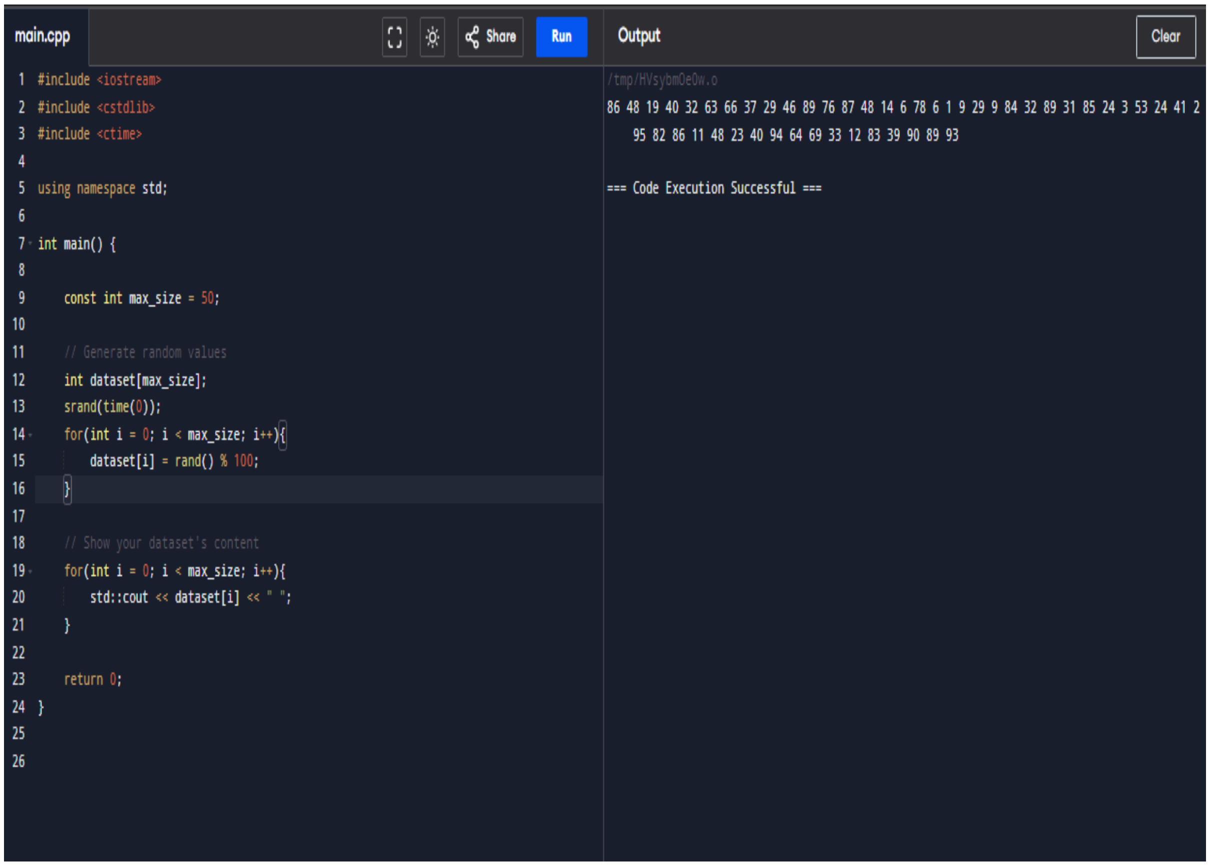
Activity No. 6	
Hands-on Activity 6.1 Searching Techniques	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: 10/15/2024**
Section: CPE21S1	**Date Submitted: 10/15/2024**
Name(s): Jhon Hendricks T. Bautista	**Instructor: Mrs. Maria Rizette Sayo**
6. Output	

