**C Programming**

**Document is made with intention to cover all the topic present in the syllabus as well as to cover list of experiment. I am using gcc compiler some things might vary from compiler to compiler.**

**Experiment no.01**

**Aim:** a) Program to demonstrate Operators Data Input and Output – getchar( ), putchar( ), scanf( ), printf( ), gets( ), puts( )

b) Program to demonstrate Operators-Arithmetic, Relational and logical, Assignment, Unary, Conditional, Bitwise, Comma, other operators

**Requirements:** C Compiler

**Theory:**

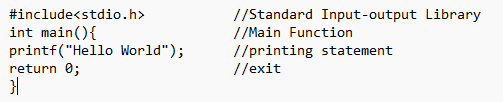
#### Data Input and Output Functions:

1. **getchar() and putchar()**
   * **getchar():** Reads the next character from standard input (keyboard) and returns it. It’s typically used for reading single characters.
   * **putchar():** Writes a single character to standard output (screen). It’s commonly used to output characters one at a time.
2. **scanf() and printf()**
   * **scanf():** Reads formatted input from standard input. It allows you to read various types of data, including integers, floats, and strings, based on specified format specifiers.
   * **printf():** Outputs formatted data to standard output. It can handle multiple data types and provides formatting options for displaying data.
3. **gets() and puts()**
   * **gets():** Reads an entire line of input from standard input until a newline character is encountered. Note: This function is unsafe and has been removed from the C standard library due to potential buffer overflow issues.
   * **puts():** Outputs a string to standard output and appends a newline character. It’s used for printing strings followed by a new line.

**Operators in C:**

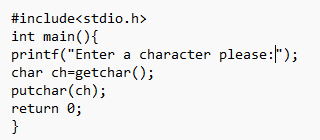
1. Arithmetic Operators (+, -, \*, /, %)
   * Used for basic mathematical operations.
2. Relational Operators (==, !=, >, <, >=, <=)
   * Used for comparing two values.
3. Logical Operators (&&, ||, !)
   * Used for logical operations and combining conditions.
4. Assignment Operators (=, +=, -=, \*=, /=, %=)
   * Used for assigning values to variables and updating them.
5. Unary Operators (+, -, ++, --, !)
   * Operate on a single operand.
6. Conditional (Ternary) Operator (?:)
   * Used to return one of two values based on a condition.
7. Bitwise Operators (&, |, ^, ~, <<, >>)
   * Operate on binary representations of integers.
8. Comma Operator (,)
   * Allows two expressions to be evaluated in sequence.

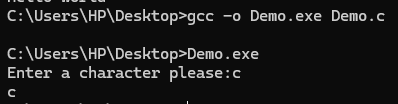
Q.1 Write a program to print “Hello World”.(10 min)



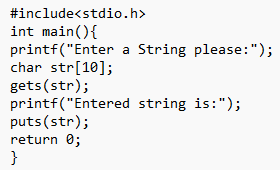


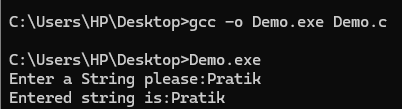
Q.2 Write a program to take character as a input from user and print it.(Use of getchar(),putchar()).(10 min)



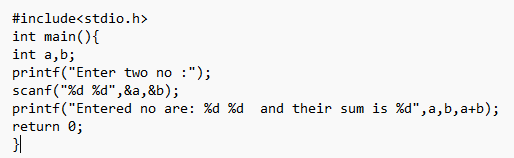


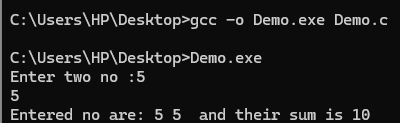
Q.3 Write a program to take String as input from user and print it.(Use of gets(),puts()).(10 min)



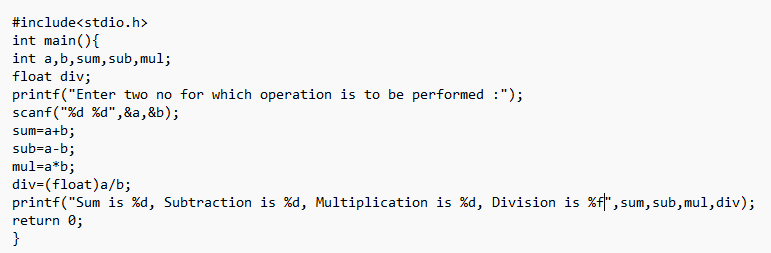


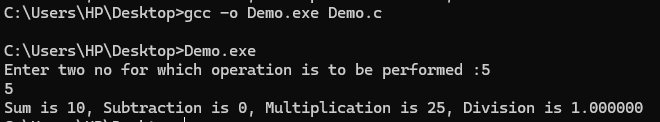
Q.4 Write a program to take two numbers as input from user and display sum of it.(Scanf(), printf()).(10 min)





Q.5 Write a program to perform Addition,Subtraction,Multiplication,Division (Arithmatic operators)(10 min)

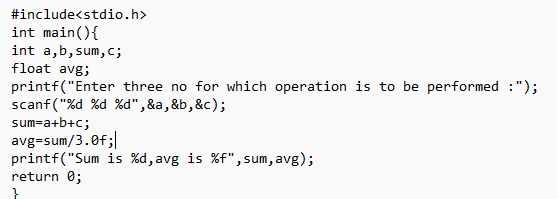


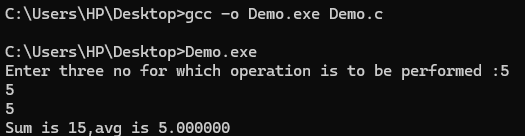


Other operators we will cover in the class.

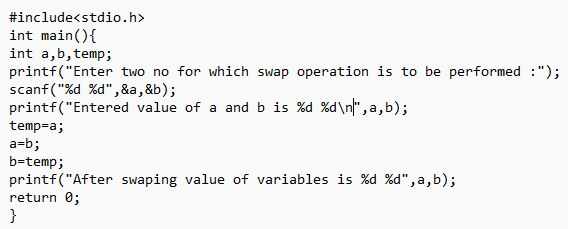
Practice Questions:

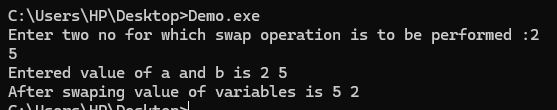
Q.1 WAP to calculate Sum and Average of three numbers.



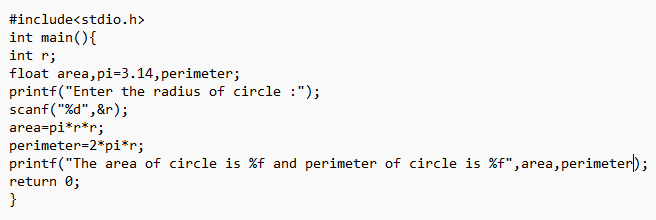


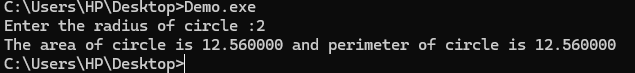
Q.2 WAP to interchange/Swap of Two variables.





Q.3 WAP to find Area and perimeter of circle.





**Conclusion:** We understood the concept and then did some question on it. We Successfully fulfilled the aim of experiment.

