# **Experiment 1**

## Program 1

**Aim ::** Write a program showing the implementation of array concatenation.

**Code ::**

#include <iostream>

int main() {

int arr1[] = {2,5,6,8,9,23};

int arr2[] = {102,105,106,108,109};

int size\_arr1 = (sizeof(arr1) / sizeof(int));

int size\_arr2 = (sizeof(arr2) / sizeof(int));

int size\_arr3 = (size\_arr1+size\_arr2);

//Memory gets allocated for new array

int arr3[size\_arr3];

//loop over first array and add its elements to final array

for (int i = 0; i < size\_arr1; i++) arr3[i] = arr1[i];

//loop over first array and add its elements to final array

for (int i = 0; i < size\_arr2; i++) arr3[i+size\_arr1] = arr2[i];

//Print resulting arrays

std::cout << "Array 1 :\t";

for (int i = 0; i < size\_arr1; i++) std::cout << arr1[i] << " ";

std::cout << std::endl;

std::cout << "Array 2 :\t";

for (int i = 0; i < size\_arr2; i++) std::cout << arr2[i] << " ";

std::cout << std::endl;

std::cout << "Array 3 :\t";

for (int i = 0; i < size\_arr3; i++) std::cout << arr3[i] << " ";

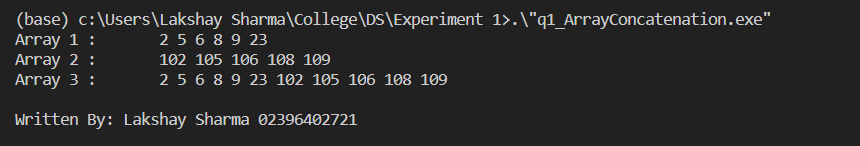
std::cout << std::endl;

std::cout << "\nWritten By: Lakshay Sharma 02396402721" << std::endl;

return 0;

}

**Output ::**

****

## Program 2

**Aim ::** Write a program showing the implementation of Linear search.

**Code ::**

#include <iostream>

template <typename T>

int linearSearch(T element, T array[], int arr\_size){

int index;

//Performing Linear Search

for(index = 0; index < arr\_size ; index++) {

if (array[index] == element) {

return index; //return at first instance

}

}

//If value is not in array

return -1;

}

int main()

{

int arr[10] = {8,9,5,4,7,3,1,5,6,12};

int size\_arr = (sizeof(arr)/sizeof(int));

//Showing user the array to perform search operation

std::cout << "Array:\t";

for(int i = 0;i<10;i++) std::cout << arr[i] << " ";

//Asking for element to find in array

int ele;

std::cout << "\nElement to find: "; std::cin >> ele;

//Linear Search Operation

int foundAt = linearSearch(ele, arr, size\_arr);

//Returning result after searching

if (foundAt > 0) std::cout << "Value <" << ele <<"> found at index: " << foundAt << std::endl;

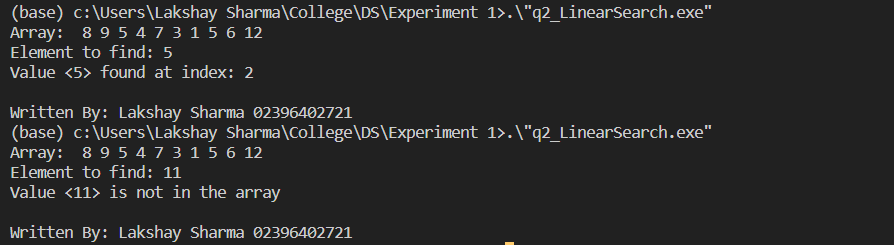
else std::cout << "Value <" << ele << "> is not in the array" << std::endl;

std::cout << "\nWritten By: Lakshay Sharma 02396402721";

return 0;

}

**Output ::**



## Program 3

**Aim ::** Write a program showing the implementation of Binary search.

**Code ::**

#include <iostream>

/\*Iterative Approach\*/

template <typename T>

int binarySearchI(T element, T array[], int lower\_index, int upper\_index){

while (lower\_index <= upper\_index) {

int middle\_index = lower\_index + (upper\_index - lower\_index) / 2;

// Check if element is present at mid

if (array[middle\_index] == element) return middle\_index;

// If element smaller than middle, ignore right half

if (element < array[middle\_index]) upper\_index = middle\_index - 1;

// If element is greater than middle, ignore left half

else lower\_index = middle\_index + 1;

}

// Value is not found in array

return -1;

}

/\*Recursive Approach\*/

template <typename T>

int binarySearchR(T element, T array[], int lower\_index, int upper\_index){

if (upper\_index >= lower\_index) {

int middle\_index = lower\_index + (upper\_index - lower\_index) / 2;

// If the element is present at the middle

if (array[middle\_index] == element) return middle\_index;

// If element is smaller than middle, then element is in left subArray

if (element < array[middle\_index])

return binarySearchR(element, array, lower\_index, middle\_index - 1);

// Else, element is in right subArray

return binarySearchR(element, array, middle\_index + 1, upper\_index);

}

// Value is not found in array

return -1;

}

int main()

{

int arr[10] = {1,2,3,5,7,11,13,17,19,23};

int size\_arr = (sizeof(arr)/sizeof(int));

//Showing user the sorted array to perform search operation

std::cout << "Array:\t";

for(int i = 0;i<10;i++) std::cout << arr[i] << " ";

//Asking for element to find in array

int ele;

std::cout << "\nElement to find: "; std::cin >> ele;

/\*Iterative Binary Search Operation\*/

int foundAt = binarySearchI(ele, arr, 0, size\_arr);

//Returning result after searching

std::cout << "Iterative Binary Search >> ";

if (foundAt > 0) std::cout << "Value <" << ele << "> found at index: " << foundAt << std::endl;

else std::cout << "Value <" << ele << "> is not in the array" << std::endl;

/\*Recursive Binary Search Operation\*/

foundAt = binarySearchR(ele, arr, 0, size\_arr);

//Returning result after searching

std::cout << "Recursive Binary Search >> ";

if (foundAt > 0) std::cout << "Value <" << ele << "> found at index: " << foundAt << std::endl;

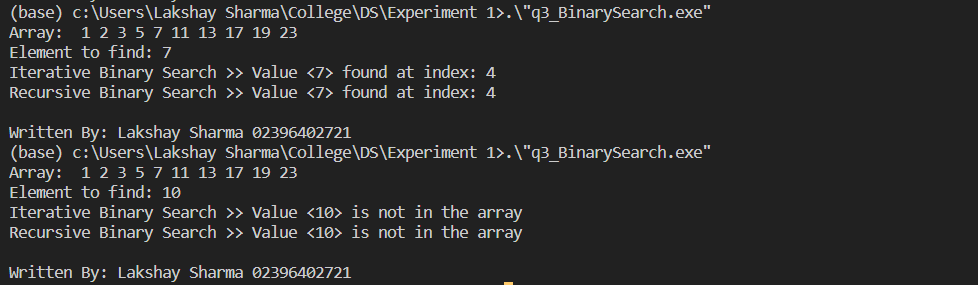
else std::cout << "Value <" << ele << "> is not in the array" << std::endl;

std::cout << "\nWritten By: Lakshay Sharma 02396402721";

return 0;

}

**Output ::**

****

## Program 4

**Aim ::** Write a program showing the implementation of Matrix Multiplication.

**Code ::**

#include <iostream>

#include <vector>

using namespace std;

// Function to perform matrix multiplication

vector<vector<int>> multiply(vector<vector<int>>& A, vector<vector<int>>& B){

int n = A.size();

int m = B[0].size();

int p = B.size();

vector<vector<int>> C(n, vector<int>(m, 0));

for (int i = 0; i < n; i++)

{

for (int j = 0; j < m; j++)

{

for (int k = 0; k < p; k++)

{

C[i][j] += A[i][k] \* B[k][j];

}

}

}

return C;

}

void print(vector<vector<int>>&A){

for (int i = 0; i < A.size(); i++)

{

for (int j = 0; j < A[0].size(); j++)

{

cout << A[i][j] << " ";

}

cout << endl;

}

}

// Driver function

int main()

{

// Initialize two matrices

vector<vector<int>> A = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};

vector<vector<int>> B = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};

cout << "Matrix A : " << endl;

print(A);

cout << "Matrix B : " << endl;

print(B);

// Perform matrix multiplication

vector<vector<int>> C = multiply(A, B);

// Print the result

cout << "Result of matrix multiplication:" << endl;

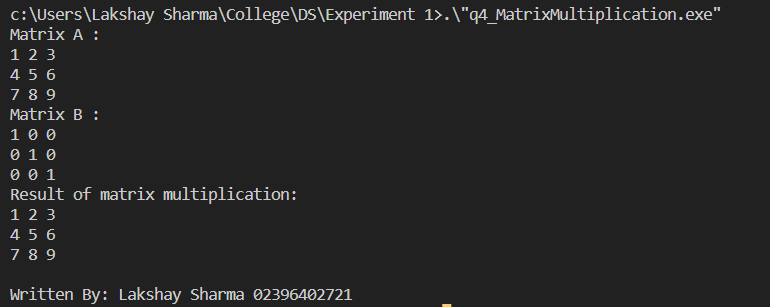
print(C);

std::cout << "\nWritten By: Lakshay Sharma 02396402721";

return 0;

