

**Data structure practicals**

Bsc (hons.) Computer science

Practical List

1. Write a program to implement singly linked list as an ADT that supports the following operations:

i. Insert an element x at the beginning of the singly linked list

ii. Insert an element x at ith position in the singly linked list

iii. Remove an element from the beginning of the singly linked list

iv. Remove an element from ith position in the singly linked list.

vi. Search for an element x in the singly linked list and return its pointer

2. Write a program to implement doubly linked list as an ADT that supports the following operations:

i. Insert an element x at the beginning of the doubly linked list

ii. Insert an element x at the end of the doubly linked list

iii. Remove an element from the beginning of the doubly linked list

iv. Remove an element from the end of the doubly linked list

3. Write a program to implement circular linked list as an ADT which supports the following

operations:

i. Insert an element x in the list

ii. Remove an element from the list

iii. Search for an element x in the list and return its pointer

4. Implement Stack as an ADT and use it to evaluate a prefix/postfix expression.

5. Implement Queue as an ADT.

6. Write a program to implement Binary Search Tree as an ADT which supports the following

operations:

i. Insert an element x

ii. Delete an element x

iii. Search for an element x in the BST

iv. Display the elements of the BST in preorder, inorder, and postorder traversal

7. Write a program to implement insert and search operation in AVL trees.

1.

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

Node(int val) {

data = val;

next = NULL;

}

};

class SinglyLinkedList {

private:

Node\* head;

public:

SinglyLinkedList() {

head = NULL;

}

void insertAtBeginning(int x) {

Node\* newNode = new Node(x);

newNode->next = head;

head = newNode;

}

void printList() {

Node\* current = head;

while (current != NULL) {

cout << current->data << " ";

current = current->next;

}

cout << endl;

}

void insertAtIndex(int x, int i) {

Node\* newNode = new Node(x);

if (i == 0) {

newNode->next = head;

head = newNode;

return;

}

Node\* current = head;

int position = 0;

while (current != NULL && position < i - 1) {

current = current->next;

position++;

}

if (current == NULL) {

cout << "Index out of range. Element not inserted." << endl;

delete newNode;

return;

}

newNode->next = current->next;

current->next = newNode;

}

void removeFromIndex(int i) {

if (head == NULL) {

cout << "List is empty. Nothing to remove." << endl;

return;

}

if (i == 0) {

Node\* temp = head;

head = head->next;

delete temp;

return;

}

Node\* current = head;

int position = 0;

while (current != NULL && position < i - 1) {

current = current->next;

position++;

}

if (current == NULL || current->next == NULL) {

cout << "Index out of range. Element not removed." << endl;

return;

}

Node\* temp = current->next;

current->next = current->next->next;

delete temp;

}

Node\* search(int x) {

Node\* current = head;

while (current != NULL) {

if (current->data == x) {

return current;

}

current = current->next;

}

return NULL;

}

};

int main() {

SinglyLinkedList myList;

int choice;

int value, index;

Node\* searchResult;

while (true) {

cout << "Menu:\n";

cout << "1. Insert at the beginning\n";

cout << "2. Insert at a specific index\n";

cout << "3. Remove from a specific index\n";

cout << "4. Search for an element\n";

cout << "5. Print the list\n";

cout << "6. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter the value to insert at the beginning: ";

cin >> value;

myList.insertAtBeginning(value);

break;

case 2:

cout << "Enter the value to insert: ";

cin >> value;

cout << "Enter the index to insert at: ";

cin >> index;

myList.insertAtIndex(value, index);

break;

case 3:

cout << "Enter the index to remove: ";

cin >> index;

myList.removeFromIndex(index);

break;

case 4:

cout << "Enter the value to search for: ";

cin >> value;

searchResult = myList.search(value);

if (searchResult != NULL) {

cout << "Element found: " << searchResult->data << endl;

} else {

cout << "Element not found." << endl;

}

break;

case 5:

myList.printList();

break;

case 6:

cout << "Exiting the program." << endl;

return 0;

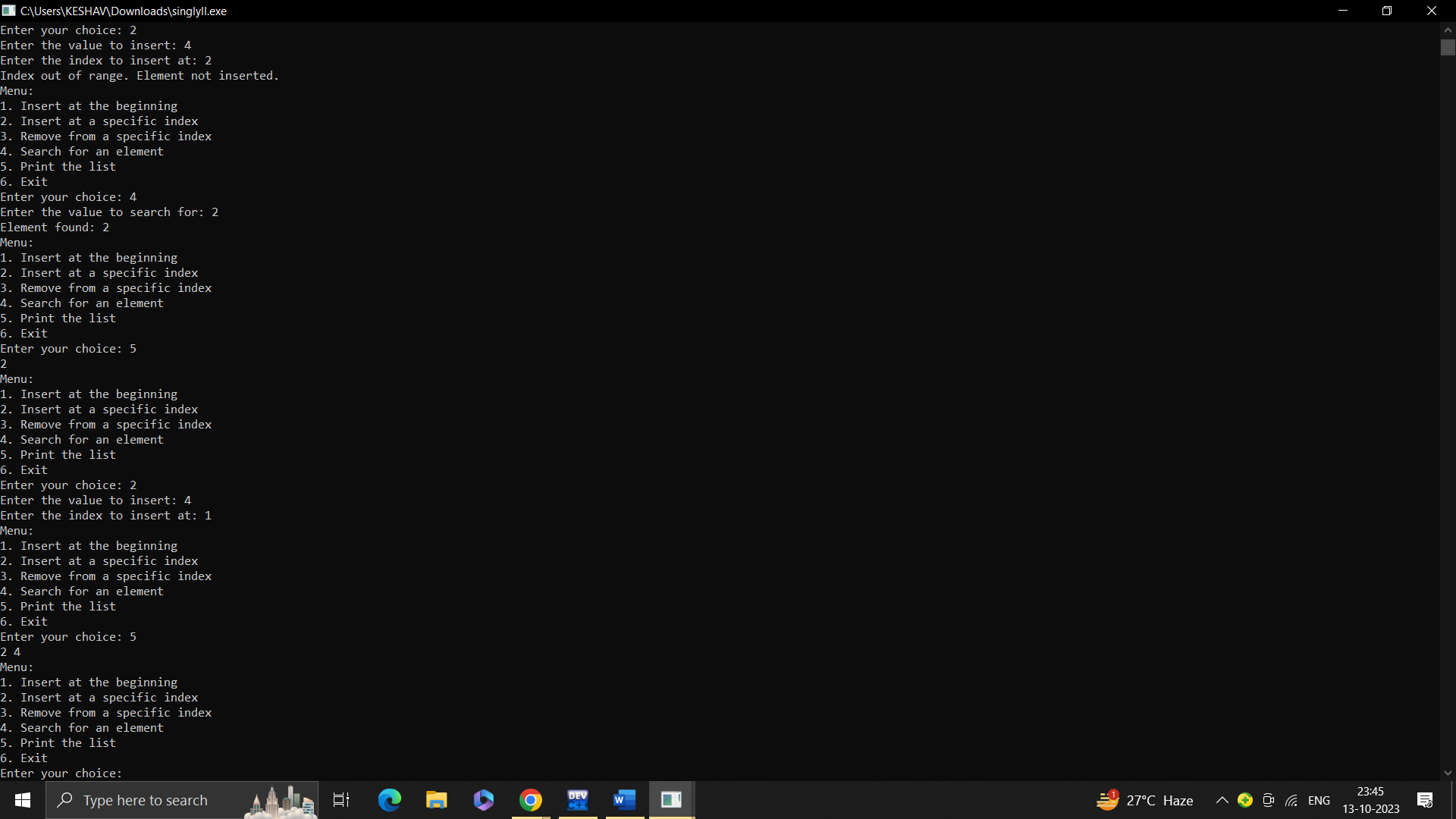
default:

cout << "Invalid choice. Please try again." << endl;

}

}

}



2.

#include<iostream>

using namespace std;

class node{

public:

int data;

node\*next,\*prev;

};

class dlinkedlist{

node\*head,\*tail;

public:

dlinkedlist(){

head=tail=NULL;

}

void addhead(int val){

if(head==NULL){

head=new node;

head->data=val;

head->next=NULL;

head->prev=NULL;

tail=head;

}

else{

node \*p=new node;

p->data=val;

p->next=head;

p->prev=NULL;

head->prev=p;

head=p;

}

}

void addtail(int val){

if(tail==NULL){

tail=new node;

tail->data=val;

tail->next=NULL;

tail->prev=NULL;

head=tail;

}

else{

node\*q=new node;

q->data=val;

q->prev=tail;

q->next=NULL;

tail->next=q;

tail=q;

}

}

void traverse(){

node\*p;

p=head;

while(p!=NULL){

cout<<p->data<<endl;

p=p->next;

}

}

void deletehead(){

if(head==NULL){

cout<<"list is empty";

}

else{

node\*p=head;

head=p->next;

head->prev=NULL;

delete p;

}

}

void deletetail(){

if(tail==NULL){

cout<<"list is empty";

}

else if(head==tail){

delete tail;

}

else{

node\*q;

q=tail->prev;

delete tail;

tail=q;

tail->next=NULL;

}

}

};

void showchoices(){

cout<<"menu"<<endl;

cout<<"1.addhead"<<endl;

cout<<"2.addtail"<<endl;

cout<<"3.display linked list"<<endl;

cout<<"4.delete head"<<endl;

cout<<"5.delete tail"<<endl;

cout<<"6.exit"<<endl;

cout<<"enter your choice:";

}

int main(){

int choice,x,y;

dlinkedlist l1;

do{

showchoices();

cin>>choice;

switch(choice){

case 1:

cout<<"enter element";

cin>>x;

l1.addhead(x);

break;

case 2:

cout<<"enter element";

cin>>x;

l1.addtail(x);

break;

case 3:

l1.traverse();

break;

case 4:

l1.deletehead();

break;

case 5:

l1.deletetail();

break;

case 6:

break;

default:

cout<<"invalid input"<<endl;

