Files

- Concept of a file streams,
- text and binary files
- stream file processing
- system created steams
- Standard library I/O functions
- file open and close
- formatting I/O functions
- character I/O functions
- Binary I/O
- command line arguments
- file status functions
- positioning functions
- Applications: Basic operations on files.

FILES:

A **file** is an external collection of related data treated as a unit.

The primary purpose of a file is to **keep a record of data**.

Record is a group of related fields. Field is a group of characters they convey meaning.

Files are stored in **auxiliary** or secondary storage devices. The two common forms of secondary storage are disk (hard disk, CD and DVD) and tape.

Each file ends with an end of file (**EOF**) at a specified byte number, recorded in file structure.

A file must first be **opened** properly **before** it can be **accessed** for reading or writing. When a file is opened an object (buffer) is created and a stream is associated with the object.

FILE NAME:

File name is a string of characters that make up a valid filename.

Every operating system uses a set of rules for naming its files.

When we want to read or write files, we must use the operating system rules when we name a file.

The file name may contain two parts, a **primary name** and an **optional** period with **extension**.

Example: input.txt
program.c

FILE INFORMATION TABLE:

A program that reads or write files needs to know several pieces of information, such as name of the file, the position of the current character in the file and so on.

C has predefined structure to hold this information.

The **stdio.h** header file defines this file structure; its name is **FILE**.

When we need a file in our program, we declare it using the FILE type.

STREAM:

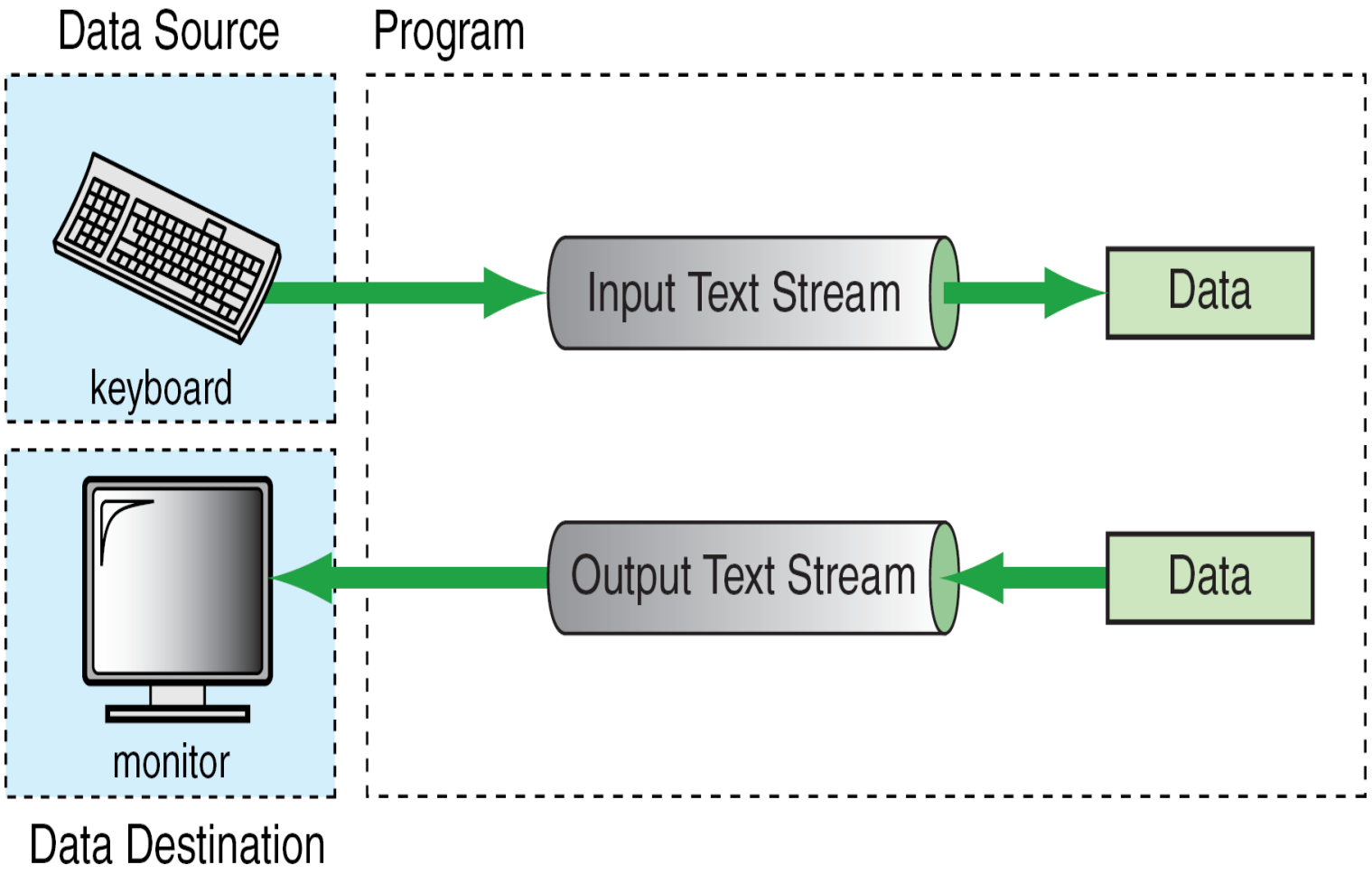
A stream is a source or destination of the data, it is associated with a physical device such as terminals (keyboard, monitor) or with a file stored in memory.

C has two forms of streams: **Text Stream and Binary Stream**.

A **text stream** consists of a sequence of characters divided into lines with each line terminated by a new line (\n).

A **binary stream** consists of sequence of data values such as integer, real, or complex using their memory representation.

A terminal keyboard and monitor can be associated only with a text stream. A keyboard is a source for a text stream; a monitor is a destination for a text stream.


Data Source
Program
keyboard
Input Text Stream
Data
Output Text Stream
Data
monitor
Data Destination

System Created Streams:

C provides standard streams to communicate with a terminal.

The first, **stdin,** points to the standard input stream which is normally connected to the keyboard.

The second, **stdout,** points to the standard output stream which is normally connected to the monitor.

The third, points to the **standard error** stream which is also normally connected to the monitor.

There is no need to open and close the standard streams. It is done automatically by the operating system.

STREAM FILE PROCESSING:

A file exists as an independent entity with a name known to the O.S.

A stream is an entity created by the program.

To use a file in our program, we must associate the program's stream name with the file name.

In general, there are four steps to processing a file.

1. Create a stream
2. Open a file
3. Process the file (read or write data)
4. Close a file

Creating a Stream:
We can create a stream when we declare it.

The declaration uses the FILE type as shown below:

FILE *fp;

The FILE type is a structure that contains the information needed for reading and writing a file and spData is a pointer to the stream.

Opening a File:
Once stream has been created, we can ready to associate the stream to a file. This is done through **open** function.

When the file is opened, the stream and the file are associated with each other.

Closing the Stream:
When file processing is complete, we close the file. After closing the file The stream is no longer available.