}

**Output ::**

****

# **Experiment 2**

## Program 1

**Aim ::** Write a program showing the implementation of Stack and Queue with array.

**Code ::**

#include <iostream>

const int MAX\_SIZE = 100;

class Stack {

private:

int arr[MAX\_SIZE];

int top;

public:

Stack() {

top = -1;

}

bool isEmpty() {

return top == -1;

}

bool isFull() {

return top == MAX\_SIZE - 1;

}

void push(int x) {

if (isFull()) {

std::cout << "Error: stack is full" << std::endl;

return;

}

top++;

arr[top] = x;

}

void pop() {

if (isEmpty()) {

std::cout << "Error: stack is empty" << std::endl;

return;

}

top--;

}

int peek() {

if (isEmpty()) {

std::cout << "Error: stack is empty" << std::endl;

return -1;

}

return arr[top];

}

};

class Queue {

private:

int arr[MAX\_SIZE];

int front;

int rear;

public:

Queue() {

front = 0;

rear = -1;

}

bool isEmpty() {

return front == rear+1;

}

bool isFull() {

return rear == MAX\_SIZE - 1;

}

void enqueue(int x) {

if (isFull()) {

std::cout << "Error: queue is full" << std::endl;

return;

}

rear++;

arr[rear] = x;

}

void dequeue() {

if (isEmpty()) {

std::cout << "Error: queue is empty" << std::endl;

return;

}

front++;

}

int peek() {

if (isEmpty()) {

std::cout << "Error: queue is empty" << std::endl;

return -1;

}

return arr[front];

}

};

int main() {

std::cout << "-------------------STACK------------------" << std::endl;

Stack s;

s.push(1);

s.push(2);

s.push(3);

std::cout << s.peek() << std::endl; // Output: 3

s.pop();

std::cout << s.peek() << std::endl; // Output: 2

s.pop();

std::cout << s.peek() << std::endl; // Output: 1

s.pop();

s.pop(); // Error: stack is empty

std::cout << std::endl << std::endl;

std::cout << "-------------------QUEUE------------------" << std::endl;

Queue q;

q.enqueue(1);

q.enqueue(2);

q.enqueue(3);

std::cout << q.peek() << std::endl; // Output: 1

q.dequeue();

std::cout << q.peek() << std::endl; // Output: 2

q.dequeue();

std::cout << q.peek() << std::endl; // Output: 3

q.dequeue();

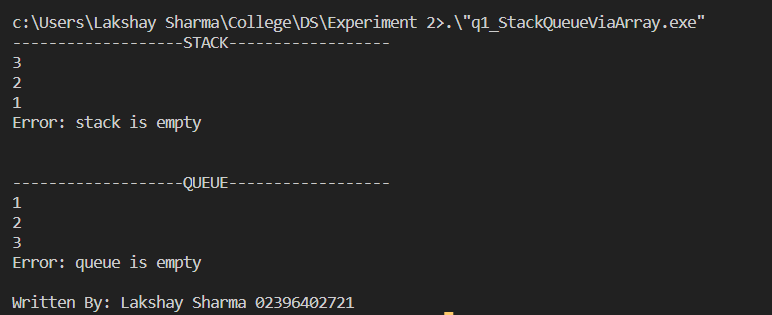
q.dequeue(); // Error: queue is empty

std::cout << "\nWritten By: Lakshay Sharma 02396402721";

return 0;

}

**Output ::**

****

## Program 2

**Aim ::** Write a program to implement two stacks using single array.

**Code ::**

#include <iostream>

#include <cstdlib>

using namespace std;

const int STACK\_SIZE = 10;

class Stack {

private:

int\* arr;

int top1, top2;

public:

Stack() {

arr = new int[STACK\_SIZE];

top1 = -1;

top2 = STACK\_SIZE;

}

~Stack() {

delete[] arr;

}

void push1(int value) {

if (top1 + 1 == top2) {

cout << "Error: stack overflow" << endl;

return;

}

arr[++top1] = value;

}

void push2(int value) {

if (top1 + 1 == top2) {

cout << "Error: stack overflow" << endl;

return;

}

arr[--top2] = value;

}

int pop1() {

if (top1 < 0) {

cout << "Error: stack underflow" << endl;

exit(EXIT\_FAILURE);

}

return arr[top1--];

}

int pop2() {

if (top2 > STACK\_SIZE - 1) {

cout << "Error: stack underflow" << endl;

exit(EXIT\_FAILURE);

}

return arr[top2++];

}

};

int main() {

Stack stack;

stack.push1(1);

stack.push1(2);

stack.push2(3);

stack.push2(4);

cout << stack.pop1() << endl;

cout << stack.pop1() << endl;

cout << stack.pop2() << endl;

cout << stack.pop2() << endl;

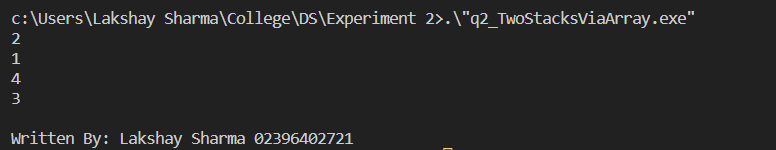
std::cout << "\nWritten By: Lakshay Sharma 02396402721";

return 0;

}

/\* Two stack Array :: 2 1 X X X X X X 3 4 \*/

**Output ::**

****

# **Experiment 3**

## Program 1

**Aim ::** Write a program showing the implementation of Linked list

**Code ::**

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

int main() {

Node\* n1 = new Node();

Node\* n2 = new Node();

Node\* n3 = new Node();

n1->data = 11; n1->next = n2;

n2->data = 13; n2->next = n3;

n3->data = 15; n3->next = NULL;

/\*

| 11 , &n2 | -> | 13 , &n3 | -> | 15 , NULL |

\*/

std::cout << "Linked List :: ";

std::cout << n1->data << " -> " << n2->data << " -> " << n3->data << std::endl;

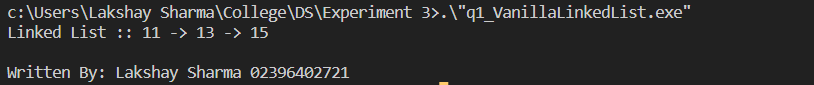
delete(n1); delete(n2); delete(n3);

std::cout << "\nWritten By: Lakshay Sharma 02396402721";

return 0;

}

**Output ::**

****

## Program 2

**Aim ::** Write a program for the insertion (at beginning,end and any position) in a Linked list.

**Code ::**

#include <iostream>

struct Node

{

int data;

Node\* next;

};

class LinkedList

{

private:

Node\* head;

int list\_length;

public:

LinkedList() {

head = NULL;

}

~LinkedList() {

Node\* current = head;

while (current != NULL) {

Node\* temp = current;

current = current->next;

delete temp;

}

}

void insertAtBeginning(int value){

Node\* newNode = new Node();

newNode->data = value;

newNode->next = head;

head = newNode;

list\_length++;

}

void insertAtEnd(int value){

if (head == NULL) {insertAtBeginning(value); return;};

Node\* temp = head;

while (temp->next != NULL) {

temp = temp->next;

}

Node\* newNode = new Node();

newNode->data = value;

newNode->next = NULL;

temp->next = newNode;

list\_length++;

}

void insertAtIndex(int value, int index){

if (index == 0) insertAtBeginning(value);

else if ((index == list\_length-1) || ( index == -1)) insertAtEnd(value);

else{

Node\* temp = head;

for (int i = 0; i < index-1; i++) temp = temp->next;

Node\* newNode = new Node();

newNode->data = value;

newNode->next = temp->next;

temp->next = newNode;

}

list\_length++;

}

friend std::ostream& operator<<(std::ostream& os, LinkedList& myList) {

Node\* temp = myList.head;

while (temp != NULL) {

std::cout << temp->data << " -> ";

temp = temp->next;

}

return os << "";

