amount of User 3 : 90000

____MENU____
1.Create Account
2.View Account Detail
3.Transfer Amount
4.Exit
Enter Your Choice: 2

Enter Customer Number to be view : -1 to stop
2_
```

```
2
Customer Number: 2      Customer Name: Ranjith      Balance: 70000
Enter Customer Number to be view : -1 to stop
1
Customer Number: 1      Customer Name: Vinu      Balance: 75000
Enter Customer Number to be view : -1 to stop
3
Customer Number: 3      Customer Name: Ram      Balance: 90000
Enter Customer Number to be view : -1 to stop
-1

____MENU____
1.Create Account
2.View Account Detail
3.Transfer Amount
4.Exit
Enter Your Choice: 3

Enter From Account No:
2

Enter To Account No:
1

Enter Amount to be transfer:1000
```



```
____MENU____
1.Create Account
2.View Account Detail
3.Transfer Amount
4.Exit
Enter Your Choice: 2

Enter Customer Number to be view : -1 to stop
2
Customer Number: 2      Customer Name: Ranjith      Balance: 69000
Enter Customer Number to be view : -1 to stop
1
Customer Number: 1      Customer Name: Vinu      Balance: 76000
Enter Customer Number to be view : -1 to stop
3
Customer Number: 3      Customer Name: Ram      Balance: 90000
Enter Customer Number to be view : -1 to stop
-1

____MENU____
1.Create Account
2.View Account Detail
3.Transfer Amount
4.Exit
Enter Your Choice: 4
```

COMMAND LINE ARGUMENTS

It is possible to pass some values from the command line to your C programs when they are executed. These values are called **command line arguments** and many times they are important for your program especially when you want to control your program from outside instead of hard coding those values inside the code.

The command line arguments are handled using **main()** function arguments where **argc** refers to the number of arguments passed, and **argv[]** is a pointer array which points to each argument passed to the program. Following is a simple example which checks if there is any argument supplied from the command line and take action accordingly

```
#include <stdio.h>
int main( int argc, char *argv[] ) {

    if( argc == 2 ) {
        printf("One argument passed. Argument is %s\n", argv[1]);
    }
    else if( argc > 2 ) {
        printf("More than one arguments passed.\n");
    }
    else {
        printf("No argument Passed.\n");
    }
}
```

When the above code is compiled and executed with single argument, it produces the following result.

```
[root@vinu ~]# ./a.out one
One argument passed. Argument is one
```

When the above code is compiled and executed with a two arguments, it produces the following result.

```
[root@vinu ~]# ./a.out one two
More than one arguments passed.
```

When the above code is compiled and executed without passing any argument, it produces the following result.

```
[root@vinu ~]# ./a.out
No argument Passed.
```

It should be noted that **argv[0]** holds the name of the program itself and **argv[1]** is a pointer to the first command line argument supplied, and ***argv[n]** is the last argument. If no arguments are supplied, **argc** will be one, and if you pass one argument then **argc** is set at 2.

You pass all the command line arguments separated by a space, but if argument itself has a space then you can pass such arguments by putting them inside double quotes "" or single quotes ' '.

C Program to Copy Content from One File to another File Using Command Line Argument:

Program:

```
#include<stdio.h>
#include<string.h>
int main(int argc,char*argv[])
{
    FILE *sptr,*dptr;
    char ch;
    if(argc!=3)
    {
        printf("Command Line Error. Need Two argument");
        return;
    }
    sptr=fopen(argv[1],"r");
    dptr=fopen(argv[2],"w");
    if(sptr==NULL || dptr==NULL)
    {
```

```
    printf("Error in opening file!");  
    return;  
}  
while((ch=fgetc(sptr))!=EOF)  
{  
    fputc(ch,dptr);  
}  
printf("File Copied Successfully.");  
fclose(sptr);  
fclose(dptr);  
}
```

Output:

```
[root@vinu ~]# vi sample.txt  
[root@vinu ~]# vi filecopy.c  
[root@vinu ~]# ./a.out sample.txt samplecopy.txt  
File Copied Successfully.[root@vinu ~]#
```

While executing the above program content of sample.txt is copied to samplecopy.txt file.