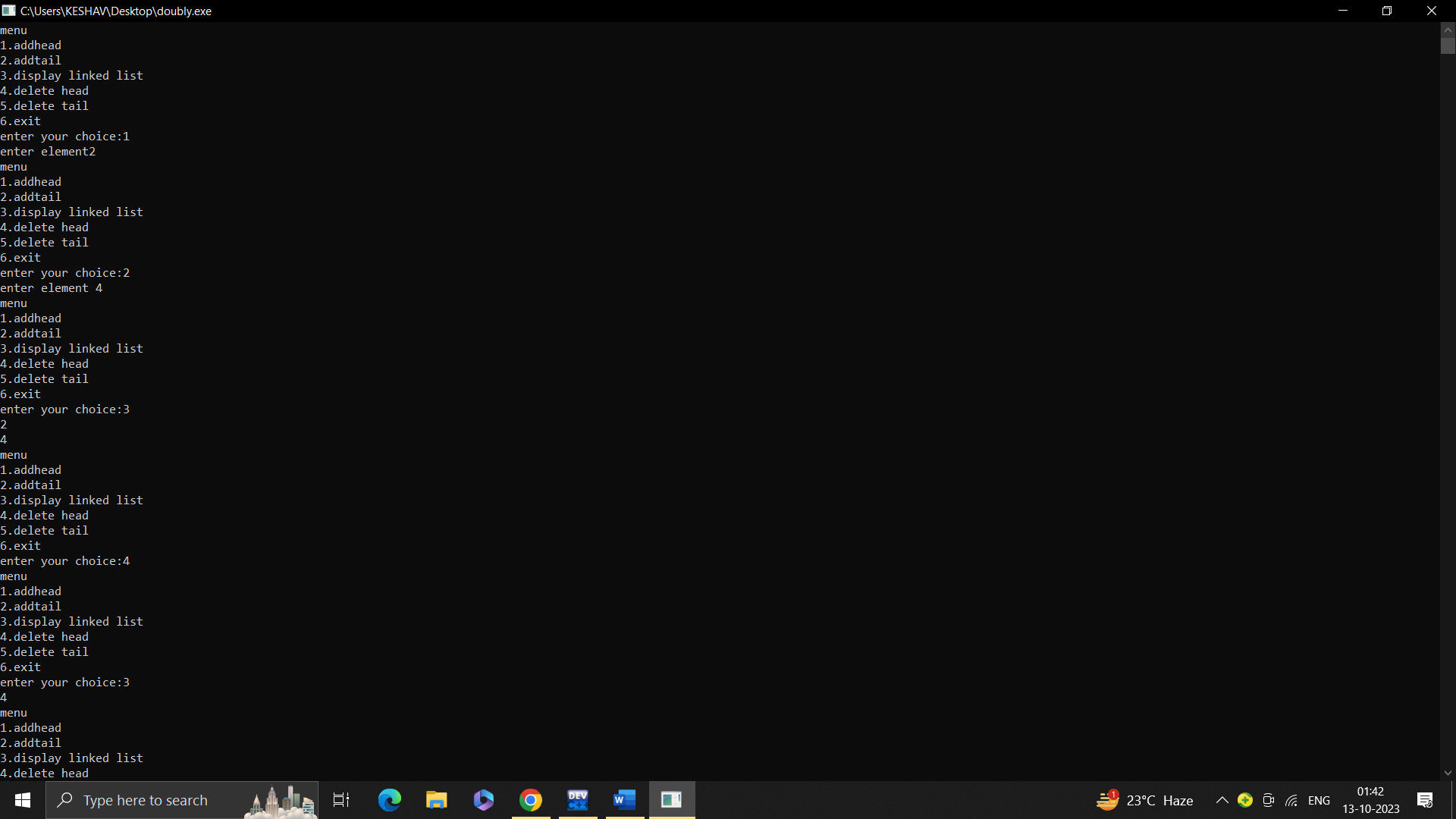
}

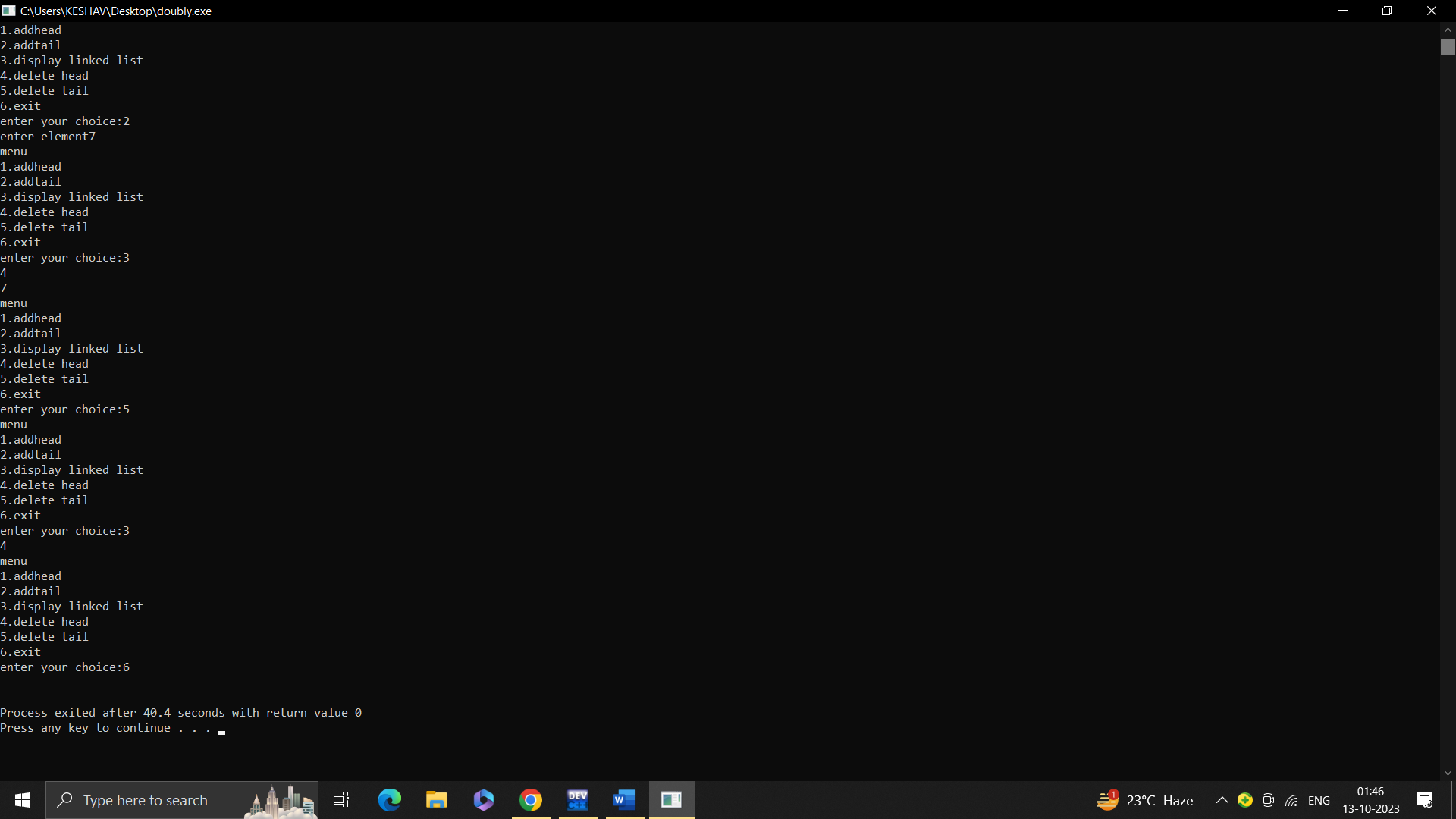
}

while(choice!=6);

return 0;

}





3.