File Open and Close:

File Open (fopen):
The function that prepares a file for processing is fopen.

It does two things: First, it makes the **connection** between the physical file and the file stream in the program.

Second, it creates a program **file structure** to store the information needed to process the file.

To open a file, we need to specify the physical filename and its mode.

Syntax: **fopen (“filename”, “mode”);**

The file mode is a string that tells C compiler how we intend to use the file: reading, writing or append.

The address of the file structure that contains the file information is returned by **fopen**.

The actual contents of the FILE are hidden from our view.

All we need to know is that we can store the address of the file structure and use it to read or write the file.

For example: **fptr1 = fopen (“mydata", "r”);**
fptr2 = fopen ("results", "w”);

Once the files are open, they stay open until you close them or end the program.

File Mode: When we open a file, we explicitly define its mode.

The mode shows how we will use the file: for reading, for writing, or for appending.

"r" (read mode):
The read mode (r) opens an existing file for reading.

When a file is opened in this mode, the file marker is positioned at the beginning of the file (first character).

The file marker is a logical element in the file structure that keeps track of our current position in the file.

The file must already exist: if it does not, NULL is returned as an error.
If we try to write a file opened in read mode, we get an error message.

Syntax: **fp=fopen ("filename" , "r");**

“w” (write mode):

The write mode (w) opens a file for writing.

If the file doesn’t exist, it is created.

If it is already exists, it is opened and all its data are erased; the file marker is positioned at the beginning of the file (first character).

It is an error to try to read from a file opened in write mode.

Syntax: **fp=fopen (“filename”,”w”);**

“a” (append mode):
The append mode (a) also opens an existing file for writing.

Instead of creating a new file, however, the writing starts after the last character; that is new data is added, or appended, at the end of the file.

If the file doesn’t exist, it is created and opened. In this case, the writing will start at the beginning of the file.

Syntax: **fp=fopen (“filename”,”a”);**

“r+” (read and write) mode:
In this mode a file is opened for both reading and writing the data.

If a file does not exist then NULL, is returned.
Syntax: **fp=fopen (“filename”,”r+”);**

"w+" (write and read) mode:

In this mode a file is opened for both writing and reading the data.

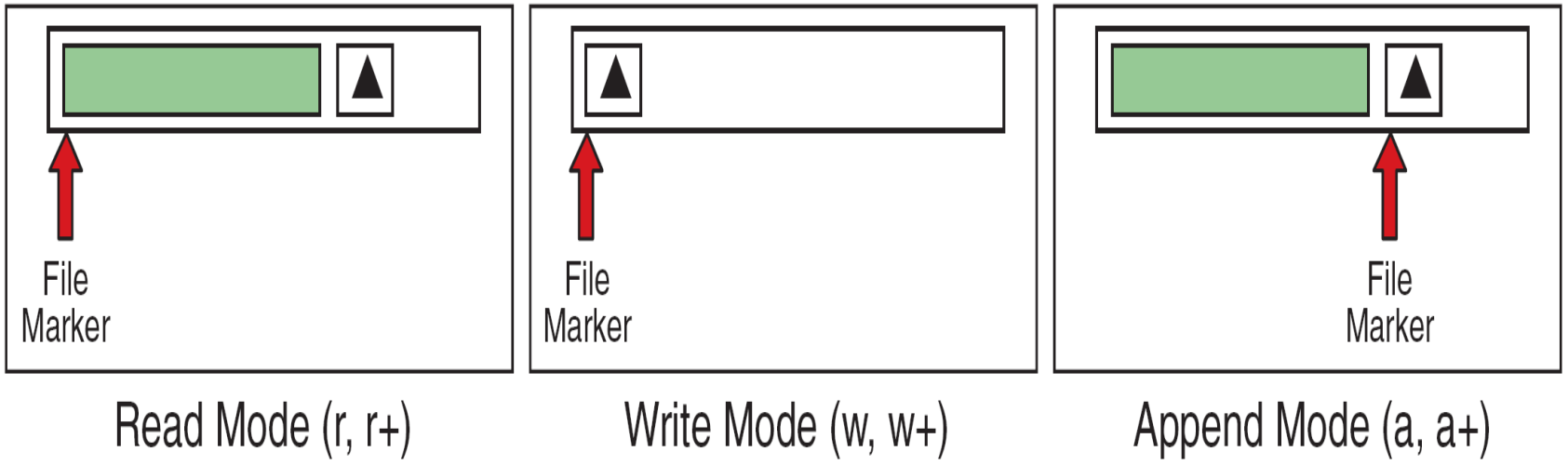
If a file already exists its contents are erased. If a file does not exist then new file created.

Syntax: **fp=fopen ("filename","w+");**

"a+" (append and read) mode:

In this mode a file is opened for reading the data as well as data can be added at the end.

Syntax: **fp=fopen ("filename", "a+");**



Mode	r	w	a	r+	w+	a+
Open State	read	write	write	read	write	write
Read Allowed	yes	no	no	yes	yes	yes
Write Allowed	no	yes	yes	yes	yes	yes
Append Allowed	no	no	yes	no	no	yes
File Must Exist	yes	no	no	yes	no	no
Contents of Existing File Lost	no	yes	no	no	yes	no

File Modes

File Close (fclose):

When we no longer need a file, we should be close it to free system resources, such as buffer space.

Closing a file ensures that all outstanding information associated with the file is flushed out from the buffers and all links to the file are broken.

Another instance where we have to close a file is to reopen the same file in a different mode.

A file is closed using the close function, **fclose.**
Syntax: **fclose (file_pointer);**

fclose () returns 0 on success (or) -1 on error.