Screenshot	

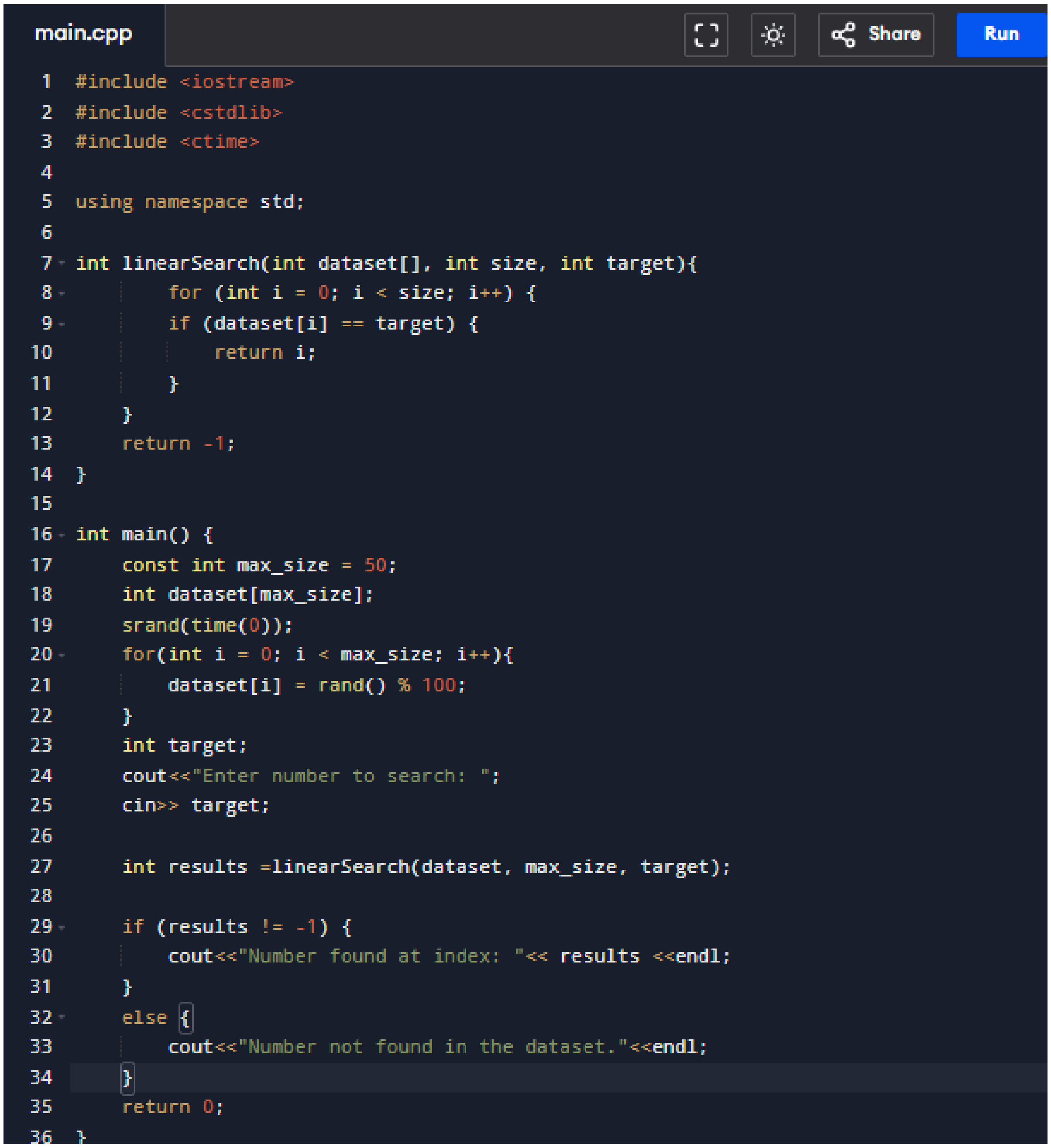
```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>

using namespace std;

int main() {

    const int max_size = 50;

    // Generate random values
    int dataset[max_size];
    srand(time(0));
    for(int i = 0; i < max_size; i++){
        dataset[i] = rand() % 100;
    }

    // Show your dataset's content
    for(int i = 0; i < max_size; i++){
        std::cout << dataset[i] << " ";
    }

    return 0;
}


