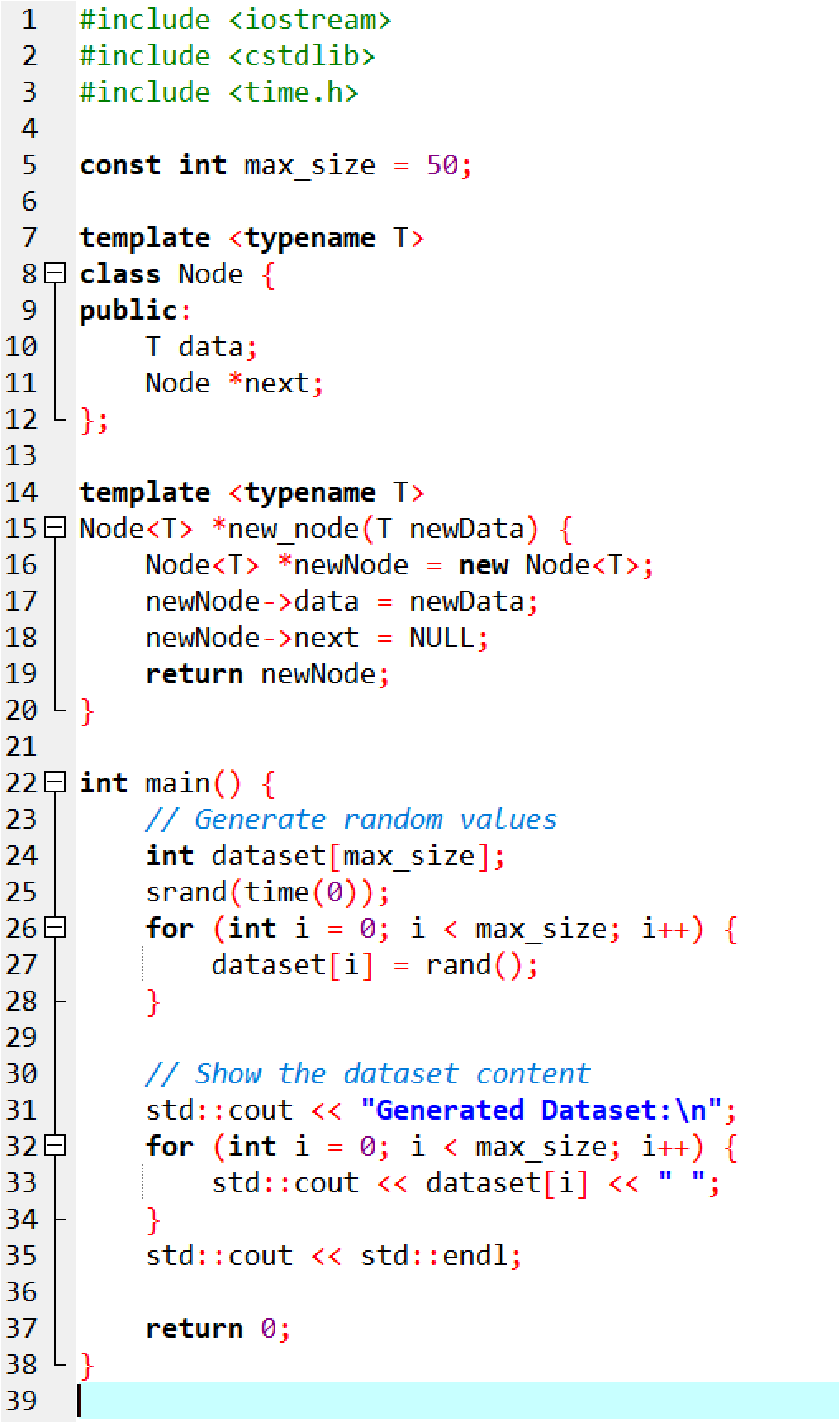
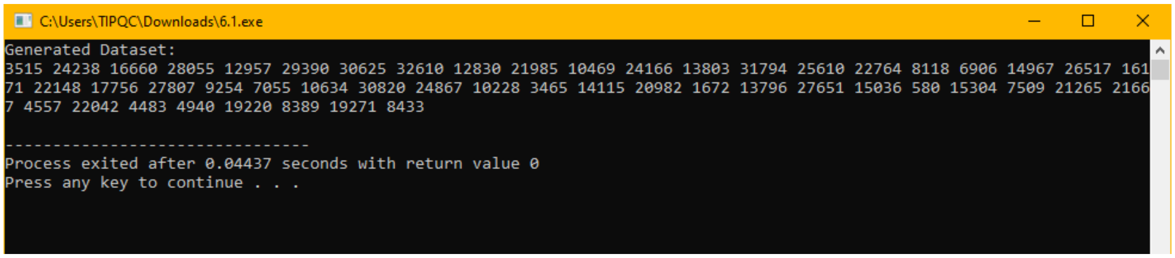
Hands-on Activity 6.1	
Hands-on Activity 6.1 Searching Techniques	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** 10-15-2024
Section: CPE21S4	**Date Submitted:** 10-15-2024
Name(s): Mamaril, Justin Kenneth I.	**Instructor:** Prof. Sayo
6. Output	

Screenshot



```cpp
#include <iostream>
#include <cstdlib>
#include <time.h>

const int max_size = 50;

template <typename T>
class Node {
public:
    T data;
    Node *next;
};

template <typename T>
Node<T> *new_node(T newData) {
    Node<T> *newNode = new Node<T>;
    newNode->data = newData;
    newNode->next = NULL;
    return newNode;
}

int main() {
    // Generate random values
    int dataset[max_size];
    srand(time(0));
    for (int i = 0; i < max_size; i++) {
        dataset[i] = rand();
    }

    // Show the dataset content
    std::cout << "Generated Dataset:\n";
    for (int i = 0; i < max_size; i++) {
        std::cout << dataset[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}

```



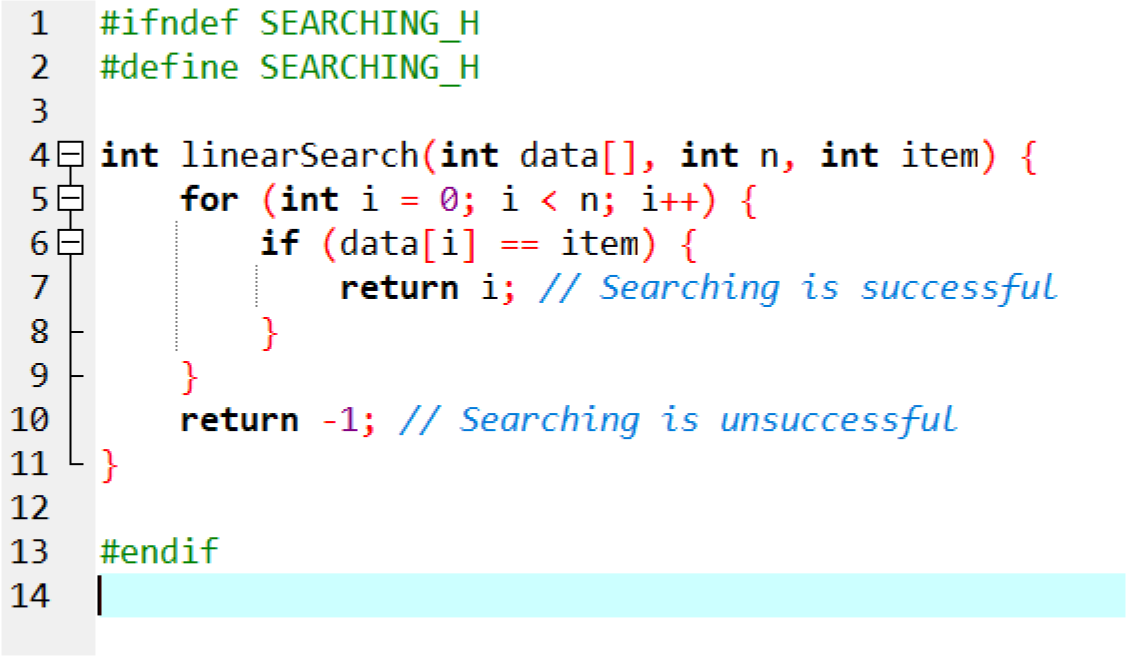
```
Generated Dataset:
3515 24238 16660 28055 12957 29390 30625 32610 12830 21985 10469 24166 13803 31794 25610 22764 8118 6906 14967 26517 161
71 22148 17756 27807 9254 7055 10634 30820 24867 10228 3465 14115 20982 1672 13796 27651 15036 580 15304 7509 21265 2166
7 4557 22042 4483 4940 19220 8389 19271 8433

--------------------------------
Process exited after 0.04437 seconds with return value 0
Press any key to continue . . .
```