#include <iostream>

#include <cstdlib>

using namespace std;

class node {

public:

node\* prev;

int data;

node\* next;

};

class cirll {

node\* tail;

public:

cirll() {

tail = NULL;

}

void insertafterpos(int data, int pos);

void deletenextnode(int pos);

int search(int val);

void addtoEmpty(int data);

void printList();

};

void cirll::addtoEmpty(int data) {

node\* temp = new node;

temp->prev = temp;

temp->data = data;

temp->next = temp;

tail = temp;

}

void cirll::insertafterpos(int data, int pos) {

if (tail == NULL) {

addtoEmpty(data);

return;

}

node\* newP = new node;

newP->data = data;

node\* temp = tail->next;

while (pos > 1) {

temp = temp->next;

pos--;

}

newP->prev = temp;

newP->next = temp->next;

temp->next->prev = newP;

temp->next = newP;

if (temp == tail) {

tail = newP;

}

}

void cirll::deletenextnode(int pos) {

if (tail == NULL || tail->next == tail) {

return;

}

node\* temp = tail->next;

while (pos > 1) {

temp = temp->next;

pos--;

}

node\* temp2 = temp->prev;

temp2->next = temp->next;

temp->next->prev = temp2;

delete temp;

if (temp == tail) {

tail = temp2;

}

}

int cirll::search(int val) {

node\* temp;

int index = 0;

if (tail == NULL) {

return -2; // -2: List is empty

}

temp = tail->next;

do {

if (temp->data == val) {

return index + 1; //inc by 1 so it starts with 1 not 0

}

temp = temp->next;

index++;

} while (temp != tail->next);

return -1; // -1: Element not found

}

void cirll::printList() {

if (tail == NULL) {

cout << "Linked list is empty" << endl;

return;

}

node\* temp = tail->next;

do {

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

temp = temp->next;

} while (temp != tail->next);

cout << endl;

}

int main() {

cirll myList;

int choice;

int data, pos;

int searchResult;

while (true) {

cout << "Menu:\n";

cout << "1. Insert after position\n";

cout << "2. Delete at position\n";

cout << "3. Search\n";

cout << "4. Print the list\n";

cout << "5. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter the value to insert: ";

cin >> data;

cout << "Enter the position to insert after: ";

cin >> pos;

myList.insertafterpos(data, pos);

break;

case 2:

cout << "Enter the position to delete: ";

cin >> pos;

myList.deletenextnode(pos);

break;

case 3:

cout << "Enter the value to search for: ";

cin >> data;

searchResult = myList.search(data);

if (searchResult == -2) {

cout << "Element not found: List is empty" << endl;

} else if (searchResult == -1) {

cout << "Element not found" << endl;

} else {

cout << "Element found at position " << searchResult << endl;

}

break;

case 4:

cout << "List: ";

myList.printList();

break;

case 5:

cout << "Exiting the program." << endl;

return 0;

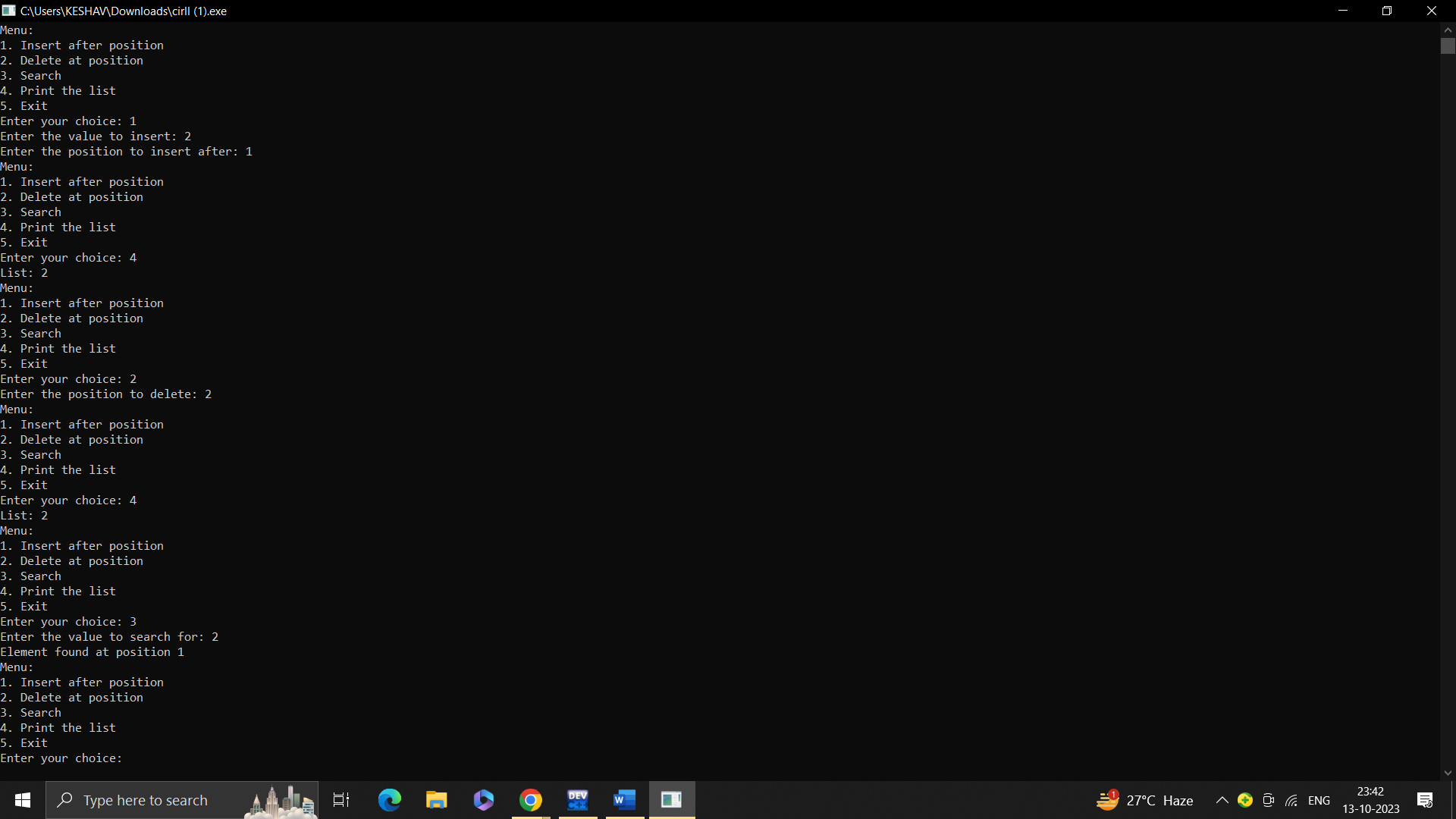
default:

cout << "Invalid choice. Please try again." << endl;

}

}

}



4.