**Experiment No.02**

**Aim:** a) Program to demonstrate Branching - If statement, If-else Statement, Multiway decision. b) Program to demonstrate Looping – while, do-while

**Requirements:** C Compiler

**Theory:**

#### Branching Statements:

**1.if Statement**

Purpose: Executes a block of code if a specified condition is true.  
if (condition) {

// code to execute if condition is true }

**2.if-else Statement**

Purpose: Executes one block of code if the condition is true, and another block if the condition is false.

if (condition) {

// code to execute if condition is true

} else {

// code to execute if condition is false

}

**3.Multiway Decision (if-else Ladder)**

Purpose: Used for complex, range-based, or multiple conditional checks involving different variables.

if (condition1) {

// code to execute if condition1 is true

} else if (condition2) {

// code to execute if condition2 is true

} else if (condition3) {

// code to execute if condition3 is true

} else {

// code to execute if none of the above conditions are true

}

**Looping:**

**1.While Loop:**

Purpose:Repeatedly executes a block of code as long as the specified condition remains true.

while (condition) {

// code to execute

}

**2.Do-While Loop:**

Purpose:Similar to the while loop, but guarantees that the code block will be executed at least once before checking the condition.

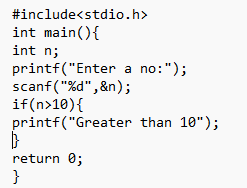
do {

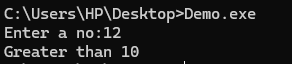
// code to execute

} while (condition);

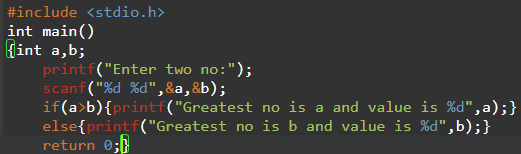
**Questions:**

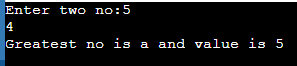
Q.1 Write a program to take input from user and check if it is greater than 10 then print “Greater than 10”.(Use of If Statement).(10 min)



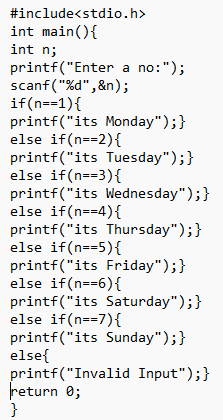


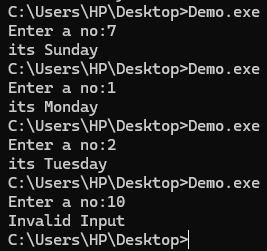
Q.2 Write a program to take two no as input and print the greatest.(Use of If-else Statement).(10 min)



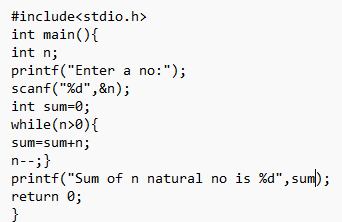


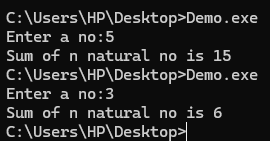
Q.3 Write a program to take a input as a no from user and print the day of the week. e.g if input is 1 then Monday,2 then Tuesday,likewise (Use of If else Ladder).(15 min)



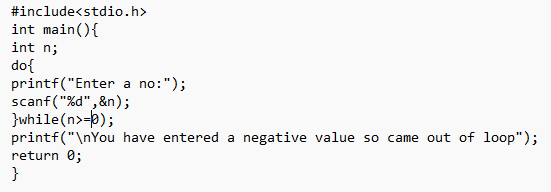


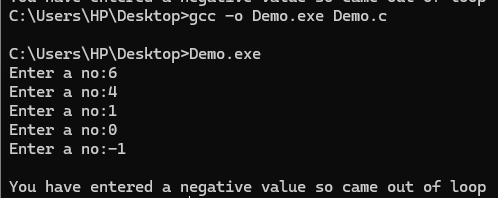
Q.4 Write a program to print sum of n natural no. (n should be user input).(Use of while loop).(15 min)





Q.5 Write a program to take input from user until user enters negative no.(Use of Do While loop).(15 min).

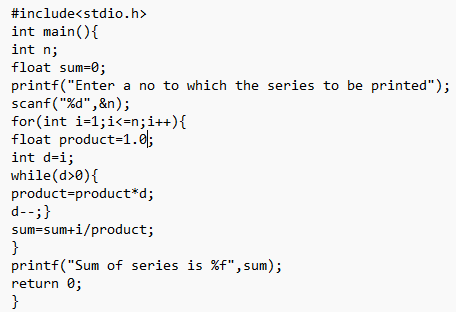


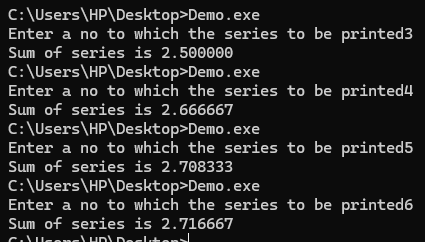


**Practice Questions:**

Q.1 WAP to calculate sum of series.

1/1!+2/2!+3/3!+……….+n/n!





Q.2 Study of Nested for loop.

WAP to print following pattern.

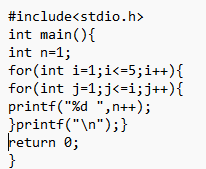
1 A

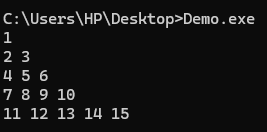
2 3 A B

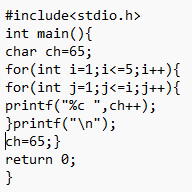
4 5 6 A B C

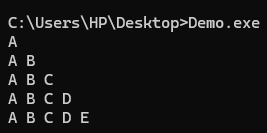
7 8 9 10 A B C D

11 12 13 14 15 A B C D E

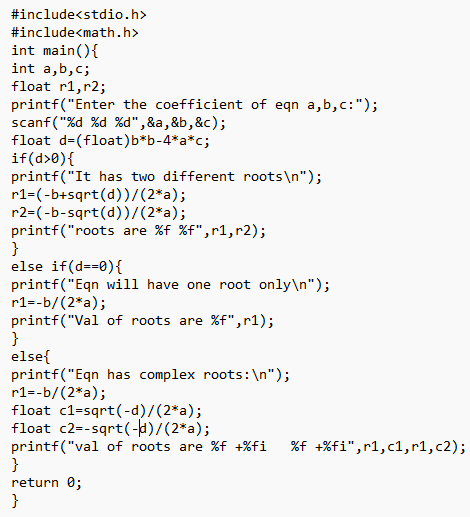


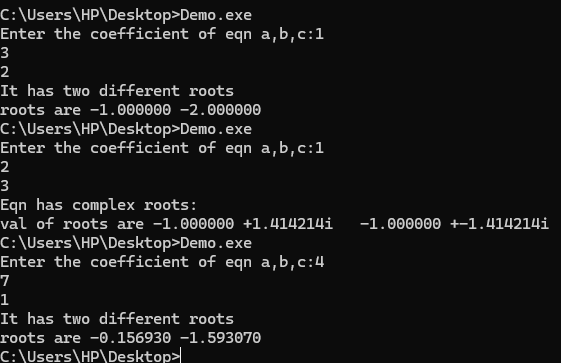




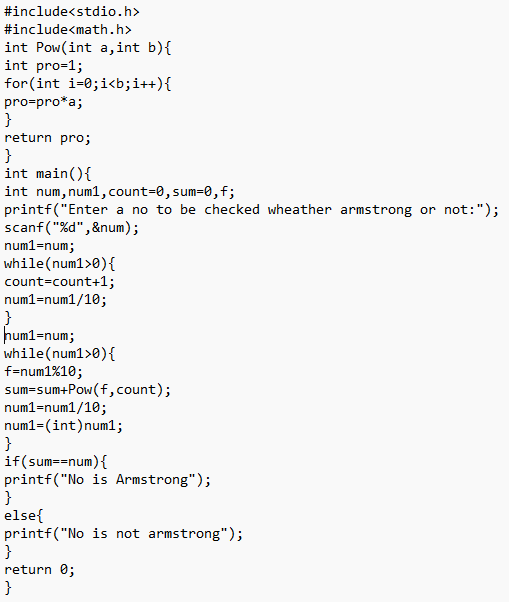


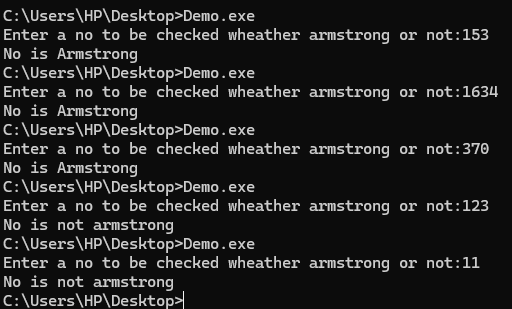
Q.3 Study of If-Else (Decision making).WAP to calculate roots of quadratic equation.