Observation

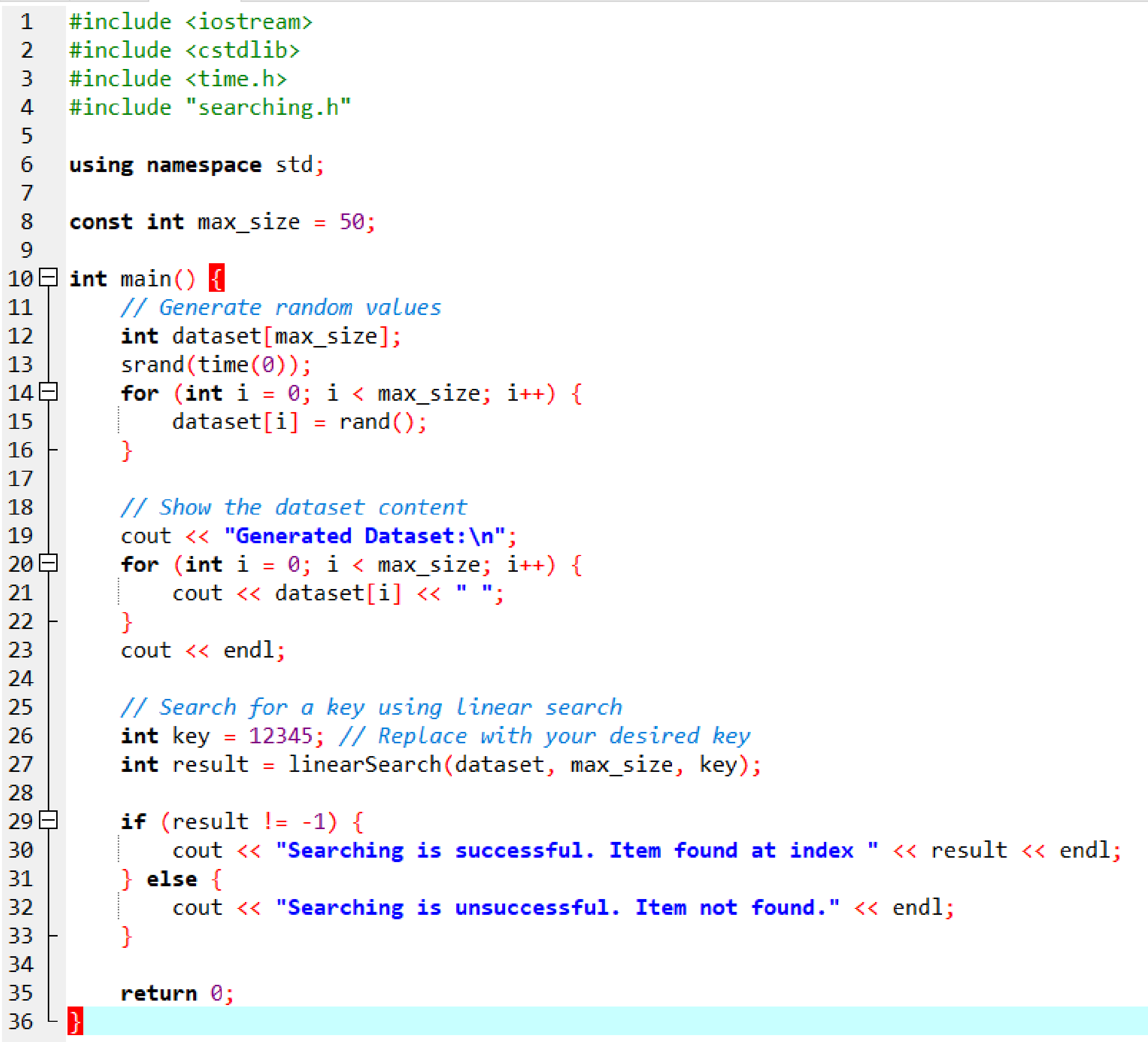
The code exhibits sequential search for arrays and linked lists, as well as binary search for sorted arrays. It produces random data, builds a linked list, and returns clear search results and comparison counts.

Table 6-1. Data Generated and Observations.

Code

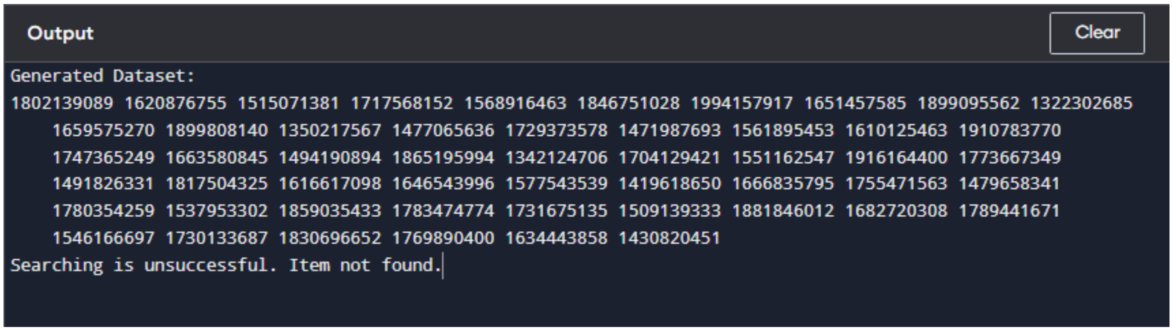


```
#ifndef SEARCHING_H
#define SEARCHING_H

int linearSearch(int data[], int n, int item) {
    for (int i = 0; i < n; i++) {
        if (data[i] == item) {
            return i; // Searching is successful
        }
    }
    return -1; // Searching is unsuccessful
}

#endif

```



```cpp
#include <iostream>
#include <cstdlib>
#include <time.h>
#include "searching.h"

using namespace std;

const int max_size = 50;

int main() {
    // Generate random values
    int dataset[max_size];
    srand(time(0));
    for (int i = 0; i < max_size; i++) {
        dataset[i] = rand();
    }

    // Show the dataset content
    cout << "Generated Dataset:\n";
    for (int i = 0; i < max_size; i++) {
        cout << dataset[i] << " ";
    }
    cout << endl;

    // Search for a key using linear search
    int key = 12345; // Replace with your desired key
    int result = linearSearch(dataset, max_size, key);

    if (result != -1) {
        cout << "Searching is successful. Item found at index " << result << endl;
    } else {
        cout << "Searching is unsuccessful. Item not found." << endl;
    }

    return 0;
}
```

Output



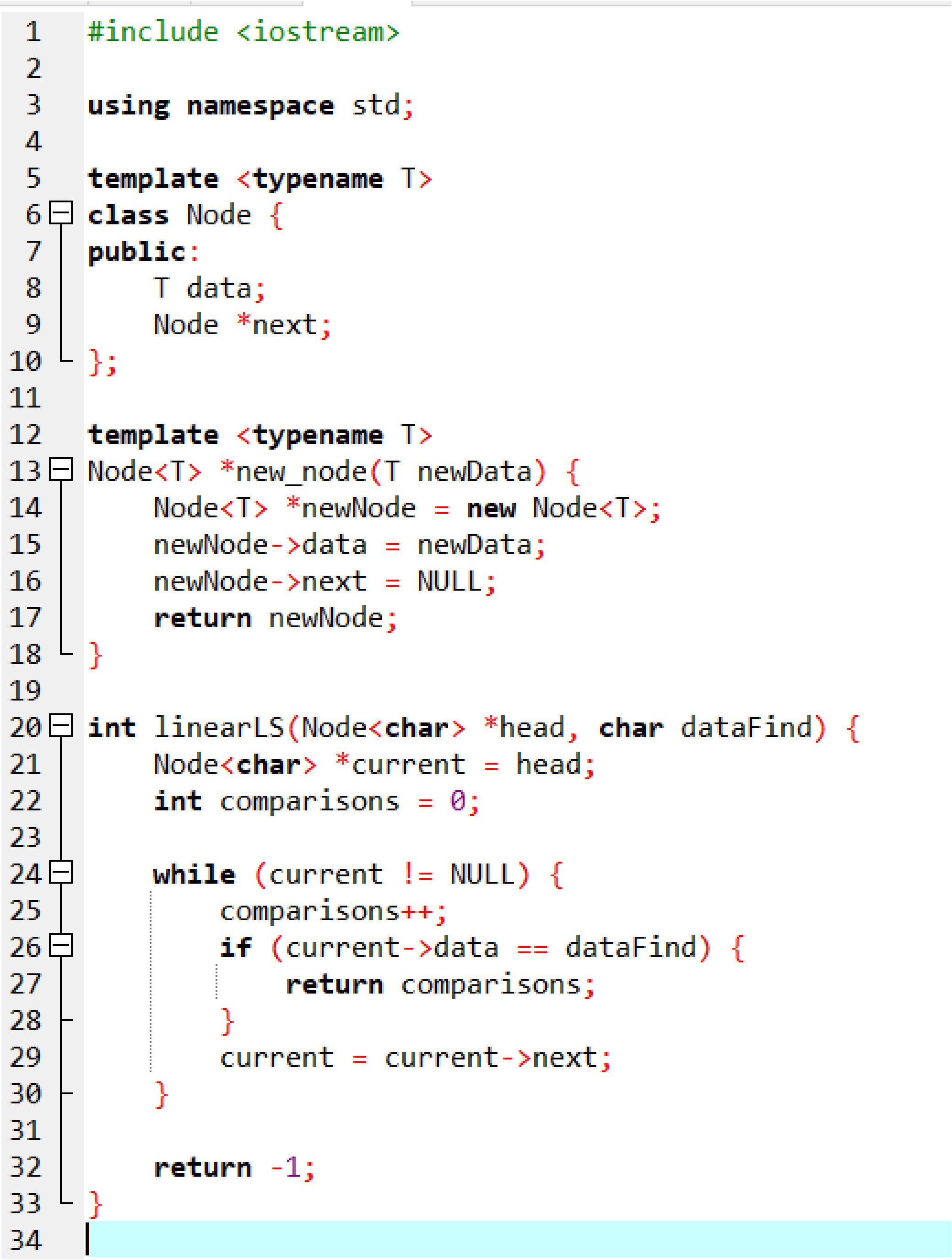
```
Generated Dataset:
1802139089 1620876755 1515071381 1717568152 1568916463 1846751028 1994157917 1651457585 1899095562 1322302685
    1659575270 1899808140 1350217567 1477065636 1729373578 1471987693 1561895453 1610125463 1910783770
    1747365249 1663580845 1494190894 1865195994 1342124706 1704129421 1551162547 1916164400 1773667349
    1491826331 1817504325 1616617098 1646543996 1577543539 1419618650 1666835795 1755471563 1479658341
    1780354259 1537953302 1859035433 1783474774 1731675135 1509139333 1881846012 1682720308 1789441671
    1546166697 1730133687 1830696652 1769890400 1634443858 1430820451
Searching is unsuccessful. Item not found.
```