```

```
/tmp/HVsybmOeOw.o
86 48 19 40 32 63 66 37 29 46 89 76 87 48 14 6 78 6 1 9 29 9 84 32 89 31 85 24 3 53 24 41 2
    95 82 86 11 48 23 40 94 64 69 33 12 83 39 90 89 93

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

Observation	For this code it gives an output of the array containing the random numbers from 0 to 99. It is contained in the array and they are printed with the use of a for loop. An important note is that the values are completely random and not sorted.

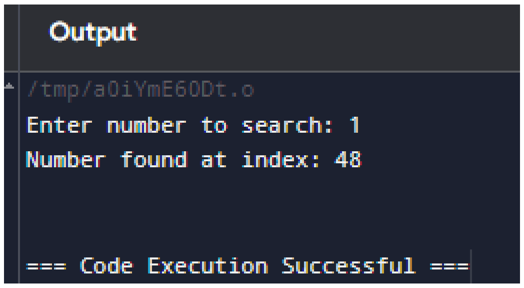
Table 6-1. Data Generated and Observations

Code



```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>

using namespace std;

int linearSearch(int dataset[], int size, int target){
        for (int i = 0; i < size; i++) {
        if (dataset[i] == target) {
            return i;
        }
    }
    return -1;
}

int main() {
    const int max_size = 50;
    int dataset[max_size];
    srand(time(0));
    for(int i = 0; i < max_size; i++){
        dataset[i] = rand() % 100;
    }
    int target;
    cout<<"Enter number to search: ";
    cin>> target;

    int results =linearSearch(dataset, max_size, target);

