Name: Tushar Attri **UID:** 22BCS15290

Section: 22BCS_IOT_620-A **Date:** 23-12-24

DOMAIN WINTER WINNING CAMP-Day(3)

1) Fibonacci Series

Code:

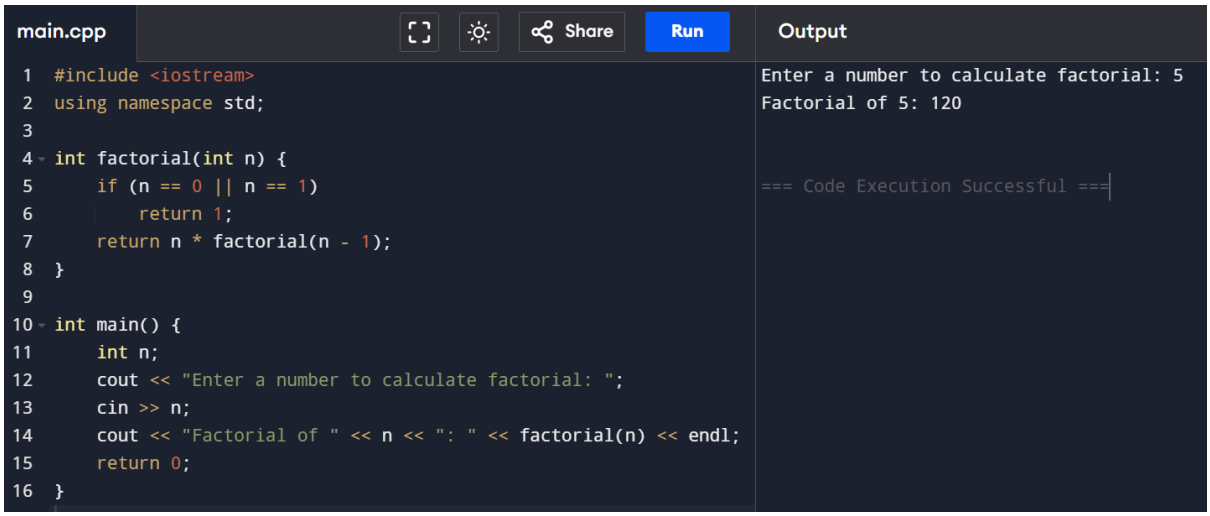
```
#include <iostream>
using namespace std;

int fibonacci(int n) {
   if (n <= 1)
      return n;
   return fibonacci(n - 1) + fibonacci(n - 2);
}

int main() {
   int n;
   cout << "Enter a number for Fibonacci calculation: ";
   cin >> n;
   cout << "Fibonacci(" << n << "): " << fibonacci(n) << endl;
   return 0;
```

Output:

2) Factorial Using Recursion

```
#include <iostream>
using namespace std;

int factorial(int n) {
    if (n == 0 || n == 1)
        return 1;
    return n * factorial(n - 1);
}

int main() {
    int n;
    cout << "Enter a number to calculate factorial: ";
    cin >> n;
    cout << "Factorial of " << n << ": " << factorial(n) << endl;
    return 0;
}
```

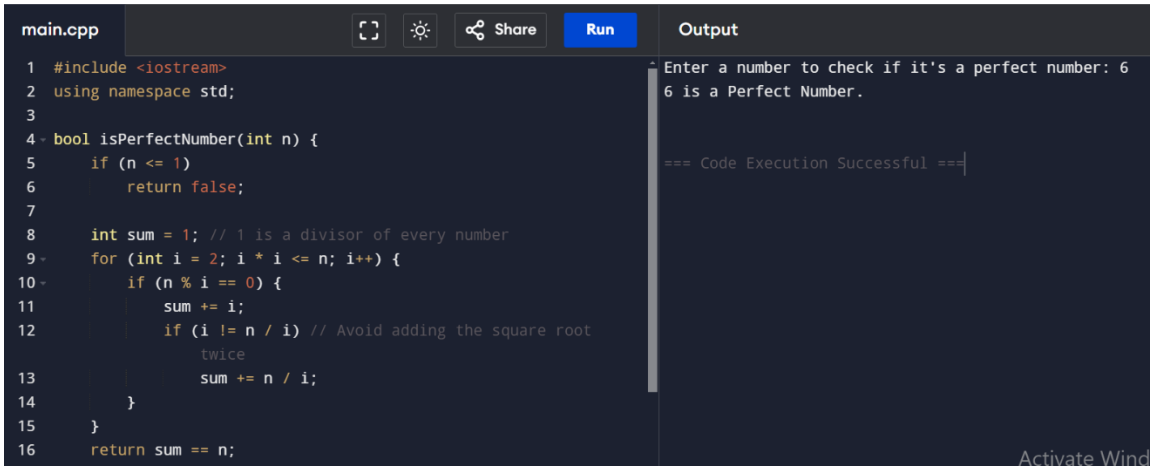
Output:



3) Perfect Number

```
#include <iostream>
using namespace std;

bool isPerfectNumber(int n) {
    if (n <= 1)
        return false;

    int sum = 1; // 1 is a divisor of every number
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            sum += i;
            if (i != n / i) // Avoid adding the square root twice
                sum += n / i;
        }
    }
    return sum == n;
}

int main() {
    int n;
    cout << "Enter a number to check if it's a perfect number: ";
    cin >> n;

    if (isPerfectNumber(n)) {
        cout << n << " is a Perfect Number." << endl;
    } else {
        cout << n << " is not a Perfect Number." << endl;
    }

    return 0;
}
```

Output:

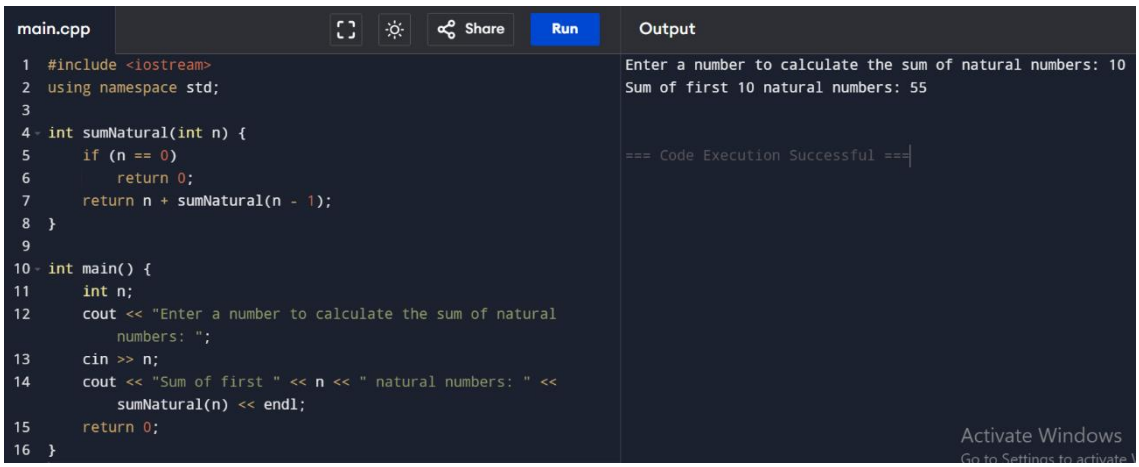


4) Sum of Natural Numbers Using Recursion

```
#include <iostream>
using namespace std;

int sumNatural(int n) {
    if (n == 0)
        return 0;
    return n + sumNatural(n - 1);
}

int main() {
    int n;
    cout << "Enter a number to calculate the sum of natural numbers: ";
    cin >> n;
    cout << "Sum of first " << n << " natural numbers: " << sumNatural(n) << endl;
```

```
    return 0;
}
```

Output:



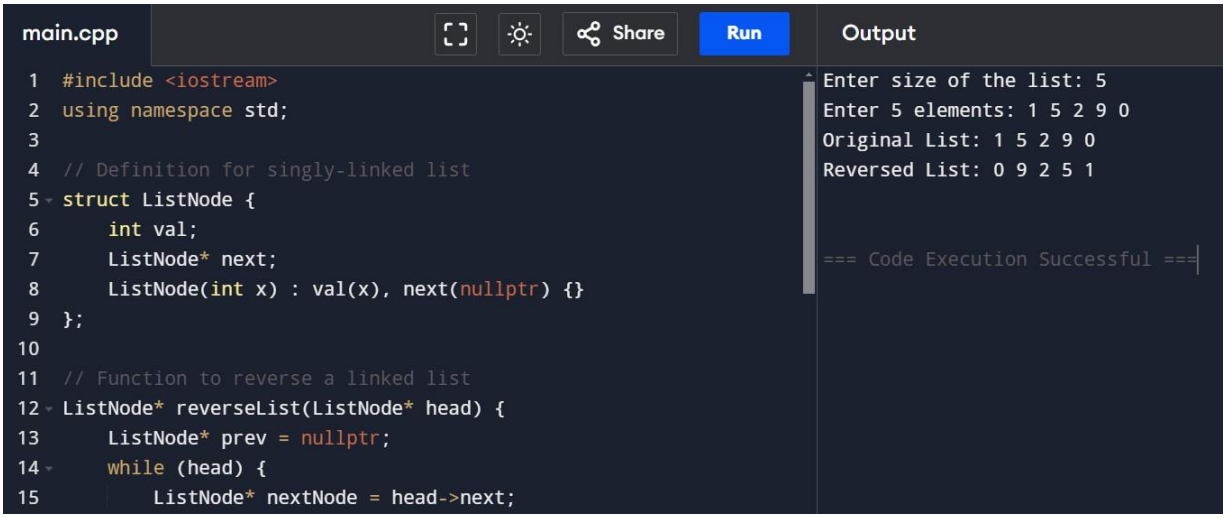
5) Reverse Linked List

```
 #include <iostream>
using namespace std;


    // Definition for singly-linked list
    struct ListNode {
        int val;
        ListNode* next;
        ListNode(int x) : val(x), next(nullptr) {}
    };


    // Function to reverse a linked list
    ListNode* reverseList(ListNode* head) {
        ListNode* prev = nullptr;
```

```
    while (head) {
        ListNode* nextNode = head->next;
        head->next = prev;
        prev = head;
        head = nextNode;
    }
    return prev;
}

// Function to create a linked list from user input
ListNode* createList(int n) {
    ListNode* head = nullptr;
    ListNode* tail = nullptr;
    cout << "Enter " << n << " elements: ";
    for (int i = 0; i < n; i++) {
        int val;
        cin >> val;
        ListNode* newNode = new ListNode(val);
        if (!head) {
            head = tail = newNode;
        } else {
            tail->next = newNode;
            tail = newNode;
        }
    }
    return head;
}

// Function to print a linked list
```

```
void printList(ListNode* head) {
    while (head) {
        cout << head->val << " ";
        head = head->next;
    }
    cout << endl;
}

int main() {
    int n;
    cout << "Enter size of the list: ";
    cin >> n;
    ListNode* head = createList(n);

    cout << "Original List: ";
    printList(head);

    head = reverseList(head);

    cout << "Reversed List: ";
    printList(head);

    return 0;
}
```

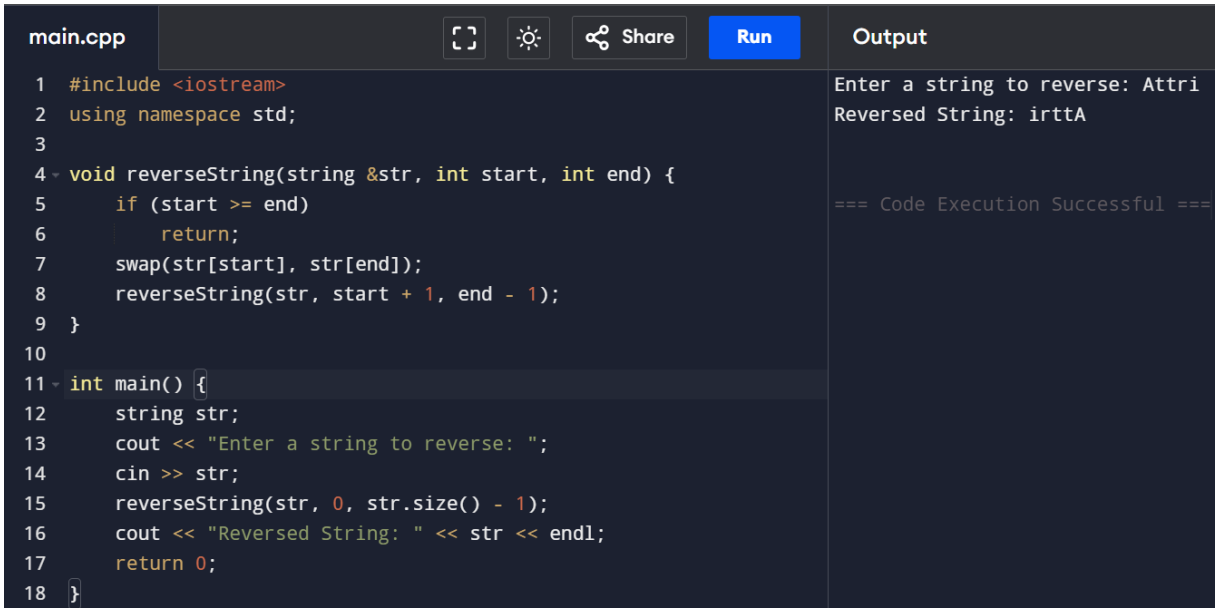
Output:



6) Reverse String

```
#include <iostream>
using namespace std;

void reverseString(string &str, int start, int end) {
   if (start >= end)
      return;
   swap(str[start], str[end]);
   reverseString(str, start + 1, end - 1);
}

int main() {
   string str;
   cout << "Enter a string to reverse: ";
   cin >> str;
   reverseString(str, 0, str.size() - 1);
   cout << "Reversed String: " << str << endl;
   return 0;
}
```

Output :



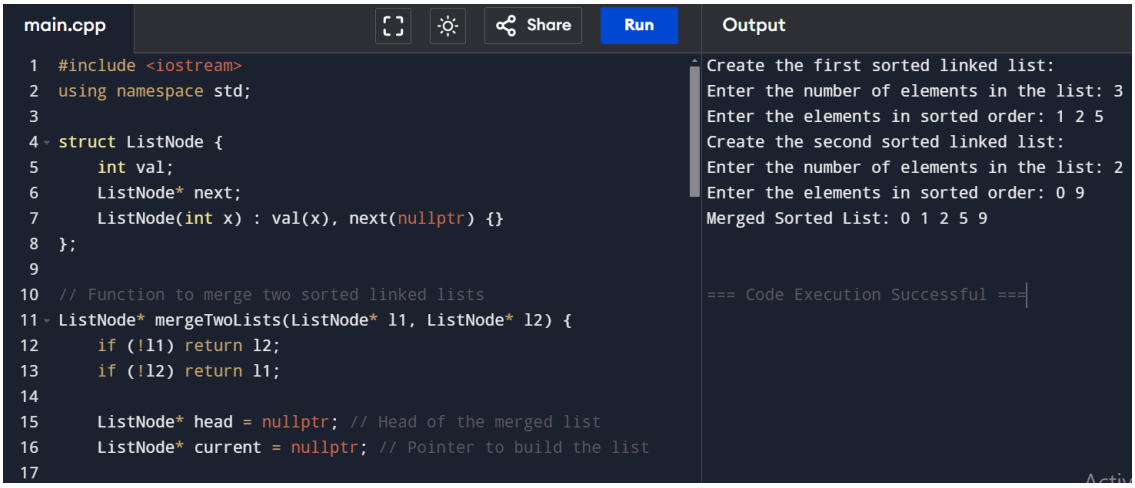
7) Merge Two Sorted Linked Lists

```
#include <iostream>
using namespace std;

struct ListNode {
    int val;
    ListNode* next;
    ListNode(int x) : val(x), next(nullptr) {}
};

// Function to merge two sorted linked lists
ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
    if (!l1) return l2;
    if (!l2) return l1;

    ListNode* head = nullptr; // Head of the merged list
    ListNode* current = nullptr; // Pointer to build the list

    // Initialize the head of the merged list
    if (l1->val < l2->val) {
        head = l1;
        l1 = l1->next;
    } else {
        head = l2;
        l2 = l2->next;
```

```
    }
    current = head;

    // Merge the remaining nodes
    while (l1 && l2) {
        if (l1->val < l2->val) {
            current->next = l1;
            l1 = l1->next;
        } else {
            current->next = l2;
            l2 = l2->next;
        }
        current = current->next;
    }

    // Append any remaining nodes
    current->next = l1 ? l1 : l2;

    return head;
}

// Function to create a linked list from user input
ListNode* createList() {
    int n, val;
    cout << "Enter the number of elements in the list: ";
    cin >> n;
    if (n == 0) return nullptr;

    cout << "Enter the elements in sorted order: ";
    cin >> val;
    ListNode* head = new ListNode(val);
    ListNode* current = head;

    for (int i = 1; i < n; ++i) {
        cin >> val;
        current->next = new ListNode(val);
        current = current->next;
    }
    return head;
}

// Function to print a linked list
void printList(ListNode* head) {
    while (head) {
```

```
        cout << head->val << " ";
        head = head->next;
    }
    cout << endl;
}

// Main function to demonstrate merging of two lists
int main() {
    cout << "Create the first sorted linked list:\n";
    ListNode* l1 = createList();

    cout << "Create the second sorted linked list:\n";
    ListNode* l2 = createList();

    ListNode* mergedList = mergeTwoLists(l1, l2);

    cout << "Merged Sorted List: ";
    printList(mergedList);

    return 0;
}
```