Observation

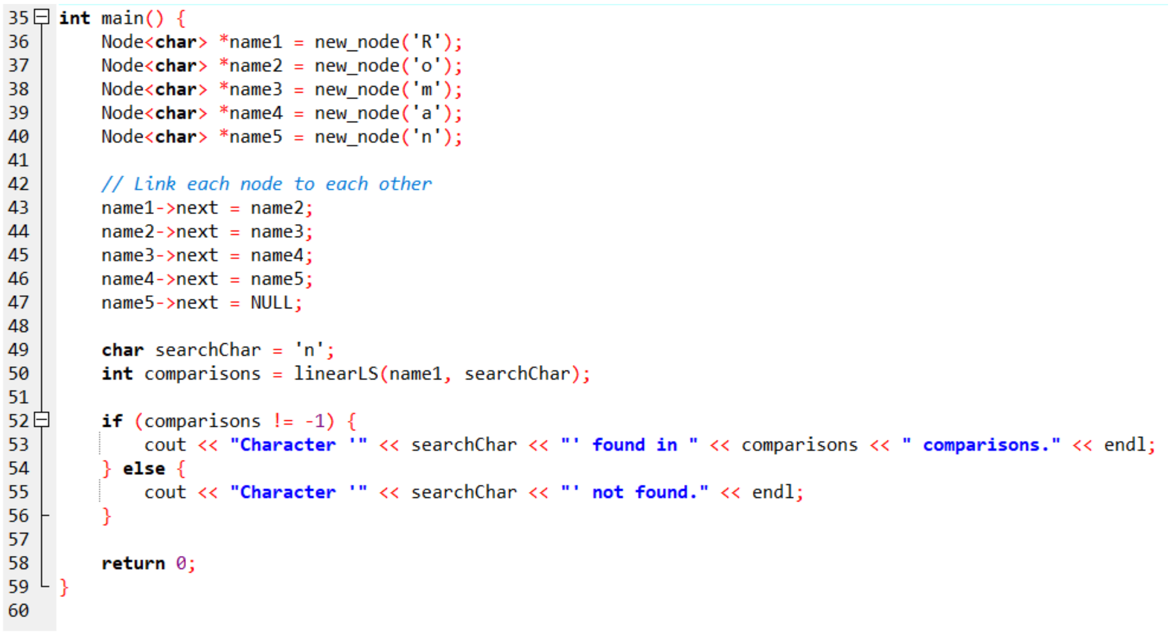
I implemented the linear search algorithm in C++ with a separate header file. I created a random dataset and applied the method to find a specific key. The code executed successfully, and the output indicated whether or not the key was found. Because of its sequential nature, the linear search algorithm has been shown to be inefficient for big datasets.

Table 6-2a. Linear Search for Arrays

Code

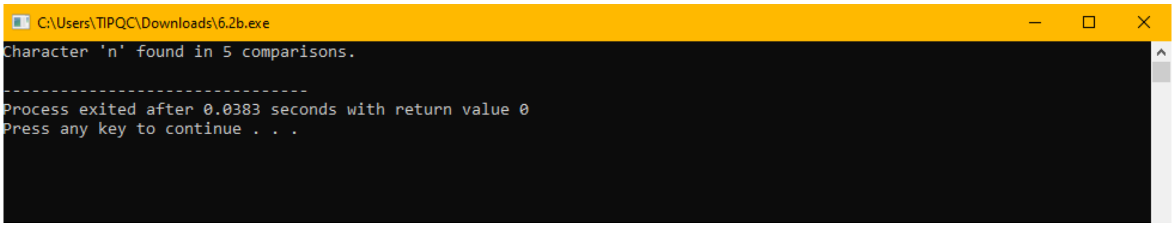


```cpp
#include <iostream>

using namespace std;

template <typename T>
class Node {
public:
    T data;
    Node *next;
};

template <typename T>
Node<T> *new_node(T newData) {
    Node<T> *newNode = new Node<T>;
    newNode->data = newData;
    newNode->next = NULL;
    return newNode;
}

int linearLS(Node<char> *head, char dataFind) {
    Node<char> *current = head;
    int comparisons = 0;

    while (current != NULL) {
        comparisons++;
        if (current->data == dataFind) {
            return comparisons;
        }
        current = current->next;
    }

    return -1;
}

```



```cpp
int main() {
    Node<char> *name1 = new_node('R');
    Node<char> *name2 = new_node('o');
    Node<char> *name3 = new_node('m');
    Node<char> *name4 = new_node('a');
    Node<char> *name5 = new_node('n');

    // Link each node to each other
    name1->next = name2;
    name2->next = name3;
    name3->next = name4;
    name4->next = name5;
    name5->next = NULL;

    char searchChar = 'n';
    int comparisons = linearLS(name1, searchChar);

    if (comparisons != -1) {
        cout << "Character '" << searchChar << "' found in " << comparisons << " comparisons." << endl;
    } else {
        cout << "Character '" << searchChar << "' not found." << endl;
    }

    return 0;
}

```

Output



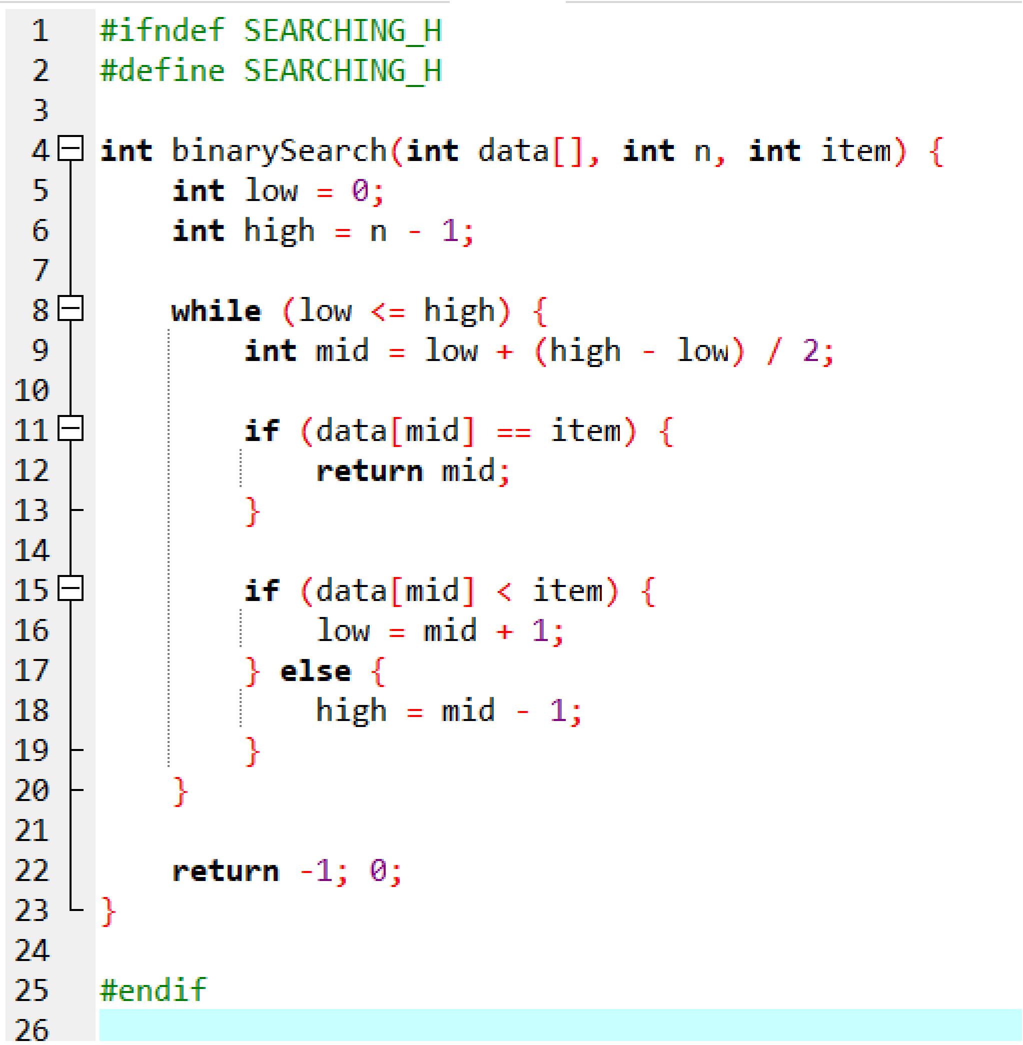
```
Character 'n' found in 5 comparisons.

--------------------------------
Process exited after 0.0383 seconds with return value 0
Press any key to continue . . .
```

Observation

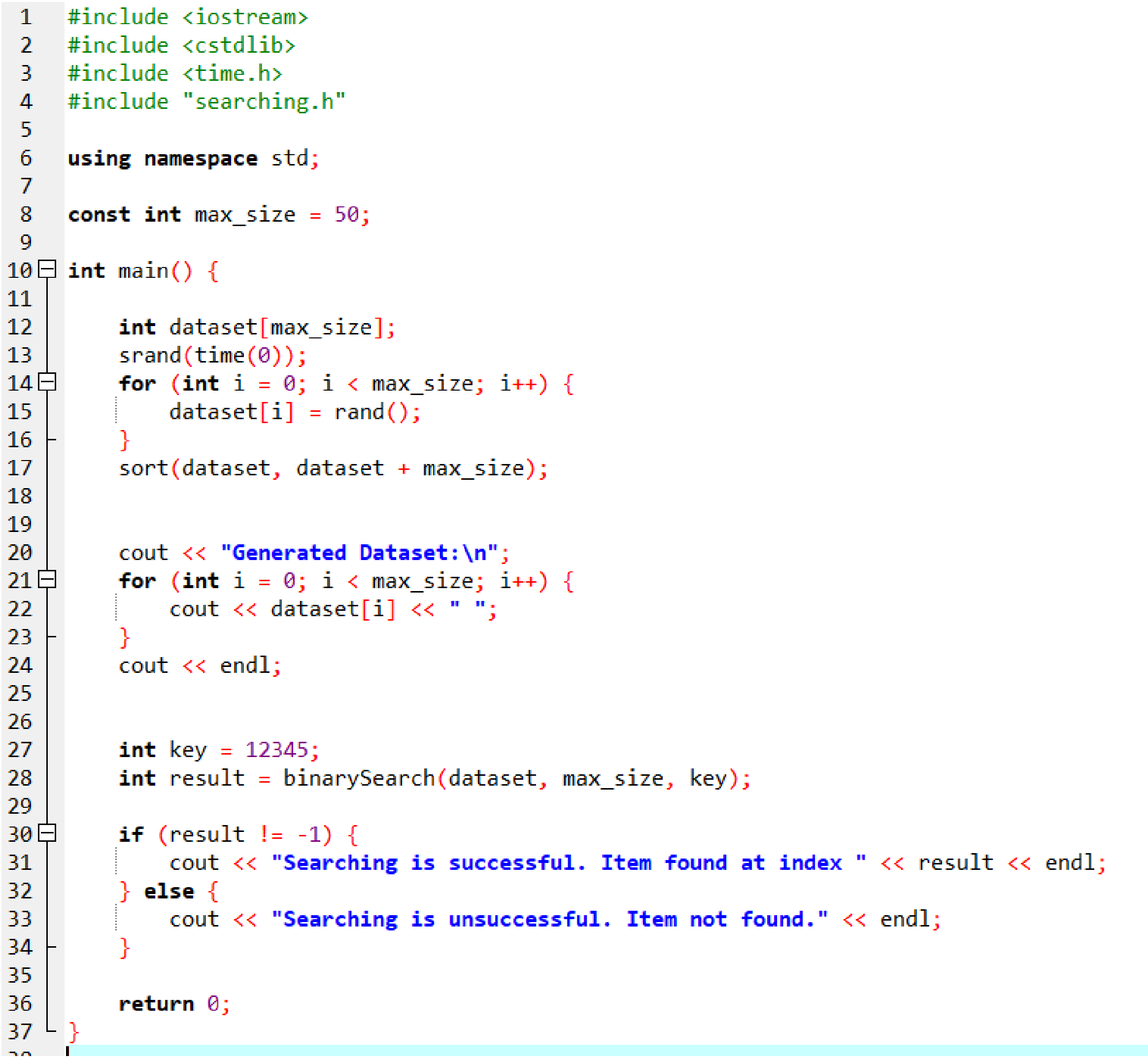
I made a linked list with my first name and used sequential search to locate a specific character. The character appeared in five comparisons, illustrating the algorithm's sequential nature.