#include <iostream>

#include <string>

using namespace std;

class Stack {

private:

int top;

int\* array;

int maxSize;

public:

Stack(int size) {

top = -1;

maxSize = size;

array = new int[size];

}

~Stack() {

delete[] array;

}

void push(int item) {

if (top == maxSize - 1) {

cout << "Stack Overflow" << endl;

return;

}

array[++top] = item;

}

int pop() {

if (top == -1) {

cout << "Stack Underflow" << endl;

return -1;

}

return array[top--];

}

bool isEmpty() {

return (top == -1);

}

};

int evaluateExpression(string exp, bool isPostfix) {

Stack stack(exp.length());

for (int i = 0; i < exp.length(); i++) {

if (isdigit(exp[i])) {

stack.push(exp[i] - '0');

} else {

int val1, val2;

if (isPostfix) {

val2 = stack.pop();

val1 = stack.pop();

} else {

val1 = stack.pop();

val2 = stack.pop();

}

switch (exp[i]) {

case '+':

stack.push(val1 + val2);

break;

case '-':

stack.push(val1 - val2);

break;

case '\*':

stack.push(val1 \* val2);

break;

case '/':

stack.push(val1 / val2);

break;

}

}

}

return stack.pop();

}

int main() {

int choice;

string exp;

bool isPostfix;

while (true) {

cout << "Menu:" << endl;

cout << "1. Evaluate postfix expression" << endl;

cout << "2. Evaluate prefix expression" << endl;

cout << "3. Exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter postfix expression: ";

cin >> exp;

isPostfix = true;

cout << "Result: " << evaluateExpression(exp, isPostfix) << endl;

break;

case 2:

cout << "Enter prefix expression: ";

cin >> exp;

isPostfix = false;

cout << "Result: " << evaluateExpression(exp, isPostfix) << endl;

break;

case 3:

cout << "Exiting program." << endl;

return 0;

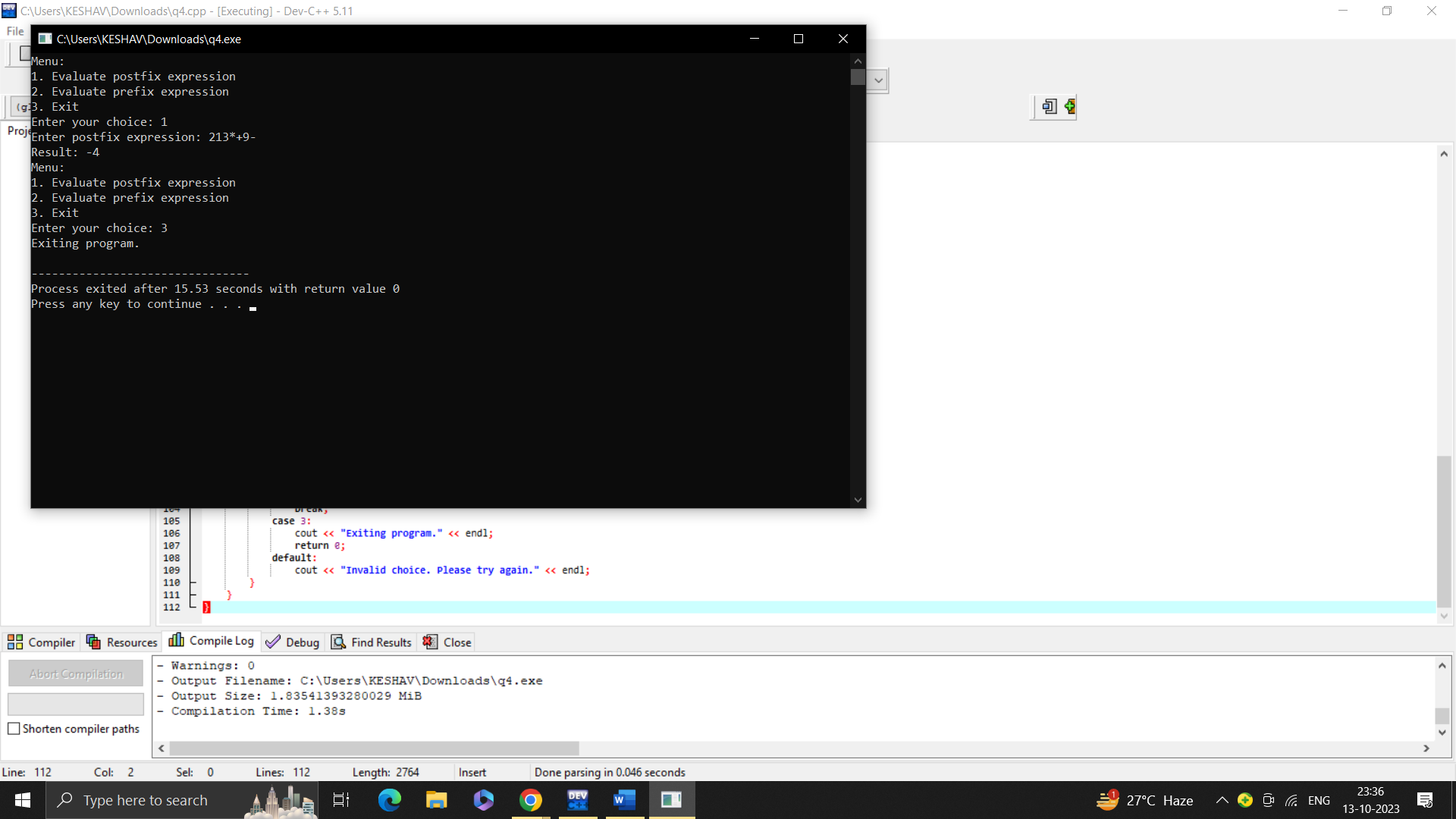
default:

cout << "Invalid choice. Please try again." << endl;

}

}

}



5.