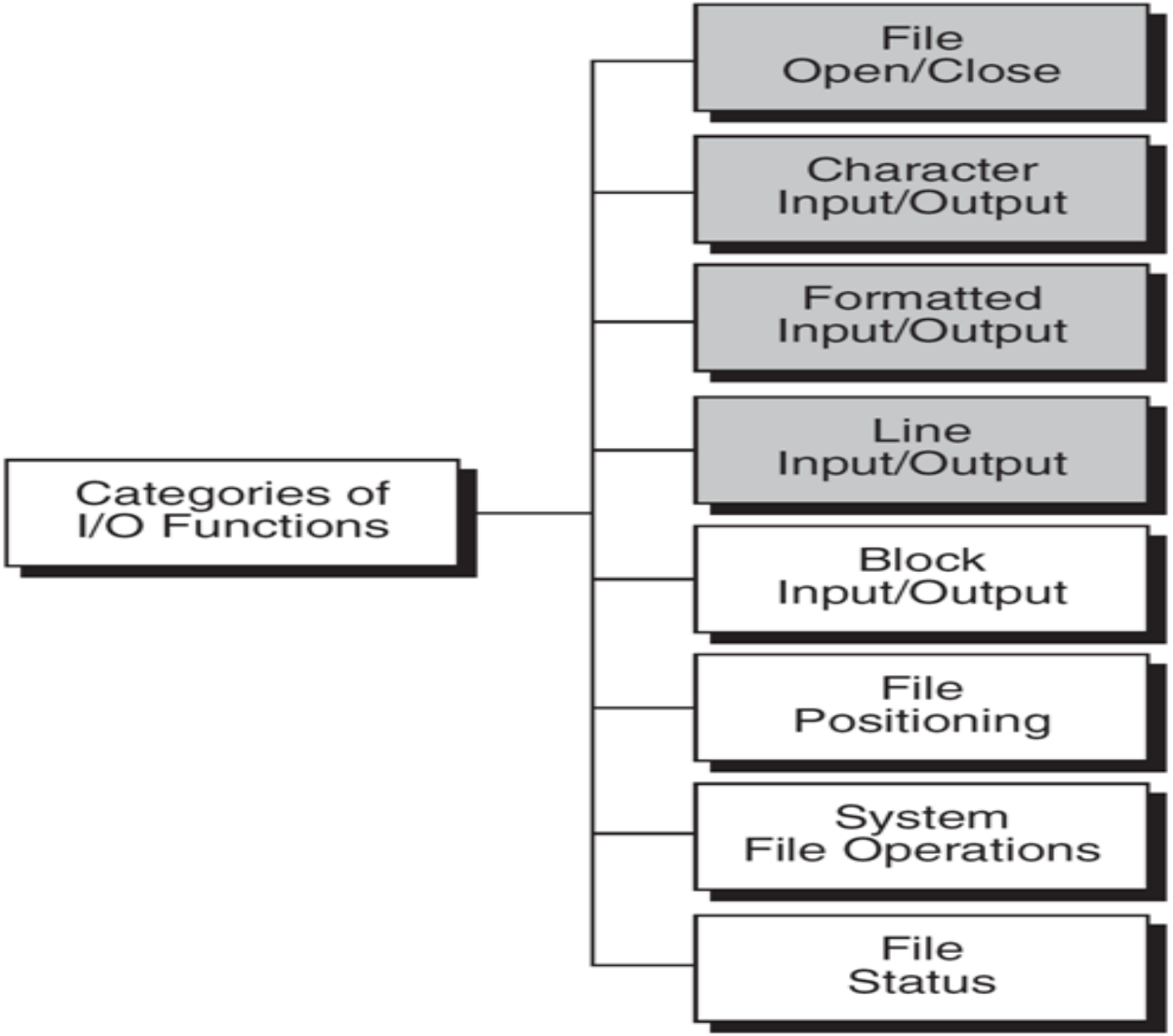
Once a file is closed, its file pointer can be reused for another file.

STANDARD LIBRARY I/O FUNCTIONS:

C includes many standard functions to input data from the keyword and output data to the monitor.

The **stdio.h** header file contains several different input/output functions declarations.

These are grouped into eight different categories.


Categories of I/O Functions
File Open/Close
Character Input/Output
Formatted Input/Output
Line Input/Output
Block Input/Output
File Positioning
System File Operations
File Status

Operations on Files:

- Open / close: fopen() and fclose()
- Character input / output : putc() /getc()
- Formatted input/output: fprintf() / fscanf()
- Line I/O : fputs() / fgets()
- Block input/output: fwrite() / fread()
- File position : ftell() / fseek / rewind()

FORMATED I/O FUNCTIONS:

We have already familiar with two formatting functions **scanf** and **printf**.

These two functions can be used only with the keyboard and monitor.

The C library defines two more general functions, **fscanf** and **fprintf**, that can be used with any **text stream**.

Reading from Files: fscanf ()

It is used to read data from a **user-specified stream.**

The general format of fscanf() is:

fscanf (stream_pointer, ”format string”, list);

The **first argument** is the **stream pointer**, it is the pointer to the streams that has been declared and associated with a text file. Remaining is **same as scanf** function arguments.

The following example illustrates the use of an input stream.

```
int a, b;
FILE *fptr1;
fptr1 = fopen (“mydata", "r”);
fscanf (fptr1, "%d %d", &a, &b);
```

The fscanf function would read values from the file "pointed" to by fptr1 and assign those values to a and b.

The only **difference** between scanf and fscanf is that scanf reads data from the stdin (input stream) and fscanf reads input from a user specified stream(stdin or file).

The following example illustrates how to **read data from keyboard** using fscanf,

fscanf (stdin,"%d", &a);

End of File:

The end-of-file indicator informs the program when there are no more data (no more bytes) to be processed.

Writing to Files: fprintf ()

It can handle a group of mixed data simultaneously.

The first argument of these functions is a file pointer which specifies the file to be used.

The general form of fprintf is:

fprintf (stream_pointer, ”format string”, list);

Where stream_pointer is a file pointer associated with a file that has been opened for writing.

The format string contains output specifications for the items in the list.

The list may include variables, constants and strings.

The following example illustrates the use of an Output stream.

```
int a = 5, b = 20;
FILE *fptr2;
fptr2 = fopen (“results", "w”);
fprintf (fptr2, "%d %d\n", a, b) ;
```

The fprintf functions would write the values stored in a and b to the file "pointed" to by fptr2.

fprintf function works like printf except that it specifies the file in which the data will be displayed.

The file can be standard output (stdout) or standard error (stderr) also.

Example,

```
fprintf (stdout,”%d”,45);        //displays 45 on Monitor.
```

//Program to perform addition of two numbers using **fscanf** and **fprintf.**

```
#include<stdio.h>
#include<stdlib.h>
main()
{
        FILE *fp1, *fp2;
        int a,b,c;
        fp1 = fopen("myfile","r");
        fscanf(fp1,"%d%d",&a,&b);
        c=a+b;
        fp2 = fopen("output","w");
        fprintf(fp2, "sum=%d\n", c);
        fclose(fp2);
        fclose(fp1);
}
```

//Program to copy contents of one file another file using **fscanf** and **fprintf.**

