# Experiment No.1

**Aim:** Write a C program,

1. To print your name and address
2. Find simple and compound interest

**Theory:**

1. To print your name and address

Algorithm:

1. Start
2. Include header file
3. Use printf() to print name and address
4. Put return 0 statement at the end of program
5. Stop

The preprocessor directive being at the first line, "#include<stdio.h>," instructs the compiler to include the standard input/output library. This library contains the "printf()" function, which we will use to print the message to the console.
The next line '**int main()** {' is the main function of our program. The execution of the program begins at this time. The word "int" before the word "**main()**" denotes that the function returns an integer number. Integer is one of the datatype in C programming language.
The next line '**printf("My name is Sachin\n");**' is the heart of this program. The command "**printf()**" or also called as function is used to print " **My name is Sachin** " to the console. The special character '**\n**' at the conclusion of the string stands for a newline character, which causes the cursor to move to the next line after the message has been printed.
The last statement, "**return 0;**" is used to end the program and it means that int main() has to return something.
Final step is to compile and run the program which will print name and address on the console if compiled successfully.

2. Find simple and compound interest

Algorithm: Simple Interest

1. Start
2. Include header file
3. Consider variables like P,R,T,SI with appropriate datatype
4. Perform SI=P*R*T/100
5. Print value of SI
5. Stop

Program will start with header file "#include<stdio.h>", additionally one more header file required "#include<math.h>" as we are going to use 'power' function. Variable is name of memory location where we store the data/value. This name can be any with some rules. Datatype is the concept which specifies what type data we are going to use in a program. Both variable and datatype is required to declare different variables like p,r,t,si.

Syntax to declare variable,

datatype variable_name;

Example:

int a;

Next values assignment is required for all considered variables called as definition.

Example:

int a=50;

Considered variables in this program are as follows,

p = principal

r = rate of interest

t = time

si = to store result

We need to define above variables as follows,

int p = 50000;

int r = 7;

int t = 5;

int si;

Next is put above values in given formula which is as follows,

si=p*r*t;

Once above calculation is over, result will be stored in si variable.

At last we will use printf() to print the result which is stored in si.

Algorithm: Compound Interest

1. Start
2. Include header file
3. Consider variables like P,R,T,CI,A with appropriate datatype
4. Perform A=P*pow((1+R/100))^T

   CI=A-P;
5. Print value of CI
5. Stop

Considered variables in this program are as follows,

p = principal

r = rate of interest

t = time

a = to store result

ci = to store result(compound interest)

We need to define above variables with float datatype as follows,

float p = 50000;

float r = 7;

float t = 5;

float a;

float ci;

Next is put above values in given formula which is as follows,

a= p*pow((1+r/100))^t

ci=a-p

Once above calculation is over, result will be stored in ci variable.

At last we will use printf() to print the result which is stored in ci.

**Conclusion:**

In this way we have successfully implemented programs which prints name, address and finds simple, compound interest by using various concepts like header file, int main(), printf(), variable, datatype etc.

# Experiment No.2

**Aim:** Write a C program to perform basic arithmetic operations using switch case.

**Theory:**

We can have a single program which will perform all the basic arithmetic operations like addition, subtraction, multiplication and division. To perform such operations arithmetic operators are required. Following are the arithmetic operators in C language,

Plus (+) Operator – To perform addition

Minus (-) Operator – To perform subtraction

Multiplication (*) Operator – To perform multiplication

Division (/) Operator – To perform division

We can use above operators on operands. An operator will operate on given data/values called as operand. Appropriate datatype to be included in given program to handle data/values.

Example:

```
int a=10,b=5,c;

c=a+b;  // addition

int a=10,b=5,c;

c=a-b;  // subtraction

int a=10,b=5,c;

c=a*b;  // multiplication

int a=10,b=5,c;

c=a/b;  // division
```

Switch Case:

Above program to be implemented by using switch case. Switch Case is concept in C language which will allow to execute different operations/different possibilities which are given in a program. Switch Case statement provides control in execution as per the requirements.

Syntax:

```
switch (appr. variable)

{

        case 1:
        // given code
        break;

        case 2:
        // given code
        break;
        case 3:
        // given code
        break;
        default:
```

```
        // given code
}
```

Once appropriate variable matched with switch case statement it will check inside switch case. If case 1 matches with choice/input provided by an user then corresponding code will get executed and control will come out of case 1 as ‘break’ has been provided. Meaning of break is it simply breaks the current case and go to the next line of program. If not then it will keep checking with each case written in a program. If no case has been match then code written in default case will get executed.