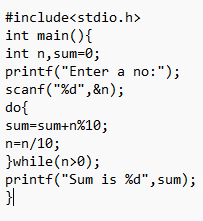


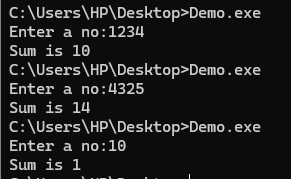
Q.4 Study of While loop.WAP to check whether it is Armstrong number or not. Study of While loop.





Q.5 Study of Do-While loop. WAP to sum of individual digits.





**Conclusion:** We understood the concept and then did some on it question. We Successfully fulfilled the aim of experiment.

**Experiment No 03**

**Aim:** Program to demonstrate Nested control structure- Switch statement, Continue statement, Break statement, Goto statement.

**Requirements:** C Compiler

**Theory:**

**switch Statement:**

* Used for multi-way branching based on the value of an expression.
* Allows for multiple discrete cases and a default case for handling unexpected values.

**continue Statement:**

* Skips the remaining code in the current iteration of a loop and proceeds to the next iteration.
* Useful for skipping specific iterations based on a condition.

**break Statement:**

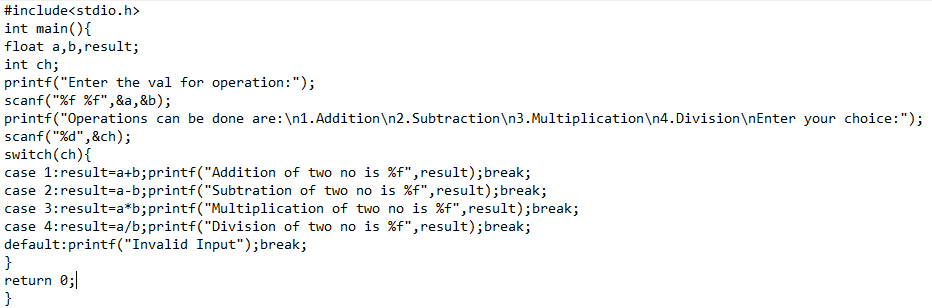
* Exits from the nearest enclosing loop or switch statement.
* Useful for terminating loops early or exiting switch cases.

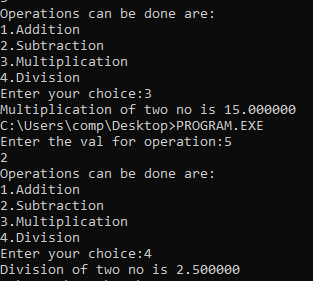
**goto Statement:**

* Transfers control to a labeled statement within the same function.
* Provides a way to jump to different parts of code, though it is generally discouraged due to potential for creating unreadable code.

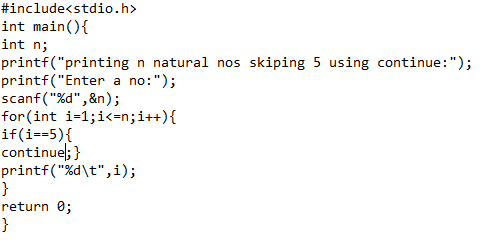
**Questions:**

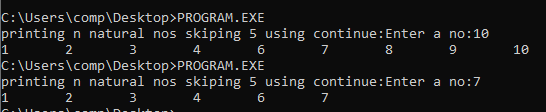
Q.1 Write a program to create a calculator which can perform Addition,Subtration,Multiplication,Division using Switch.(20 min)(break will also use).



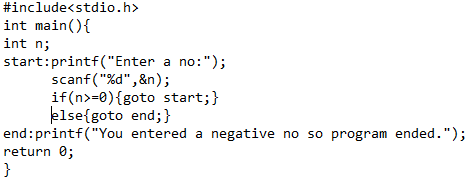


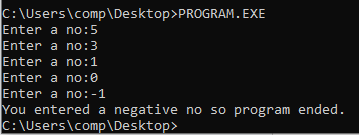
Q.2 Write a program to print n natural no except 5 using continue statement.(5 min).





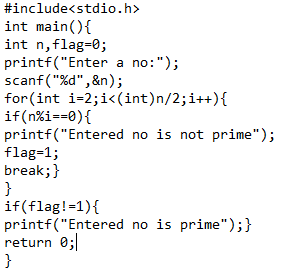
Q.3 Write a program to take user input till user not enter a negative no using GOTO statement.(10 min).

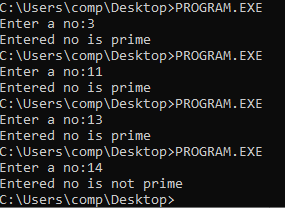




Practice Questions:

Q.1 Study of Break statement.WAP to check whether entered no. is prime or not.





**Conclusion:** We understood the concept and then did some on it question. We Successfully fulfilled the aim of experiment.

**Experiment No 04**

**Aim:** Program to demonstrate Function, Passing Arguments to a Function

(call by value and call by reference).

**Requirements:** C Compiler

**Theory:**

**Function Definition:**

* A function in C is a block of code designed to perform a specific task. It has a name, a return type, and may take arguments.

Syntax:

return\_type function\_name(parameter1, parameter2, ...) {

// code to execute

return value; // optional, depending on return type

}

**Function Call:**

* To execute a function, you call it by its name and pass the required arguments.

Syntax:

function\_name(arguments);

### Passing Arguments to Functions:

1. **Call by Value:**
   * **Definition:** Copies the value of the argument into the function's parameter. Changes made to the parameter inside the function do not affect the original argument.
   * **Usage:** Use when you don’t need to modify the original variable.

**Syntax:**   
void function\_name(int param) {

// param is a copy of the argument }

**2.Call by Reference:**

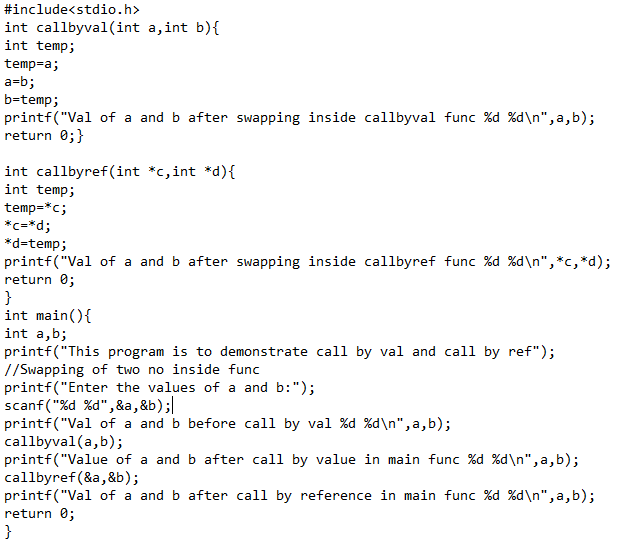
* **Definition:** Passes the address (reference) of the argument to the function. The function can modify the original variable because it operates on the actual memory location.
* **Usage:** Use when you need the function to modify the original variable or when dealing with large data structures to avoid copying.
* **Syntax:**void function\_name(int \*param) {