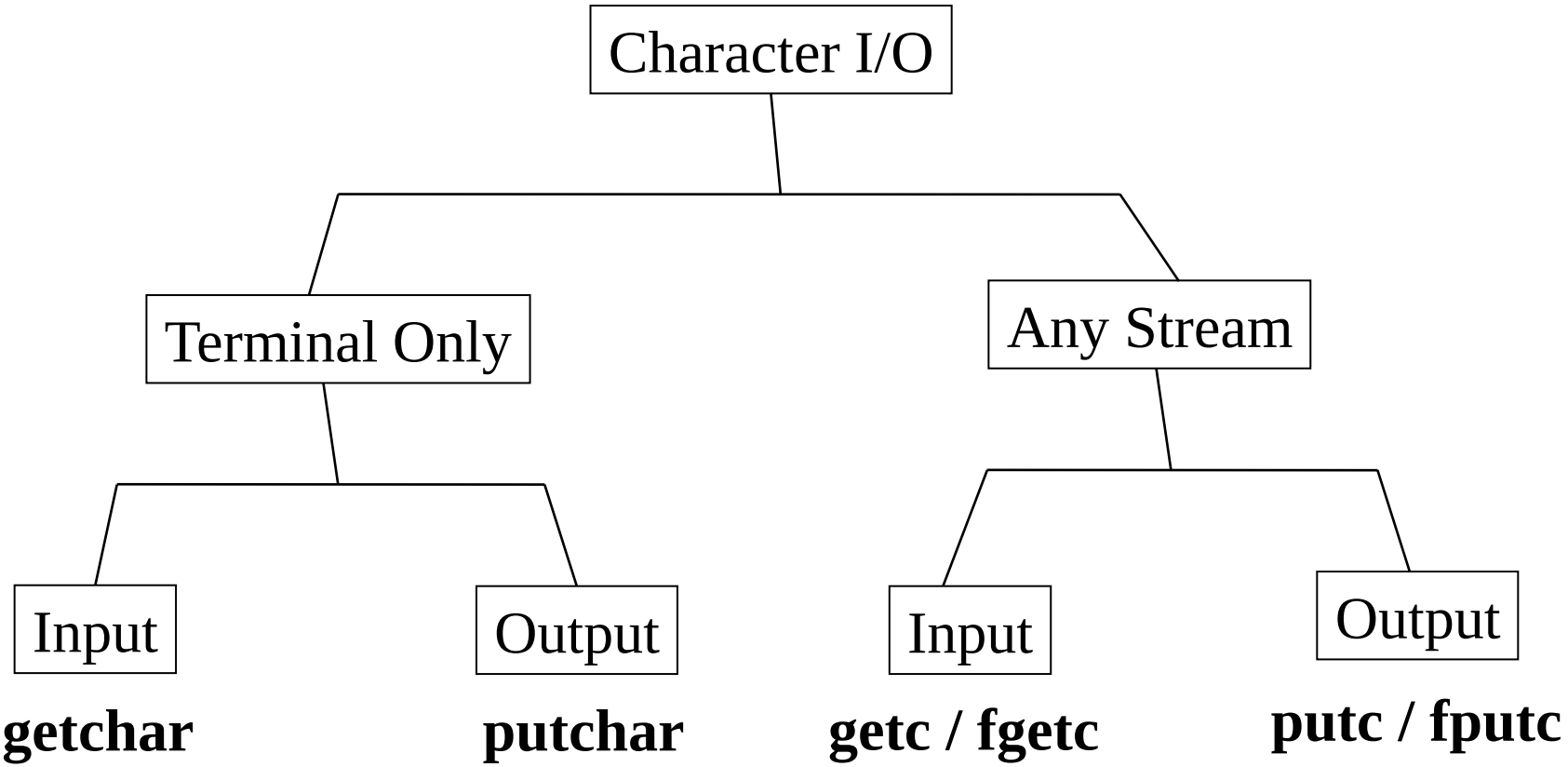
```
#include<stdio.h>
#include<stdlib.h>
main()
{
        FILE *spIn, *spOut;
        int numIn;
        spIn = fopen("input.txt","r");
        spOut = fopen("output.txt","w");

        while((fscanf(spIn,"%d",&numIn)) == 1)
                fprintf(spOut, "%d\n", numIn);

        fclose(spIn);
        fclose(spOut);
}
```

CHARACTER I/O FUNCTIONS:

Character input functions read one character at a time from a text stream.
Character output functions write one character at the time to a text stream.

These functions can be divided into two categories:

Terminal Character I/O, Terminal and File Character I/O



TERMINAL CHARACTER I/O:

C declares a set of character input/output functions that can only be used with the standard streams:

standard input (**stdin**), standard output (**stdout**).

Read a character: **getchar ()**

It reads the next character from the standard input stream and return its value.

Syntax: **int getchar (void);**

Its return value is integer. Up on successful reading returns the ASCII value of character otherwise it returns EOF.

Write a character: **putchar ()**

It writes one character to the monitor.

Syntax: **int putchar (int ch);**

Its return value is integer. Up on successful writing returns the ASCII value of character. Otherwise returns EOF.

TERMINAL AND FILE CHARACTER I/O :

The terminal character input/output functions are designed for convenience; we don’t need to specify the stream.

Here, we can use a more general set of functions that can be used with both the standard streams and a file.

These functions require an argument that specifies the stream associated with a terminal device or a file.

When used with a terminal device, the streams are declared and opened by the system, the standard input stream (stdin) for the keyword and standard output stream (stdout) for the monitor.

When used with a file, we need to explicitly declare the stream, it is our responsibility to open the stream and associate with the file.

Read a character: getc () and fgetc ()

The getc functions read the next character from the **stream**, which can be a **user-defined stream or stdin**, and converts it in to an integer.

This function has one argument which is the file pointer declared as FILE or stdin (in case of standard input stream).

If the read detects an end of file, the function returns EOF, EOF is also returned if any error occurs.

The functionality of getc / fgetc is **same**.

Syntax:

int getc (FILE *fpIn);	or	**int getc(stdin);**
int fgetc (FILE *fpIn);	or	**int fgetc(stdin);**

Write a Character: putc () and fputc ()

The putc function writes a character to the stream which can be a user-defined stream, stdout, or stderr.

The functionality of putc/ fputc is same.

The functions, putc or fputc takes two arguments.

The first parameter is the character to be written and the second parameter is the file.

The second parameter is the file pointer declared as FILE or stdout or stderr.

If the character is successfully written, the function returns it. If any error occurs, it returns EOF.

Syntax:

```
int putc (char, *fp);
int fputc (char, *fp);
```

/*Program to copy content of one file to another file using getc/putc*/

```
#include<stdio.h>
#include<stdlib.h>
main()
{
    char ch=' ';
    FILE *fp1,*fp2;
    fp1=fopen("file1.c","r");
    fp2=fopen("file2.c",”w");
    if(fp1) {
        while(ch!=EOF)
        {
            ch=getc(fp1);
            putc(ch,fp2);
        }
    }
    else
        printf("\n File1 doesnot exists");
    fcloseall();
}
```

/*Program to append one file to another file using getc/putc*/

```
#include<stdio.h>
#include<stdlib.h>
main()
{
    char ch=' ';
    FILE *fp1,*fp2;
    fp1=fopen("file1.c","r");
    fp2=fopen("file2.c",”a");
    if(fp1) {
        while(ch!=EOF)
        {
            ch=getc(fp1);
            putc(ch,fp2);
        }
    }
    else
        printf("\n File1 doesnot exists");
    fcloseall();
}
```

/*Program to copy the contents of one file to another file using command line arguments*/

```
#include<stdio.h>
#include<stdlib.h>
main(int argc,char *av[]) {
    FILE *fpt1,*fpt2;
    char ch;
    fpt1=fopen(av[1],"r");
    fpt2=fopen(av[2],"w");
    if(fpt1)
    {
        while(ch!=EOF)    {
            ch=getc(fpt1);
            putc(ch,fpt2);
        }
    }
    else
        printf("\n File does not exists");

    fcloseall();
}
```

LINE INPUT/OUTPUT

fgets() function

fgets() function is used to read string(array of characters) from the file.

Syntax of fgets() function

```
fgets(char str[],int n,FILE *fp);
```

The fgets() function takes three arguments, first is the string read from the file, second is size of string(character array) and third is the file pointer from where the string will be read.