    if (results != -1) {
        cout<<"Number found at index: "<< results <<endl;
    }
    else {
        cout<<"Number not found in the dataset."<<endl;
    }
    return 0;
}
```

Output



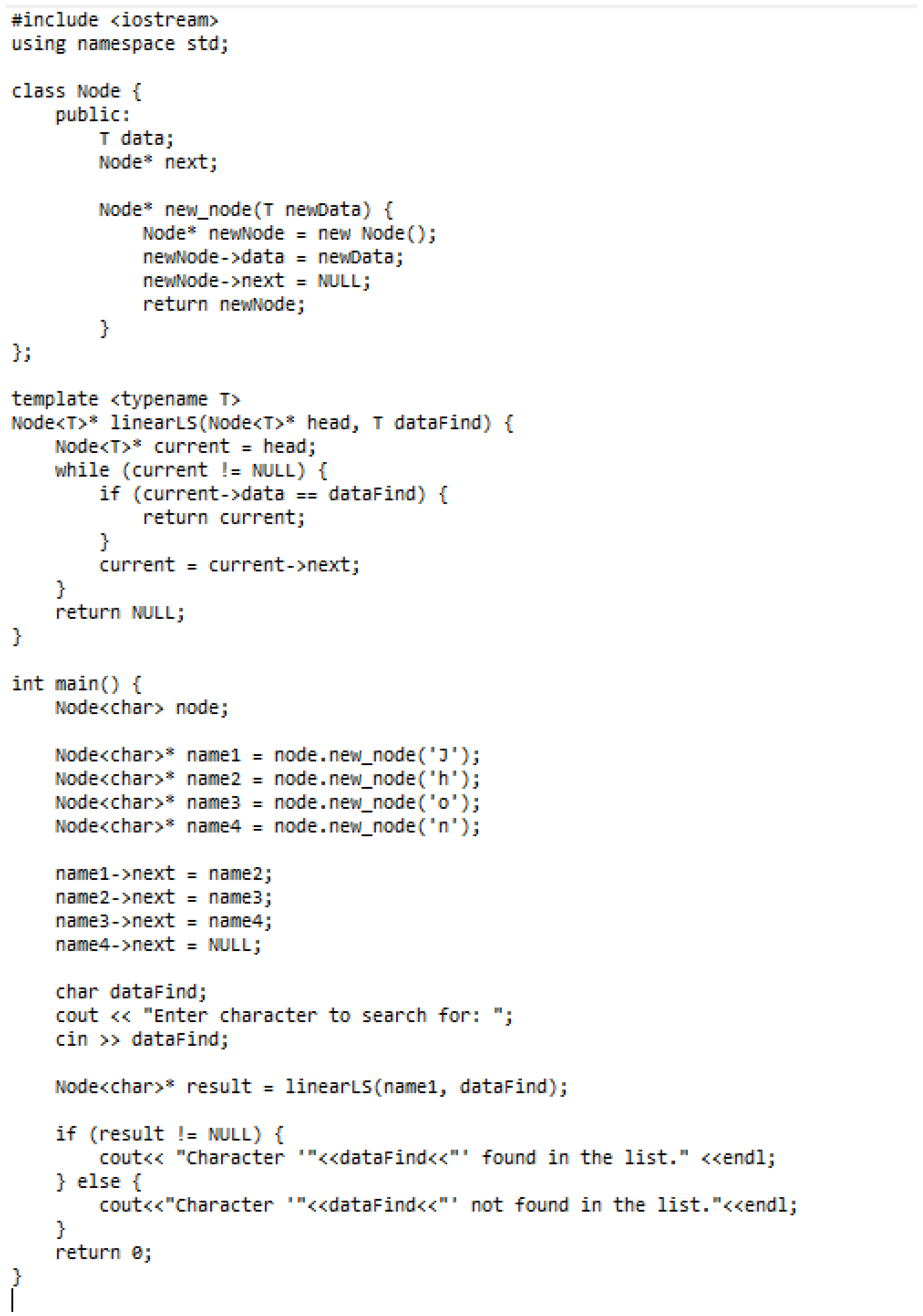
```
/tmp/a0iYmE60Dt.o
Enter number to search: 1
Number found at index: 48


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

Observation

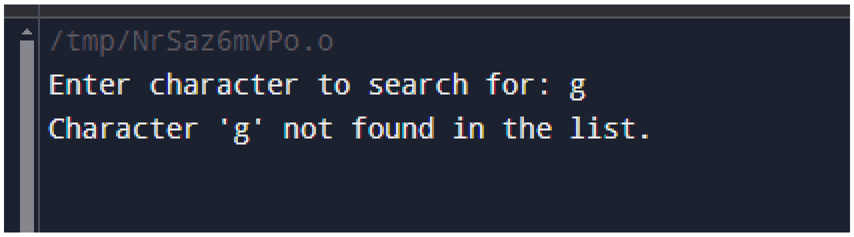
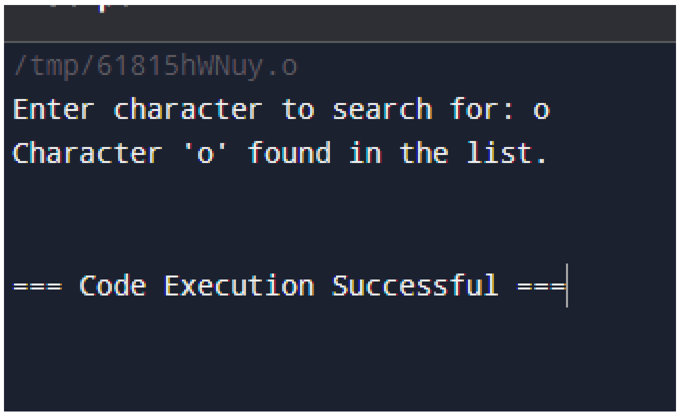
In this program, we are tasked to create a function using the linear search in an array. With the code for generating random numbers into an array, the linear search works by getting a wanted value to be found. This is then checked in the array even if it is not sorted and we output the index.