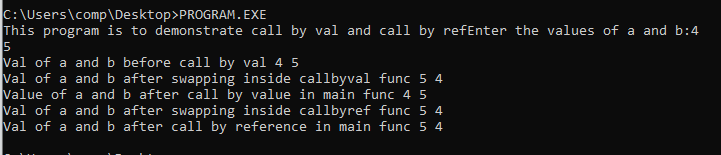
// \*param is a reference to the original argument

}

**Questions:**

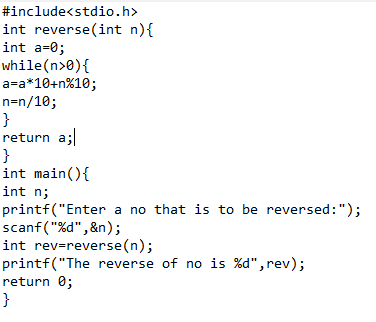
Q.1 Write a program to swap two no using functions and passing argument to the function using call by value and call by reference. (30 min).

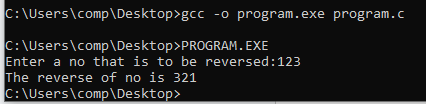




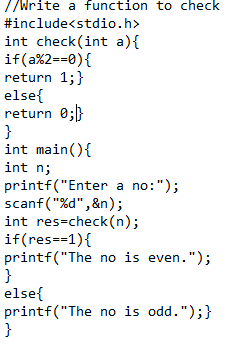
practice Questions:

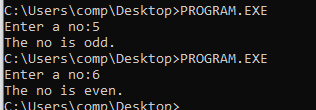
Q.1 WAP to reverse a number using functions.





Q.2 Write a function to check if a number is even or odd. The function should take an integer as an argument and return 1 if the number is even, or 0 if the number is odd.





**Conclusion:** We understood the concept and then did some on it question. We Successfully fulfilled the aim of experiment.

**Experiment No 05**

**Aim:** a) Implement an iterative function for factorial/ Fibonacci etc.

b) Implement a recursive function for factorial/ Fibonacci etc.

**Requirements:** C Compiler

**Theory:**

### 1. Iterative Function:

#### Definition:

An iterative function uses loops (for, while, do-while) to repeat a block of code until a condition is met.

#### Key Characteristics:

* **Loops:** Utilizes constructs like for, while, or do-while to repeat actions.
* **State Management:** Manages iteration using loop control variables.
* **Efficiency:** Generally has lower overhead compared to recursion, as it avoids the overhead of function calls and stack usage.

### 2. Recursive Function:

#### Definition:

A recursive function calls itself to solve a smaller instance of the problem, with each call moving closer to a base case that terminates the recursion.

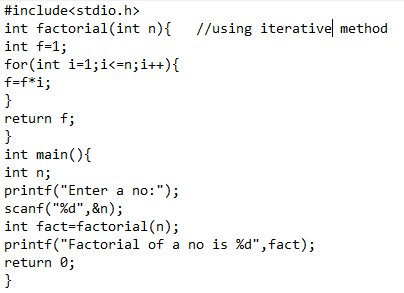
#### Key Characteristics:

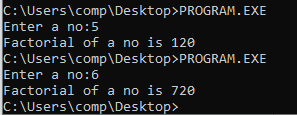
* **Base Case:** Defines when the recursion should stop to prevent infinite recursion.
* **Recursive Case:** The part where the function calls itself, breaking the problem into smaller subproblems.
* **Stack Usage:** Each recursive call adds a new layer to the call stack, which can lead to high memory usage or stack overflow for deep recursions.

### Comparison:

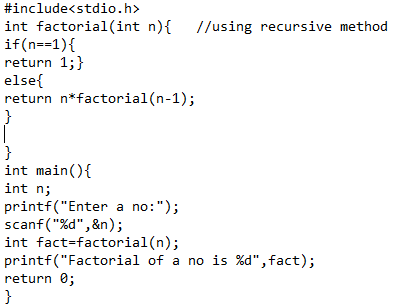
* **Iterative Function:**
  + Uses explicit loops to repeat tasks.
  + Generally more memory-efficient because it does not require additional stack space.
  + Preferred when the problem size is large or when recursion depth could lead to stack overflow.
* **Recursive Function:**
  + Uses function calls to repeat tasks, with each call handling a smaller part of the problem.
  + Can be more intuitive and elegant for problems that naturally fit a recursive structure (e.g., tree traversal).
  + Can lead to high memory usage and potential stack overflow if not properly managed.

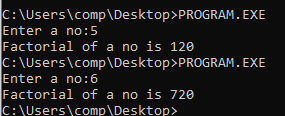
Q.1 Write a program to calculate the factorial of a no using iterative method.(use of Function)(20 min)





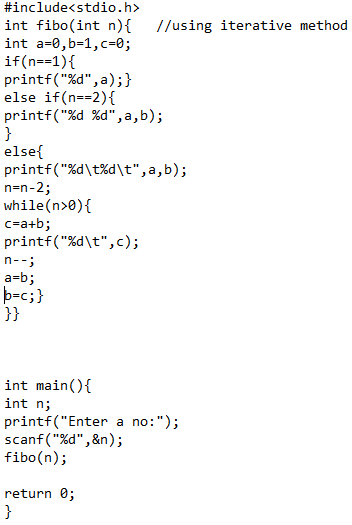
Q.2 Write a program to calculate the factorial of a no using Recursive method.(use function)(20 min)

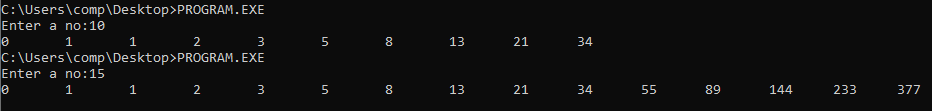




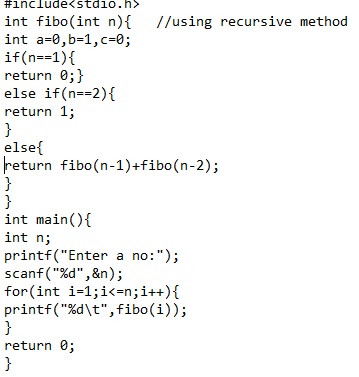
Practice questions:

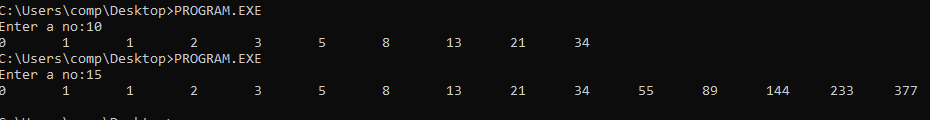
Q.1 Write a program to calculate the fibonacci series using iterative method.(20 min)





Q.2 Write a program to calculate the fibonacci series using Recursive method.(20 min)





**Conclusion:** We understood the concept and then did some on it question. We Successfully fulfilled the aim of experiment.