}

};

int main() {

LinkedList l1;

l1.insertAtBeginning(1); // 1

std::cout << "insertAtBeginning(1) Linked List :: \t" << l1 << std::endl;

l1.insertAtBeginning(2); // 2 -> 1

std::cout << "insertAtBeginning(2) Linked List :: \t" << l1 << std::endl;

l1.insertAtEnd(3); // 2 -> 1 -> 3

std::cout << "insertAtEnd(3) Linked List :: \t\t" << l1 << std::endl;

l1.insertAtEnd(4); // 2 -> 1 -> 3 -> 4

std::cout << "insertAtEnd(4) Linked List :: \t\t" << l1 << std::endl;

l1.insertAtIndex(5,0); // 5 -> 2 -> 1 -> 3 -> 4

std::cout << "insertAtIndex(5,0) Linked List :: \t" << l1 << std::endl;

l1.insertAtIndex(6,3); // 5 -> 2 -> 1 -> 3 -> 4

std::cout << "insertAtIndex(6,3) Linked List :: \t" << l1 << std::endl;

l1.insertAtIndex(7,-1); // 5 -> 2 -> 1 -> 3 -> 4

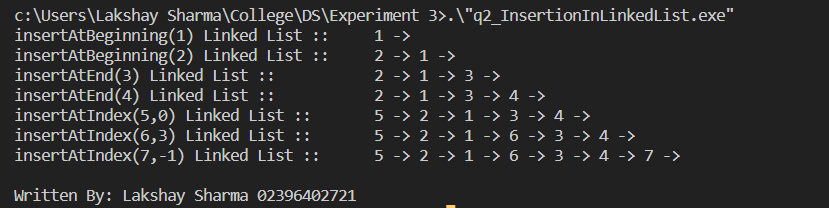
std::cout << "insertAtIndex(7,-1) Linked List :: \t" << l1 << std::endl;

std::cout << "\nWritten By: Lakshay Sharma 02396402721";

return 0;

}

**Output ::**

****

## Program 3

**Aim ::** Write a program for the deletion (at beginning,end and any position) in a Linked list..

**Code ::**

#include <iostream>

struct Node

{

int data;

Node\* next;

};

class LinkedList

{

private:

Node\* head;

int list\_length;

public:

LinkedList() {

head = NULL;

}

~LinkedList() {

Node\* current = head;

while (current != NULL) {

Node\* temp = current;

current = current->next;

delete temp;

}

}

void insertAtBeginning(int value){

Node\* newNode = new Node();

newNode->data = value;

newNode->next = head;

head = newNode;

list\_length++;

}

void insertAtEnd(int value){

if (head == NULL) {insertAtBeginning(value); return;};

Node\* temp = head;

while (temp->next != NULL) {

temp = temp->next;

}

Node\* newNode = new Node();

newNode->data = value;

newNode->next = NULL;

temp->next = newNode;

list\_length++;

}

void deleteAtBeginning(){

if(list\_length != 0){

Node\* temp = head;

head = head->next;

delete temp;

list\_length--;

}

else std::cout << "List is empty" << std::endl;

}

void deleteAtEnd(){

if(list\_length != 0){

Node\* temp = head;

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

temp = temp->next;

}

delete temp->next;

temp->next = NULL;

list\_length--;

}

else std::cout << "List is empty" << std::endl;

}

void deleteAtIndex(int index){

if(list\_length != 0){

if (index == 0) deleteAtBeginning();

else if ((index == list\_length-1) || ( index == -1)) deleteAtEnd();

else{

Node\* temp = head;

for (int i = 0; i < index-1; i++) temp = temp->next;

Node\* toDelete = temp->next;

temp->next = temp->next->next;

delete(toDelete);

}

list\_length--;

}

else std::cout << "List is empty" << std::endl;

}

friend std::ostream& operator<<(std::ostream& os, LinkedList& myList) {

Node\* temp = myList.head;

while (temp != NULL) {

std::cout << temp->data << " -> ";

temp = temp->next;

}

return os << "";

}

};

int main()

{

LinkedList l1;

l1.insertAtEnd(1);

l1.insertAtEnd(2);

l1.insertAtEnd(3);

l1.insertAtEnd(4);

l1.insertAtEnd(5);

l1.insertAtEnd(6);

l1.insertAtEnd(7);

l1.insertAtEnd(8);

//1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8

std::cout << "Linked List :: \t\t\t\t" << l1 << std::endl;

l1.deleteAtBeginning(); // 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8

std::cout << "deleteAtBeginning() Linked List :: \t" << l1 << std::endl;

l1.deleteAtEnd(); // 2 -> 3 -> 4 -> 5 -> 6 -> 7

std::cout << "deleteAtEnd() Linked List :: \t\t" << l1 << std::endl;

l1.deleteAtIndex(0); // 3 -> 4 -> 5 -> 6 -> 7

std::cout << "deleteAtIndex(0) Linked List :: \t" << l1 << std::endl;

l1.deleteAtIndex(3); // 3 -> 4 -> 5 -> 7

std::cout << "deleteAtIndex(3) Linked List :: \t" << l1 << std::endl;

l1.deleteAtIndex(-1); // 3 -> 4 -> 5

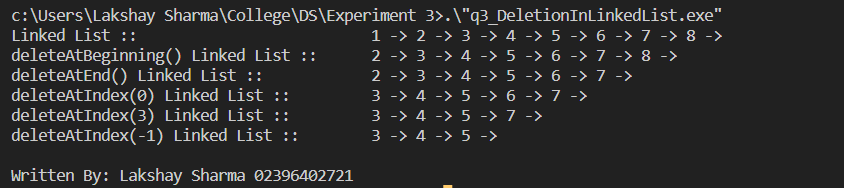
std::cout << "deleteAtIndex(-1) Linked List :: \t" << l1 << std::endl;

std::cout << "\nWritten By: Lakshay Sharma 02396402721";

return 0;

}

**Output ::**

****

## Program 4

**Aim ::** Write a program for the reverse of a Linked list.

**Code ::**

#include <iostream>

struct Node

{

int data;

Node\* next;

};

class LinkedList

{

private:

Node\* head;

int list\_length;

public:

LinkedList() {

head = NULL;

}

~LinkedList() {

Node\* current = head;

while (current != NULL) {

Node\* temp = current;

current = current->next;

delete temp;

}

}

void insertAtBeginning(int value){

Node\* newNode = new Node();

newNode->data = value;

newNode->next = head;

head = newNode;

list\_length++;

}

void insertAtEnd(int value){

if (head == NULL) {insertAtBeginning(value); return;};

Node\* temp = head;

while (temp->next != NULL) {

temp = temp->next;

}

Node\* newNode = new Node();

newNode->data = value;

newNode->next = NULL;

temp->next = newNode;

list\_length++;

}

void reverse() {

Node\* current = head;

Node\* prev = NULL;

Node\* next = NULL;

while (current != NULL) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

head = prev;

}

friend std::ostream& operator<<(std::ostream& os, LinkedList& myList) {

Node\* temp = myList.head;

while (temp != NULL) {

std::cout << temp->data << " -> ";

temp = temp->next;

}

return os << "";

}

};

int main()

{

LinkedList l1;

l1.insertAtEnd(1);

l1.insertAtEnd(2);

l1.insertAtEnd(3);

l1.insertAtEnd(4);

l1.insertAtEnd(5);

l1.insertAtEnd(6);

l1.insertAtEnd(7);

l1.insertAtEnd(8);

//1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8

std::cout << "Linked List :: \t\t" << l1 << std::endl;

l1.reverse(); // 8 -> 7 -> 6 -> 5 -> 4 -> 3 -> 2 -> 1

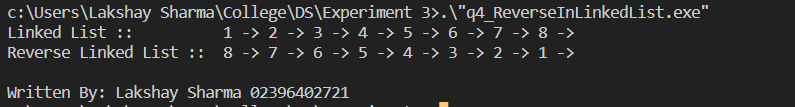
std::cout << "Reverse Linked List :: \t" << l1 << std::endl;

std::cout << "\nWritten By: Lakshay Sharma 02396402721";

return 0;

}

**Output ::**

****

# **Experiment 4**

## Program 1

**Aim ::** Write a program showing the implementation of Doubly and Circular linked list?

**Code ::**

#include <iostream>

struct Node{

int data;

Node\* next;

Node\* prev;

};

int main() {

Node\* d1 = new Node();

Node\* d2 = new Node();

Node\* d3 = new Node();

Node\* d4 = new Node();

Node\* c1 = new Node();

Node\* c2 = new Node();

Node\* c3 = new Node();

d1->prev = NULL; d1->data = 11; d1->next = d2;

d2->prev = d1; d2->data = 12; d2->next = d3;

d3->prev = d2; d3->data = 13; d3->next = d4;

d4->prev = d3; d4->data = 14; d4->next = NULL;

Node\* temp = d1;

std::cout << "Doubly Linked List :: \t\t";

while (temp->next != NULL){

std::cout << temp->data << " " << char(29) << " ";

temp = temp->next;

}

std::cout << temp->data << std::endl;

c1->prev = c3; c1->data = 21; c1->next = c2;

c2->prev = c2; c2->data = 22; c2->next = c3;

c3->prev = c1; c3->data = 23; c3->next = c1;

Node\* temp2 = c1;

std::cout << "Circular Linked List :: \t" << char(4) << " ";

while (temp2->next != c1){

std::cout << temp2->data << " " << char(29) << " ";

temp2 = temp2->next;

}

std::cout << temp2->data << " " <<char(4) << std::endl;

