Function definition

A Function is a self-contained block of statement that performs a specific task when called. A function is also called as sub program or procedure or subroutine.

Every C Program can be thought of as a collection of these functions. Function is a set of instructions to carry out a particular task. Function after its execution returns a single value. Generally, the functions are classified into two types:

1. Standard functions
2. User-defined functions.

The Standard functions are also called *library functions* or *built-in functions*. All standard functions, such as *sqrt(), abs(), log(), sin()* etc. are provided in the library of function. But, most of the applications need other functions than those available in the software, those are known as *user-defined functions.*

Advantages of functions

Several advantages of dividing the program into functions include:

- Modularity
- Reduction in code redundancy
- Enabling code reuse
- Better readability

Parts of a function

A function has the following parts:

1. Function prototype declaration.
2. Function definition.
3. Function call.
4. Actual arguments and Formal arguments.
5. The *return* statement.

Function Declaration

Function prototype declaration consist function return type, function name, and argument list. A function must be declared before it is used

Syntax

return_type function_name(parameter_list);

parameter_list: type param1,type param2,type param3, etc

Examples

```
double sqrt(double);
int func1();
void func2(char c);
```

Function Definition also known as function implementation, means composing a function.

Every Function definition consists of two Parts:

1. Header of the function,
2. Body of the function.

Syntax

return_type function_name(parameter_list)

```
{
        // Function body
}
```

The body of a function consists of a set of statements enclosed within braces. The *return* statement is used to return the result of the computations done in the called function and/or to return the program control back to the calling function. A function can be defined in any part of the program text or within a library

Example:

```
void welcome( )
{
        printf("hello world\n";
}
```

Function Call

A function call has the following syntax:

<function name>(<argument list>);

Example:

sum(a,b);

4.2 Types of functions

Functions can be divided into 4 categories:

- A function with no arguments and no return value
- A function with no arguments and a return value
- A function with an argument or arguments and returning no value
- A function with arguments and returning a values

A function with no arguments and no return value

Called function does not have any arguments

Not able to get any value from the calling function

Not returning any value

There is no data transfer between the calling function and called function.

Example:

```
#include<stdio.h>
void welcome();          //function prototype declaration

int main()
{
        welcome();  // calling function   or // Function Call

        return 0;
}
void welcome()      //called function  or // Function Definitio
```

```
{
   printf("Hi students..!");    //Function body
}
```

OUTPUT

Hi students..!

A function with no arguments and a return value

Does not get any value from the calling function

Can give a return value to calling program

Example

```
#include <stdio.h>
int send();
int main()
{
        int x;
        x=send()
        printf("\n You entered : %d", x);
        return 0;
}
int send()
{
        int n;
        printf("\n Enter a number: ");
        scanf("%d", &n);
        return n;
}
```

OUTPUT

Enter a number: 5
You entered: 5

A function with an argument(s) and returning no value

A function has argument(s).

A calling function can pass values to function called , but calling function not receive any value.

Data is transferred from calling function to the called function but no data is transferred from the called function to the calling function.

Generally Output is printed in the Called function.

Example

```
#include<stdio.h>
void add(int x, int y);
int main()
{
        add(30,15);
        add(63,49);
```

```
        add(952,321);
        return 0;;
}
void add(int a, int b)
{
        int result;
        result = a+b;
        printf("Sum of %d and %d is %d\n" ,a, b, result);
}
```

OUTPUT

Sum of 30 and 15 is 45
Sum of 63 and 49 is 112
Sum of 952 and 321 is 1273

A function with arguments and returning a values

Argument are passed by calling function to the called function
Called function return value to the calling function
Data returned by the function can be used later in our program for further calculation
Mostly used in programming because it can two way communication

Example

```
#include <stdio.h>

int add(int x, int y);
int main()
{
        int z;
        z=add(952,321);
        printf("Result %d\n", add(30,55));
        printf("Result %d\n", z);
        return 0;
}
int add(int a, int b)
{
        int result;
        result = a + b;
        return (result);
}
```

OUTPUT

Result = 85
Result = 1273

Inter Function Communication in C:

When a function gets executed in the program, the execution control is transferred from calling a function to called function and executes function definition, and finally comes back to the calling

function. In this process, both calling and called functions have to communicate with each other to exchange information. The process of exchanging information between calling and called functions is called inter-function communication.
In C, the inter function communication is classified as follows...