**Experiment No 06**

**Aim:** a) Program to demonstrate Storage Classes –Auto, Extern, Static, Register.

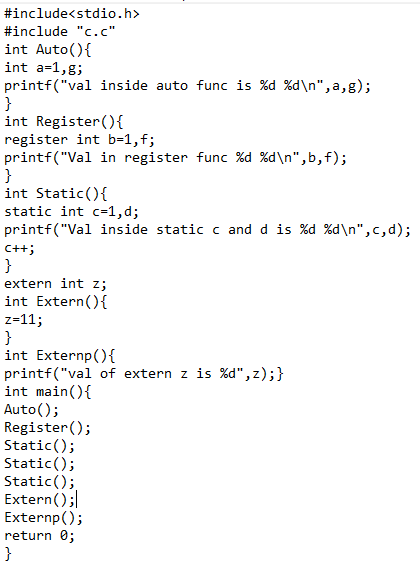
**Requirements:** C Compiler

**Theory:  
Storage Classes:**

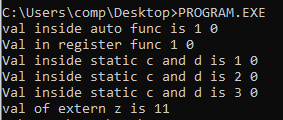
1. **auto:**
   * Definition: Default storage class for local variables. The auto keyword is optional and typically omitted.
   * Scope: Local to the block in which it is declared.
   * Lifetime: Exists only during the execution of the block in which it is declared.
2. **extern:**
   * Definition: Used to declare a variable that is defined in another file or elsewhere in the same file. It extends the visibility of a variable to other files.
   * Scope: Global, accessible across multiple files or from different functions in the same file.
   * Lifetime: Exists for the entire duration of the program.
3. **static:**
   * Definition: Used to declare a variable that retains its value between function calls or to limit the visibility of a variable to the file in which it is declared.
   * Scope: Local to the function or file in which it is declared, but retains its value across function calls or within the file.
   * Lifetime: Exists for the entire duration of the program.
4. **register:**
   * Definition: Suggests to the compiler that the variable should be stored in a CPU register for faster access.
   * Scope: Local to the block in which it is declared.
   * Lifetime: Exists only during the execution of the block in which it is declared.
   * Note: The register keyword is a hint to the compiler and does not guarantee that the variable will be stored in a register.

**Question:**

Q.1 Write a C program that illustrates the usage of the four primary storage classes: auto, extern, static, and register.(30 min)







**Conclusion:** We understood the concept and then did some on it question. We Successfully fulfilled the aim of experiment.

**Experiment No 07**

**Aim:** a)Program to demonstrate Array 1D, b) Program to demonstrate Array 2D.

**Requirements:** C Compiler

**Theory:  
Arrays in C:**

1. **One-Dimensional Array (1-D Array):**
   * Definition: A collection of elements of the same type stored in contiguous memory locations. It is essentially a list of elements.

Syntax:  
 data\_type array\_name[size];

1. **Two-Dimensional Array (2-D Array):**
   * Definition: An array of arrays, which can be visualized as a matrix or table with rows and columns.
   * Syntax:  
     data\_type array\_name[rows][columns];

**1-D Array:**

* Structure: Linear; a single list of elements.
* Access: Uses a single index.
* Memory Allocation: Contiguous block of memory.
* Typical Uses: Lists, sequences.

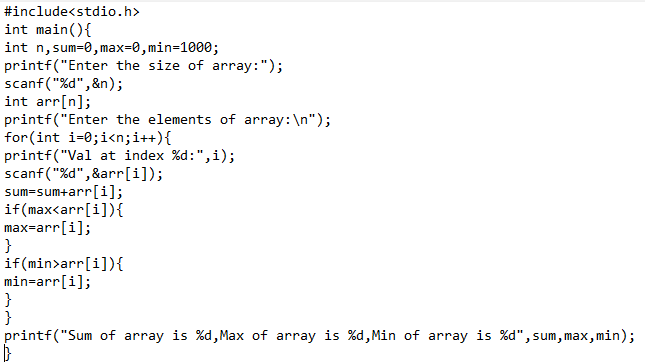
**2-D Array:**

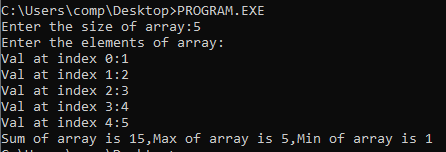
* Structure: Matrix; a table of elements with rows and columns.
* Access: Uses two indices (row and column).
* Memory Allocation: Contiguous block of memory with multiple rows.
* Typical Uses: Matrices, grids, tables.

### Key Points:

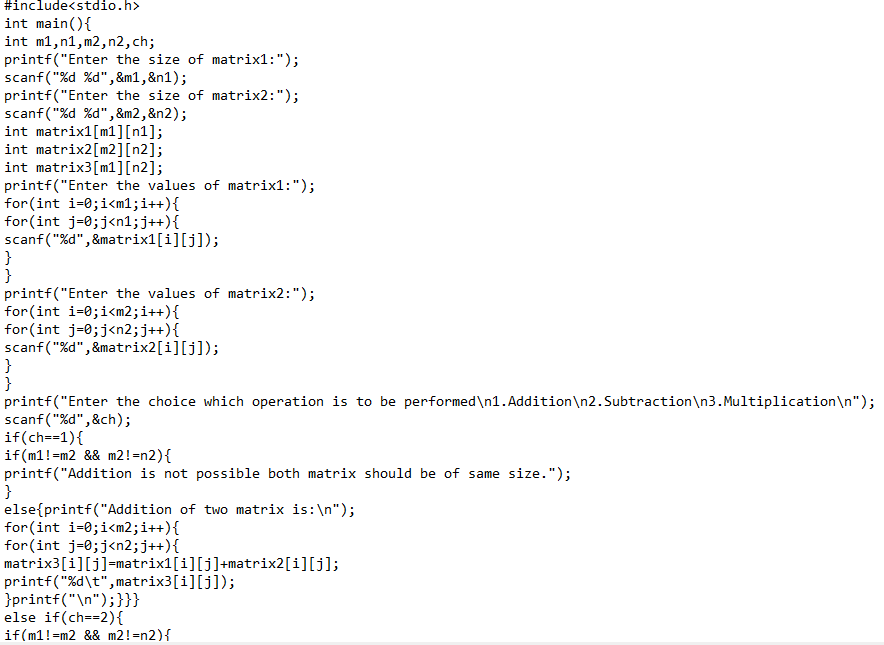
* **Contiguous Memory:** Both 1-D and 2-D arrays store elements in contiguous memory locations, which allows efficient access and manipulation.
* **Indexing:** The index for a 1-D array starts from 0 and goes up to size-1. For a 2-D array, indexing starts from [0][0] for the first element and goes up to [rows-1][columns-1].
* **Initialization:** Both types of arrays can be initialized during declaration or through loops.
* **Accessing Elements:** Direct access via indices makes arrays suitable for tasks that require efficient random access.

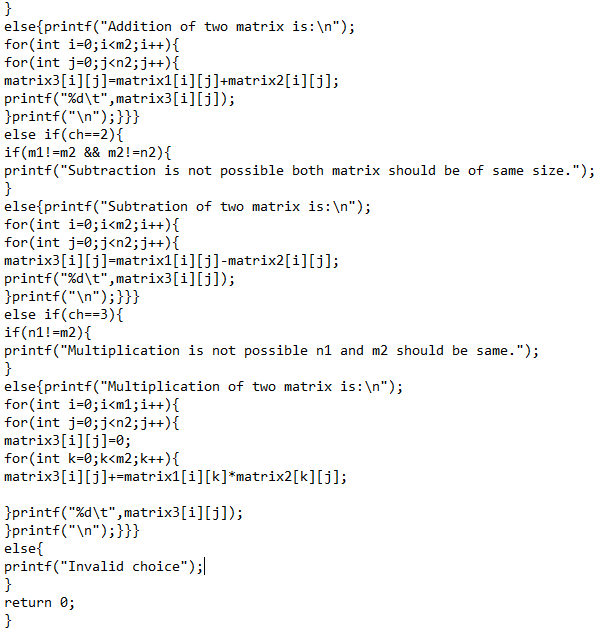
Q.1 Write a program to create a 1D array and print the greatest element present in the array, smallest element present in the array, sum of all the elements present inside it.(20 min)