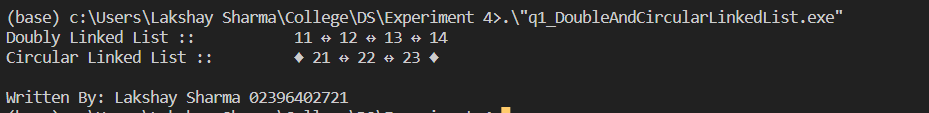
delete(d1);delete(d2);delete(d3);delete(d4);delete(c1);delete(c2);delete(c3);

std::cout << "\nWritten By: Lakshay Sharma 02396402721";

return 0;

}

**Output ::**

****

## Program 2

**Aim ::** Write a program for the insertion (at beginning, end and any position) in a Doubly and Circular linked list.

**Code ::**

#include <iostream>

class DoublyLinkedList {

private:

// Definition for a node in the linked list

struct Node {

int data;

Node\* next;

Node\* prev;

};

Node\* head;

Node\* tail;

int size;

public:

DoublyLinkedList() {

head = NULL;

tail = NULL;

size = 0;

}

// Insert a new node at the front of the list

void insertAtBeginning(int data) {

Node\* newNode = new Node;

newNode->data = data;

newNode->prev = NULL;

newNode->next = head;

if (head != NULL) {

head->prev = newNode;

}

head = newNode;

if (tail == NULL) {

tail = newNode;

}

size++;

}

// Insert a new node at the back of the list

void insertAtEnd(int data) {

Node\* newNode = new Node;

newNode->data = data;

newNode->next = NULL;

newNode->prev = tail;

if (tail != NULL) {

tail->next = newNode;

}

tail = newNode;

if (head == NULL) {

head = newNode;

}

size++;

}

void insertAtPos(int pos, int data) {

Node \*newNode = new Node;

newNode->data = data;

newNode->prev = NULL;

newNode->next = NULL;

if (pos == 0) {

insertAtBeginning(data);

return;

}

Node \*temp = head;

int i = 0;

while (i < pos - 1 && temp->next != NULL) {

temp = temp->next;

i++;

}

if (temp->next == NULL) {

std::cout << "Invalid position!" << std::endl;

}

else {

newNode->prev = temp;

newNode->next = temp->next;

temp->next->prev = newNode;

temp->next = newNode;

}

}

// Prints out the list

friend std::ostream& operator<<(std::ostream& os, DoublyLinkedList& myList) {

Node\* temp = myList.head;

while (temp != NULL) {

std::cout << temp->data << " " << char(29) << " ";

temp = temp->next;

}

return os << "";

}

// Check if the list is empty

bool empty() {

return size == 0;

}

};

class CircularLinkedList {

private:

struct Node {

int data;

Node \*next;

};

Node \*head;

public:

CircularLinkedList() {

head = NULL;

}

void insertAtEnd(int data) {

Node \*newNode = new Node;

newNode->data = data;

newNode->next = NULL;

if (head == NULL) {

head = newNode;

newNode->next = head;

}

else {

Node \*temp = head;

while (temp->next != head) {

temp = temp->next;

}

temp->next = newNode;

newNode->next = head;

}

}

void insertAtBegin(int data) {

Node \*newNode = new Node;

newNode->data = data;

newNode->next = NULL;

if (head == NULL) {

head = newNode;

newNode->next = head;

}

else {

Node \*temp = head;

while (temp->next != head) {

temp = temp->next;

}

temp->next = newNode;

newNode->next = head;

head = newNode;

}

}

void insertAtPos(int pos, int data) {

Node \*newNode = new Node;

newNode->data = data;

newNode->next = NULL;

if (pos == 0) {

insertAtBegin(data);

return;

}

Node \*temp = head;

int i = 0;

while (i < pos - 1 && temp->next != head) {

temp = temp->next;

i++;

}

if (temp->next == head) {

std::cout << "Invalid position!" << std::endl;

} else {

newNode->next = temp->next;

temp->next = newNode;

}

}

// Prints out the list

friend std::ostream& operator<<(std::ostream& os, CircularLinkedList& myList) {

Node\* temp = myList.head;

while (temp->next != myList.head) {

std::cout << temp->data << " " << char(29) << " ";

temp = temp->next;

}

return os << temp->data;

}

};

int main() {

DoublyLinkedList list;

list.insertAtEnd(1);

std::cout << "< insert at end = 1 > Doubly Linked List :: \t\t" << list <<std::endl;

list.insertAtEnd(2);

std::cout << "< insert at end = 2 > Doubly Linked List :: \t\t" << list <<std::endl;

list.insertAtBeginning(3);

std::cout << "< insert at begin = 3 > Doubly Linked List :: \t\t" << list <<std::endl;

list.insertAtBeginning(4);

std::cout << "< insert at begin = 4 > Doubly Linked List :: \t\t" << list <<std::endl;

list.insertAtPos(2, 5);

std::cout << "< insert at pos(2) = 5 > Doubly Linked List :: \t\t" << list <<std::endl << std::endl;

CircularLinkedList clist;

clist.insertAtEnd(11);

std::cout << "< insert at end = 1 > Circular Linked List :: \t\t" << clist <<std::endl;

clist.insertAtEnd(12);

std::cout << "< insert at end = 2 > Circular Linked List :: \t\t" << clist <<std::endl;

clist.insertAtBegin(13);

std::cout << "< insert at begin = 3 > Circular Linked List :: \t" << clist <<std::endl;

clist.insertAtBegin(14);

std::cout << "< insert at begin = 4 > Circular Linked List :: \t" << clist <<std::endl;

clist.insertAtPos(2, 15);

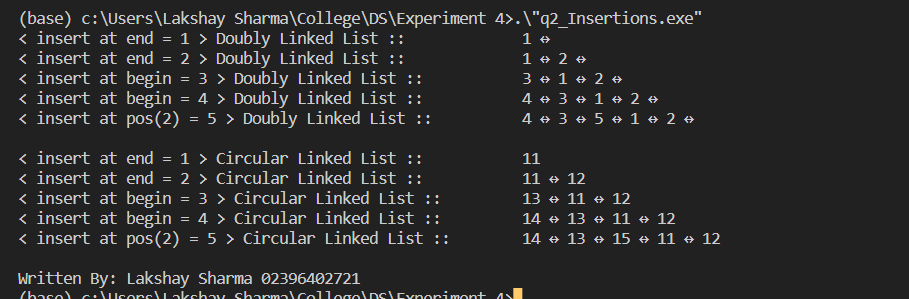
std::cout << "< insert at pos(2) = 5 > Circular Linked List :: \t" << clist <<std::endl;

std::cout << "\nWritten By: Lakshay Sharma 02396402721";

return 0;

}

**Output ::**

****

## Program 3

**Aim ::** Write a program for the deletion (at beginning, end and any position) in a Doubly and Circular linked list.

**Code ::**

#include <iostream>

class DoublyLinkedList {

private:

// Definition for a node in the linked list

struct Node {

int data;

Node\* next;

Node\* prev;

};

Node\* head;

Node\* tail;

int size;

public:

DoublyLinkedList() {

head = NULL;

tail = NULL;

size = 0;

}

// Insert a new node at the back of the list

void insertAtEnd(int data) {

Node\* newNode = new Node;

newNode->data = data;

newNode->next = NULL;

newNode->prev = tail;

if (tail != NULL) {

tail->next = newNode;

}

tail = newNode;

if (head == NULL) {

head = newNode;

}

size++;

}

// Remove the front node of the list

void deleteAtBeginning() {

if (head == NULL) {

return;

}

Node\* temp = head;

head = head->next;

if (head != NULL) {

head->prev = NULL;

}

delete temp;

size--;

}

// Remove the back node of the list

void deleteAtEnd() {

if (tail == NULL) {

return;

}

Node\* temp = tail;

tail = tail->prev;

if (tail != NULL) {

tail->next = NULL;

}

delete temp;

size--;

}

void deleteAtPos(int pos) {

Node \*temp = head;

if (pos == 0) { deleteAtBeginning(); return;}

int i = 0;

while (i < pos && temp->next != NULL) {

temp = temp->next;

i++;

}

if (temp->next == NULL) {

std::cout << "Invalid position!" << std::endl;

}

else {

temp->prev->next = temp->next;

temp->next->prev = temp->prev;

delete temp;

}

}

// Prints out the list

friend std::ostream& operator<<(std::ostream& os, DoublyLinkedList& myList) {

Node\* temp = myList.head;

while (temp != NULL) {

std::cout << temp->data << " " << char(29) << " ";

temp = temp->next;

}

return os << "";

}

// Check if the list is empty

bool empty() {

return size == 0;

}

};

class CircularLinkedList {

private:

struct Node {

int data;

Node \*next;

};

Node \*head;

public:

CircularLinkedList() {

head = NULL;

}

void insertAtEnd(int data) {

Node \*newNode = new Node;

newNode->data = data;

newNode->next = NULL;

if (head == NULL) {

head = newNode;

newNode->next = head;

}

else {

Node \*temp = head;

while (temp->next != head) {

temp = temp->next;