The fgets() function will return NULL value when it reads EOF(end-of-file**).**

Example:fgets(str,80,*fp1)

fputs() function

The fputs() function is used to write string(array of characters) to the file

Syntax of fputs() function

```
fputs(char str[], FILE *fp);
```

The fputs() function takes two arguments, first is the string to be written to the file and second is the file pointer where the string will be written.

/*Program to copy content of one file to another file using fgets /fputs*/

```
#include<stdio.h>
#include<stdlib.h>
main()
{
    FILE *fp1,*fp2;
    char str[80];
    fp1=fopen("file1.c","r");
    fp2=fopen("file2.c",”w");
    if(fp1) {
        while((fgets(str,80,fp1))!=NULL)
        {
            fputs(str,fp2);
        }
    }
    else
        printf("\n File1 doesnot exists");
    fcloseall();
}
```

TEXT FILES AND BINARY FILES:

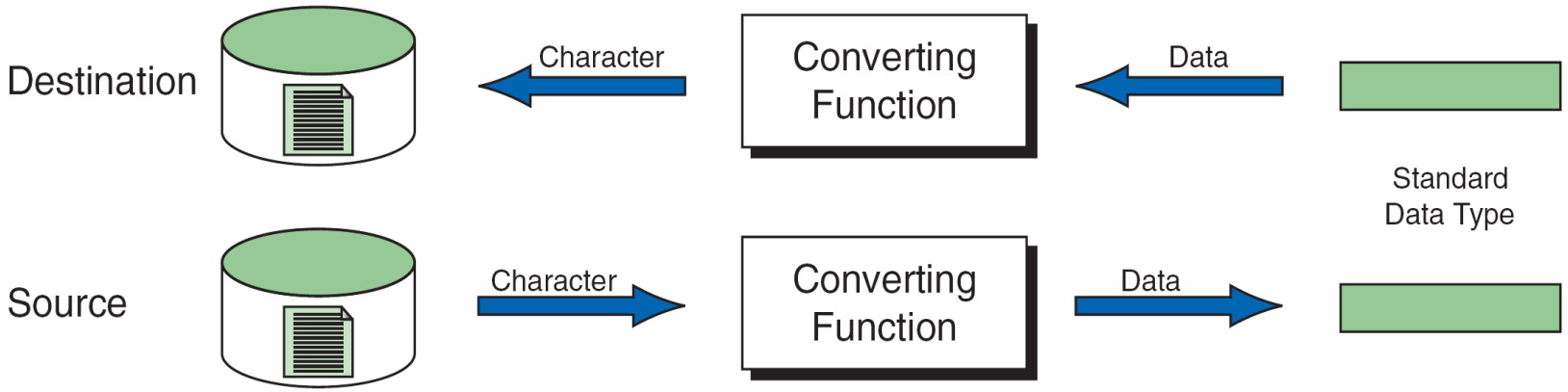
Text File: It is a file in which data are stored using only **characters**; a text file is written using text stream.

Non-character data types are converted to a sequence of characters before they are stored in the file.

In the text format, data are organized into lines, terminated by **newline** character. The text files are in **human readable form** and they can be created and read using any text editor.

Text files are read and written using input / output functions that convert characters to data types: **scanf and printf, getchar and putchar, fgets and fputs.**

The following figure shows the data transfer in text file:



Reading and Writing Text Files

Binary File:

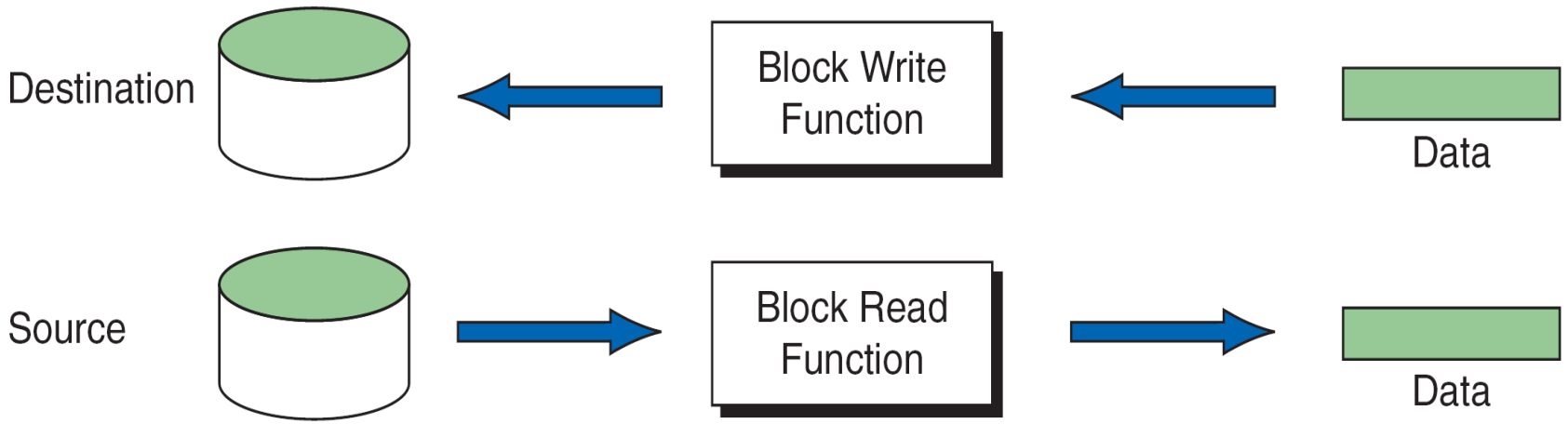
A **binary file** is a collection of data stored in the internal format of the computer.

The binary files are **not in human readable** form.

There are **no** lines or **newline** characters.

Binary files are read and written using binary streams known as **block input / output** functions.

The following figure shows the data transfer in binary file:



Block Input and Output

Differences between Text File and Binary File

Text File	Binary File
Data is stored as lines of characters with each line terminated by newline.	Data is stored on the disk in the same way as it is represented in the computer memory.
Human readable format.	Not in human readable format.
There is a special character called end-of-file(EOF) marker at the end of the file.	There is an end-of-file marker.
Data can be read using any of the text editors.	Data can be read only by specific programs written for them.

Opening Binary Files

The basic operation is unchanged for binary files, only the mode changes.

Just like text files, binary files must be closed when they are not needed anymore using fclose ().

Mode	Meaning
wb	This mode opens a binary file in write mode. Example: **fp=fopen (“data.dat”,”wb”);**
rb	This mode opens a binary file in read mode. Example: **fp=fopen (“data.dat”,”rb”);**
ab	This mode opens a binary file in append mode. Example: **fp=fopen (“data.dat”,”ab”);**
w+b	This mode opens/creates a binary file in write and read mode. Example: **fp=fopen (“data.dat”,”w+b”);**
r+b	This mode opens a pre-existing binary file in read and write mode. Example: **fp=fopen (“data.dat”,”r+b”);**
a+b	This mode opens/creates a binary file in append mode. Example: **fp=fopen (“data.dat”,”a+b”);**

Binary Modes of Opened File

Opening Text / Binary Files:

Binary file Mode	Text file mode		Meaning
wb	w		writing
rb	r		reading
ab	a		appending
w+b	w+		write and read
r+b	r+		read and write
a+b	a+		read and append

Modes of Opened File

BLOCK I/O FUNCTIONS:

C language uses the block input and output functions to read and write data to binary files.