Algorithm:

1. Start
2. Include header file
3. Define variables int a,b,c, with values
4. Perform addition, subtraction, multiplication and division
5. Save the result in c variable
6. Write switch case to give choice
7. Include all above operations in different cases of switch case
8. Give appropriate choice/input to program to check an output
9. Stop

**Conclusion:**

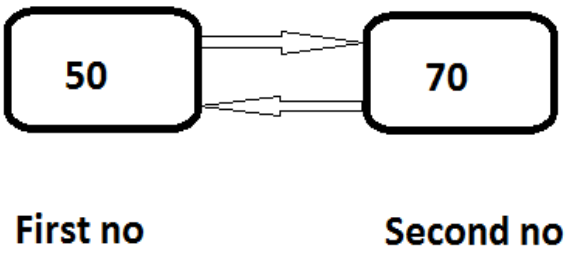
In this way we have successfully implemented program which performs all basic arithmetic operations like addition, subtraction, multiplication and division by using switch case statement which allows to have control in execution of a given program.

# Experiment No.3

**Aim:** Write a C program to swap two numbers using third variable.

**Theory:**

It is possible to perform swapping operation in C language. Here we need to swap two numbers by using third variable. Swapping means interchanging the given two numbers. In this program, we will take data/numbers from user. 'scanf()' is use to get the data/numbers from an user.




Syntax:

scanf("format specifier", & var_name);

Example:

scanf("%d", &a);

%d is format specifier which tells compiler that the type of data/numbers being given as an input. & refers the memory address and a is name of variable through which input will be provided by an user.

Algorithm:

1. Start
2. Include header file
3. Declare variable like int a,b,c
4. Use scanf() to take both numbers as an input with variable a and b
5. Perform swapping by using third variable c
6. c=a;

   a=b;

   b=c;
7. Print both numbers after swapping
8. Stop

With the help of scanf() we will take both numbers as an input.

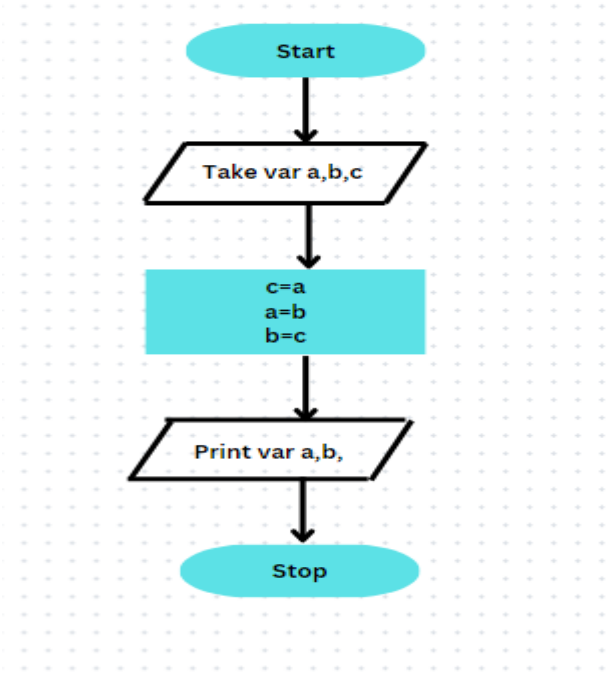
c=a; // at first copy value of a into c

a=b;  // then copy value of b into a

b=c;  // and finally copy value of c into b

Above simple logic will swap to number as provided by an user and we will print it by using printf().

Flowchart: Graphical representation




**Conclusion:**

In this way we have successfully implemented program for swapping of two numbers by using third variable. We have performed swapping of two numbers entered by an user.

# Experiment No.4

**Aim:** Write a C program to print Fibonacci series.

**Theory:**

With respect to fibonacci series, next number is the addition of previous two numbers. The first two numbers in fibonacci series are 0 and 1 respectively. It possible to print such series in C language by using one of control statement called as 'for loop'.

