**WEEK – 3**

**3. Write a program to perform  
insertion, deletion and searching operations on a skip list.**

**CODE:**

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <cstring>

const int MAX\_LEVEL = 3;

// Node structure for the skip list

struct Node {

int value;

Node\*\* forward;

Node(int level, int &value) {

forward = new Node\*[level + 1];

std::memset(forward, 0, sizeof(Node\*) \* (level + 1));

this->value = value;

}

~Node() {

delete[] forward;

}

};

// SkipList class

class SkipList {

private:

Node\* header;

int level;

public:

SkipList() {

header = new Node(MAX\_LEVEL, \*(new int(-1)));

level = 0;

}

~SkipList() {

delete header;

}

// Generate random level for a node

int randomLevel() {

int lvl = 0;

while (lvl < MAX\_LEVEL && rand() % 2 == 0) {

lvl++;

}

return lvl;

}

// Insert value into the skip list

void insertElement(int &value) {

Node\* current = header;

Node\* update[MAX\_LEVEL + 1];

std::memset(update, 0, sizeof(Node\*) \* (MAX\_LEVEL + 1));

for (int i = level; i >= 0; i--) {

while (current->forward[i] != nullptr && current->forward[i]->value < value) {

current = current->forward[i];

}

update[i] = current;

}

current = current->forward[0];

if (current == nullptr || current->value != value) {

int newLevel = randomLevel();

if (newLevel > level) {

for (int i = level + 1; i <= newLevel; i++) {

update[i] = header;

}

level = newLevel;

}

Node\* newNode = new Node(newLevel, value);

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

newNode->forward[i] = update[i]->forward[i];

update[i]->forward[i] = newNode;

}

std::cout << "Inserted element: " << value << std::endl;

}

}

// Delete element from the skip list

void deleteElement(int &value) {

Node\* current = header;

Node\* update[MAX\_LEVEL + 1];

std::memset(update, 0, sizeof(Node\*) \* (MAX\_LEVEL + 1));

for (int i = level; i >= 0; i--) {

while (current->forward[i] != nullptr && current->forward[i]->value < value) {

current = current->forward[i];

}

update[i] = current;

}

current = current->forward[0];

if (current != nullptr && current->value == value) {

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

if (update[i]->forward[i] != current)

break;

update[i]->forward[i] = current->forward[i];

}

delete current;

while (level > 0 && header->forward[level] == nullptr) {

level--;

}

std::cout << "Deleted element: " << value << std::endl;

}

}

// Search for an element in the skip list

bool searchElement(int &value) {

Node\* current = header;

for (int i = level; i >= 0; i--) {

while (current->forward[i] != nullptr && current->forward[i]->value < value) {

current = current->forward[i];

}

}

current = current->forward[0];

return (current != nullptr && current->value == value);

}

// Display the skip list

void displayList() {

std::cout << "Skip List:" << std::endl;

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

Node\* node = header->forward[i];

std::cout << "Level " << i << ": ";

while (node != nullptr) {

std::cout << node->value << " ";

node = node->forward[i];

}

std::cout << std::endl;

}

}

};

int main() {

// Initialize random seed

srand(time(nullptr));

SkipList skipList;

int choice;

int element;

while (true) {

std::cout << "\nSkip List Menu:\n";

std::cout << "1. Insert element\n";

std::cout << "2. Delete element\n";

std::cout << "3. Search element\n";

std::cout << "4. Display list\n";

std::cout << "5. Exit\n";

std::cout << "Enter your choice: ";

std::cin >> choice;

switch (choice) {

case 1:

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

std::cin >> element;

skipList.insertElement(element);

break;

case 2:

std::cout << "Enter element to delete: ";

std::cin >> element;

skipList.deleteElement(element);

break;

case 3:

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

std::cin >> element;

if (skipList.searchElement(element)) {

std::cout << "Element " << element << " found in the skip list." << std::endl;

} else {

std::cout << "Element " << element << " not found in the skip list." << std::endl;

}

break;

case 4:

skipList.displayList();

break;

case 5:

std::cout << "Exiting program.\n";

return 0;

default:

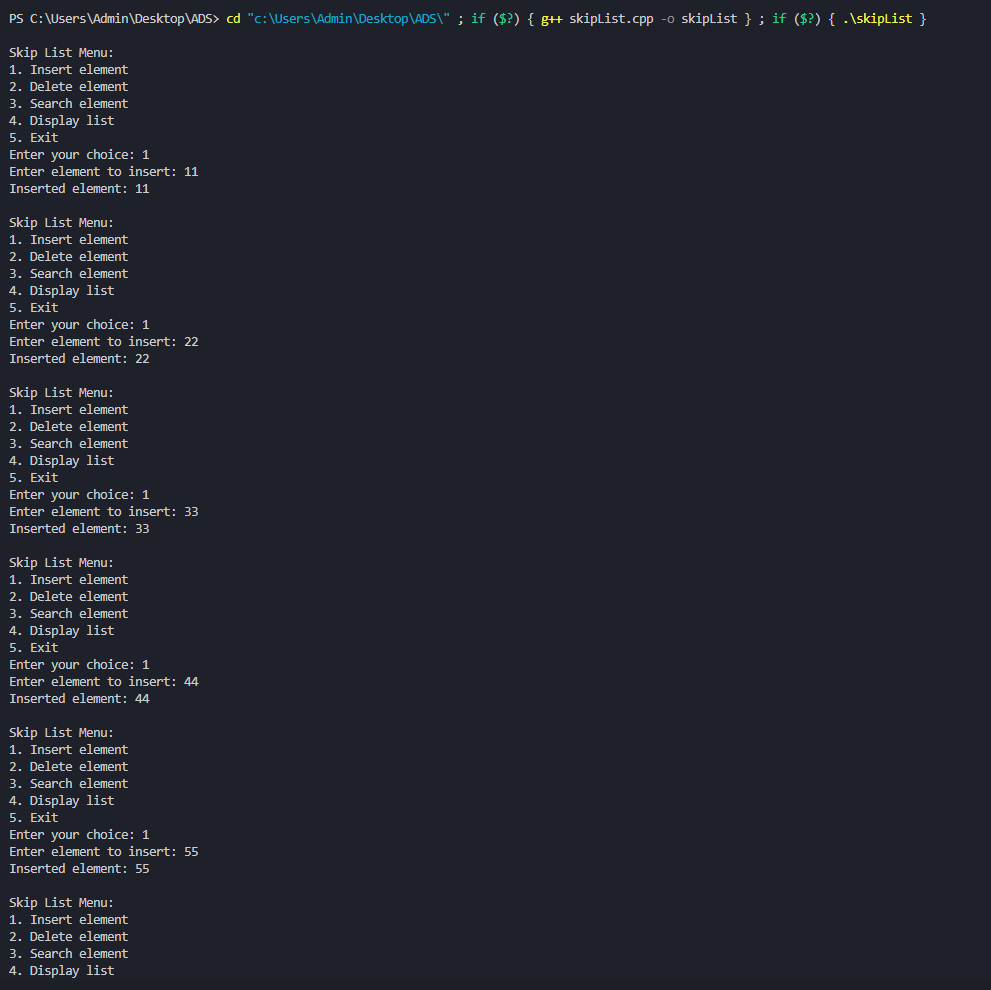
std::cout << "Invalid choice. Please try again.\n";

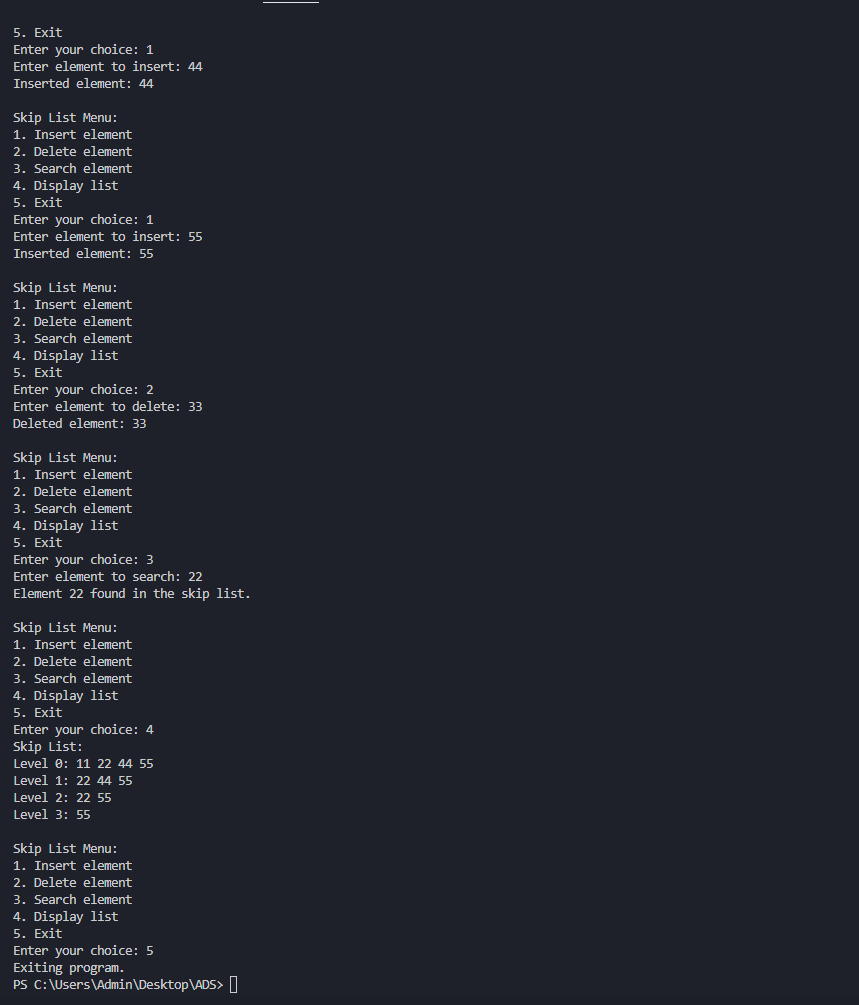
}

}

return 0;

}

**OUTPUT:**

****