Table 6-2a. Linear Search for Arrays

Code



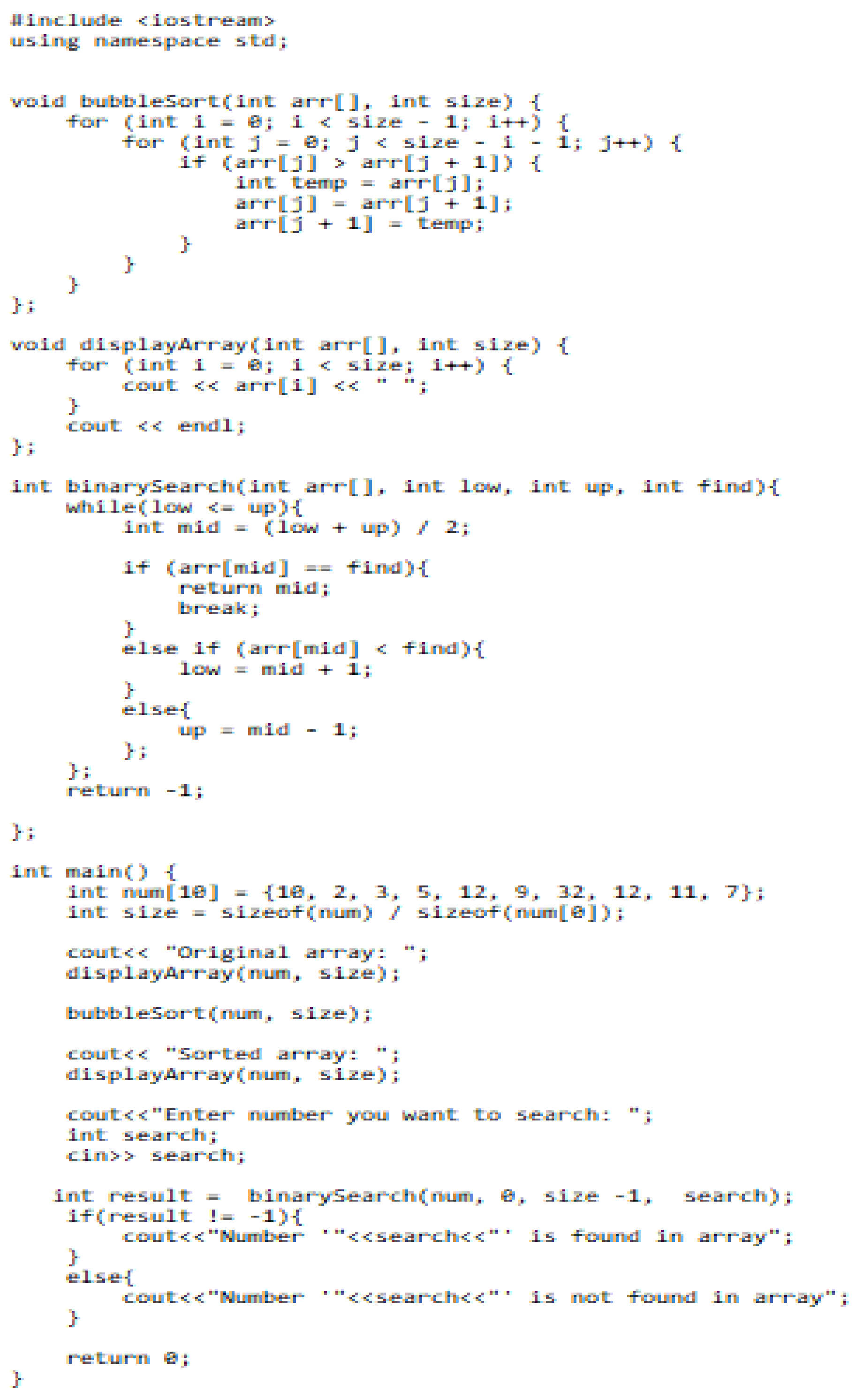
```
#include <iostream>
using namespace std;

class Node {
    public:
        T data;
        Node* next;

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

template <typename T>
Node<T>* linearLS(Node<T>* head, T dataFind) {
    Node<T>* current = head;
    while (current != NULL) {
        if (current->data == dataFind) {
            return current;
        }
        current = current->next;
    }
    return NULL;
}

int main() {
    Node<char> node;

    Node<char>* name1 = node.new_node('J');
    Node<char>* name2 = node.new_node('h');
    Node<char>* name3 = node.new_node('o');
    Node<char>* name4 = node.new_node('n');

    name1->next = name2;
    name2->next = name3;
    name3->next = name4;
    name4->next = NULL;

    char dataFind;
    cout << "Enter character to search for: ";
    cin >> dataFind;

    Node<char>* result = linearLS(name1, dataFind);

    if (result != NULL) {
        cout<< "Character '"<<dataFind<<"' found in the list." <<endl;
    } else {
        cout<<"Character '"<<dataFind<<"' not found in the list."<<endl;
    }
    return 0;
}
```