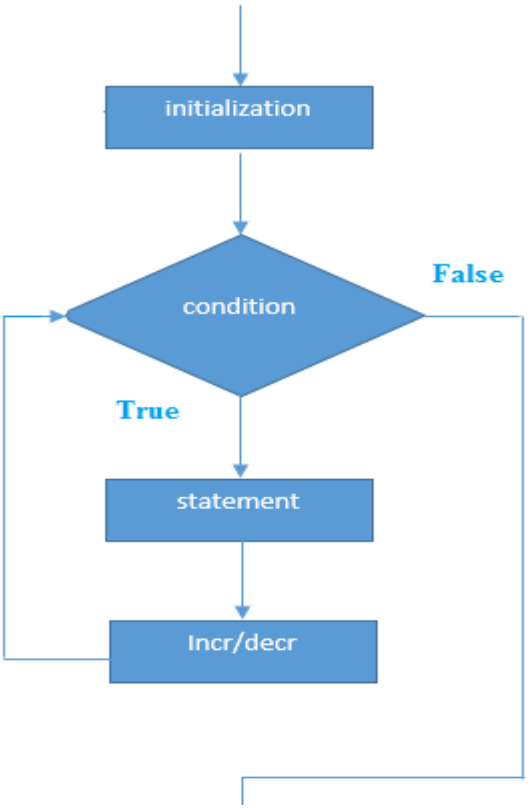
Basically for loop provides flexibility in writing the program for given problem statement. When we want to execute specific code many times we can use for loop. It avoids writing of repetitive code. It means use of for loop makes given program more efficient.

Syntax:

for(initialization;condition;increment/decrement)

Example:

for(i=1;i<5;i++)




Programming example:

```
#include<stdio.h>
int main()

{
        int i=1;
        for(i=1;i<5;i++)
        {
                printf("This is for loop\n");
        }
        return 0;
```

}

Output:

This is for loop
This is for loop
This is for loop
This is for loop

Algorithm:

1. Start

2. Declare variables n1=0,n2=1,a,i,n3;

3. Print n1 and n2

4. Use for(i=2;i<10;i++)

5. Perform n3=n1+n2

6. Print value of n3

7. Update values of n1 and n2 like

    n1=n2

    n2=n3

8.Stop

We have to declare variables n1=0,n2=1 as these numbers are at starting of Fibonacci series. By using printf() function we will print the values of n1 and n2.

Next we have to use for(i=2;i<10;i++) loop which will iterate for 10 times and in each iteration we are performing n3=n1+n2 which is nothing but the addition of previous two numbers to get the right number at next position. n3 will provide us next numbers in the series each time for loop executes.

Then we need to update values of n1 and n2 like,

n1=n2
n2=n3

After updating values like above, value of i will be incremented by 1and for loop will execute as written.
This way with each iteration of for loop next number in the Fibonacci series will get printed.

**Conclusion:** In this way we have successfully implemented program in which we have used for loop to print the Fibonacci series till given condition.

# Experiment No.5

**Aim:** Write a C program to print prime number.

**Theory:** Number which is divisible by 1 or itself called as prime number. 0 and 1 are not prime numbers. 2 is the only even prime number. We should have a program which will handle given explanation.

Algorithm:

1. Start
2. Input any number var a
3. Check whether given number is equal to 1 or 2 and show message
4. Check whether given number is if(a%==0), print that it is not prime number
5. Check whether given number is if(a%!=0), print that it is prime number
6. Stop

Here we can use goto statement. When we used it, it actually breaks execution and jumps to the expected program line.

Syntax: goto label_name;

```
label_name:
{
  ------------
  ------------
}
```

Example: goto demo;

```
demo:
{
  ------------
  ------------
}
```

Program:

```
#include<stdio.h>
int main()
{
        int a;
        printf("enter no");
        scanf("%d",&a);
```

```
        if(a==1)
        {
                printf("1 is not prime\n");
                goto demo;
        }
        if(a==2)
        {
                printf("2 is prime\n");
                goto demo;
        }
        if(a%2==0)
        {

                printf("not prime\n");
        }

        else
        {
                printf("prime\n");
        }
        demo:
                {

                        printf("End of program");
                }
    return 0;
}
```

Value of a will be taken from user. if(a==1) it will show it is not prime number message and control will go to demo. if(a==2) it will show it is prime number message and control will go to demo. And at last if(a%2==0) it will show it is not prime number message and control will go to demo, else it will show it is prime number message and control will go to demo.

**Conclusion:** In this way, we have successfully checked whether given number is prime or not.

# Experiment No.6

**Aim:** Write a C program to sort an array of integers.

**Theory:** An Array is a collection of related data items which will share a common name.

It is a finite collection of values of similar data type stored in adjacent memory locations that is accessed using one common name.