Table 6-2b. Linear Search for Linked List

Code

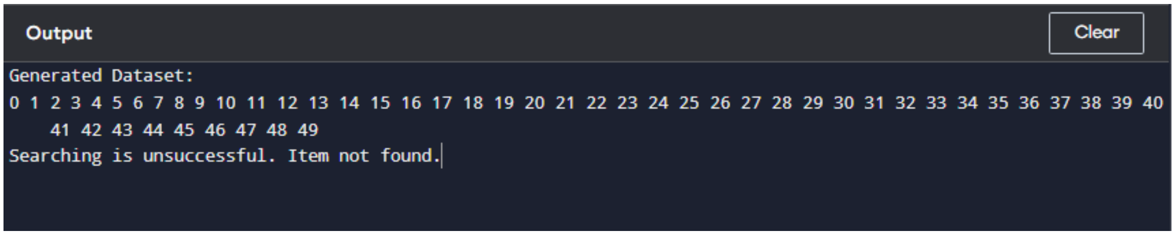


```
#ifndef SEARCHING_H
#define SEARCHING_H

int binarySearch(int data[], int n, int item) {
    int low = 0;
    int high = n - 1;

    while (low <= high) {
        int mid = low + (high - low) / 2;

        if (data[mid] == item) {
            return mid;
        }

        if (data[mid] < item) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }

    return -1; 0;
}

#endif

```



```cpp
#include <iostream>
#include <cstdlib>
#include <time.h>
#include "searching.h"

using namespace std;

const int max_size = 50;

int main() {

    int dataset[max_size];
    srand(time(0));
    for (int i = 0; i < max_size; i++) {
        dataset[i] = rand();
    }
    sort(dataset, dataset + max_size);


    cout << "Generated Dataset:\n";
    for (int i = 0; i < max_size; i++) {
        cout << dataset[i] << " ";
    }
    cout << endl;


    int key = 12345;
    int result = binarySearch(dataset, max_size, key);

    if (result != -1) {
        cout << "Searching is successful. Item found at index " << result << endl;
    } else {
        cout << "Searching is unsuccessful. Item not found." << endl;
    }

    return 0;
}
```

Output



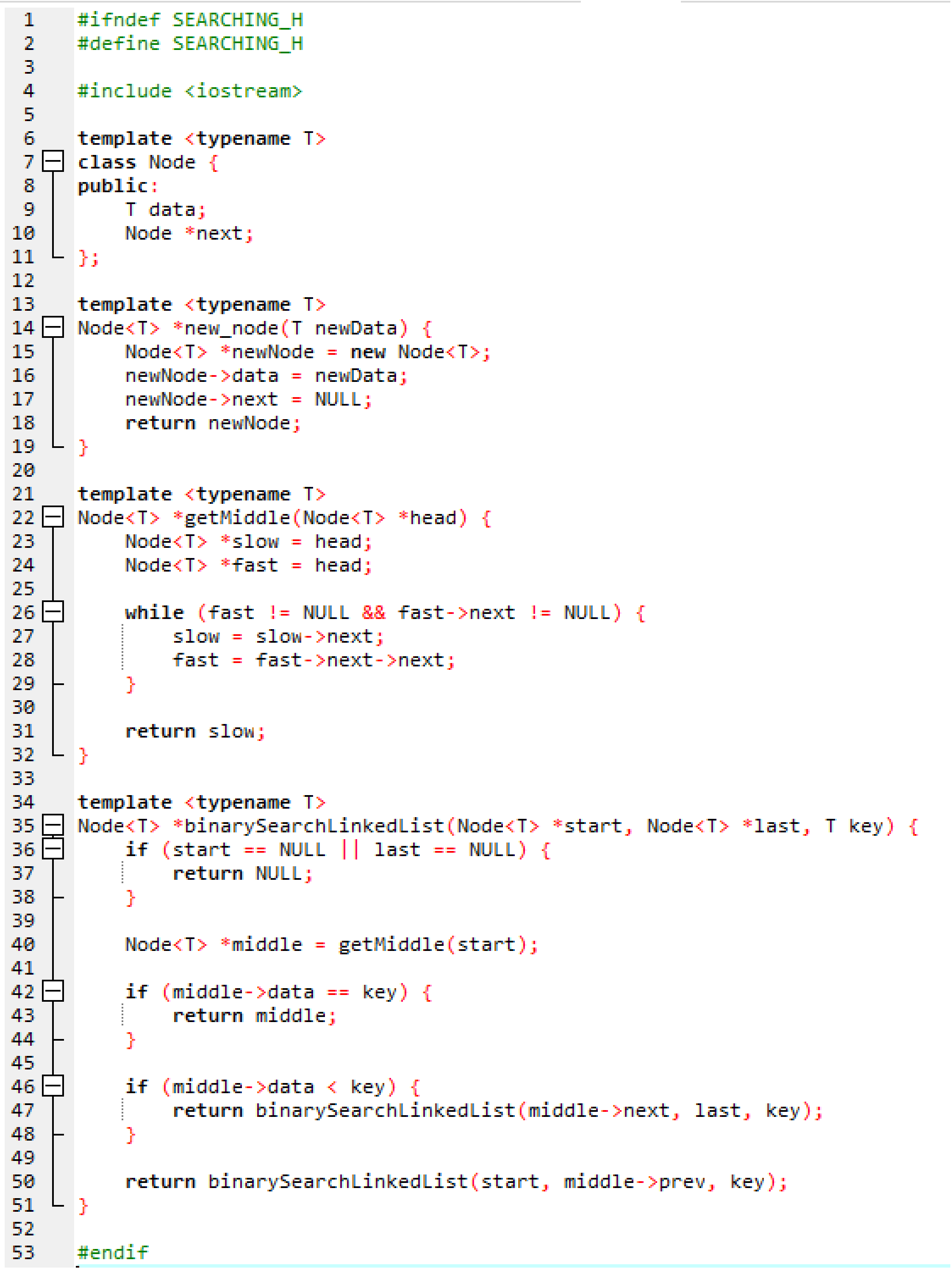
```
Generated Dataset:
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
    41 42 43 44 45 46 47 48 49
Searching is unsuccessful. Item not found.
```

Observation

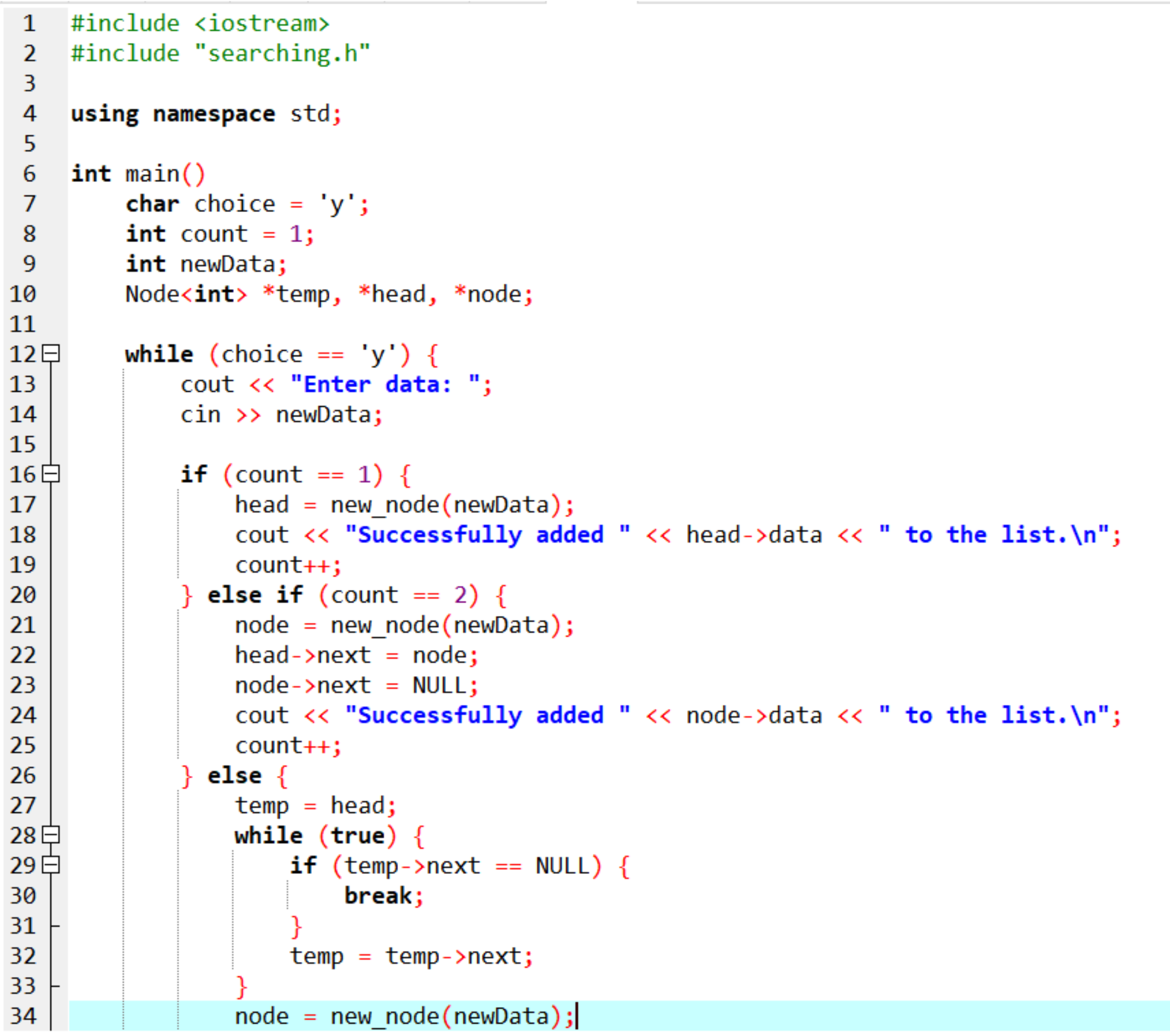
I implemented binary search, which is more efficient than linear search for large, sorted datasets. It divides the search space in half with each iteration, quickly narrowing down the search area.