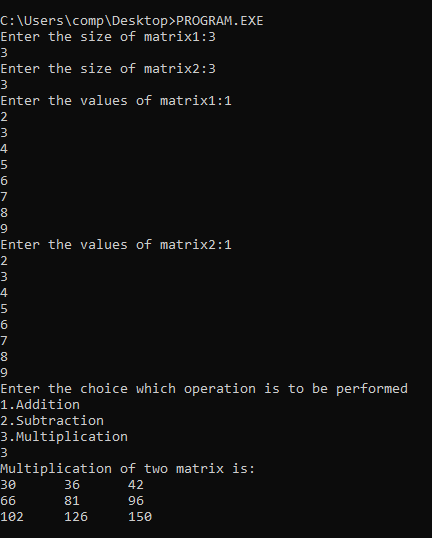




Q.2 Write a program for Matrix addition, subtraction and multiplication (Use 2D array).(40 min)



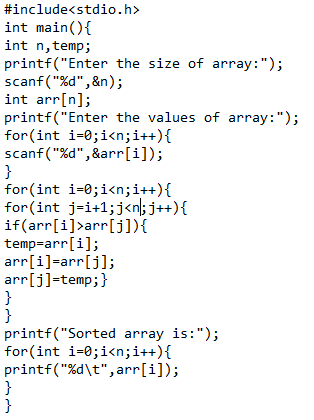


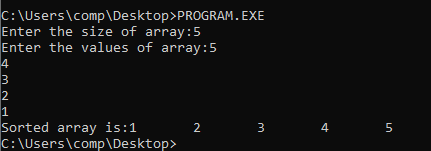


We will understand the concept of multi-dimensional array in theory class

Practice questions:

Q.1 WAP to sort a given array in ascending order.





**Conclusion:** We understood the concept and then did some on it question. We Successfully fulfilled the aim of experiment.

**Experiment No 08**

**Aim:** a)Program to demonstrate String b) Program to demonstrate String arrays of string.

**Requirements:** C Compiler

**Theory:**

### Strings in C

#### Definition:

A string in C is a sequence of characters terminated by a null character ('\0'). In C, strings are represented as arrays of characters.

#### Characteristics:

**Declaration:**char str[size];

* + char: Data type for characters.
  + str: Name of the string.
  + size: Number of characters including the null terminator.
* Initialization: Strings can be initialized at the time of declaration.

char str[] = "Hello, World!";

char first\_char = str[0]; // Accesses 'H'

**Standard Library Functions:**

* strlen(): Returns the length of the string.
* strcpy(): Copies one string to another.
* strcat(): Concatenates two strings.
* strcmp(): Compares two strings.

### Arrays of Strings

#### Definition:

An array of strings is a two-dimensional array of characters, where each row represents a string. This can be used to store multiple strings.

Declaration: char array\_of\_strings[rows][columns];

Can be initialized like char array\_of\_strings[3][20] = { "Apple", "Banana","Cherry"};

**Strings:**

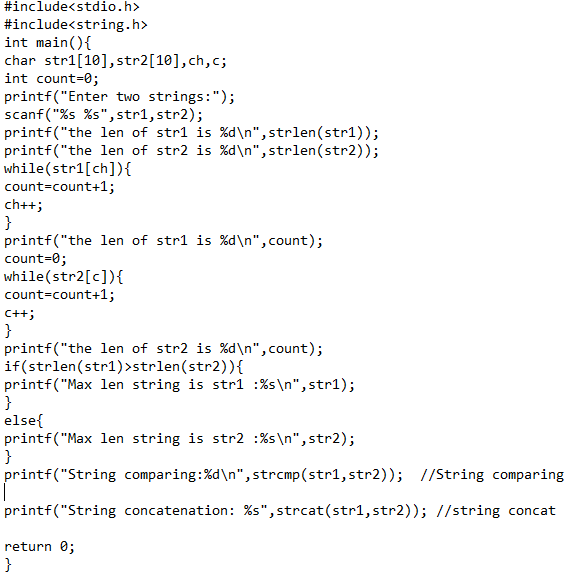
* Strings are handled as arrays of characters terminated by '\0'.
* Standard library functions are used for common string operations such as copying, concatenating, and comparing.

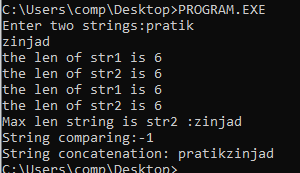
**Arrays of Strings:**

* An array of strings is essentially a 2-D character array where each row is a string.
* Useful for storing and manipulating lists of strings (e.g., a list of names).

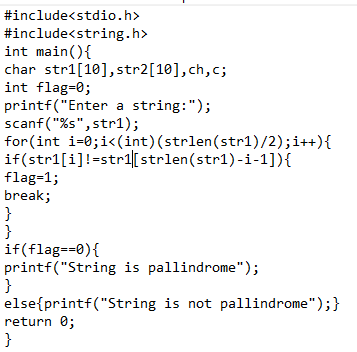
**Questions:**

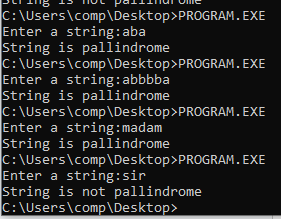
Q.1 Write a program to take user input as two different strings from user and perform the following operations: a) calculate and print the length of both the string.(both ways) b) print the string of max length c) compare the two strings d) concatenate the strings and print the result.





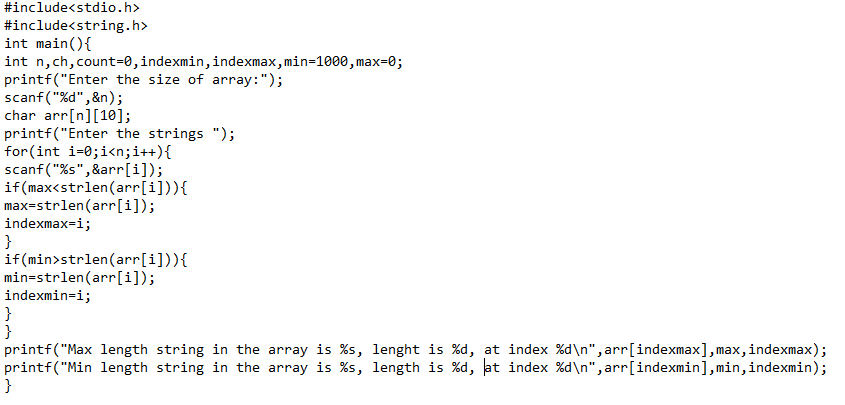
Q.2 Write a program to check weather the entered string is palindrome or not.

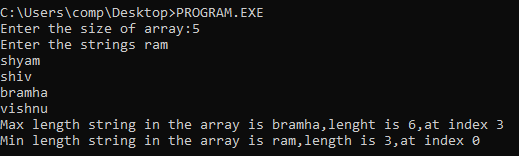




Q.3 WAP to take array of strings and perform the following operations:

a)print the string and index of string which has max length b) print the the string and index of string which has min length.





**Conclusion:** We understood the concept and then did some on it question. We Successfully fulfilled the aim of experiment.

**Experiment No 09**

**Aim:** Write a program to store and display information of a student

etc. using structures.

a) Define a structure.

b) Read and store details.

c) Display the stored information.