}

temp->next = newNode;

newNode->next = head;

}

}

void deleteAtEnd() {

Node \*temp = head;

Node \*prev = NULL;

while (temp->next != head) {

prev = temp;

temp = temp->next;

}

prev->next = head;

delete temp;

}

void deleteAtBegin() {

Node \*temp = head;

Node \*prev = head;

while (temp->next != head) {

temp = temp->next;

}

temp->next = head->next;

delete prev;

head = temp->next;

}

void deleteAtPos(int pos) {

Node \*temp = head;

Node \*prev = NULL;

if (pos == 0) { deleteAtBegin(); return; };

int i = 0;

while (i < pos && temp->next != head) {

prev = temp;

temp = temp->next;

i++;

}

if (temp->next == head) {

std::cout << "Invalid position!" << std::endl;

} else {

prev->next = temp->next;

delete temp;

}

}

// Prints out the list

friend std::ostream& operator<<(std::ostream& os, CircularLinkedList& myList) {

Node\* temp = myList.head;

while (temp->next != myList.head) {

std::cout << temp->data << " -> ";

temp = temp->next;

}

return os << temp->data;

}

};

int main() {

DoublyLinkedList list;

list.insertAtEnd(1);

list.insertAtEnd(2);

list.insertAtEnd(3);

list.insertAtEnd(4);

list.insertAtEnd(5);

std::cout << "< inserted 1,2,3,4,5 > Doubly Linked List :: \t" << list <<std::endl;

list.deleteAtEnd();

std::cout << "< delete at end > Doubly Linked List :: \t" << list <<std::endl;

list.deleteAtBeginning();

std::cout << "< delete at begin > Doubly Linked List :: \t" << list <<std::endl;

list.deleteAtPos(1);

std::cout << "< delete at pos=1 > Doubly Linked List :: \t" << list <<std::endl << std::endl;

CircularLinkedList clist;

clist.insertAtEnd(1);

clist.insertAtEnd(2);

clist.insertAtEnd(3);

clist.insertAtEnd(4);

clist.insertAtEnd(5);

std::cout << "< inserted 1,2,3,4,5 > Circular Linked List :: \t" << clist <<std::endl;

clist.deleteAtEnd();

std::cout << "< delete at end > Circular Linked List :: \t" << clist <<std::endl;

clist.deleteAtBegin();

std::cout << "< delete at begin > Circular Linked List :: \t" << clist <<std::endl;

clist.deleteAtPos(1);

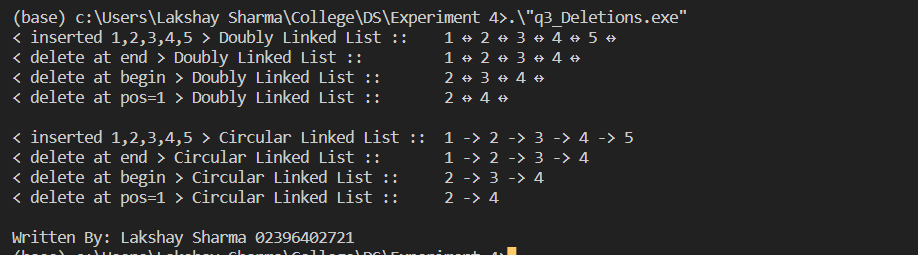
std::cout << "< delete at pos=1 > Circular Linked List :: \t" << clist <<std::endl;

std::cout << "\nWritten By: Lakshay Sharma 02396402721";

return 0;

}

**Output ::**

****

## Program 4

**Aim ::** Write a program for the reverse of a Doubly and Circular linked list.

**Code ::**

#include <iostream>

class DoublyLinkedList {

private:

// Definition for a node in the linked list

struct Node {

int data;

Node\* next;

Node\* prev;

};

Node\* head;

Node\* tail;

int size;

public:

DoublyLinkedList() {

head = NULL;

tail = NULL;

size = 0;

}

// Insert a new node at the back of the list

void insertAtEnd(int data) {

Node\* newNode = new Node;

newNode->data = data;

newNode->next = NULL;

newNode->prev = tail;

if (tail != NULL) {

tail->next = newNode;

}

tail = newNode;

if (head == NULL) {

head = newNode;

}

size++;

}

void reverse(){

Node\* temp = head;

Node\* tempp = head->prev;

Node\* tempn = head->next;

while (temp->next != NULL){

temp = temp->next;

tempp = temp->prev->prev;

tempn = temp;

temp->prev->next = tempp;

temp->prev->prev = tempn;

}

tempp = temp->prev;

tempn = temp->next;

temp->next = tempp;

temp->prev = tempn;

head = temp;

}

// Prints out the list

friend std::ostream& operator<<(std::ostream& os, DoublyLinkedList& myList) {

Node\* temp = myList.head;

while (temp != NULL) {

std::cout << temp->data << " " << char(29) << " ";

temp = temp->next;

}

return os << "";

}

};

class CircularLinkedList {

private:

struct Node {

int data;

Node \*next;

};

Node \*head;

public:

CircularLinkedList() {

head = NULL;

}

void insertAtEnd(int data) {

Node \*newNode = new Node;

newNode->data = data;

newNode->next = NULL;

if (head == NULL) {

head = newNode;

newNode->next = head;

}

else {

Node \*temp = head;

while (temp->next != head) {

temp = temp->next;

}

temp->next = newNode;

newNode->next = head;

}

}

void reverse() {

if(head != NULL) {

Node\* prev = head;

Node\* temp = head;

Node\* curr = head->next;

prev->next = prev;

while(curr != head) {

temp = curr->next;

curr->next = prev;

head->next = curr;

prev = curr;

curr = temp;

}

head = prev;

}

}

// Prints out the list

friend std::ostream& operator<<(std::ostream& os, CircularLinkedList& myList) {

Node\* temp = myList.head;

while (temp->next != myList.head) {

std::cout << temp->data << " -> ";

temp = temp->next;

}

return os << temp->data;

}

};

int main(){

DoublyLinkedList list;

list.insertAtEnd(1);

list.insertAtEnd(2);

list.insertAtEnd(3);

list.insertAtEnd(4);

list.insertAtEnd(5);

std::cout << "< original > Doubly Linked List :: \t" << list <<std::endl;

list.reverse();

std::cout << "< reversed > Doubly Linked List :: \t" << list <<std::endl << std::endl;

CircularLinkedList clist;

clist.insertAtEnd(1);

clist.insertAtEnd(2);

clist.insertAtEnd(3);

clist.insertAtEnd(4);

clist.insertAtEnd(5);

std::cout << "< original > Circular Linked List :: \t" << clist <<std::endl;

clist.reverse();

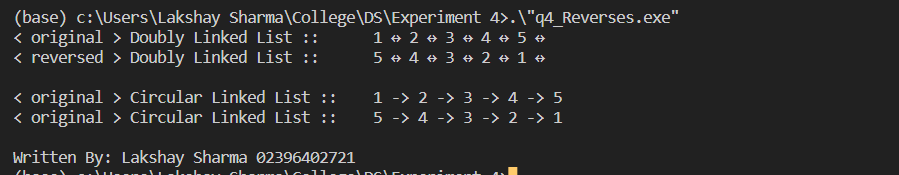
std::cout << "< original > Circular Linked List :: \t" << clist <<std::endl;

std::cout << "\nWritten By: Lakshay Sharma 02396402721";

return 0;

}

**Output ::**

****

# **Experiment 5**

## Program 1

**Aim ::** Write a program showing the implementation of Stack and Queue with Linked list.

**Code ::**

#include <iostream>

struct Node {

int data;

Node\* next;

};

// Stack class with push, pop, and peek operations

class Stack {

private:

Node\* top;

public:

Stack() { top = NULL; }

void push(int data) {

Node\* newNode = new Node;

newNode->data = data;

newNode->next = top;

// Set the new node as the top of the stack

top = newNode;

}

int pop() {

// Return -1 if the stack is empty

if (top == NULL) return -1;

int data = top->data;

Node\* temp = top;

top = top->next;

delete temp;

return data;

}

int peek() {

// Return -1 if the stack is empty

if (top == NULL) return -1;

return top->data;

}

};

// Queue class with enqueue, dequeue, and peek operations

class Queue {

private:

Node\* head;

Node\* tail;

public:

Queue() { head = tail = NULL; }

void enqueue(int data) {

Node\* newNode = new Node;

newNode->data = data;

newNode->next = NULL;

// If the queue is empty, set the new node as both the head and tail

if (tail == NULL) {

head = tail = newNode;

}

// Otherwise, set the new node as the tail and update the tail pointer

else {

tail->next = newNode;

tail = newNode;

}

}

int dequeue() {

// Return -1 if the queue is empty

if (head == NULL) return -1;

int data = head->data;

Node\* temp = head;

head = head->next;

// If the head is now null, set the tail to be null as well (queue is empty)

if (head == NULL) tail = NULL;

delete temp;

return data;