As we know that data are stored in memory in the form of 0’s and 1’s.

When we read and write the binary files, the data are transferred just as they are found in memory and hence there are no format conversions.

The block read function is file read(**fread**) and the block write function is file write (**fwrite**).

File Read: fread ()

It reads a specified number of bytes from a binary file and places them into memory at the specified location.

The function declaration is as follows:

int fread (void *pInArea, int elementsize, int count, FILE *sp);

The first parameter, pInArea, is a pointer to the input area in memory.
The data read from the file should be stored in memory.

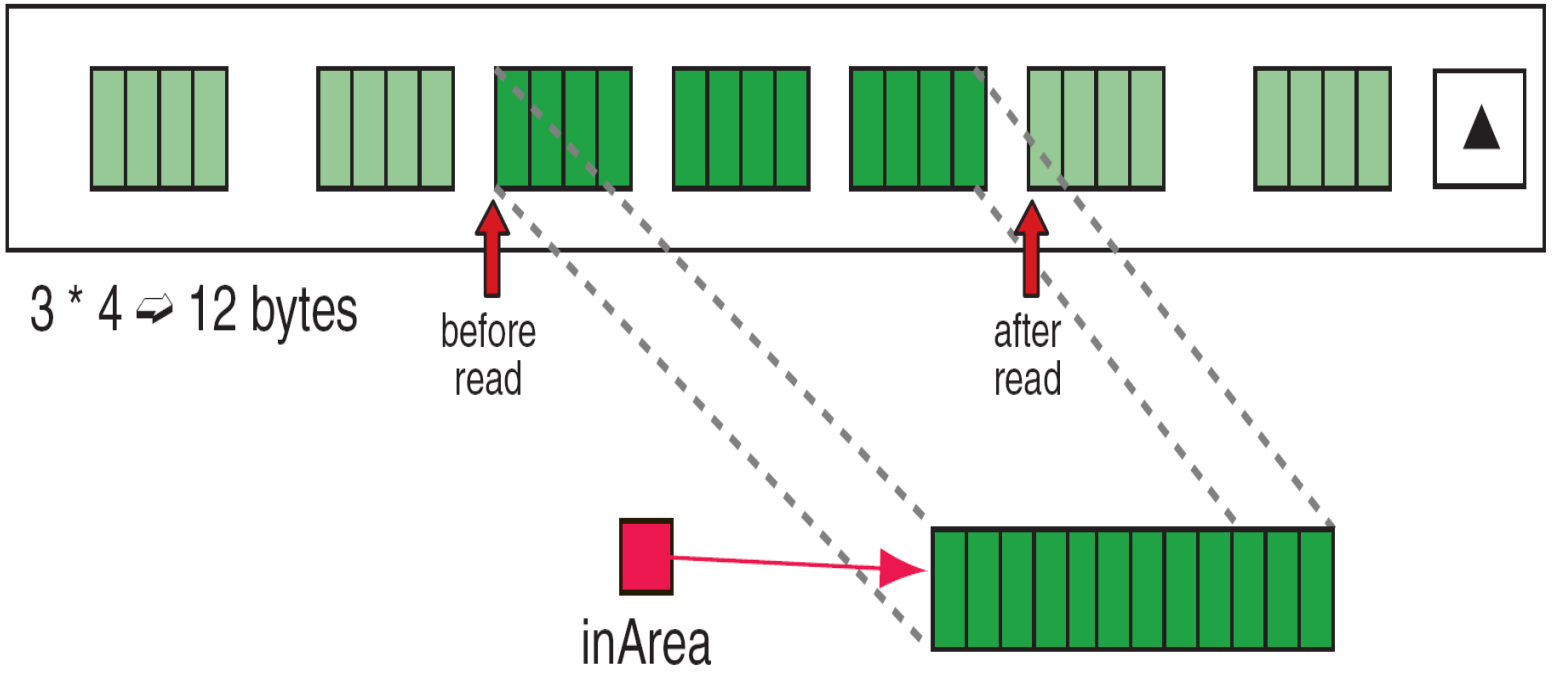
For this purpose, it is required to allocate the sufficient memory and address of the first byte is stored in pInArea.

The next two elements, elementSize and count, are multiplied to determine how much data are to be transferred.
The size is normally specified using the sizeof operator and the count is normally one when reading structures.

The last argument is the pointer to the file we want to read from.

This function returns the number of items read. If no items have been read or when error has occurred or EOF encountered, the function returns 0.

Below is an example of a file read that reads data into an array of integers. When fread is called, it transfers the next three integers from the file to the array, inArea.




```
fread (inArea, sizeof (int), 3, spData);
```

File Read Operation

File Write: fwrite ()

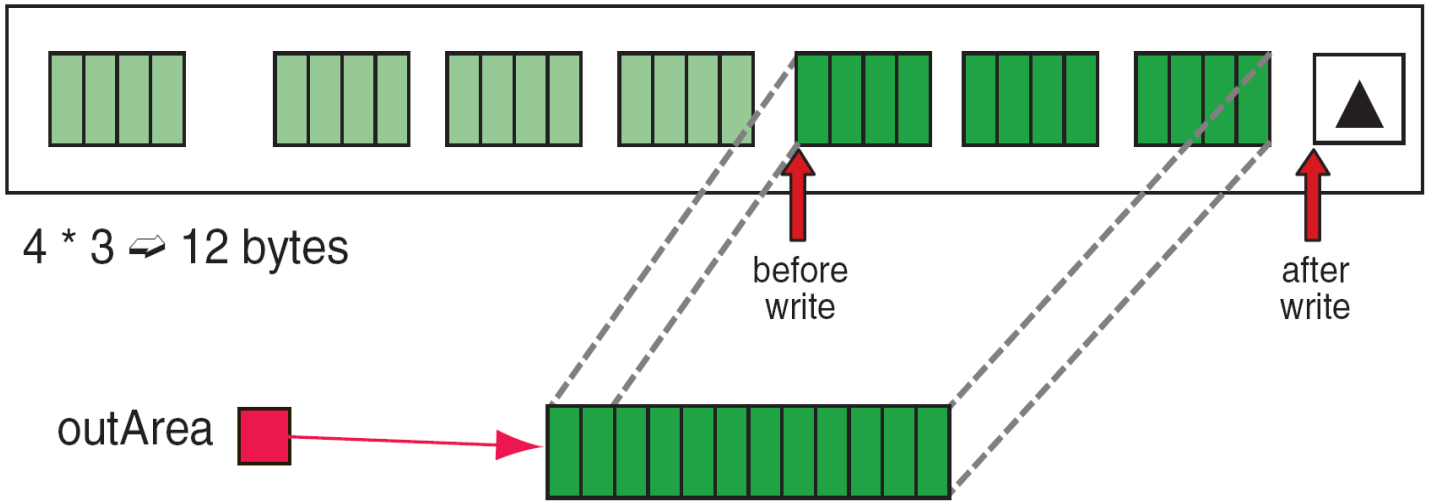
It writes specified number of items to a binary file.