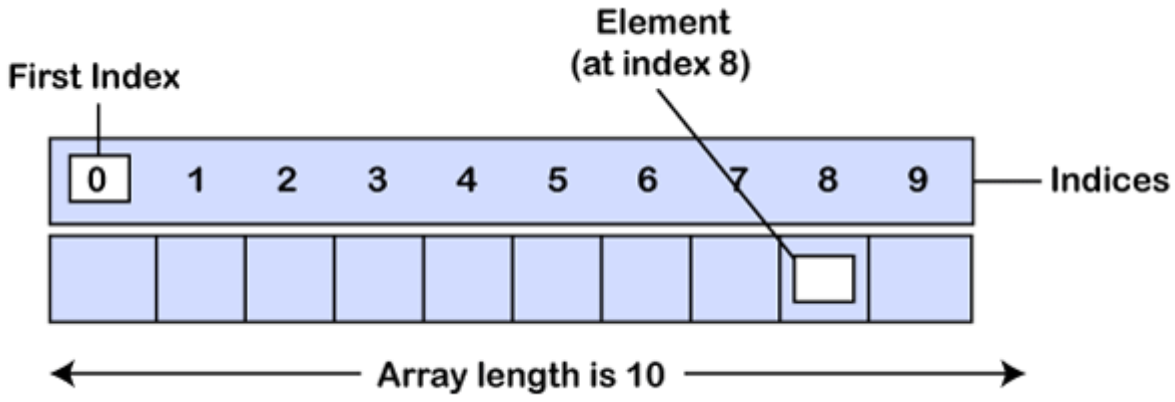
Each value of the array is called as an element. Elements are accessed using index, which is a number representing the position of the element. By finite we mean that, there are specific number of elements in an array and by similar we mean that, all elements in the array are of same type.

Example of an Array:

Suppose we have to store the roll numbers of the 100 students the we have to declare 100 variables named as roll1, roll2, roll3, ……. roll100 which is very difficult job. Concept of C programming arrays is introduced in C which gives the capability to store the 100 roll numbers in the contiguous memory which has 100 blocks and which can be accessed by single variable name.

1. C Programming Arrays is the Collection of Elements
2. C Programming Arrays is collection of the Elements of the same data type.
3. All Elements are stored in the Contiguous memory
4. All elements in the array are accessed using the subscript variable (index).

Pictorial representation of C Programming Arrays




**Declaring an array** -An array is declared just like an ordinary variable but with the addition of number of elements that the array should contain. A particular value is indicated by writing a number called 'index or subscript' number in brackets for after the array name.

Syntax: **type variable name[size];**

The type specifies data type of element that will be contained in an array such as int, float, char, and double. The size indicates the maximum number of elements that can be stored inside the array.

E.g. The following declaration declares n array of 20 elements where each element is of int type.

**int marks [20];**

Array can be of any variable type. The ability to use a single name to represent a collection to refer to an item by specifying the item number enables us to develop efficient programs.

Program-

```
#include<stdio.h>
int main()
{
	int a,i,j,temp;
	int arr[5]={20,17,35,9,37};
	for(a=0;a<5;a++)
	{
		printf("%d ",arr[a]);
	}
	for(i=0;i<5;i++)
	{
		for(j=i+1;j<5;j++)
		{
			if(arr[i]>arr[j])
			{
				temp=arr[i];
				arr[i]=arr[j];
				arr[j]=temp;
			}
		}
	}
	printf("\n");
	for(a=0;a<5;a++)
	{
		printf("%d ",arr[a]);
	}
	return 0;
}
```

Output - 20 17 35 9 37
9 17 20 35 37

**Conclusion:** In this way we have understood the concept of array and used the same concept to sort an integers successfully.

# Experiment No.7

**Aim:** Write a C program to take input of name, roll no and marks obtained by a student in 4 subject of 100 marks each and display the name, roll no with percentage score secured**.**

**Theory:** A structure is a container to store multiple types of variables. The variables stored in a structure are called member variables. Member variables are related to each other, and they are combined to provide complete information about an object.

- The structure is declared using the "struct" keyword.
- Structure allocates contiguous memory to all its member variables.
- You can only declare member variables in a structure.
- You cannot initialize the member variables during their declaration.
- To access the member variables, use the dot operator.
- Structure member variables are initialized in the same order as their declaration.