- **Downward Communication**
- **Upward Communication**
- **Bi-directional Communication**

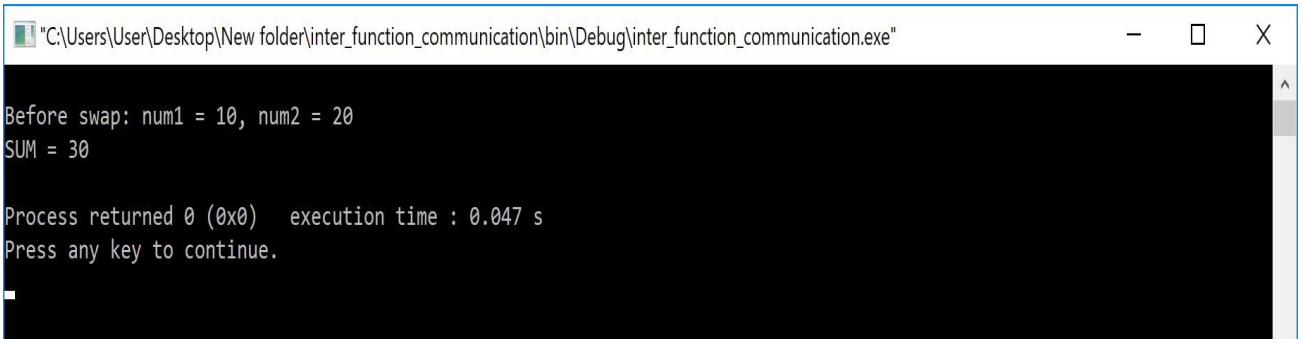
Downward Communication

In this type of inter function communication, the data is transferred from calling function to called function but not from called function to calling function. The functions with parameters and without return value are considered under downward communication. In the case of downward communication, the execution control jumps from calling function to called function along with parameters and executes the function definition, and finally comes back to the calling function without any return value. For example consider the following program...

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

void main(){
  int num1, num2 ;
  void addition(int, int) ; // function declaration
  clrscr() ;
  num1 = 10 ;
  num2 = 20 ;

  printf("\nBefore swap: num1 = %d, num2 = %d", num1, num2) ;
  addition(num1, num2) ; // calling function

  getch() ;
}
void addition(int a, int b)  // called function
{

  printf("SUM = %d", a+b) ;
```

}

Output:



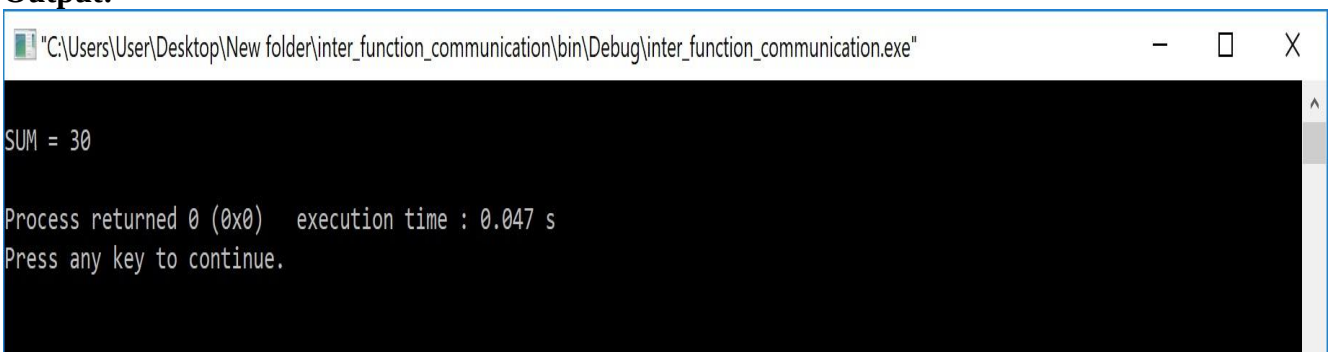
Upward Communication

In this type of inter-function communication, the data is transferred from called function to calling-function but not from calling-function to called-function. The functions without parameters and with return value are considered under upward communication. In the case of upward communication, the execution control jumps from calling-function to called-function without parameters and executes the function definition, and finally comes back to the calling function along with a return value. For example, consider the following program...

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

void main(){
  int result ;
  int addition() ; // function declaration
  clrscr() ;

  result = addition() ; // calling function

  printf("SUM = %d", result) ;
  getch() ;
}
int addition()  // called function
{
```

```
  int num1, num2 ;
  num1 = 10;
  num2 = 20;
  return (num1+num2) ;
}
```