**Requirements:** C Compiler

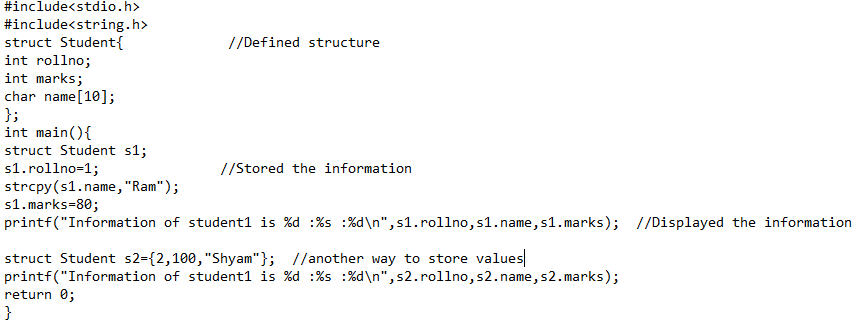
**Theory:**

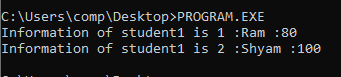
#### Definition:

A structure in C is a user-defined data type that allows grouping variables of different types under a single name. Structures are used to represent a record.

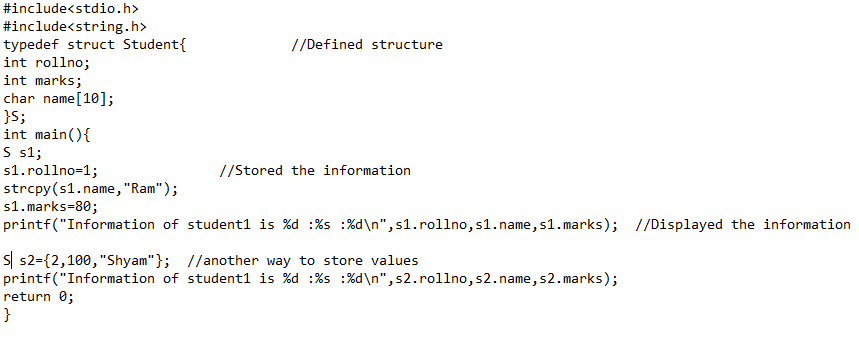
#### Characteristics:

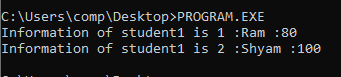
* Declaration: Defines the structure template.
* Initialization: Creates variables of the structure type and initializes them.
* Accessing Members: Accesses individual members of the structure using the dot (.) operator.





using typedef keyword:





practice Question:

We will cover basics in theory class so no need to worry.

Write a program to store and display information of a employee

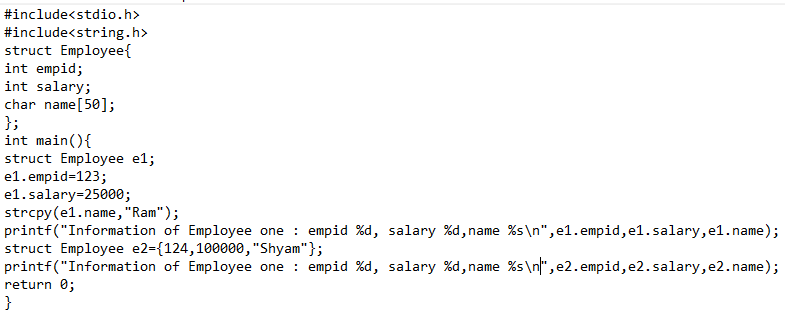
etc.

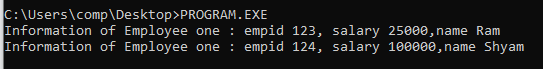
using structures.

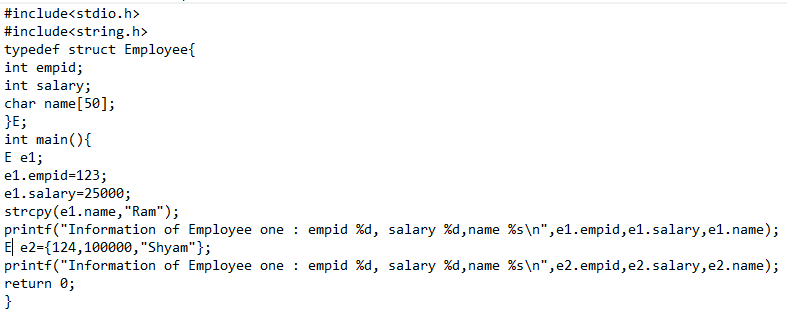
a) Define a structure.

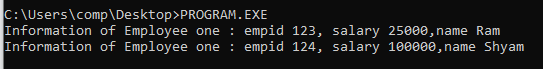
b) Read and store details.

c) Display the stored information.









**Conclusion:** We understood the concept and then did some on it question. We Successfully fulfilled the aim of experiment.

**Experiment No 10**

**Aim:** Program to demonstrate pointers

a) Define a node structure.

b) Implement functions to insert, delete, and display nodes.

**Requirements:** C Compiler

**Theory: Pointers in C**

#### Definition:

A pointer is a variable that stores the memory address of another variable. Pointers are a powerful feature in C programming that allows for dynamic memory allocation, efficient array manipulation, and the implementation of complex data structures like linked lists.

#### Characteristics:

* **Declaration:** Pointers are declared using the asterisk (\*) symbol.

data\_type \*pointer\_name;

data\_type: The type of data the pointer points to.

pointer\_name: The name of the pointer variable.

**Initialization:** Pointers are usually initialized to NULL or the address of a variable.

int \*ptr = NULL;

int var = 10;

ptr = &var; // ptr now holds the address of var

**Dereferencing:** Accessing the value at the memory address stored in the pointer is done using the dereference operator (\*).

int value = \*ptr; // Accesses the value at the address stored in ptr

**Pointer Arithmetic:** Pointers can be incremented or decremented to point to the next or previous memory location.

ptr++; // Points to the next memory location (e.g., next integer in an array)

**Dynamic Memory Allocation:** Functions like malloc(), calloc(), and free() are used to allocate and deallocate memory dynamically.

Code:

//Linked list implementation

#include<stdio.h>

#include<stdlib.h>

struct node{

int val;

struct node\* next;

};

void InsertAtBegin(struct node\*\* head,int data){

struct node\* newnode=(struct node\*)malloc(sizeof(struct node));

newnode->val=data;

newnode->next=\*head;

\*head=newnode;

}

void InsertAtEnd(struct node\*\* head,int data){

struct node\* newnode=(struct node\*)malloc(sizeof(struct node));

struct node\* temp=\*head;

newnode->val=data;

newnode->next=NULL;

while(temp->next!=NULL){

temp=temp->next;

}

temp->next=newnode;

}

void Display(struct node\* head){

while(head!=NULL){

printf("%d->\t",head->val);

head=head->next;

}

printf("\n");

}

void InsertAtIndex(struct node\*\* head,int index,int data){

struct node\* newnode=(struct node\*)malloc(sizeof(struct node));

int count=1;

struct node\* temp=\*head;

newnode->val=data;

while(count!=index-1){

count=count+1;

temp=temp->next;

}

newnode->next=temp->next;

temp->next=newnode;

}

void DeleteAtStart(struct node\*\* head){

struct node\* temp=\*head;

\*head=temp->next;

free(temp);

}

void InsertAtafterVal(struct node\*\* head,int val,int data){

struct node\* newnode=(struct node\*)malloc(sizeof(struct node));

newnode->val=data;

struct node\* temp=\*head;

while(temp->val!=val){

temp=temp->next;

}

newnode->next=temp->next;

temp->next=newnode;

}

void InsertAtBeforeVal(struct node\*\* head,int val,int data){

struct node\* newnode=(struct node\*)malloc(sizeof(struct node));

newnode->val=data;

struct node\* temp=\*head;

while(temp->next->val!=val){

temp=temp->next;

}newnode->next=temp->next;

temp->next=newnode;