Table 6-3a. Binary Search for Arrays

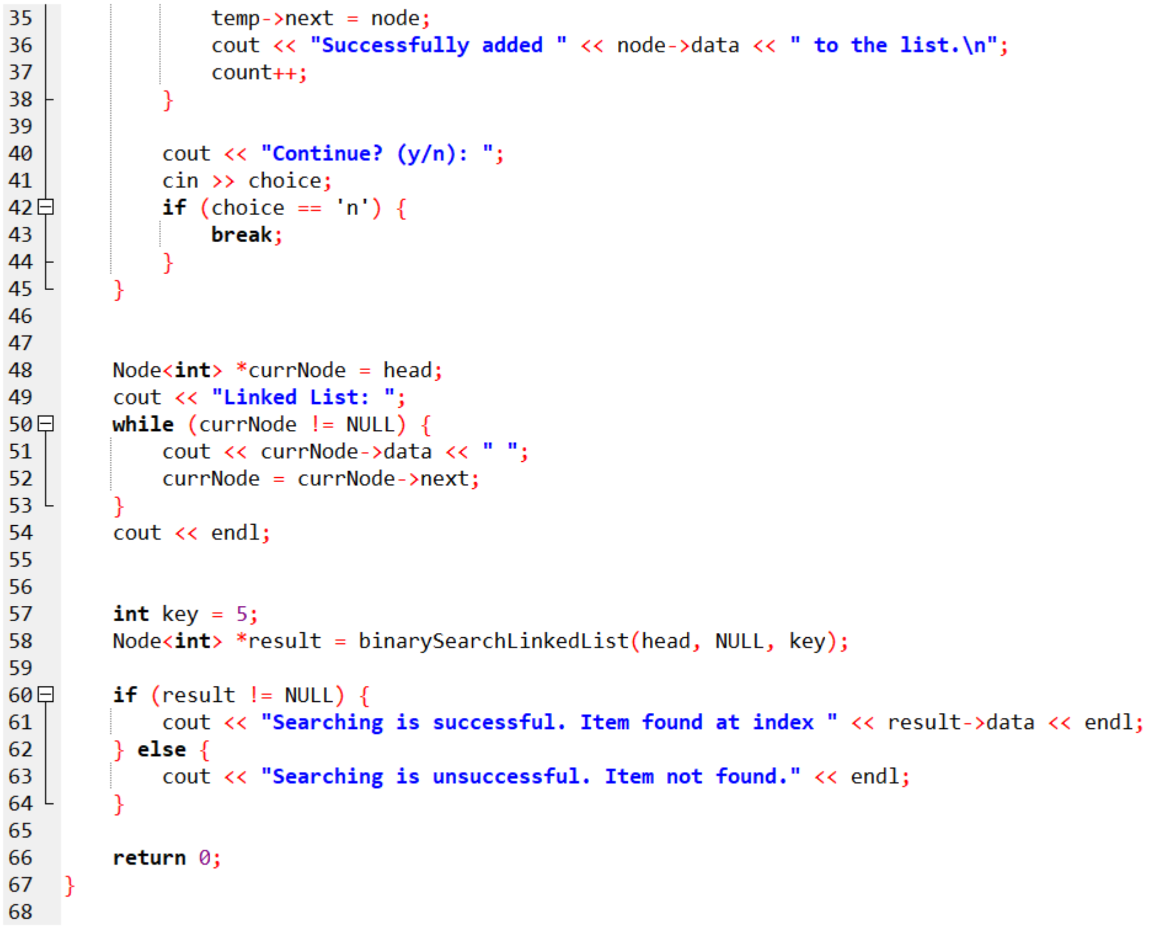
Code



```cpp
#ifndef SEARCHING_H
#define SEARCHING_H

#include <iostream>

template <typename T>
class Node {
public:
    T data;
    Node *next;
};

template <typename T>
Node<T> *new_node(T newData) {
    Node<T> *newNode = new Node<T>;
    newNode->data = newData;
    newNode->next = NULL;
    return newNode;
}

template <typename T>
Node<T> *getMiddle(Node<T> *head) {
    Node<T> *slow = head;
    Node<T> *fast = head;

    while (fast != NULL && fast->next != NULL) {
        slow = slow->next;
        fast = fast->next->next;
    }

    return slow;
}

template <typename T>
Node<T> *binarySearchLinkedList(Node<T> *start, Node<T> *last, T key) {
    if (start == NULL || last == NULL) {
        return NULL;
    }

    Node<T> *middle = getMiddle(start);

    if (middle->data == key) {
        return middle;
    }

    if (middle->data < key) {
        return binarySearchLinkedList(middle->next, last, key);
    }

    return binarySearchLinkedList(start, middle->prev, key);
}

#endif
```

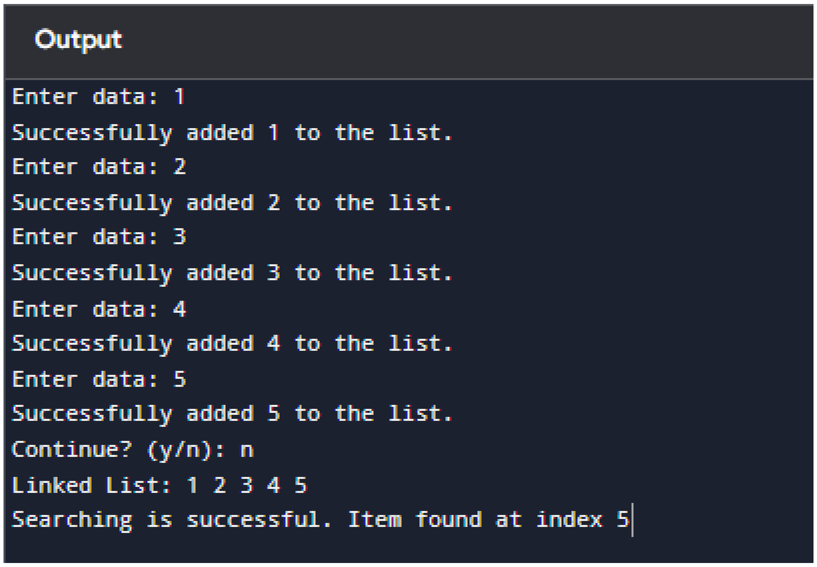


```cpp
#include <iostream>
#include "searching.h"

using namespace std;

int main()
    char choice = 'y';
    int count = 1;
    int newData;
    Node<int> *temp, *head, *node;

    while (choice == 'y') {
        cout << "Enter data: ";
        cin >> newData;

        if (count == 1) {
            head = new_node(newData);
            cout << "Successfully added " << head->data << " to the list.\n";
            count++;
        } else if (count == 2) {
            node = new_node(newData);
            head->next = node;
            node->next = NULL;
            cout << "Successfully added " << node->data << " to the list.\n";
            count++;
        } else {
            temp = head;
            while (true) {
                if (temp->next == NULL) {
                    break;
                }
                temp = temp->next;
            }
            node = new_node(newData);
```



```
            temp->next = node;
            cout << "Successfully added " << node->data << " to the list.\n";
            count++;
        }

        cout << "Continue? (y/n): ";
        cin >> choice;
        if (choice == 'n') {
            break;
        }
    }


    Node<int> *currNode = head;
    cout << "Linked List: ";
    while (currNode != NULL) {
        cout << currNode->data << " ";
        currNode = currNode->next;
    }
    cout << endl;


    int key = 5;
    Node<int> *result = binarySearchLinkedList(head, NULL, key);

    if (result != NULL) {
        cout << "Searching is successful. Item found at index " << result->data << endl;
    } else {
        cout << "Searching is unsuccessful. Item not found." << endl;
    }

    return 0;
}

```

Output



```
Enter data: 1
Successfully added 1 to the list.
Enter data: 2
Successfully added 2 to the list.
Enter data: 3
Successfully added 3 to the list.
Enter data: 4
Successfully added 4 to the list.
Enter data: 5
Successfully added 5 to the list.
Continue? (y/n): n
Linked List: 1 2 3 4 5
Searching is successful. Item found at index 5
```