The function declaration is as follows,

int fwrite (void *pOutArea, int elementSize, int count, FILE *sp);

The parameters for file write correspond exactly to the parameters for the file read function.

Example for file write operation:




```
fwrite (outArea, sizeof (int), 3, spOut);
```

File **Write Operation**

/*Program to reverse the first n characters of a file.*/

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
main() {
    FILE *fp;
    char str[30],rev[30];
    int n,j,i;
    printf("\nEnter the numof character u want reverse:");
    scanf("%d",&n);
    fp=fopen("file3.c","r+b");
    fread(&str,sizeof(char),n,fp);
    for(i=0,j=n-1;i<n;i++,j--)   {
            rev[i]=str[j];
    }
    rev[i]='\0';
    rewind(fp);
    fwrite(&rev,sizeof(char),n,fp);
    fcloseall();
}
```

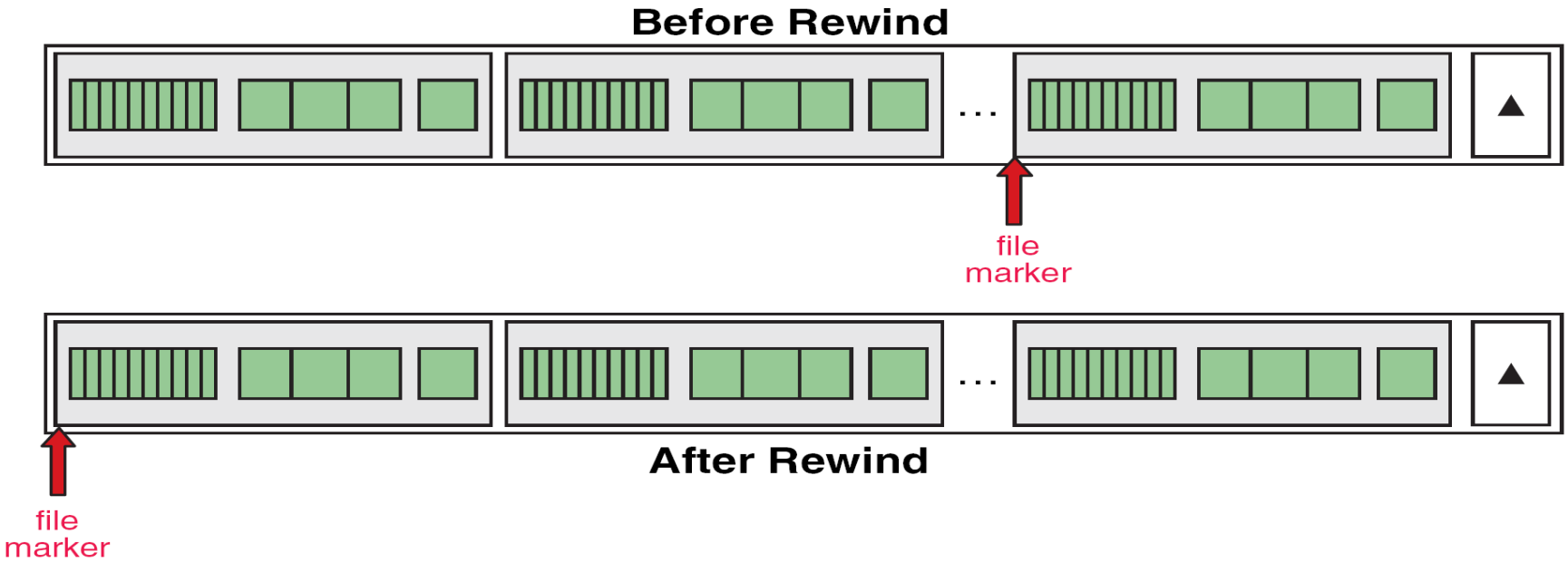
Rewind File (rewind):

It simply sets the file position indicator to the beginning of the file.

Syntax: **void rewind(FILE *stream);**

It helps us in reading a file **more than once**, without having to close and open the file.

A common use of the rewind function is to change a work file from a write state to a read state.



Rewind File

Current Location (ftell):

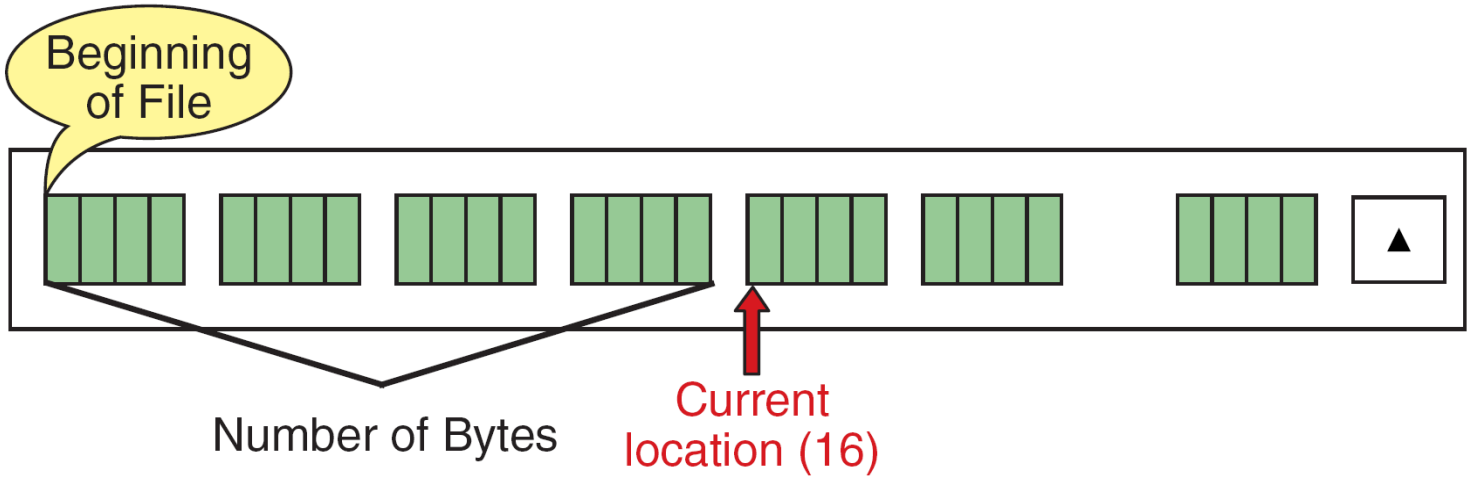
It reports the current position of the file marker in the file, relative to the beginning of the file.

It measures the position in the file by the number of bytes, relative to zero, from the beginning of the file.

Syntax: **long int ftell(FILE *stream);**

It also returns the number of bytes from the beginning of the file.

If ftell encounters an error, it returns -1.



Current Location (*ftell*) Operation

Set Position: (fseek):

It is used to move the file position to a desired location within the file.

Syntax: **int fseek(FILE *stream, long offset, int wherefrom);**

The offset specifies the number of positions to be moved from the location specified by position.

The position can take one of the following three values:

Value	Meaning
0	Beginning of file.
1	Current position.
2	End of file.

The offset may be positive(means forward), or negative (means backward).

When the operation is successful, fseek returns a zero.

If we attempt to move the file pointer beyond the file boundaries, an error occurs and fseek returns -1.

Operations of the fseek function

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