}

int peek() {

// Return -1 if the queue is empty

if (head == nullptr) return -1;

return head->data;

}

};

int main() {

// Create a stack and push some values onto it

Stack stack;

stack.push(1);

stack.push(2);

stack.push(3);

// Pop and print the values from the stack

std::cout << "Stack :: " ;

std::cout << stack.pop() << " ";

std::cout << stack.pop() << " ";

std::cout << stack.pop() << std::endl << std::endl;

// Create a queue and enqueue some values

Queue queue;

queue.enqueue(1);

queue.enqueue(2);

queue.enqueue(3);

// Dequeue and print the values from the queue

std::cout << "Queue :: " ;

std::cout << queue.dequeue() << " ";

std::cout << queue.dequeue() << " ";

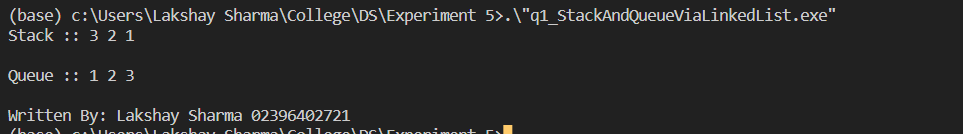
std::cout << queue.dequeue() << std::endl;

std::cout << "\nWritten By: Lakshay Sharma 02396402721";

return 0;

}

**Output ::**

****

# **Experiment 6**

## Program 1

**Aim ::** Write a program to create a Binary tree and perform traversal ( pre-order, post-order, In-order).

**Code ::**

#include <iostream>

using namespace std;

struct Node {

int data;

Node \*left;

Node \*right;

};

Node\* createNode(int data) {

Node \*newNode = new Node;

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

void preOrderTraversal(Node \*root) {

if (root == NULL) return;

cout << root->data << " ";

preOrderTraversal(root->left);

preOrderTraversal(root->right);

}

void inOrderTraversal(Node \*root) {

if (root == NULL) return;

inOrderTraversal(root->left);

cout << root->data << " ";

inOrderTraversal(root->right);

}

void postOrderTraversal(Node \*root) {

if (root == NULL) return;

postOrderTraversal(root->left);

postOrderTraversal(root->right);

cout << root->data << " ";

}

int main() {

Node \*root = createNode(1);

root->left = createNode(2);

root->right = createNode(3);

root->left->left = createNode(4);

root->left->right = createNode(5);

cout << "Pre-order traversal: ";

preOrderTraversal(root);

cout << endl;

cout << "In-order traversal: ";

inOrderTraversal(root);

cout << endl;

cout << "Post-order traversal: ";

postOrderTraversal(root);

cout << endl;

std::cout << "\nWritten By: Lakshay Sharma 02396402721";

return 0;

}

**Output ::**

/\* Actual Tree 🡪

1

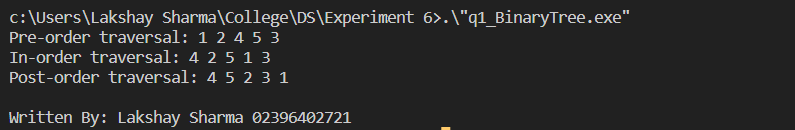
/ \

2 3

/ \

4 5

\*/

****

## Program 2

**Aim ::** Write a program to create a Binary search tree and perform traversal ( pre-order, post-order, In-order).

**Code ::**

#include <iostream>

using namespace std;

struct Node {

int data;

Node \*left;

Node \*right;

};

Node\* createNode(int data) {

Node \*newNode = new Node;

newNode->data = data;

newNode->left = NULL;

newNode->right = NULL;

return newNode;

}

Node\* insert(Node \*root, int data) {

if (root == NULL) root = createNode(data);

else if (data <= root->data) root->left = insert(root->left, data);

else root->right = insert(root->right, data);

return root;

}

void preOrderTraversal(Node \*root) {

if (root == NULL) return;

cout << root->data << " ";

preOrderTraversal(root->left);

preOrderTraversal(root->right);

}

void inOrderTraversal(Node \*root) {

if (root == NULL) return;

inOrderTraversal(root->left);

cout << root->data << " ";

inOrderTraversal(root->right);

}

void postOrderTraversal(Node \*root) {

if (root == NULL) return;

postOrderTraversal(root->left);

postOrderTraversal(root->right);

cout << root->data << " ";

}

int main() {

Node \*root = NULL;

root = insert(root, 5);

root = insert(root, 3);

root = insert(root, 7);

root = insert(root, 2);

root = insert(root, 4);

root = insert(root, 6);

root = insert(root, 8);

cout << "Pre-order traversal: ";

preOrderTraversal(root);

cout << endl;

cout << "In-order traversal: ";

inOrderTraversal(root);

cout << endl;

cout << "Post-order traversal: ";

postOrderTraversal(root);

cout << endl;

std::cout << "\nWritten By: Lakshay Sharma 02396402721";

return 0;

}

**Output ::**

/\* Actual Tree

5

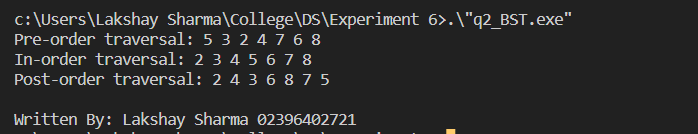
/ \

3 7

/ \ / \

2 4 6 8

\*/

****

# **Experiment 7**

## Program 1

**Aim ::** Write a program showing the implementation of Insertion sort?

**Code ::**

#include <iostream>

#include <vector>

// Function to sort an array using insertion sort

void insertionSort(std::vector<int>& arr)

{

int n = arr.size();

// Iterate over the array, starting from the second element

for (int i = 1; i < n; i++) {

// Get the current element

int curr = arr[i];

// Compare the current element with the elements to its left, until it is in the correct position

int j = i - 1;

while (j >= 0 && arr[j] > curr) {

// Shift the elements to the right to make room for the current element

arr[j + 1] = arr[j];

j--;

}

// Insert the current element into its correct position

arr[j + 1] = curr;

std::cout << "Step " << i << " : \t" ;

for (int x : arr) {

std::cout << x << " ";

}

std::cout << std::endl;

}

}

int main()

{

// Test the insertion sort function

std::vector<int> arr = { 4, 2, 1, 3, 6, 5 };

std::cout << "Input : \t" ;

for (int x : arr) {

std::cout << x << " ";

}

std::cout << std::endl;

insertionSort(arr);

std::cout << "Output : \t" ;

for (int x : arr) {

std::cout << x << " ";

}

std::cout << std::endl;

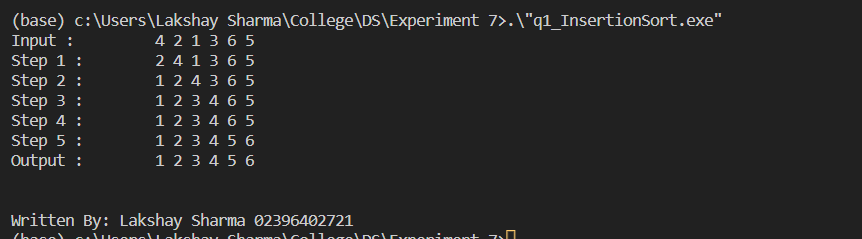
std::cout << std::endl;

std::cout << "\nWritten By: Lakshay Sharma 02396402721";

return 0;

}

**Output ::**

****

## Program 2

**Aim ::** Write a program showing the implementation of Merge sort?

**Code ::**

#include <iostream>

#include <vector>

// Function to merge two sorted arrays

std::vector<int> merge(const std::vector<int>& left, const std::vector<int>& right)

{

// Create a result vector

std::vector<int> result;

// Set up two indices, one for each array

int i = 0, j = 0;

// Iterate until one of the indices reaches the end of its array

while (i < left.size() && j < right.size()) {

// Compare the elements at the current indices and add the smaller one to the result

if (left[i] < right[j]) {

result.push\_back(left[i++]);

}

else {

result.push\_back(right[j++]);

}

}

// Add the remaining elements from the left array, if any

while (i < left.size()) {

result.push\_back(left[i++]);

}

// Add the remaining elements from the right array, if any

while (j < right.size()) {

result.push\_back(right[j++]);

}

std::cout << "Merged : \t" ;

for (int x : result) {

std::cout << x << " ";

}

std::cout << "\n" ;

return result;

}

// Recursive function to sort an array using merge sort

std::vector<int> mergeSort(std::vector<int>& arr) {

// Base case: If the array has 1 or fewer elements, it is already sorted

if (arr.size() <= 1) {

return arr;

}

// Split the array in half

int mid = arr.size() / 2;

std::vector<int> left(arr.begin(), arr.begin() + mid);

std::vector<int> right(arr.begin() + mid, arr.end());

std::cout << "Left : \t" ;

for (int x : left) {

std::cout << x << " ";

}

std::cout << "\t\t" ;

std::cout << "Right : \t" ;

for (int x : right) {

std::cout << x << " ";