Output:



Bi - Directional Communication:

In this type of inter-function communication, the data is transferred from calling-function to called function and also from called function to calling-function. The functions with parameters and with return value are considered under bi-directional communication. In the case of bi-directional communication, the execution control jumps from calling-function to called function along with parameters and executes the function definition and finally comes back to the calling function along with a return value. For example, consider the following program...

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

void main(){
  int num1, num2, result ;
  int addition(int, int) ; // function declaration
  clrscr() ;

  num1 = 10 ;
  num2 = 20 ;
```

```
  result = addition(num1, num2) ; // calling function


  printf("SUM = %d", result) ;
  getch() ;
}
int addition(int a, int b)  // called function
{
  return (a+b) ;
}
```

Output:

Recursion

A function that calls itself repetitively is known as a *recursive function*. And, this technique is known as ***recursion***. The function calls itself repetitively until certain condition is satisfied.

The recursive function has following types of statements:

- A statement or *condition* to determine if the function is calling itself again.
- A *function call* which should have argument.
- Conditional statement(if-else)
- A *return* statement.

Example: C Program that calculates factorial of a given number using recursion

```
#include <stdio.h>
long int factorial(int);
int main()
{
        int x;
        long int f;
        prinf("Enter a number:");
        scanf("%d",&x);
        f=factorial(x);
        printf("Factorial of %d is %ld\n",x,f);
        return 0;
```

```
}
long int factorial(int n)
{
        if(n==0 ) //base condtion
                return 1;
        else
        return n*factorial(n-1);
}
```

OUTPUT

Factorial of 5 is 120

Parameter passing mechanism:

When a function gets executed in the program, the execution control is transferred from calling-function to called function and executes function definition, and finally comes back to the calling function. When the execution control is transferred from calling-function to called-function it may carry one or number of data values. These data values are called as **parameters.**
In C, there are two types of parameters and they are as follows...

- **Actual Parameters**
- **Formal Parameters**

The **actual parameters** are the parameters that are specified in calling function. The **formal parameters** are the parameters that are declared at called function. When a function gets executed, the copy of actual parameter values are copied into formal parameters.

There are two ways that a C function can be called from a program. They are,

1. Call by value
2. Call by reference

Call by Value

In call by value method, the value of the variable is passed to the function as parameter. The value of the actual parameter can not be modified by formal parameter. Different Memory is allocated for both actual and formal parameters. Because, value of actual parameter is copied to formal parameter.

Example

```
#include<stdio.h>
void swap(int , int );
int main()
{
        int a = 22, b = 44;
        printf("\nValues before swap \n a = %d and b = %d", a, b);
        swap(a,b);
        printf(" \nValues after swap \n a = %d and b = %d", a, b);
        return 0;
```

```
}
void swap(int x, int y)
{
        int temp;
        temp = x;
        x = y;
        y = temp;
}
```

OUTPUT

```
Values before swap
a =22 and b = 44
Values after swap
a =22 and b = 44
```

Call by reference

In *call by reference* method, the address of the variable is passed to the function

as parameter. The value of the actual parameter can be modified by formal parameter. Same memory is used for both actual and formal parameters since only address is used by both parameters.

Example