Syntax:

```
struct structure_name
{
        data_type member1;
        data_type member2;
        .
        .
        data_type member n;
}var_name;
```

Example:

```
struct emp
{
        int id;
        char name;
        float sal;
}s;   // to access structure members
```

Program:

```
#include<stdio.h>
struct student
{
        int r;
        char a[10];
        float m1,m2,m3,p;
}s;

int main()
{
        printf("Accept info\n");
        printf("Enter roll no\n");
        scanf("%d",&s.r);
        printf("Enter name\n");
        scanf("%s",&s.a);
```

```
        printf("Enter sub 1 marks\n");
        scanf("%f",&s.m1);
        printf("Enter sub 2 marks\n");
        scanf("%f",&s.m2);
        printf("Enter sub 3 marks\n");
        scanf("%f",&s.m3);

        printf("Display info\n");
        printf("Roll no=%d\n",s.r);
        printf("Name=%s\n",s.a);
        s.p=((s.m1+s.m2+s.m3)/300)*100;
        printf("Percentage=%.2f",s.p);

        return 0;
}
```

Output:

```
        Accept info
        Enter roll no 10
        Enter name Sachin
        Enter sub 1 marks 50
        Enter sub 1 marks 60
        Enter sub 1 marks 40

        Display info
        Roll no=10
        Name=Sachin
        Percentage=50.00
```

**Conclusion:** In this way we have understood the concept of structures and used the same concept to take input of name, roll no and marks obtained by a student in 4 subject of 100 marks each and display the name, roll no with percentage score secured**.**

# Experiment No.8

**Aim:** Write a C program to implement concept of pointer.

**Theory:**

Pointers in C are variables that store memory addresses. They are used to directly access and manipulate memory locations, providing a powerful mechanism for efficient memory management and dynamic allocation.

Key Points about Pointers:

- Memory Address Storage: Pointers store memory address of the variables. They "point" to the location in memory where data is stored.
- Declaration: Pointers are declared using an asterisk(*) before the variable name, indicating that the variable is a pointer. For example: int *ptr; declares a pointer named ptr that can point to an integer.
- Address-of Operator (&): The & operator is used to get the memory address of a variable. For instance: &num gives the address of the variable num.
- Dereferencing Operator (*): The * operator is used to access the value store data pointer's memory address. For example: *ptr accesses the value at the memory location pointed to by ptr.
- Pointer Arithmetic: Pointers can be incremented, decremented, and manipulated using arithmetic operations to navigate through memory addresses.
- Dynamic Memory Allocation: Pointers are used in functions like malloc() and free() to dynamically allocate and deallocate memory at runtime, enabling flexible memory usage.
- Pointer to Pointer: Pointers can also point to other pointers, allowing multi-level indirection.
- Passing Pointers to Functions: Pointers are commonly used to pass addresses of variables to functions, allowing functions to modify the original data.

Algorithm:

1. Start

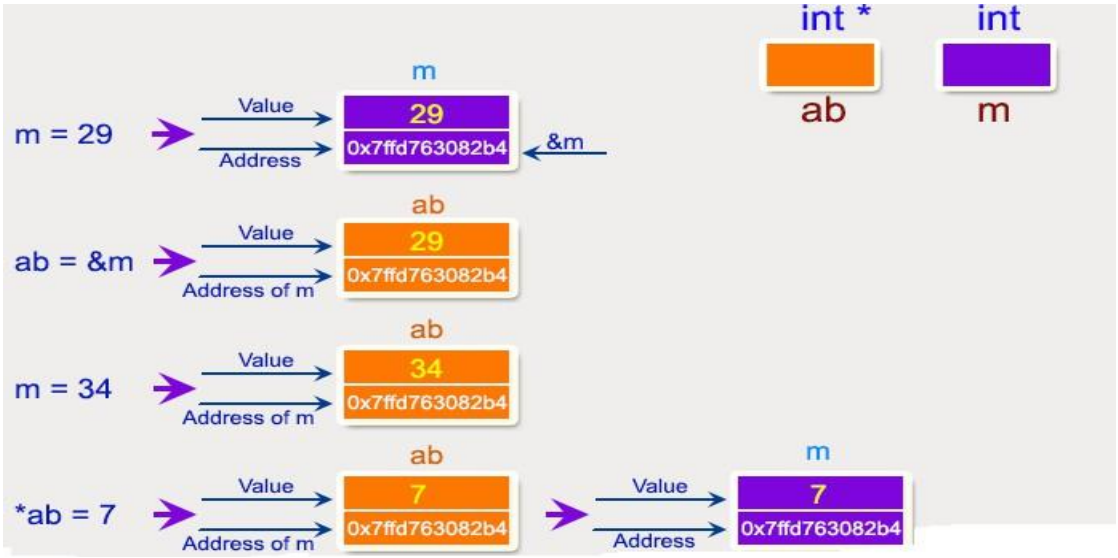
2. Declare an integer variable num.