#include<iostream>

#define max 10

using namespace std;

template<class T>

class queue{

int q[max];

int f,r,n;

public:

queue(){

f=r=-1;

n=0;

}

void enqueue(T val){

if(n==max)

cout<<"queue is full";

else{

r=(r+1)%max;

q[r]=val;

n=n+1;

if(n==1){

f=r;

}

}

}

void dequeue(){

if(n==0)

cout<<"queue is empty";

else{

cout<<q[f];

f=(f+1)%max;

n=n-1;

}

}

void display(){

if(n==0)

cout<<"queue is empty";

else{

cout<<"queue elements are:";

for(int i=0;i<=r;i++){

cout<<q[i]<<" ";

}

}

}

};

int main(){

int x;char y;

queue<int>q1;

queue<char>q2;

int ch;

cout<<"1. insert element in queue as a number"<<endl;

cout<<"2. insert element in queue as a char"<<endl;

cout<<"3. deletes element from queue (number)"<<endl;

cout<<"4. deletes element from queue (char)"<<endl;

cout<<"5. display element of queue (number)"<<endl;

cout<<"6. display element of queue(char)"<<endl;

cout<<"7.exit"<<endl;

do{

cout<<endl<<"enter your choice: ";

cin>>ch;

cout<<endl;

switch(ch){

case 1:

cout<<"enter no. of queue: ";

cin>>x;

q1.enqueue(x);

break;

case 2:

cout<<"enter character for queue: ";

cin>>y;

q2.enqueue(y);

break;

case 3:

q1.dequeue();

break;

case 4:

q2.dequeue();

break;

case 5:

q1.display();

break;

case 6:

q2.display();

break;

case 7:

break;

default:

cout<<"invalid choice";

break;

}

}

while(ch!=7);

return 0;

}

6.

#include<iostream>

using namespace std;

class Node {

public:

int data;

Node\* left;

Node\* right;

Node(int val) {

data = val;

left = NULL;

right = NULL;

}

};

class BST {

Node\* root;

public:

BST() {

root = NULL;

}

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

if (root == NULL) {

return new Node(x);

}

if (x < root->data) {

root->left = insert(root->left, x);

} else if (x > root->data) {

root->right = insert(root->right, x);

}

return root;

}

void insert(int x) {

root = insert(root, x);

}

Node\* findMin(Node\* root) {

while (root->left != NULL) {

root = root->left;

}

return root;

}

Node\* deleteNode(Node\* root, int x) {

if (root == NULL) {

return root;

}

if (x < root->data) {

root->left = deleteNode(root->left, x);

} else if (x > root->data) {

root->right = deleteNode(root->right, x);

} else {

if (root->left == NULL) {

Node\* temp = root->right;

delete root;

return temp;

} else if (root->right == NULL) {

Node\* temp = root->left;

delete root;

return temp;

}

Node\* temp = findMin(root->right);

root->data = temp->data;

root->right = deleteNode(root->right, temp->data);

}

return root;

}

void deleteNode(int x) {

root = deleteNode(root, x);

}

bool search(Node\* root, int x) {

if (root == NULL) {

return false;

}

if (root->data == x) {

return true;

} else if (x < root->data) {

return search(root->left, x);

} else {

return search(root->right, x);

}

}

bool search(int x) {

return search(root, x);

}

void preorder(Node\* root) {

if (root == NULL) return;

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

preorder(root->left);

preorder(root->right);

}

void inorder(Node\* root) {

if (root == NULL) return;

inorder(root->left);

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

inorder(root->right);

}

void postorder(Node\* root) {

if (root == NULL) return;

postorder(root->left);

postorder(root->right);

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

}

void preorder() {

preorder(root);

cout << endl;

}

void inorder() {

inorder(root);

cout << endl;

}

void postorder() {

postorder(root);

cout << endl;

}

};

int main() {

BST t;

int ch, val;

do {

cout << "Binary Search Tree Menu:" << endl;

cout << "1. Insert element x" << endl;

cout << "2. Delete element x" << endl;

cout << "3. Search element x" << endl;

cout << "4. Pre-order Traversal" << endl;

cout << "5. In-order Traversal" << endl;

cout << "6. Post-order Traversal" << endl;

cout << "7. Exit" << endl;

cout << "Enter your choice: ";

cin >> ch;

switch (ch) {

case 1:

cout << "Enter the value to insert: ";

cin >> val;

t.insert(val);

break;

case 2:

cout << "Enter the value to delete: ";

cin >> val;

t.deleteNode(val);

break;

case 3:

cout << "Enter the value to search: ";

cin >> val;

if (t.search(val)) {

cout << val << " is found in the tree." << endl;

} else {

cout << val << " is not found in the tree." << endl;

}

break;

case 4:

cout << "Pre-order Traversal: ";

t.preorder();

break;

case 5:

cout << "In-order Traversal: ";

t.inorder();

break;

case 6:

cout << "Post-order Traversal: ";

t.postorder();

break;

case 7:

cout << "Exiting program." << endl;

break;

default:

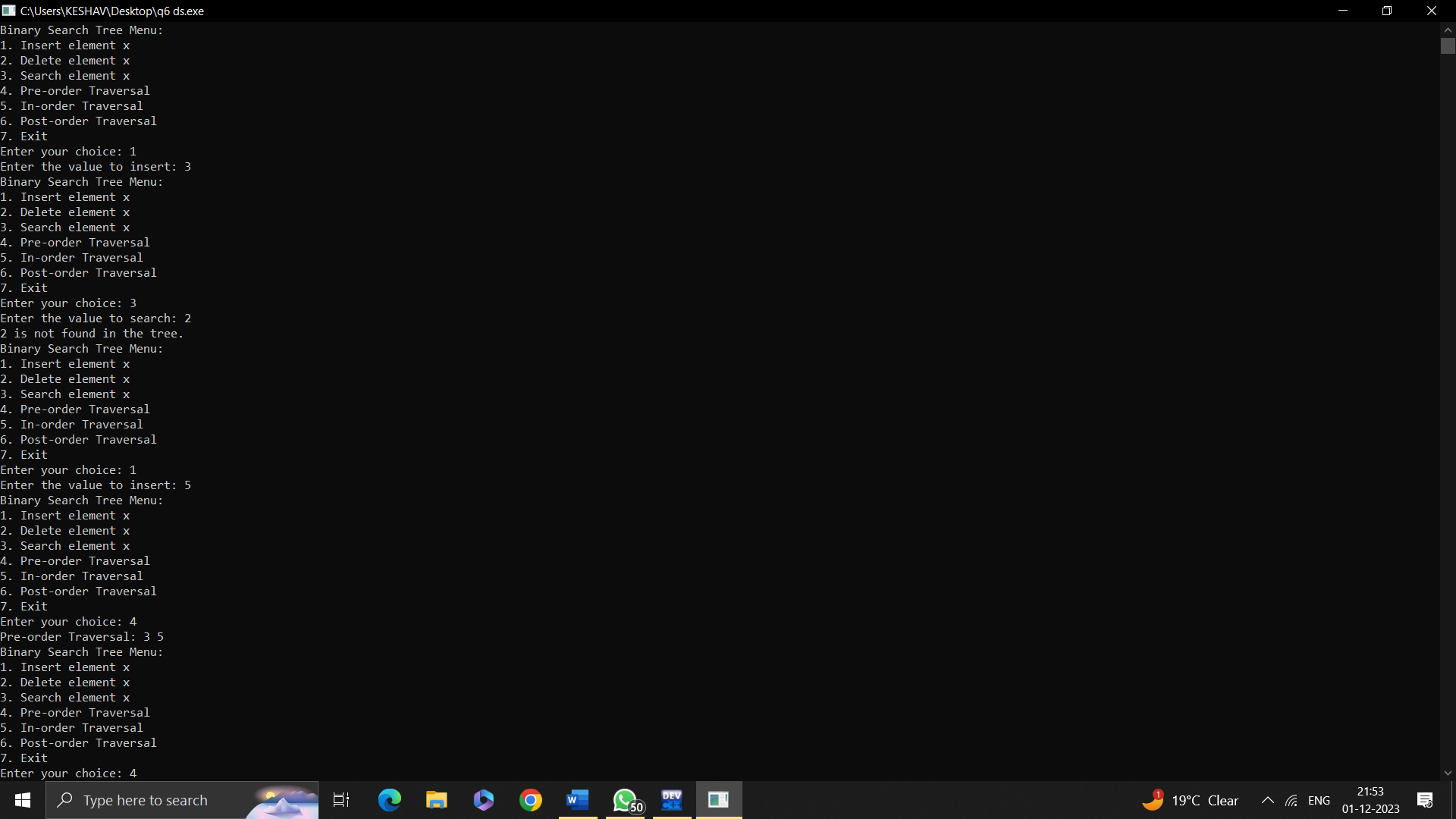
cout << "Invalid choice. Please try again." << endl;

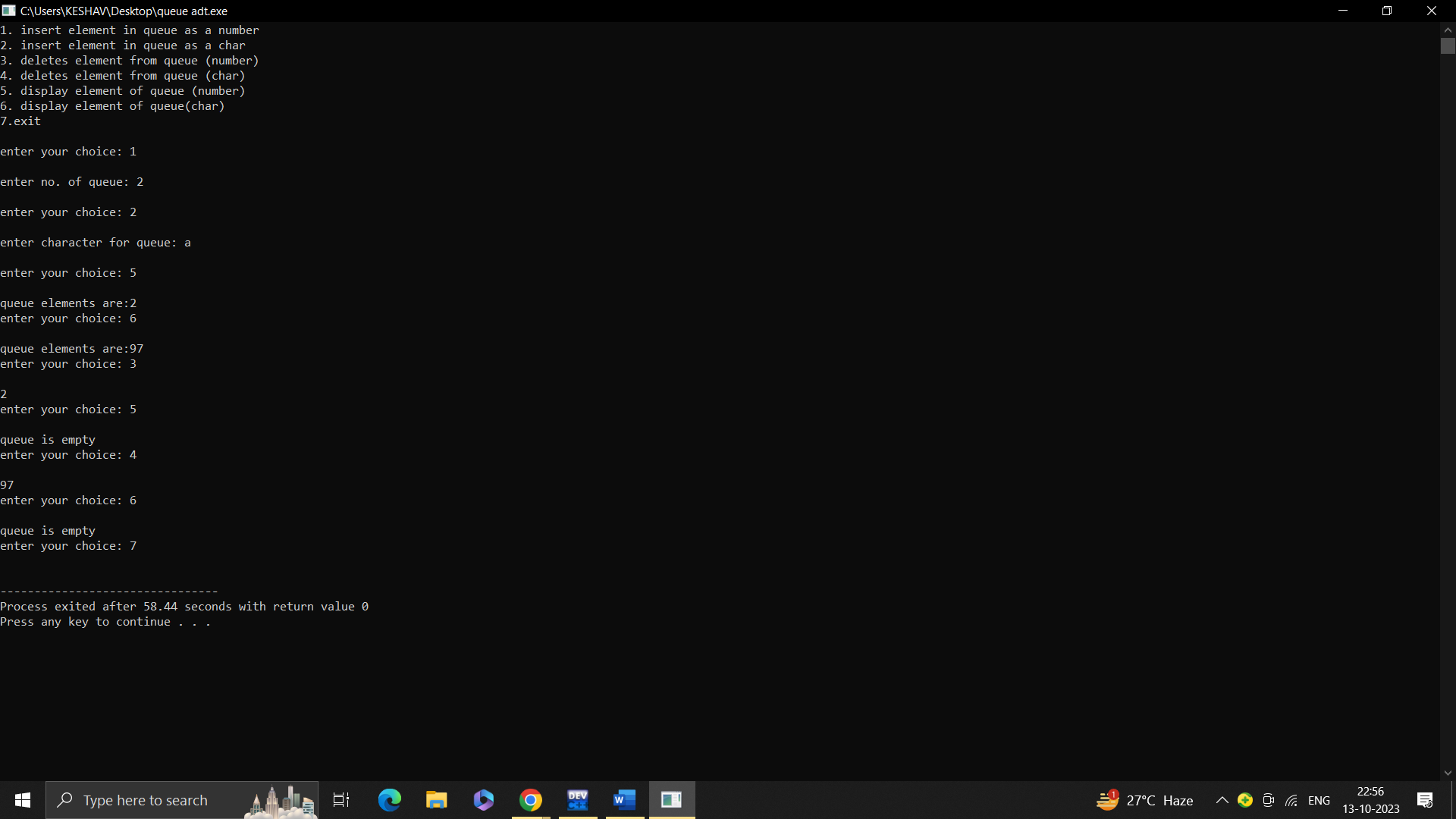
}

} while (ch != 7);

return 0;

}





7.