```
#include<stdio.h>
void swap(int *, int *);
int main()
{
        int a = 22, b = 44;
        printf("\nValues before swap \n a = %d and b = %d", a, b);
        swap(&a,&b);
        printf(" \nValues after swap \n a = %d and b = %d", a, b);
        return 0;
}
void swap(int *x, int *y)
{
        int temp;
        temp = *x;
        *x = *y;
        *y = temp;
}
```

OUTPUT

```
Values before swap
a =22 and b = 44
Values after swap
```

a =44 and b = 22

Standard Functions in C

The standard functions are built-in functions. In C programming language, the standard functions are declared in header files and defined in .dll files. In simple words, the standard functions can be defined as "the ready made functions defined by the system to make coding more easy". The standard functions are also called as **library functions** or **pre-defined functions**.

In C when we use standard functions, we must include the respective header file using **#include** statement. For example, the function **printf()** is defined in header file **stdio.h** (Standard Input Output header file). When we use **printf()** in our program, we must include **stdio.h** header file using **#include<stdio.h>** statement.

C Programming Language provides the following header files with standard functions.

Header File	Purpose	Example Functions
stdio.h	Provides functions to perform standard I/O operations	printf(), scanf()
conio.h	Provides functions to perform console I/O operations	clrscr(), getch()
math.h	Provides functions to perform mathematical operations	sqrt(), pow()
string.h	Provides functions to handle string data values	strlen(), strcpy()
stdlib.h	Provides functions to perform general functions/td>	calloc(), malloc()
time.h	Provides functions to perform operations on time and date	time(), localtime()
ctype.h	Provides functions to perform - testing and mapping of character data values	isalpha(), islower()

Header File	Purpose	Example Functions
setjmp.h	Provides functions that are used in function calls	setjump(), longjump()
signal.h	Provides functions to handle signals during program execution	signal(), raise()
assert.h	Provides Macro that is used to verify assumptions made by the program	assert()
locale.h	Defines the location specific settings such as date formats and currency symbols	setlocale()
stdarg.h	Used to get the arguments in a function if the arguments are not specified by the function	va_start(), va_end(), va_arg()
errno.h	Provides macros to handle the system calls	Error, errno
graphics.h	Provides functions to draw graphics.	circle(), rectangle()
float.h	Provides constants related to floating point data values	
stddef.h	Defines various variable types	
limits.h	Defines the maximum and minimum values of various variable types like char, int and long	

FILES

File is a collection of bytes that is stored on secondary storage devices like hard disk. There are two kinds of files in a system. They are,

6. ASCII Text Files
7. Binary Files

A file has a beginning and an end; it has a current position, typically defined as on many bytes from the beginning. You can move the current position to any other point in file. A new current position can be specified as an offset from the beginning the file.

Types of files

ASCII Text Files

3. A text file can be a stream of charact ers that a computer can process sequentially.
4. It is processed only in forward direction.

- It is opened for one kind of operation (reading, writing, or appending) at any give time.
- It can read only one character at a time.

1. A binary file is collection of bytes.
2. In „C" both a both a byte and a character are equivalent.
3. A binary file is also referred to as a character stream, but there are two essential differences.

6.11 File streams

A stream is a series of bytes of data flow from your program to a file or vice-versa. A stream is an abstract representation of any external source or destination for data, so the keyword, the command line on your display and files on disk are all examples of stream. „C" provides numbers of function for reading and writing t o or from the streams on any external devices. There are two formats of streams which are as follows:

1. Text Stream
2. Binary Stream

Text Stream

1. It consists of sequence of characters, depending on the compilers.
2. Each character line in a text stream may be terminated by a newline character.
3. Text streams are used for textual data, which has a consistent appearance from one environment to another or from one machine to another.

Binary Stream

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

6.12 File Operations

In C, you can perform four major operations on the file, either text or binary:

- Opening a file
- Closing a file
- Reading a file

- Writing in a file

6.12.1 Opening a file

To perform any operation (read or write), the file has to be brought into memory from the storage device. Thus, bringing the copy of file from disk to memory is called opening the file.

The *fopen()* function is used to open a file and associates an I/O stream with it. The general form of fopen() is

fopen(const char *filename, const char * mode);

This function takes two arguments. The first argument is the name of the file to be opened while the second argument is the mode in which the file is to be opened.

NOTE: Before opening a file, we have to create a file pointer that is created using **FILE** structure is defined in the "stdio.h" header file.

For example