Output :

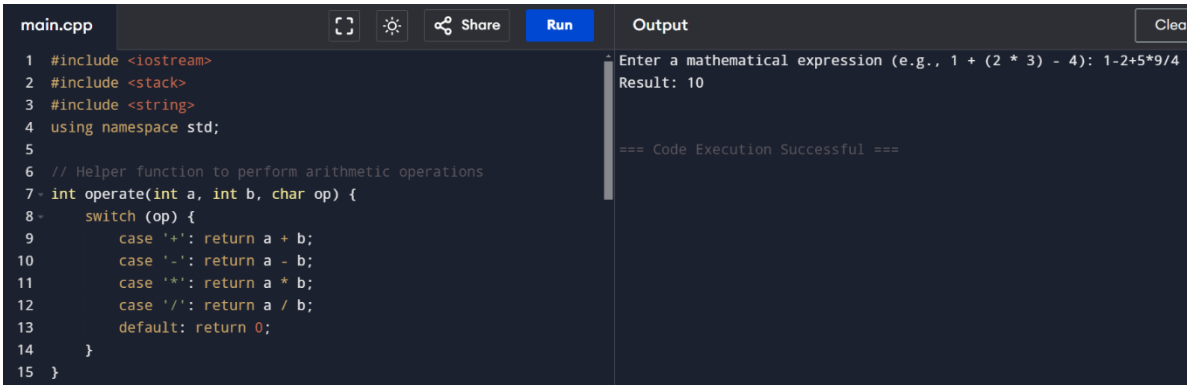


8) Basic Calculator

```
#include <iostream>
#include <stack>
#include <string>
using namespace std;

// Helper function to perform arithmetic operations
int operate(int a, int b, char op) {
   switch (op) {
      case '+': return a + b;
      case '-': return a - b;
      case '*': return a * b;
      case '/': return a / b;
      default: return 0;
   }
}

// Function to evaluate the expression
int calculate(string s) {
   stack<int> nums;    // Stack to store numbers
   stack<char> ops;    // Stack to store operators
   int num = 0;
   int result = 0;
   int sign = 1;

   for (int i = 0; i < s.length(); ++i) {
      char c = s[i];

      if (isdigit(c)) {
         num = num * 10 + (c - '0');
      } else if (c == '+' || c == '-') {
         result += sign * num;
         num = 0;
         sign = (c == '+') ? 1 : -1;
      } else if (c == '(') {
         nums.push(result);
         ops.push(sign);
         result = 0;
         sign = 1;
      } else if (c == ')') {
         result += sign * num;
         num = 0;
         result = nums.top() + ops.top() * result;
         nums.pop();
         ops.pop();
      } else if (c == '*' || c == '/') {
         // Handle multiplication or division immediately
         while (i + 1 < s.length() && isspace(s[i + 1])) i++; // Skip spaces
         int nextNum = 0;
```

```
            i++;
            while (i < s.length() && isdigit(s[i])) {
                nextNum = nextNum * 10 + (s[i] - '0');
                i++;
            }
            i--; // Adjust for next iteration
            num = (c == '*') ? num * nextNum : num / nextNum;
        }
    }

    result += sign * num;
    return result;
}

int main() {
    string expression;
    cout << "Enter a mathematical expression (e.g., 1 + (2 * 3) - 4): ";
    getline(cin, expression);

    cout << "Result: " << calculate(expression) << endl;

    return 0;
}
```

Output :