Observation

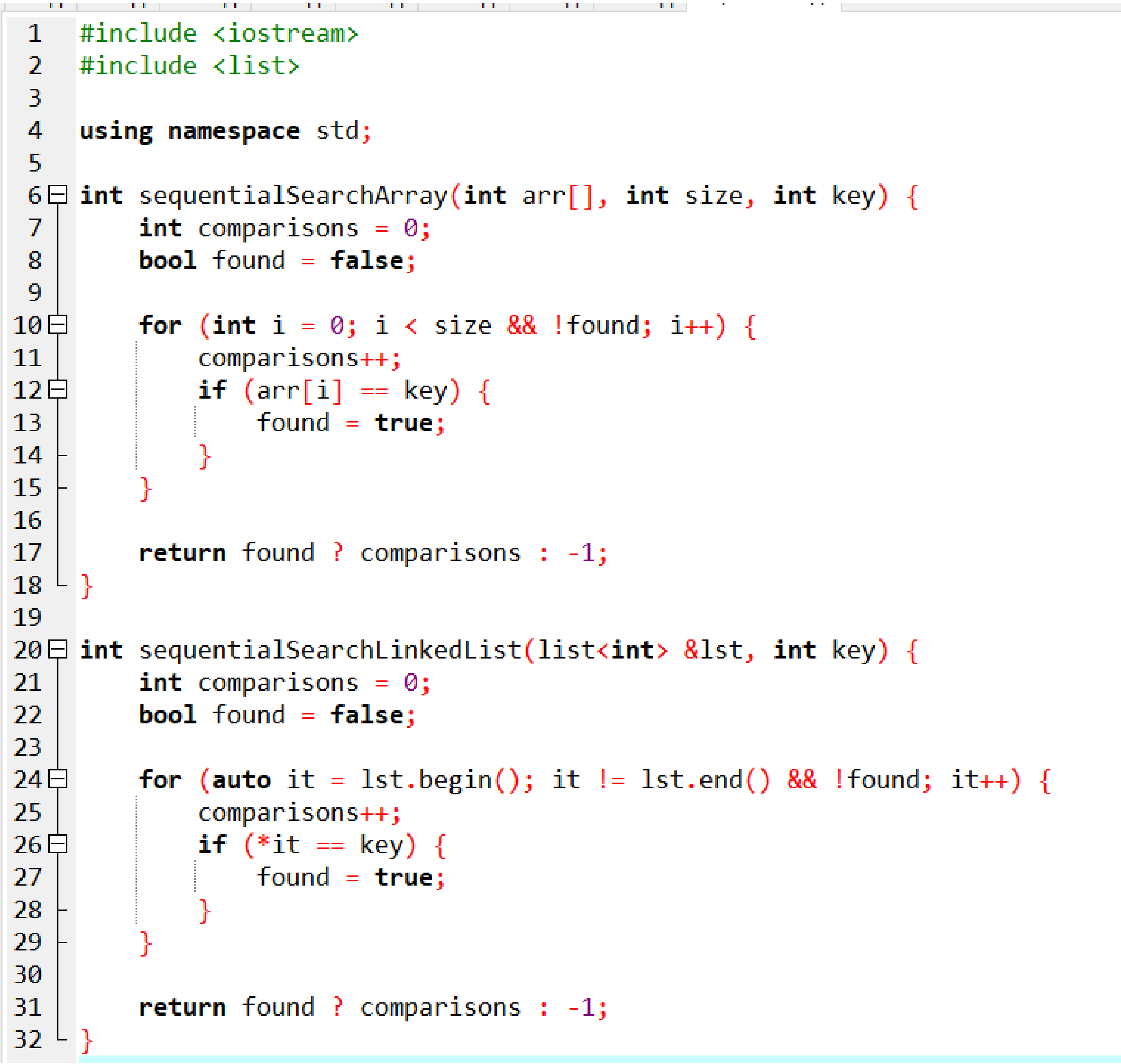
I built binary search for linked lists, which is faster than linear search for sorted data. The getMiddle function successfully locates the middle node, allowing for quick searching in the linked list.

Table 6-3b. Binary Search for Linked List

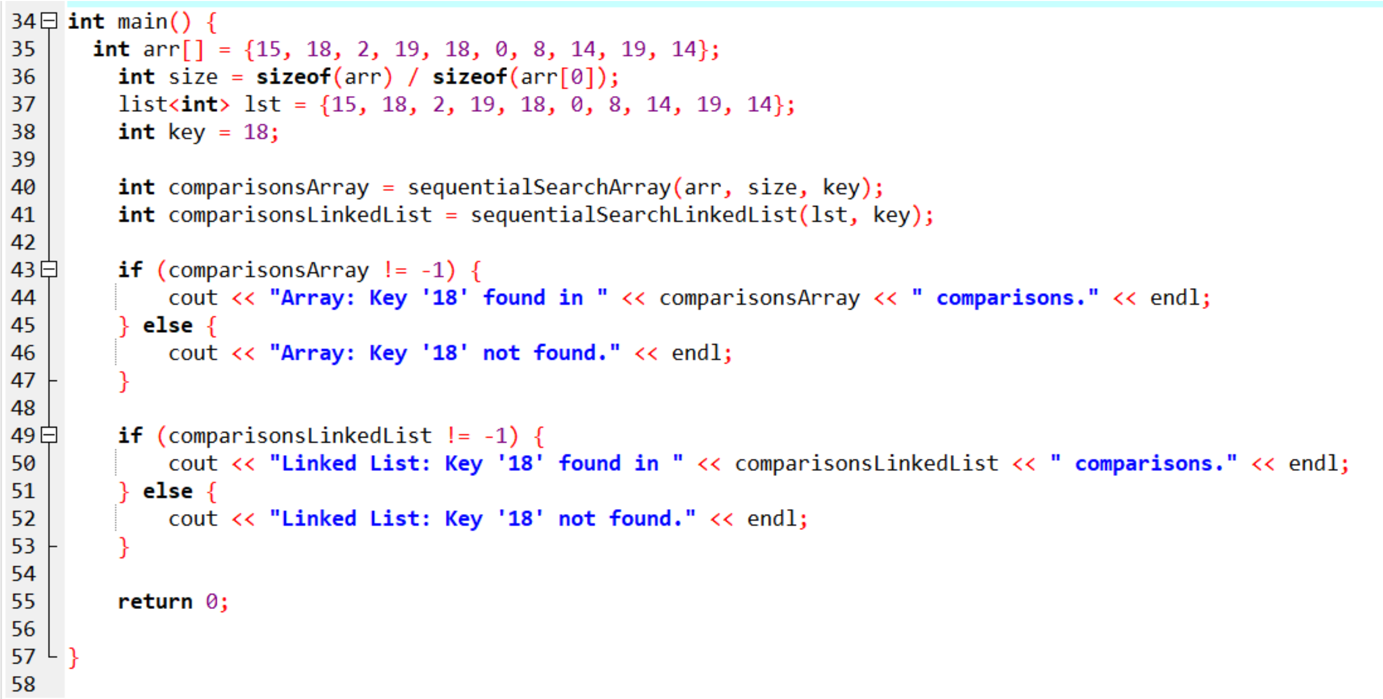
7. Supplementary Activity

PROBLEM 1

CODE:

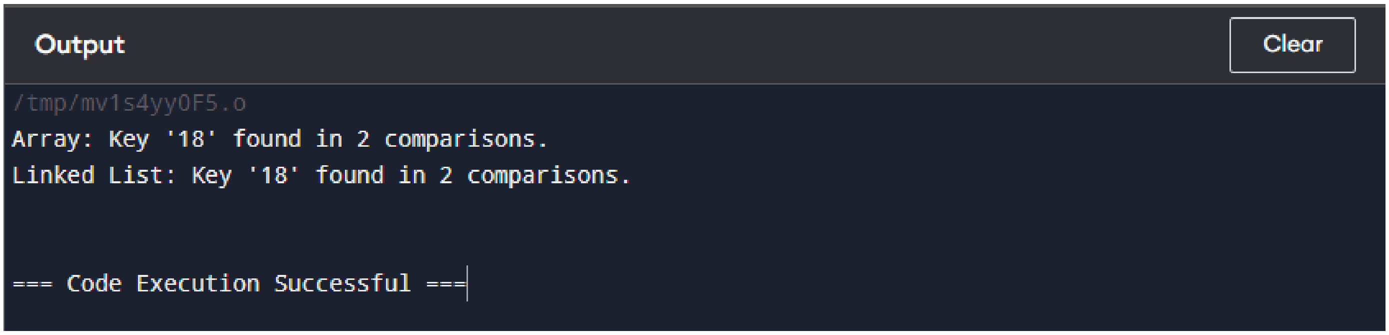


```cpp
#include <iostream>
#include <list>

using namespace std;

int sequentialSearchArray(int arr[], int size, int key) {
    int comparisons = 0;
    bool found = false;

    for (int i = 0; i < size && !found; i++) {
        comparisons++;
        if (arr[i] == key) {
            found = true;
        }
    }

    return found ? comparisons : -1;
}

int sequentialSearchLinkedList(list<int> &lst, int key) {
    int comparisons = 0;
    bool found = false;

    for (auto it = lst.begin(); it != lst.end() && !found; it++) {
        comparisons++;
        if (*it == key) {
            found = true;
        }
    }

    return found ? comparisons : -1;
}
```



```cpp
int main() {
  int arr[] = {15, 18, 2, 19, 18, 0, 8, 14, 19, 14};
    int size = sizeof(arr) / sizeof(arr[0]);
    list<int> lst = {15, 18, 2, 19, 18, 0, 8, 14, 19, 14};
    int key = 18;

    int comparisonsArray = sequentialSearchArray(arr, size, key);
    int comparisonsLinkedList = sequentialSearchLinkedList(lst, key);

    if (comparisonsArray != -1) {
        cout << "Array: Key '18' found in " << comparisonsArray << " comparisons." << endl;
    } else {
        cout << "Array: Key '18' not found." << endl;
    }

    if (comparisonsLinkedList != -1) {
        cout << "Linked List: Key '18' found in " << comparisonsLinkedList << " comparisons." << endl;
    } else {
        cout << "Linked List: Key '18' not found." << endl;
    }

    return 0;

}
```