```
FILE *fp;
if(fp = fopen("myfile.txt","r")) == NULL)
{
        printf("Error opening a file");
        exit(1);
```

This opens a file named *myfile.txt* in *read* mode.

File opening modes

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, then 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 doesn‟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
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

6.12.2 Closing a file

To close a file and dis-associate it with a stream, use *fc lose()* function. It returns zero if successful else returns EOF if error occurs. The *fc loseall()* closes all the files opened previously. The general form is:

fclose(file pointer);

6.12.3 Reading a file

When we want to read contents from existing file, then we require opening that file into read mode that means “r” mode. To read file follow the below steps

- Initialize the file variable
- Open a file in read mode
- Accept information from file
- Write it into the output devices
- Close the file

Example

```
#include<stdio.h>
int main()
{
        FILE *fp;
        char c;
        fp = fopen("test.txt", "w");
        printf("Enter your Input\n");
        while((c =getchar()) != EOF)
                putc(c,fp);
        fclose(fp);
        printf("\nYou have entered\n");
        fp = fopen("test.txt","r");
        while((c =getc(fp)) != EOF)
                printf("%c ",c);
```

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

Output

Enter your Input
Hi Students…! How are you?
All the best for your end exams. **(press CTRL+Z for stopping)**

C program to copy contents of one file to another file:

```
#include <stdio.h>
#include <stdlib.h> // For exit()

int main()
{
        FILE *fptr1, *fptr2;
        char filename[100], c;

        printf("Enter the filename to open for reading \n");
        scanf("%s", filename);

        // Open one file for reading
        fptr1 = fopen(filename, "r");
        if (fptr1 == NULL)
        {
                printf("Cannot open file %s \n", filename);
                exit(0);
        }

        printf("Enter the filename to open for writing \n");
        scanf("%s", filename);

        // Open another file for writing
        fptr2 = fopen(filename, "w");
        if (fptr2 == NULL)
        {
                printf("Cannot open file %s \n", filename);
                exit(0);
        }

        // Read contents from file
        c = fgetc(fptr1);
        while (c != EOF)
        {
                fputc(c, fptr2);
                c = fgetc(fptr1);
        }
```

```
        printf("\nContents copied to %s", filename);

        fclose(fptr1);
        fclose(fptr2);
        return 0;
}
```

Reading from a file

The reading from a file operation is performed using the following pre-defined file handling methods.

1. **getc()**
2. **getw()**
3. **fscanf()**
4. **fgets()**
5. **fread()**

getc(*file_pointer) **-** This function is used to read a character from specified file which is opened in reading mode. It reads from the current position of the cursor. After reading the character the cursor will be at next character.

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


int main(){


  FILE *fp;
  char ch;
  clrscr();
  fp = fopen("MySample.txt","r");
  printf("Reading character from the file: %c\n",getc(fp));
  ch = getc(fp);
  printf("ch = %c", ch);
  fclose(fp);
  getch();
```

```
  return 0;
}
```

getw(*file_pointer) **-** This function is used to read an integer value form the specified file which is opened in reading mode. If the data in file is set of characters then it reads ASCII values of those characters.