}

std::cout << std::endl;

// Recursively sort the left and right halves

left = mergeSort(left);

right = mergeSort(right);

// Merge the sorted left and right halves

return merge(left, right);

}

int main() {

// Test the merge sort function

std::vector<int> arr = { 4, 2, 1, 3, 6, 5 };

std::cout << "Input : \t" ;

for (int x : arr) {

std::cout << x << " ";

}

std::cout << std::endl;

arr = mergeSort(arr);

std::cout << "Output : \t" ;

for (int x : arr) {

std::cout << x << " ";

}

std::cout << std::endl;

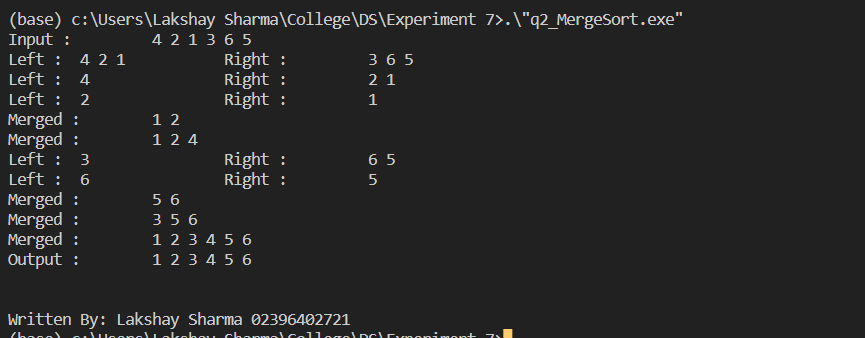
std::cout << std::endl;

std::cout << "\nWritten By: Lakshay Sharma 02396402721";

return 0;

}

**Output ::**

****

## Program 3

**Aim ::** Write a program showing the implementation of Selection sort?

**Code ::**

#include <iostream>

#include <vector>

// Function to sort an array using selection sort

void selectionSort(std::vector<int>& arr)

{

int n = arr.size();

// Iterate over the array, starting from the first element

for (int i = 0; i < n - 1; i++) {

// Find the index of the minimum element in the unsorted portion of the array

int minIndex = i;

for (int j = i + 1; j < n; j++) {

if (arr[j] < arr[minIndex]) {

minIndex = j;

}

std::cout << "Step " << i << "." << j << " : \t" ;

for (int x : arr) {

std::cout << x << " ";

}

std::cout << "\t<" << arr[minIndex] << ">" << std::endl;

}

// Swap the minimum element with the first element of the unsorted portion

int temp = arr[i];

arr[i] = arr[minIndex];

arr[minIndex] = temp;

std::cout << "Step " << i << "->" << " : \t" ;

for (int x : arr) {

std::cout << x << " ";

}

std::cout << std::endl << std::endl;

}

}

int main()

{

// Test the selection sort function

std::vector<int> arr = { 4, 2, 1, 3, 6, 5 };

std::cout << "Input : \t" ;

for (int x : arr) {

std::cout << x << " ";

}

std::cout << std::endl;

selectionSort(arr);

std::cout << "Output : \t" ;

for (int x : arr) {

std::cout << x << " ";

}

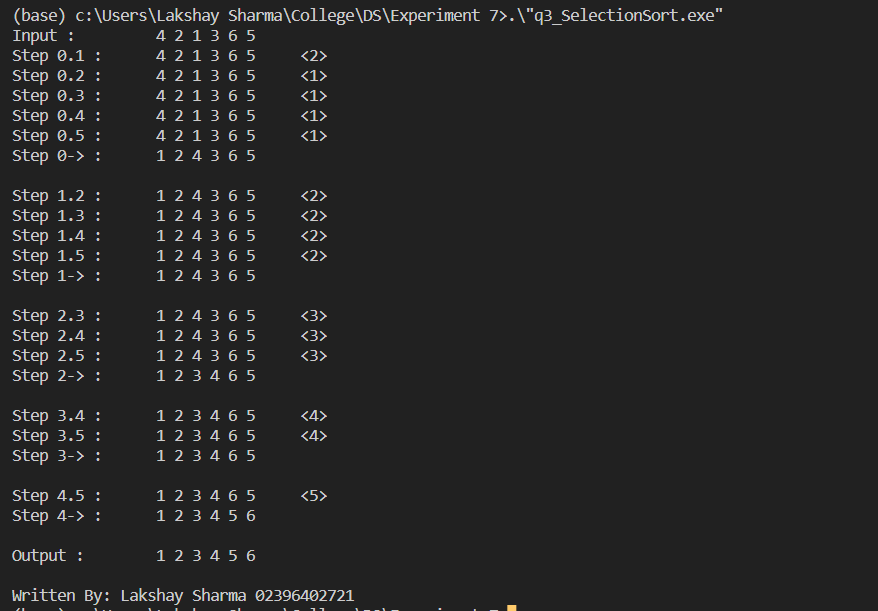
std::cout << std::endl;

std::cout << "\nWritten By: Lakshay Sharma 02396402721";

return 0;

}

**Output ::**

****

## Program 4

**Aim ::** Write a program showing the implementation of Quick sort?

**Code ::**

#include <iostream>

#include <vector>

// Function to partition an array around a pivot

int partition(std::vector<int>& arr, int low, int high)

{

// Choose the pivot as the last element in the array

int pivot = arr[high];

// Set up two indices, one for the left side of the pivot and one for the right side

int i = low - 1;

for (int j = low; j < high; j++) {

// If the current element is smaller than or equal to the pivot, swap it with the element at the left index

if (arr[j] <= pivot) {

i++;

int temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

// Swap the pivot with the element at the left index + 1

int temp = arr[i + 1];

arr[i + 1] = arr[high];

arr[high] = temp;

// Return the index of the pivot

return i + 1;

}

// Recursive function to sort an array using quick sort

void quickSort(std::vector<int>& arr, int low, int high)

{

if (low < high) {

// Partition the array around a pivot and get the pivot's index

int pivotIndex = partition(arr, low, high);

std::cout << "Step-> : \t" ;

for (int x = 0; x < arr.size() ; x++) {

if (x == pivotIndex) std::cout << "<" << arr[x] << ">" << " ";

else std::cout << arr[x] << " ";

// std::cout << arr[x] << " ";

}

std::cout << std::endl;

// Recursively sort the left and right halves of the array

quickSort(arr, low, pivotIndex - 1);

quickSort(arr, pivotIndex + 1, high);

}

}

int main()

{

// Test the quick sort function

std::vector<int> arr = { 4, 2, 1, 3, 6, 5 };

std::cout << "Input : \t" ;

for (int x : arr) {

std::cout << x << " ";

}

std::cout << std::endl;

quickSort(arr,0, arr.size()-1);

std::cout << "Output : \t" ;

for (int x : arr) {

std::cout << x << " ";

}

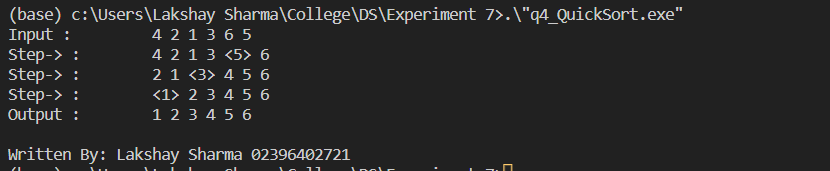
std::cout << std::endl;

std::cout << "\nWritten By: Lakshay Sharma 02396402721";

return 0;

}

**Output ::**

****

## Program 5

**Aim ::** Write a program showing the implementation of Bubble sort?

**Code ::**

#include <iostream>

#include <vector>

// Function to sort an array using bubble sort

void bubbleSort(std::vector<int>& arr)

{

int n = arr.size();

// Iterate over the array, starting from the second element

for (int i = 0; i < n - 1; i++) {

// Flag to track if any swaps were made in the current pass

bool swapped = false;

// Iterate over the unsorted portion of the array

for (int j = 0; j < n - i - 1; j++) {

// If the current element is greater than the next element, swap them

if (arr[j] > arr[j + 1]) {

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

swapped = true;

}

std::cout << "Step " << i << "." << j <<" : \t" ;

for (int x : arr) {

std::cout << x << " ";

}

std::cout << std::endl;

}

std::cout << std::endl;

// If no swaps were made, the array is already sorted

if (!swapped) {

break;

}

}

}

int main()

{

// Test the bubble sort function

std::vector<int> arr = { 4, 2, 1, 3, 6, 5 };

std::cout << "Input : \t" ;

for (int x : arr) {

std::cout << x << " ";

}

std::cout << std::endl;

bubbleSort(arr);

std::cout << "Output : \t" ;

for (int x : arr) {

std::cout << x << " ";