#include <iostream>

using namespace std;

class Node {

public:

int key;

int height;

Node\* left;

Node\* right;

Node(int k) : key(k), height(1), left(NULL), right(NULL) {}

};

class AVLTree {

private:

Node\* root;

int height(Node\* node) {

return (node == NULL) ? 0 : node->height;

}

int max(int a, int b) {

return (a > b) ? a : b;

}

int getBalance(Node\* node) {

return (node == NULL) ? 0 : height(node->left) - height(node->right);

}

Node\* rotateRight(Node\* y) {

Node\* x = y->left;

Node\* T = x->right;

x->right = y;

y->left = T;

y->height = max(height(y->left), height(y->right)) + 1;

x->height = max(height(x->left), height(x->right)) + 1;

return x;

}

Node\* rotateLeft(Node\* x) {

Node\* y = x->right;

Node\* T = y->left;

y->left = x;

x->right = T;

x->height = max(height(x->left), height(x->right)) + 1;

y->height = max(height(y->left), height(y->right)) + 1;

return y;

}

Node\* insertRec(Node\* node, int key) {

if (node == NULL)

return new Node(key);

if (key < node->key)

node->left = insertRec(node->left, key);

else if (key > node->key)

node->right = insertRec(node->right, key);

else

return node;

node->height = 1 + max(height(node->left), height(node->right));

int balance = getBalance(node);

if (balance > 1 && key < node->left->key)

return rotateRight(node);

if (balance < -1 && key > node->right->key)

return rotateLeft(node);

if (balance > 1 && key > node->left->key) {

node->left = rotateLeft(node->left);

return rotateRight(node);

}

if (balance < -1 && key < node->right->key) {

node->right = rotateRight(node->right);

return rotateLeft(node);

}

return node;

}

Node\* minValueNode(Node\* node) {

Node\* current = node;

while (current->left != NULL)

current = current->left;

return current;

}

Node\* searchRec(Node\* root, int key) {

if (root == NULL || root->key == key) {

return root;

}

if (key < root->key) {

return searchRec(root->left, key);

}

return searchRec(root->right, key);

}

Node\* deleteRec(Node\* root, int key) {

if (root == NULL)

return root;

if (key < root->key)

root->left = deleteRec(root->left, key);

else if (key > root->key)

root->right = deleteRec(root->right, key);

else {

if ((root->left == NULL) || (root->right == NULL)) {

Node\* temp = (root->left != NULL) ? root->left : root->right;

if (temp == NULL) {

temp = root;

root = NULL;

}

else

\*root = \*temp;

delete temp;

}

else {

Node\* temp = minValueNode(root->right);

root->key = temp->key;

root->right = deleteRec(root->right, temp->key);

}

}

if (root == NULL)

return root;

root->height = 1 + max(height(root->left), height(root->right));

int balance = getBalance(root);

if (balance > 1 && getBalance(root->left) >= 0)

return rotateRight(root);

if (balance > 1 && getBalance(root->left) < 0) {

root->left = rotateLeft(root->left);

return rotateRight(root);

}

if (balance < -1 && getBalance(root->right) <= 0)

return rotateLeft(root);

if (balance < -1 && getBalance(root->right) > 0) {

root->right = rotateRight(root->right);

return rotateLeft(root);

}

return root;

}

void inorderRec(Node\* root) {

if (root != NULL) {

inorderRec(root->left);

cout << root->key << " ";

inorderRec(root->right);

}

}

void preorderRec(Node\* root) {

if (root != NULL) {

cout << root->key << " ";

preorderRec(root->left);

preorderRec(root->right);

}

}

void postorderRec(Node\* root) {

if (root != NULL) {

postorderRec(root->left);

postorderRec(root->right);

cout << root->key << " ";

}

}

public:

AVLTree() : root(NULL) {}

void insert(int key) {

root = insertRec(root, key);

}

void remove(int key) {

root = deleteRec(root, key);

}

bool search(int key) {

Node\* result = searchRec(root, key);

return (result != NULL);

}

void inorder() {

inorderRec(root);

}

void preorder() {

preorderRec(root);

}

void postorder() {

postorderRec(root);

}

};

int main() {

AVLTree avl;

int choice, key;

while (true) {

cout << "\nAVL Tree Menu:\n";

cout << "1. Insert an element\n";

cout << "2. Search for an element\n";

cout << "3. Remove an element\n";

cout << "4. Inorder Traversal\n";

cout << "5. Preorder Traversal\n";

cout << "6. Postorder Traversal\n";

cout << "7. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter the element to insert: ";

cin >> key;

avl.insert(key);

break;

case 2:

cout << "Enter the element to search: ";

cin >> key;

if (avl.search(key)) {

cout << "Element " << key << " found in the tree.\n";

}

else {

cout << "Element " << key << " not found in the tree.\n";

}

break;

case 3:

cout << "Enter the element to remove: ";

cin >> key;

avl.remove(key);

break;

case 4:

cout << "Inorder Traversal: ";

avl.inorder();

cout << endl;

break;

case 5:

cout << "Preorder Traversal: ";

avl.preorder();

cout << endl;

break;

case 6:

cout << "Postorder Traversal: ";

avl.postorder();

cout << endl;

break;

case 7:

cout << "Exiting the program.\n";

return 0;

default:

cout << "Invalid choice. Please try again.\n";

}

}

}