}

void DeleteAtIndex(struct node\*\* head,int index){

struct node\* temp=\*head;

int count=1;

while(count!=index-1){

temp=temp->next;

count=count+1;

}

struct node\* a=temp->next;

temp->next=a->next;

free(a);

}

void DeleteBeforeVal(struct node\*\* head,int val){

struct node\* temp=\*head;

while(temp->next->next->val!=val){

temp=temp->next;

}struct node\* a=temp->next;

temp->next=a->next;

free(a);

}

void DeleteAfterVal(struct node\*\* head,int val){

struct node\* temp=\*head;

while(temp->val!=val){

temp=temp->next;

}struct node\* a=temp->next;

temp->next=a->next;

free(a);

}

void DeleteAtEnd(struct node\*\* head){

struct node\* temp=\*head;

while(temp->next->next!=NULL){

temp=temp->next;

}//printf("%d\n",temp->val);

temp->next=NULL;

temp=temp->next;

free(temp);

}

int main(){

struct node\* head=NULL;

InsertAtBegin(&head,2);

InsertAtBegin(&head,1);

InsertAtEnd(&head,5);

Display(head);

InsertAtIndex(&head,3,3);

InsertAtIndex(&head,4,4);

Display(head);

DeleteAtStart(&head);

DeleteAtEnd(&head);

InsertAtafterVal(&head,3,10);

InsertAtBeforeVal(&head,10,9);

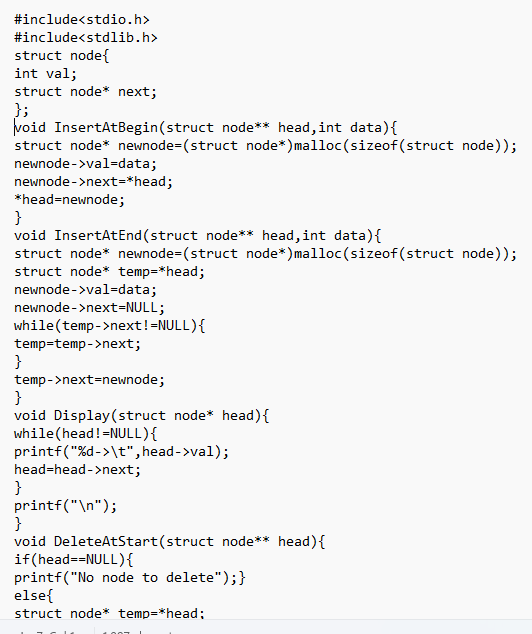
DeleteAtIndex(&head,3);

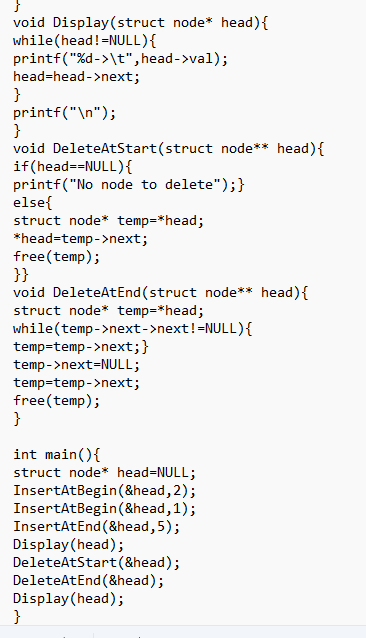
DeleteBeforeVal(&head,4);

DeleteAfterVal(&head,3);

Display(head);

}





**Conclusion:** We understood the concept and then did some on it question. We Successfully fulfilled the aim of experiment.

**Experiment No 11**

**Aim:** Program to demonstrate files Write a program to maintain a simple student/employee etc. database using file handling.

a) Open a file to store student records.

b) Implement functions to add, update, and display records.

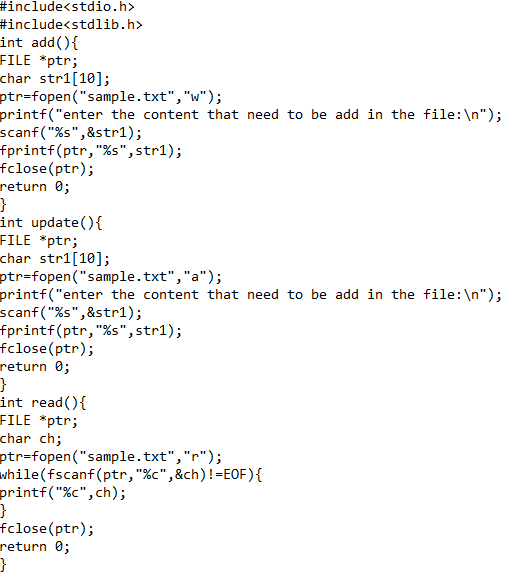
c) Ensure data persistence by saving changes to the file.

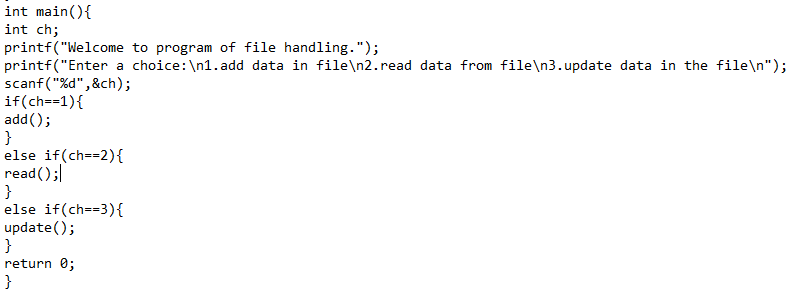
**Requirements:** C Compiler

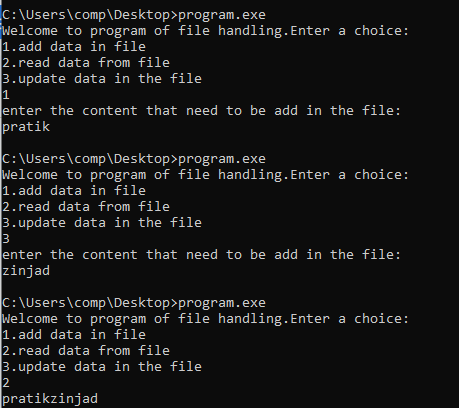
**Theory:   
File Operations:**

1. **Opening a File:**
   * Function: fopen()
   * Modes:
     + "r": Read mode. Opens an existing file for reading.
     + "w": Write mode. Creates a new file or truncates an existing file for writing.
     + "a": Append mode. Opens a file for writing at the end of the file.
     + "r+": Read and write mode. Opens an existing file for both reading and writing.
     + "w+": Read and write mode. Creates a new file or truncates an existing file for both reading and writing.
     + "a+": Read and write mode. Opens a file for reading and appending.
2. **Reading from a File:**
   * **Functions:**
     + fgetc(): Reads a single character from a file.
     + fgets(): Reads a line from a file.
     + fread(): Reads a block of data from a file.
3. **Writing to a File:**
   * **Functions:**
     + fputc(): Writes a single character to a file.
     + fputs(): Writes a string to a file.
     + fwrite(): Writes a block of data to a file.
4. **Closing a File:**
   * Function: fclose()
5. **Error Handling:**
   * Check if file operations succeed by verifying the return values of file functions and using feof(), ferror(), and clearerr() for error handling.

Code:







**Conclusion:** We understood the concept and then did some on it question. We Successfully fulfilled the aim of experiment.

**Experiment No 12**

**Aim:** Implement one small application using Function, Files, Structure and

Pointers concepts you have learnt in C (eg. : Simple Library Management

System

1.Functions: Add, display, and search books. 2. Files: Store and retrieve

book data. 3. Structures: Represent a book. 4. Pointers: Manage the list of

books dynamically.

**Requirements:** C Compiler