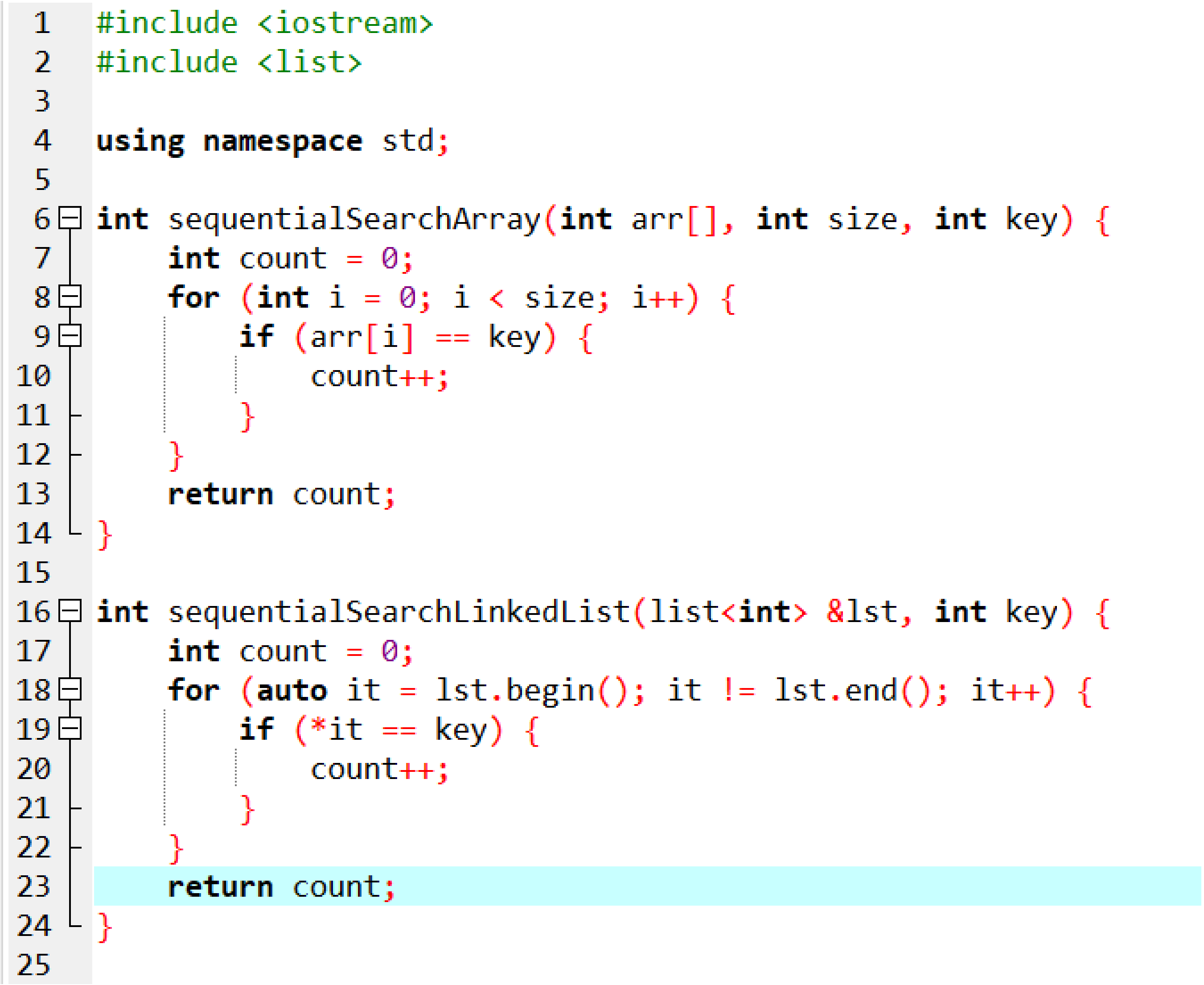
OUTPUT:



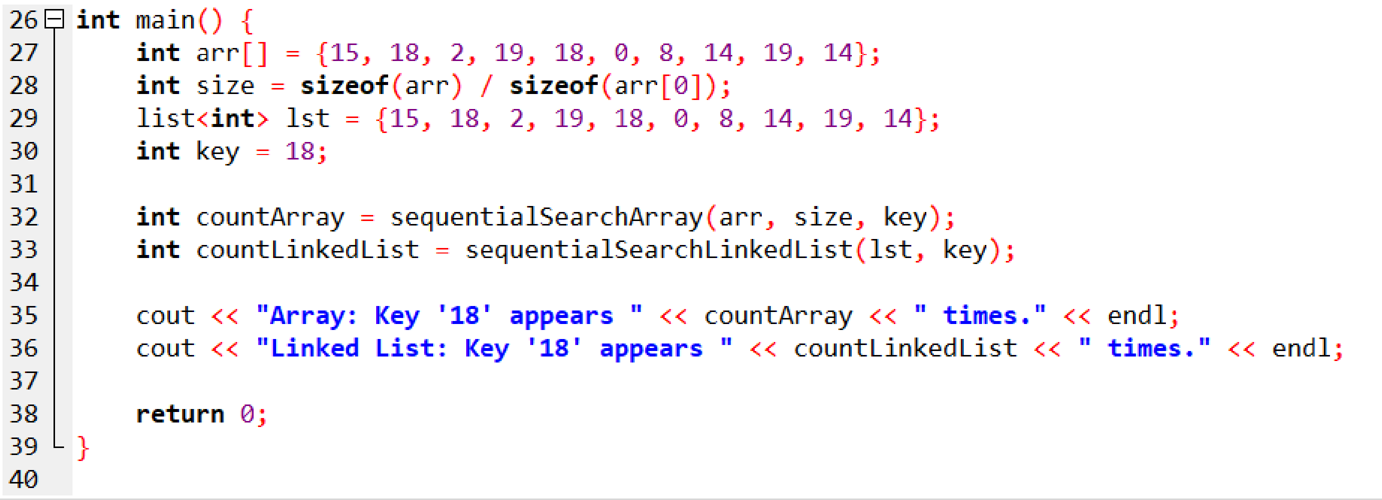
```
/tmp/mv1s4yy0F5.o
Array: Key '18' found in 2 comparisons.
Linked List: Key '18' found in 2 comparisons.


=== Code Execution Successful ===
```

PROBLEM 2

CODE:

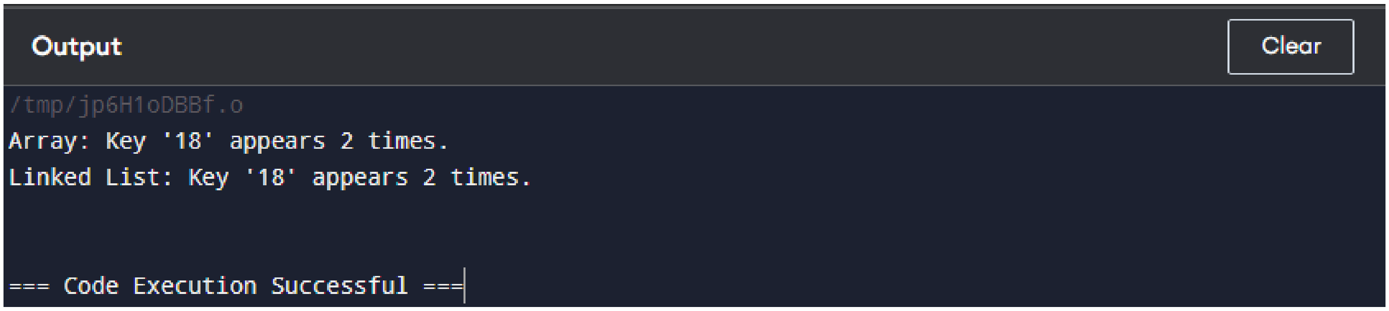


```cpp
#include <iostream>
#include <list>

using namespace std;

int sequentialSearchArray(int arr[], int size, int key) {
    int count = 0;
    for (int i = 0; i < size; i++) {
        if (arr[i] == key) {
            count++;
        }
    }
    return count;
}

int sequentialSearchLinkedList(list<int> &lst, int key) {
    int count = 0;
    for (auto it = lst.begin(); it != lst.end(); it++) {
        if (*it == key) {
            count++;
        }
    }
    return count;
}

```



```cpp
int main() {
    int arr[] = {15, 18, 2, 19, 18, 0, 8, 14, 19, 14};
    int size = sizeof(arr) / sizeof(arr[0]);
    list<int> lst = {15, 18, 2, 19, 18, 0, 8, 14, 19, 14};
    int key = 18;

    int countArray = sequentialSearchArray(arr, size, key);
    int countLinkedList = sequentialSearchLinkedList(lst, key);

    cout << "Array: Key '18' appears " << countArray << " times." << endl;
    cout << "Linked List: Key '18' appears " << countLinkedList << " times." << endl;

    return 0;
}

```

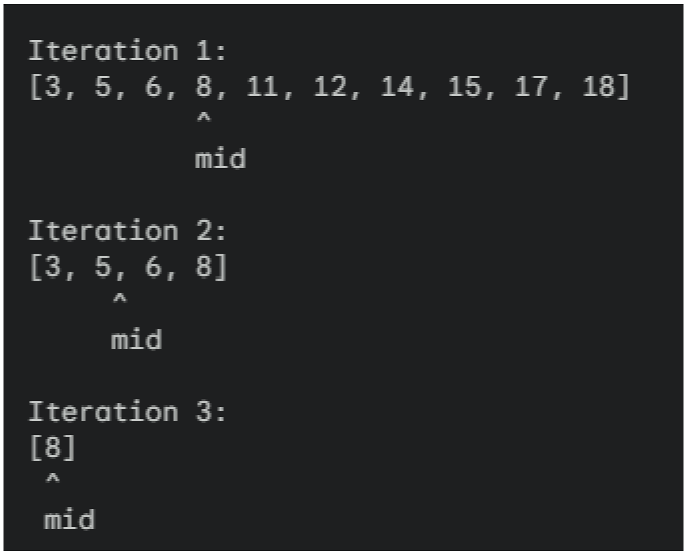
OUTPUT:



```
/tmp/jp6H1oDBBf.o
Array: Key '18' appears 2 times.
Linked List: Key '18' appears 2 times.


=== Code Execution Successful ===
```