```
#include<stdio.h>                //Program to demonstrate file positioning functions.
main(){
    FILE *fp;
    char arr[30];      int i = 0;  long int n;
    fp = fopen("input.txt", "r");
    n = ftell(fp);
    printf("\nThe current posiotion of file marker=%d",n);
    fseek(fp,-5,2);   //go backward by 5 bytes from the end
    n = ftell(fp);
    printf("\nThe current posiotion of file marker=%d",n);
    printf("\nCharacters read from file are:");
    for(i = 0; i < 5; i++) {
            arr[i] = fgetc(fp);
            fputc(arr[i],fp);
            printf("%c",arr[i]);
    }
        rewind(fp);       //reset offset to beginning
        n = ftell(fp);
        printf("\nThe current posiotion of file marker=%d",n);
        fseek(fp,5,1);
        printf("\nCharacters read from file are:");
        for(i = 0; i < 5; i++)          {
                arr[i] = fgetc(fp);
                fputc(arr[i],fp);
                printf("%c",arr[i]);
        }
}
```

Different types of files to opened to write the contents

File Type	Extension
C Source File	.c
Text File	.txt
Data File	.dat

I/O Function used in FILES

- fopen() : Create or open a file for reading or writing.
- fclose() : Close a file after reading or writing it.
- fseek() : Move (Seek) to a certain location in a file.
- ftell() : Returns the current position of the file pointer.
- rewind() : Rewind a file back to its beginning and leave it open.
- fprintf() : Formatted write.
- fscanf() : Formatted read.
- fwrite() : Unformatted write. (The block read and write functions.)
- fread() : Unformatted read.
- putc() : Write a single byte to a file.
- getc() : Read a single byte from a file.
- fgets() : To read one line from a file at a time.
- fputs() : To write one line to a file at a time.
- fcloseall() : To close all opened files in a program

1. writing the data into the file

```
#include<stdio.h>
int main()
{
    FILE *fp;
    char ch;
    fp=fopen("file.txt","w");       // open a file in write mode
    printf("\nEnter data to be stored in to the file:");
    while((ch=getchar())!=EOF)   // writing the data into the file
    fputc(ch,fp);
    fclose(fp);
    return 0;
}
```

2. To Read a file contents and print on the screen

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
char ch;
FILE *fp;
fp = fopen("abc.txt","r");     // open a file  in read mode
if( fp == NULL )
{
printf("Error while opening the file.\n");

}
printf("The contents of file are :\n", );
while( ( ch = fgetc(fp) ) != EOF )
printf("%c",ch);
fclose(fp);
return 0;
}
```

3.1 copy the data from one file to another file(using fgetc,fputc)

```
#include<stdio.h>
int main()
{
    FILE *fp1,*fp2;
    char ch;
    fp1=fopen("file.txt",”r");        // open a file in read mode
     fp2=fopen(“copy.txt”,”w”);
    while((ch=fgetc(fp1))!=EOF)   // writing the data into the file
    fputc(ch,fp2);
    fclose(fp);
    return 0;
}
```

3.2 copy content of one file to another file using fgets /fputs

```
#include<stdio.h>
#include<stdlib.h>
main()
{
    FILE *fp1,*fp2;
    char str[80];
    fp1=fopen("file1.c","r");
    fp2=fopen("file2.c",”w");
    if(fp1) {
        while((fgets(str,80,fp1))!=NULL)
        {
            fputs(str,fp2);
        }
    }
    else
        printf("\n File1 doesnot exists");
    fcloseall();
}
```

3.3 copy the contents of one file to another file using command line arguments

```
#include<stdio.h>
#include<stdlib.h>
main(int argc,char *argv[]) {
    FILE *fpt1,*fpt2;
    char ch;
    fpt1=fopen(argv[1],"r");
    fpt2=fopen(argv[2],"w");
    if(fpt1)
    {
        while(ch!=EOF)    {
                ch=getc(fpt1);
                putc(ch,fpt2);
        }
    }
    else
        printf("\n File does not exists");

    fcloseall();
}
```

3.4 copy the contents of one file to another file (read file names from keyboard)

```
#include<stdio.h>                    #include<stdlib.h>
void  main()
{
    FILE *fp1,*fp2;
    char  str1[30],str2[30],ch=‘  ‘;
    printf(“enter source file name”);
    scanf(“%s”,str1);
    printf(“enter destination name”);
    scanf(“%s”,str2);
    fp1=fopen(str1,"r");
    fp2=fopen(str2,"w");
    if(fp1)
    {
          while(ch!=EOF)  {
          ch=getc(fp1);
          putc(ch,fp2);       }
    }
    else
          printf("\n File does not exists");

    fcloseall();
```

3.4 copy the contents of one file to another file (read file names from keyboard)

```
#include<stdio.h>                    #include<stdlib.h>
void  main()
{
    FILE *fp1,*fp2;
    char  str1[30],str2[30],ch=‘  ‘;
    printf(“enter source file name”);
    scanf(“%s”,str1);
    printf(“enter destination name”);
    scanf(“%s”,str2);
    fp1=fopen(str1,"r");
    fp2=fopen(str2,"w");
    if(fp1)
    {
          while(ch!=EOF)  {
          ch=getc(fp1);
          putc(ch,fp2);       }
    }
    else
          printf("\n File does not exists");

    fcloseall();
```

4. Write a program to append file to another file

```
#include<stdio.h>
void  main()
{
    FILE *fp1,*fp2;
    char ch;
    fp1=fopen(“biodata.txt”,”a");
    fp2=fopen(“marks.txt”,”r");
    if(fp2)
    {
        while((ch=fgetc(fp2))!=EOF)        {
        fputc(ch,fp1);      }
    }
    else
        printf("\n File does not exists");

    fcloseall();
}
```

/5.Program to reverse the first n characters of a file.*/

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
main() {
    FILE *fp;
    char str[30],rev[30];
    int n,j,i;
    printf("\nEnter the numof character u want reverse:");
    scanf("%d",&n);
    fp=fopen("file3.c","r+b");
    fread(&str,sizeof(char),n,fp);
    for(i=0,j=n-1;i<n;i++,j--)   {
            rev[i]=str[j];
    }
    rev[i]='\0';
    rewind(fp);
    fwrite(&rev,sizeof(char),n,fp);
    fcloseall();
}
```