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

int main(){

  FILE *fp;
  int i,j;
  clrscr();
  fp = fopen("MySample.txt","w");
  putw(65,fp);  // inserts A
  putw(97,fp);  // inserts a
  fclose(fp);
  fp = fopen("MySample.txt","r");
  i = getw(fp);   // reads 65 - ASCII value of A
  j = getw(fp);   // reads 97 - ASCII value of a
  printf("SUM of the integer values stored in file = %d", i+j);  // 65 + 97 = 162
  fclose(fp);
  getch();

  return 0;
}
```

fscanf(*file_pointer, typeSpecifier, &variableName) - This function is used to read multiple datatype values from specified file which is opened in reading mode.

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



int main(){


  char str1[10], str2[10], str3[10];
  int year;
  FILE * fp;
  clrscr();
  fp = fopen ("file.txt", "w+");
  fputs("We are in 2016", fp);
  rewind(fp);    // moves the cursor to begining of the file
  fscanf(fp, "%s %s %s %d", str1, str2, str3, &year);
  printf("Read String1 - %s\n", str1 );
  printf("Read String2 - %s\n", str2 );
  printf("Read String3 - %s\n", str3 );
  printf("Read Integer - %d", year );
  fclose(fp);
  getch();


  return 0;
}
```

fgets(variableName, numberOfCharacters, *file_pointer) - This method is used for reading a set of characters from a file which is opened in reading mode starting from the current cursor position. The fgets() function reading terminates with reading NULL character.

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

```
int main(){

  FILE *fp;
  char *str;
  clrscr();
  fp = fopen ("file.txt", "r");
  fgets(str,6,fp);
  printf("str = %s", str);
  fclose(fp);
  getch();

  return 0;
}
```

fread(source, sizeofReadingElement, numberOfCharacters, FILE *pointer) - This function is used to read specific number of sequence of characters from the specified file which is opened in reading mode.

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


int main(){

  FILE *fp;
  char *str;
  clrscr();
  fp = fopen ("file.txt", "r");
  fread(str,sizeof(char),5,fp);
  str[strlen(str)+1] = 0;
  printf("str = %s", str);
```

```
  fclose(fp);
  getch();

  return 0;
}
```

Writing into a file

The writing into a file operation is performed using the following pre-defined file handling methods.

1. **putc()**
2. **putw()**
3. **fprintf()**
4. **fputs()**
5. **fwrite()**

- ***putc(char, *file_pointer) -*** This function is used to write/insert a character to the specified file when the file is opened in writing mode.

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


int main(){

  FILE *fp;
  char ch;
  clrscr();
  fp = fopen("C:/TC/EXAMPLES/MySample.txt","w");
  putc('A',fp);
  ch = 'B';
  putc(ch,fp);
  fclose(fp);
  getch();
```

```
  return 0;
}
```

putw(int, *file_pointer) - This function is used to writes/inserts an integer value to the specified file when the file is opened in writing mode.

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


int main(){

  FILE *fp;
  int i;
  clrscr();
  fp = fopen("MySample.txt","w");
  putw(66,fp);
  i = 100;
  putw(i,fp);
  fclose(fp);
  getch();

  return 0;
}
```

fprintf(*file_pointer, "text") - This function is used to writes/inserts multiple lines of text with mixed data types (char, int, float, double) into specified file which is opened in writing mode.

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


int main(){

  FILE *fp;
```

```
  char *text = "\nthis is example text";
  int i = 10;
  clrscr();
  fp = fopen("MySample.txt","w");
  fprintf(fp,"This is line1\nThis is line2\n%d", i);
    fprintf(fp,text);
  fclose(fp);
  getch();

  return 0;
}
```

fputs("string", *file_pointer) - TThis method is used to insert string data into specified file which is opened in writing mode.

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


int main(){

  FILE *fp;
  char *text = "\nthis is example text";
  clrscr();
  fp = fopen("MySample.txt","w");
  fputs("Hi!\nHow are you?",fp);
  fclose(fp);
  getch();

  return 0;
}
```

fwrite("StringData", sizeof(char), numberOfCharacters, FILE *pointer) - This function is used to insert specified number of characters into a binary file which is opened in writing mode.

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


int main(){

  FILE *fp;
  char *text = "Welcome to C Language";
  clrscr();
  fp = fopen("MySample.txt","wb");
  fwrite(text,sizeof(char),5,fp);
  fclose(fp);
  getch();

  return 0;
}
```

Closing a file

Closing a file is performed using a pre-defined method fclose().

fclose(*f_ptr)
The method fclose() returns '0'on success of file close otherwise it returns EOF (End Of File).