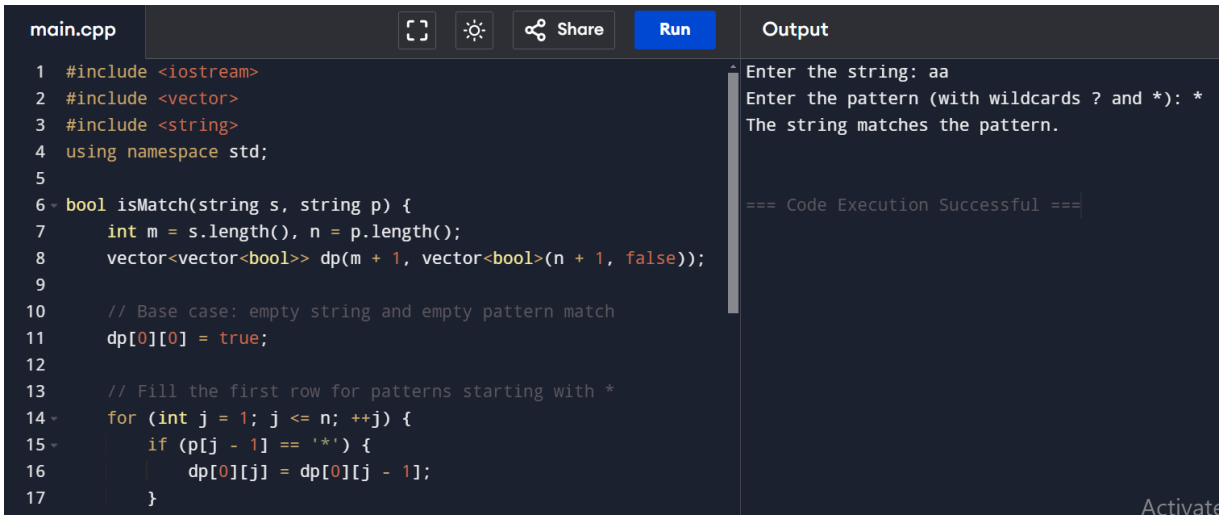


9) Wildcard Matching

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

bool isMatch(string s, string p) {
   int m = s.length(), n = p.length();
   vector<vector<bool>> dp(m + 1, vector<bool>(n + 1, false));

   // Base case: empty string and empty pattern match
   dp[0][0] = true;

   // Fill the first row for patterns starting with *
   for (int j = 1; j <= n; ++j) {
      if (p[j - 1] == '*') {
         dp[0][j] = dp[0][j - 1];
      }
   }

   // Fill the DP table
   for (int i = 1; i <= m; ++i) {
      for (int j = 1; j <= n; ++j) {
         if (p[j - 1] == '*') {
            dp[i][j] = dp[i - 1][j] || dp[i][j - 1];
         } else if (p[j - 1] == '?' || s[i - 1] == p[j - 1]) {
            dp[i][j] = dp[i - 1][j - 1];
         }
      }
   }

   return dp[m][n];
}

int main() {
   string s, p;
   cout << "Enter the string: ";
   cin >> s;
   cout << "Enter the pattern (with wildcards ? and *): ";
   cin >> p;

   if (isMatch(s, p)) {
      cout << "The string matches the pattern." << endl;
   } else {
      cout << "The string does not match the pattern." << endl;
   }
```

```
    return 0;
}
```

Output :



10) Cheapest Flight with K Stops

```
 #include <iostream>
#include <vector>
#include <queue>
#include <tuple>
#include <climits>

using namespace std;

int findCheapestPrice(int n, vector<vector<int>>& flights, int src, int dst, int K) {
   // Adjacency list representation
   vector<vector<pair<int, int>>> graph(n);
   for (const auto& flight : flights) {
      graph[flight[0]].emplace_back(flight[1], flight[2]);
   }

   // Min-heap to store {cost, current_city, stops_remaining}
   priority_queue<tuple<int, int, int>, vector<tuple<int, int, int>>, greater<>> pq;
   pq.emplace(0, src, K + 1); // Start with 0 cost, source city, and K+1 stops

   while (!pq.empty()) {
      auto [cost, city, stops] = pq.top();
      pq.pop();

      // If we reach the destination, return the cost
      if (city == dst) {
         return cost;
      }

      // If we have stops remaining, explore neighbors
      if (stops > 0) {
         for (const auto& [next_city, price] : graph[city]) {
            pq.emplace(cost + price, next_city, stops - 1);
         }
      }
   }

   // If no route is found
   return -1;
}

int main() {
   int n, m;
   cout << "Enter the number of cities and flights: ";
   cin >> n >> m;

   vector<vector<int>> flights(m, vector<int>(3));
   cout << "Enter the flights (from, to, price):" << endl;
   for (int i = 0; i < m; ++i) {
```

```
        cin >> flights[i][0] >> flights[i][1] >> flights[i][2];
    }

    int src, dst, K;
    cout << "Enter the source, destination, and maximum stops: ";
    cin >> src >> dst >> K;

    int result = findCheapestPrice(n, flights, src, dst, K);
    if (result != -1) {
        cout << "The cheapest price is: " << result << endl;
    } else {
        cout << "No route available within " << K << " stops." << endl;
    }

    return 0;
}
```

Output :