3. Declare a pointer variable ptr of type integer.

4. Assign the address of num to the pointer variable ptr.

5. Print values and address of num and ptr
6. Stop

Pictorial Representation:




Program:

```
#include<stdio.h>
int main()
{
    int num = 10;     //Declare an integer variable num and assign it the value 10
    int *ptr;         // Declare a pointer variable ptr of integer type
    ptr = &num;       //Assign the address of num to the pointer ptr using the & operator
    printf("Value of num: %d\n", num);          // Print the value of num directly
    printf("Address of num: %p\n", &num);   // Print the address of num
    printf("Value of  ptr :%d\n",*ptr);    // Print the value of ptr
    printf("Address of  ptr :%p",&ptr);  // Print the address of ptr
    return 0;
}
```

Output:

Value of num: 10

Address of num: 00000AXG

Value of  ptr: 10

Address of  ptr: 0000HFE

**Conclusion:**

In this C program, we've explored the fundamental concept of pointers, showcasing their pivotal role in memory management and data manipulation. By utilizing pointers, we've demonstrated how to access, modify, and interact with memory addresses directly, empowering us to create more efficient and flexible code. Through exercises involving

pointers, we've highlighted their significance in tasks like dynamic memory allocation, array handling, and the creation of intricate data structures. Understanding pointers is essential in harnessing the full potential of the C language, enabling programmers to optimize memory usage, enhance program efficiency, and tackle complex programming challenges with precision and control.

# Experiment No.9

**Aim:** Write a C program to implement concept of file handling.

**Theory:**
File handling in C involves working with files on the system, enabling reading from and writing to files. It allows programs to interact with external files, store data persistently, and manipulate file contents. File handling in C is the process of working with files in a program. It involves operations such as reading from and writing to files, manipulating file contents, and managing file resources. The Standard C Library provides functions and tools for file handling, allowing programs to interact with files on the system.

Key Functions in File Handling:

fopen() - Opening a File:

- Syntax: FILE *fopen(const char *filename, const char *mode);
- Opens a file named filename in a specified mode ("r", "w", "a", "r+", "w+", "a+", etc.). Modes indicate if the file is opened for reading, writing, or appending. If the file doesn't exist in write or append mode, a new file is created.
- Syntax: FILE *fopen(const char *filename, const char *mode);
- The fopen() function is used to open a file by specifying the file name and mode ("r", "w", "a", etc.).
- It returns a file pointer that is used for subsequent file operations.

fclose() - Closing a File:

- Syntax: intfclose(FILE *stream);
- The fclose() function is used to close a file once all necessary operations are done.
- It flushes any buffered data to the file and releases the file resources.

Reading from a File - Writing to a File:

- Syntax: intfprintf(FILE *stream, const char *format, ...);
- Syntax: intfscanf(FILE *stream, const char *format, ...);
- Writes formatted data to the file pointed to by stream. Works similar to printf() but writes to a file instead of the standard output.
- fprintf() allows writing various data types to a file using format specifiers.
- Functions like fprintf(), fscanf(), fputc(), fgetc(), fgets(), and fputs() are used to perform read and write operations on files.
- fprintf() and fscanf() are used for formatted read and write operations.
- fputc() and fgetc() perform character-based read and write operations.
- fgets() and fputs() handle string-based read and write operations.
- Reads formatted data from the file pointed to by stream. Similar to scanf() but reads from a file instead of the standard input.

- fscanf() reads various data types from a file using format specifiers.

Other Functions:

- feof(): Checks for the end-of-file indicator.
- fgetc(), fputc(): Read and write characters respectively.
- fgets(), fputs(): Read and write strings respectively.
- rewind(), fseek(), ftell(): Move the file pointer to a specific location or retrieve the current position.