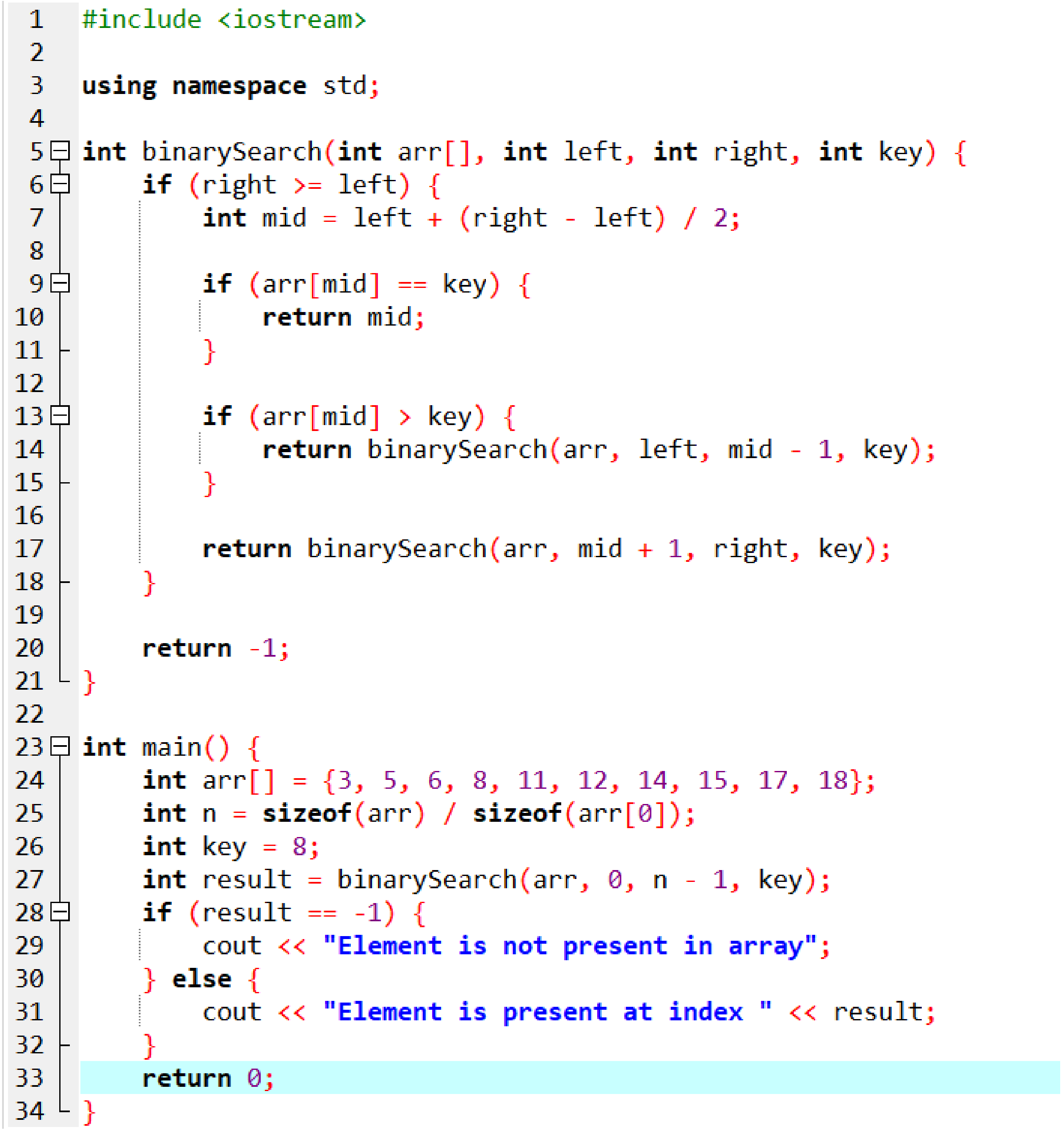
PROBLEM 3

DIAGRAM:



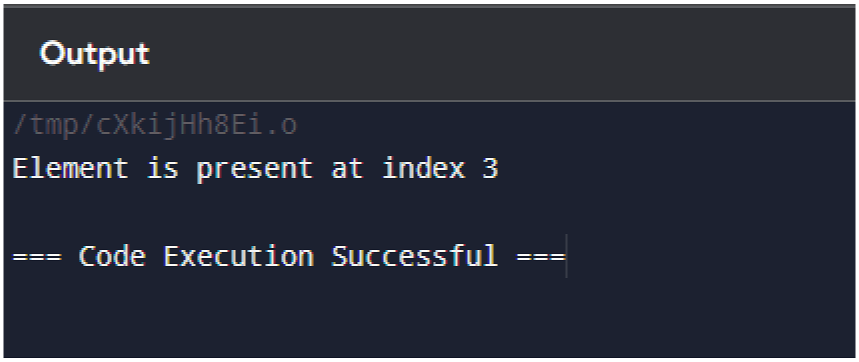
```
Iteration 1:
[3, 5, 6, 8, 11, 12, 14, 15, 17, 18]
          ^
          mid

Iteration 2:
[3, 5, 6, 8]
     ^
     mid

Iteration 3:
[8]
 ^
 mid
```

CODE:



```cpp
#include <iostream>

using namespace std;

int binarySearch(int arr[], int left, int right, int key) {
    if (right >= left) {
        int mid = left + (right - left) / 2;

        if (arr[mid] == key) {
            return mid;
        }

        if (arr[mid] > key) {
            return binarySearch(arr, left, mid - 1, key);
        }

        return binarySearch(arr, mid + 1, right, key);
    }

    return -1;
}

int main() {
    int arr[] = {3, 5, 6, 8, 11, 12, 14, 15, 17, 18};
    int n = sizeof(arr) / sizeof(arr[0]);
    int key = 8;
    int result = binarySearch(arr, 0, n - 1, key);
    if (result == -1) {
        cout << "Element is not present in array";
    } else {
        cout << "Element is present at index " << result;
    }
    return 0;
}
```

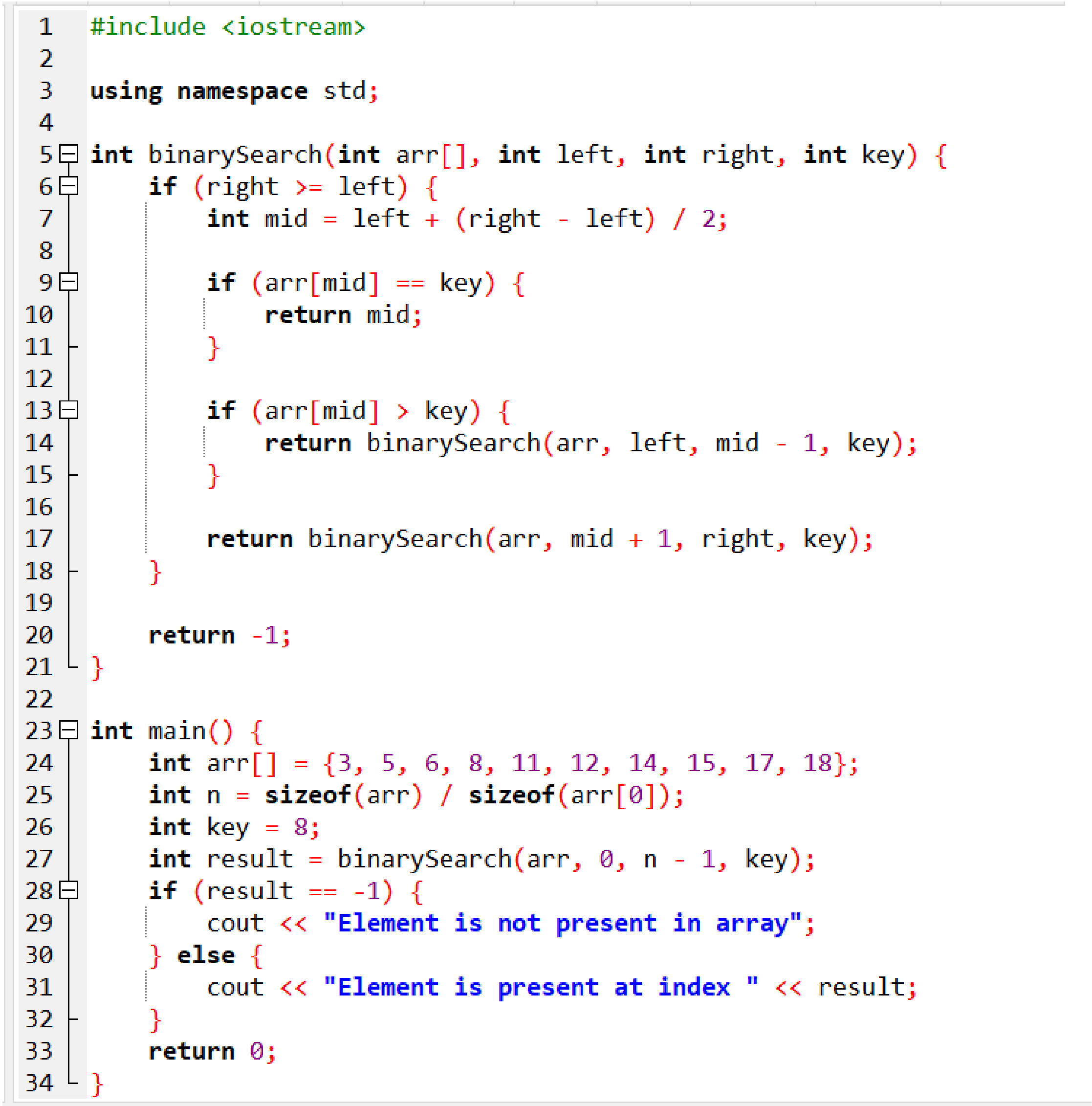
OUTPUT:



```
/tmp/cXkijHh8Ei.o
Element is present at index 3

=== Code Execution Successful ===
```

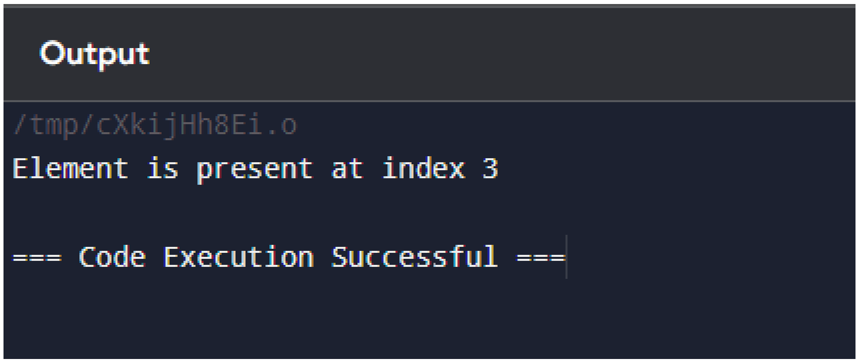
PROBLEM 4

CODE:



```cpp
#include <iostream>

using namespace std;

int binarySearch(int arr[], int left, int right, int key) {
    if (right >= left) {
        int mid = left + (right - left) / 2;

        if (arr[mid] == key) {
            return mid;
        }

        if (arr[mid] > key) {
            return binarySearch(arr, left, mid - 1, key);
        }

        return binarySearch(arr, mid + 1, right, key);
    }

    return -1;
}

int main() {
    int arr[] = {3, 5, 6, 8, 11, 12, 14, 15, 17, 18};
    int n = sizeof(arr) / sizeof(arr[0]);
    int key = 8;
    int result = binarySearch(arr, 0, n - 1, key);
    if (result == -1) {
        cout << "Element is not present in array";
    } else {
        cout << "Element is present at index " << result;
    }
    return 0;
}
```

OUTPUT:



```
/tmp/cXkijHh8Ei.o
Element is present at index 3

=== Code Execution Successful ===
```

8. Conclusion
I gained a thorough understanding of C++ searching algorithms, including binary and sequential search. I was able to put these algorithms to use and assess their performance. I want to learn more about complex searching strategies and go deeper into algorithm analysis in order to improve.
9. Assessment Rubric