}

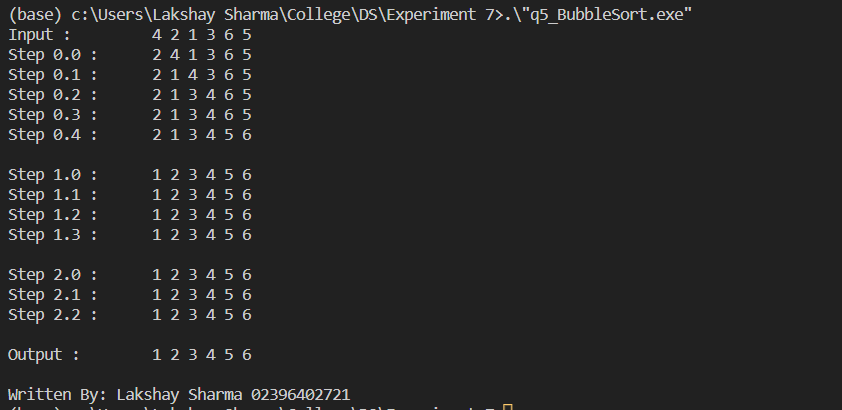
std::cout << std::endl;

std::cout << "\nWritten By: Lakshay Sharma 02396402721";

return 0;

}

**Output ::**

****

## Program 6

**Aim ::** Write a program showing the implementation of Heap sort (can be either min or max heap)?

**Code ::**

#include <iostream>

#include <vector>

// Function to heapify a subtree rooted at index i

// The subtree is assumed to already satisfy the max heap property, except for possibly the root node

void heapify(std::vector<int>& arr, int n, int i)

{

// Set up variables to store the root node and its children

int largest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

// If the left child is larger than the root, set the largest value to the left child

if (left < n && arr[left] > arr[largest]) {

largest = left;

}

// If the right child is larger than the largest value so far, set the largest value to the right child

if (right < n && arr[right] > arr[largest]) {

largest = right;

}

// If the root node is not the largest value, swap it with the largest value and heapify the affected subtree

if (largest != i) {

int temp = arr[i];

arr[i] = arr[largest];

arr[largest] = temp;

heapify(arr, n, largest);

}

}

// Function to sort an array using heap sort

void heapSort(std::vector<int>& arr)

{

int n = arr.size();

// Build a max heap

for (int i = n / 2 - 1; i >= 0; i--) {

heapify(arr, n, i);

}

// Extract elements from the heap one by one, placing them at the end of the array

for (int i = n - 1; i >= 0; i--) {

// Swap the root node (maximum value) with the last element in the heap

int temp = arr[0];

arr[0] = arr[i];

arr[i] = temp;

// Heapify the remaining elements

heapify(arr, i, 0);

}

}

int main()

{

// Test the heap sort function

std::vector<int> arr = { 4, 2, 1, 3, 6, 5 };

std::cout << "Input : \t" ;

for (int x : arr) {

std::cout << x << " ";

}

std::cout << std::endl;

heapSort(arr);

std::cout << "Output : \t" ;

for (int x : arr) {

std::cout << x << " ";

}

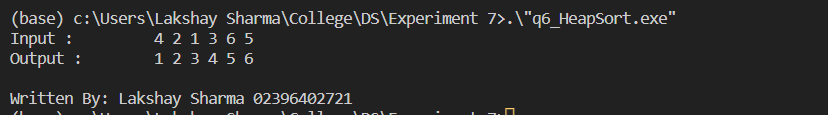
std::cout << std::endl;

std::cout << "\nWritten By: Lakshay Sharma 02396402721";

return 0;

}

**Output ::**

****

# **Experiment 8**

## Program 1

**Aim ::** Write a program to implement searching using hashing method?

**Code ::**

#include <iostream>

#include <vector>

using namespace std;

const int TABLE\_SIZE = 128; // The size of the hash table

// A hash table element, which consists of a key and a value

struct Element {

int key;

int value;

};

// A hash table, which consists of an array of elements and a count of elements

struct HashTable {

vector<Element> elements;

int count;

};

// A hash function that returns a hash value for a given key

int hash\_function(int key) {

return key % TABLE\_SIZE;

}

// A function to create a new hash table

HashTable\* create\_hash\_table() {

HashTable\* table = new HashTable;

table->count = 0;

table->elements.resize(TABLE\_SIZE);

return table;

}

// A function to insert a new element into a hash table

void insert(HashTable\* table, int key, int value) {

Element e;

e.key = key;

e.value = value;

// Find the index at which to insert the element

int index = hash\_function(key);

while (table->elements[index].key != 0) {

index = (index + 1) % TABLE\_SIZE;

}

// Insert the element

table->elements[index] = e;

table->count++;

}

// A function to search for an element with a given key in a hash table

int search(HashTable\* table, int key) {

// Find the index at which the element with the given key should be located

int index = hash\_function(key);

while (table->elements[index].key != key) {

index = (index + 1) % TABLE\_SIZE;

if (table->elements[index].key == 0) {

// The element was not found

return -1;

}

}

// The element was found

return table->elements[index].value;

}

int main() {

HashTable\* table = create\_hash\_table();

insert(table, 1, 10);

insert(table, 2, 20);

insert(table, 3, 30);

insert(table, 4, 40);

insert(table, 5, 50);

cout << "Searching at (3) : " << search(table, 3) << endl; // 30

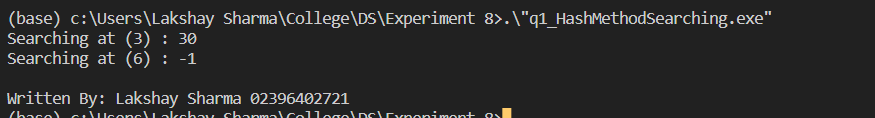
cout << "Searching at (6) : "<< search(table, 6) << endl; // -1 (not found)

std::cout << "\nWritten By: Lakshay Sharma 02396402721";

return 0;

}

**Output ::**

****

## Program 2

**Aim ::** Write a program to create a graph and perform DFS and BFS?

**Code ::**

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

// A graph node, which consists of an id and a list of neighboring nodes

struct Node {

int id;

vector<Node\*> neighbors;

};

// A graph, which consists of a list of nodes

struct Graph {

vector<Node\*> nodes;

};

// A function to create a new graph

Graph\* create\_graph() {

Graph\* g = new Graph;

return g;

}

// A function to create a new node

Node\* create\_node(int id) {

Node\* n = new Node;

n->id = id;

return n;

}

// A function to add an edge between two nodes

void add\_edge(Node\* a, Node\* b) {

a->neighbors.push\_back(b);

b->neighbors.push\_back(a);

}

// A function to perform DFS on a graph

void dfs(Graph\* g, Node\* n, vector<bool>& visited) {

// Mark the current node as visited

visited[n->id] = true;

// Print the node's id

cout << n->id << " ";

// Recursively visit the unvisited neighbors

for (Node\* neighbor : n->neighbors) {

if (!visited[neighbor->id]) {

dfs(g, neighbor, visited);

}

}

}

// A function to perform BFS on a graph

void bfs(Graph\* g, Node\* n) {

queue<Node\*> q;

vector<bool> visited(g->nodes.size(), false);

// Mark the current node as visited and enqueue it

visited[n->id] = true;

q.push(n);

while (!q.empty()) {

// Dequeue a node and print its id

Node\* node = q.front();

q.pop();

cout << node->id << " ";

// Enqueue the unvisited neighbors

for (Node\* neighbor : node->neighbors) {

if (!visited[neighbor->id]) {

visited[neighbor->id] = true;

q.push(neighbor);

}

}

}

}

int main() {

Graph\* g = create\_graph();

// Create some nodes

Node\* a = create\_node(0);

Node\* b = create\_node(1);

Node\* c = create\_node(2);

Node\* d = create\_node(3);

Node\* e = create\_node(4);

Node\* f = create\_node(5);

// Add the nodes to the graph

g->nodes.push\_back(a);

g->nodes.push\_back(b);

g->nodes.push\_back(c);

g->nodes.push\_back(d);

g->nodes.push\_back(e);

g->nodes.push\_back(f);

// Add some edges

add\_edge(a, b);

add\_edge(a, c);

add\_edge(b, d);

add\_edge(c, d);

add\_edge(d, e);

add\_edge(e, f);

// Perform DFS on the graph, starting from node 0

cout << "DFS: ";

vector<bool> visited(g->nodes.size(), false);

dfs(g, a, visited);

cout << endl;

// Perform BFS on the graph, starting from node 0

cout << "BFS: ";

bfs(g, a);

cout << endl;

cout << "\nWritten By: Lakshay Sharma 02396402721";

return 0;

}

**Output ::**

/\*

b

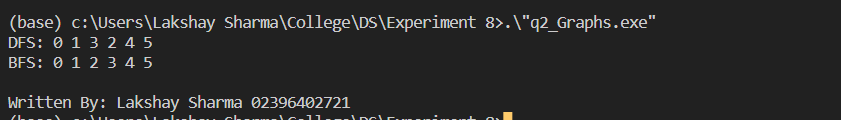
/ \

a d -- e --- f

\ /

c

\*/

****