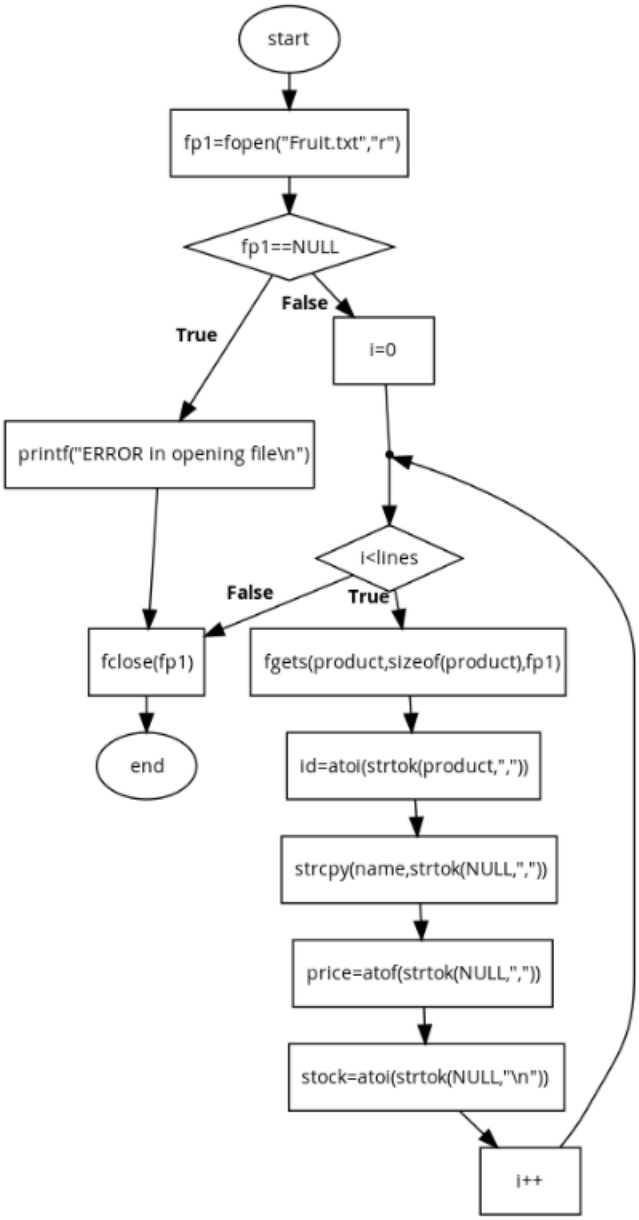
File Modes:

- "r": Read mode. Opens a file for reading. The file must exist.
- "w": Write mode. Opens a file for writing. If the file exists, it truncates it; if not, it creates a new file.
- "a": Append mode. Opens a file for writing at the end of the file. If the file doesn't exist, it creates a new file.
- "r+": Read/update mode. Opens a file for both reading and writing. The file must exist.
- "w+": Write/update mode. Opens a file for reading and writing. If the file exists, it truncates it; if not, it creates a new file.
- "a+": Append/update mode. Opens a file for reading and writing at the end of the file. If the file doesn't exist, it creates a new file.

File Handling Steps:

- Open the File: Use fopen() to open the file in the desired mode.
- Perform Operations: Use functions like fprintf(), fscanf(), fgetc(), fputc(), etc., to read from or write to the file.
- Close the File: Use fclose() to close the file after performing all necessary operations.

Flow Chart:




Example Usage:

```
FILE *filePointer;

filePointer = fopen("example.txt", "w");

if (filePointer != NULL) {

fprintf(filePointer, "File handling in C");

fclose(filePointer);

} else {

printf("File could not be opened.\n");

}
```

- #include<stdio.h> : Include the standard I/O library header file.
- int main() { ... } : Entry point of the program; the main function.
- FILE *filePointer; : Declaration of a file pointer.
- filePointer = fopen("example.txt", "w");
  : Opens a file named "example.txt" in write mode ("w"). If the file doesn't exist, it creates one; if it exists, it truncates it.
- if (filePointer == NULL) { ... } : Checks if the file was opened successfully. If the file couldn't be opened, it prints an error message and exits the program.
- fprintf(filePointer, "Hello, File Handling in C!");
  : Writes data ("Hello, File Handling in C!") to the file using fprintf().
- fclose(filePointer); : Closes the file using fclose().
- return 0; : Indicates successful termination of the main() function.

**Conclusion:**

In this C program, we've delved into the realm of file handling, unveiling the capabilities that allow us to interact with external files seamlessly. Through file opening, reading, writing, and closing operations, we've showcased how to manipulate file content, demonstrating the significance of file pointers and file streams in managing file data efficiently. The program illustrates the versatility of C's file handling functions, enabling us to create, modify, and access files, thereby facilitating tasks such as data storage, retrieval, and processing. Mastering file handling in C equips programmers with the ability to integrate external data seamlessly into their applications, enabling robust file management and manipulation for diverse real-world applications.

# Experiment No.10

**Aim:** Write a C program to create calculator using function.

**Theory:**

Program Overview:

The program prompts the user to enter an arithmetic operator (+, -, *, /).

It then asks for two numbers.

Based on the operator selected, the program performs the corresponding arithmetic operation using separate functions for each operation.

Algorithm for Simple Calculator Program:

Include Required Header:

Include the standard input/output library (<stdio.h>).

Function Declarations:

Declare functions for addition, subtraction, multiplication, and division (add(), subtract(), multiply(), divide()).

Each function takes two floating-point numbers as parameters and returns a floating-point result.

Main Function:

Declare variables: operator, number1, number2, and result.

Prompt the user to input an arithmetic operator (+, -, *, /).

Read the operator using scanf().

Prompt the user to input two numbers.

Read the numbers using scanf().

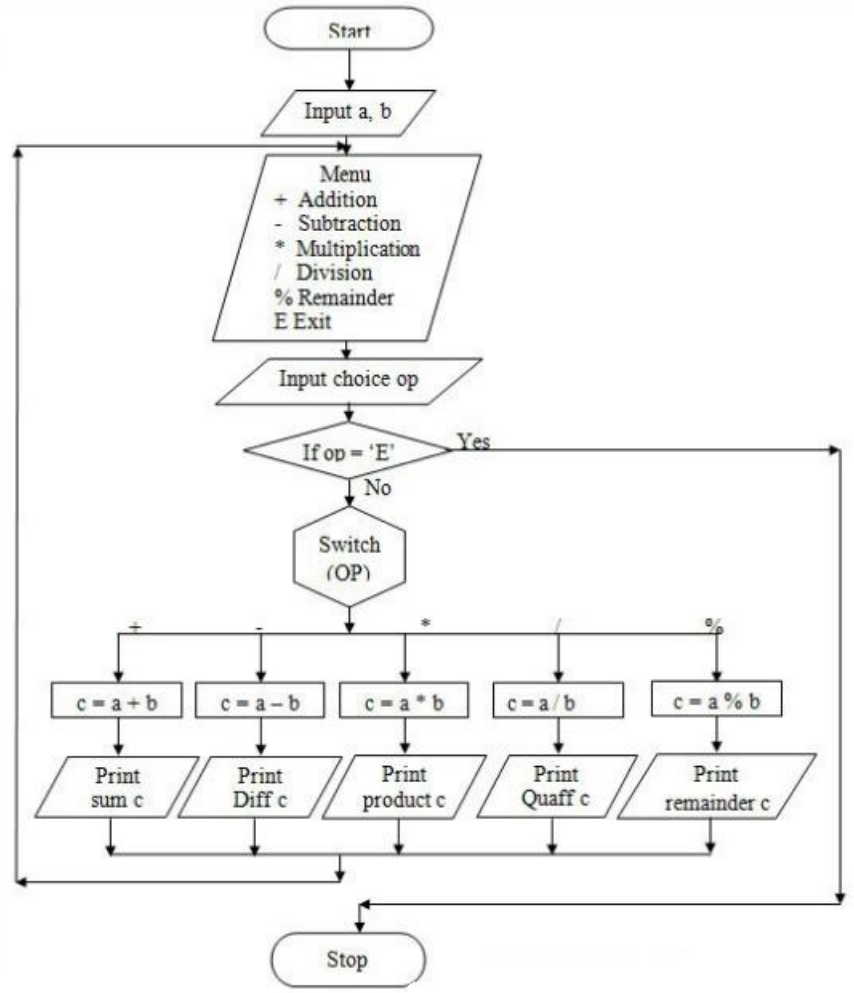
Operator Selection (Switch Statement):

Use a switch statement to perform the selected arithmetic operation.

Call the corresponding function based on the entered operator.

Store the result in the result variable.

If an invalid operator is entered, print an error message and exit the program.




Arithmetic Function Definitions:

Define functions for addition, subtraction, multiplication, and division.

Implement the arithmetic operations within each function.

Return the calculated result.

Error Handling for Division by Zero:

In the divide() function, check if the second number (num2) is zero.

If num2 is zero, print an error message and return zero.

Output Result:

Display the calculated result of the arithmetic operation using printf().

Example:

Input:

Operator: * (multiplication)

Numbers: 5, 3

Output:

Result: 5 * 3 = 15

This algorithm outlines the steps involved in the program, from handling user input for the operator and numbers to performing the arithmetic operation and displaying the result or handling errors. Error handling is implemented for division by zero.

**Conclusion:**

In this C program, we've constructed a versatile calculator using functions, showcasing the modularity and reusability that functions offer in programming. By breaking down the calculator operations into separate functions for addition, subtraction, multiplication, division, and more, we've illustrated how functions streamline the code, enhance readability, and enable easy maintenance. This implementation emphasizes the importance of structuring code into smaller, specialized functions, enabling us to build complex functionalities while maintaining code clarity and organization. Leveraging functions in this calculator program not only demonstrates their role in code organization but also highlights their ability to encapsulate logic, promoting code efficiency and facilitating future scalability and modifications.