Output	
	```
	/tmp/NrSaz6mvPo.o
	Enter character to search for: g
	Character 'g' not found in the list.
	```
	 
	```
	/tmp/61815hWNuy.o
	Enter character to search for: o
	Character 'o' found in the list.


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

Observation	In this program, it needs a linked list containing each character of my first name. I used a class for initializing the linked list. Afterwards I declare the value for each node. With the use of a linear search, I need to get a user input for the character to be searched. It checks if the user input can be found in the values of the nodes.

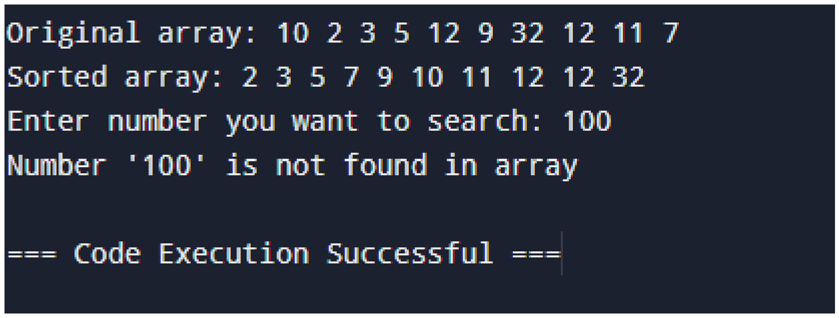
Table 6-2b. Linear Search for Linked List

Code



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

void bubbleSort(int arr[], int size) {
    for (int i = 0; i < size - 1; i++) {
        for (int j = 0; j < size - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
};

void displayArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
};

int binarySearch(int arr[], int low, int up, int find){
    while(low <= up){
        int mid = (low + up) / 2;

        if (arr[mid] == find){
            return mid;
            break;
        }
        else if (arr[mid] < find){
            low = mid + 1;
        }
        else{
            up = mid - 1;
        };
    };
    return -1;

};

int main() {
    int num[10] = {10, 2, 3, 5, 12, 9, 32, 12, 11, 7};
    int size = sizeof(num) / sizeof(num[0]);

    cout<< "Original array: ";
    displayArray(num, size);

    bubbleSort(num, size);

    cout<< "Sorted array: ";
    displayArray(num, size);

    cout<<"Enter number you want to search: ";
    int search;
    cin>> search;

   int result =  binarySearch(num, 0, size -1,  search);
    if(result != -1){
        cout<<"Number '"<<search<<"' is found in array";
    }
    else{
        cout<<"Number '"<<search<<"' is not found in array";
    }

    return 0;
}
```

Output

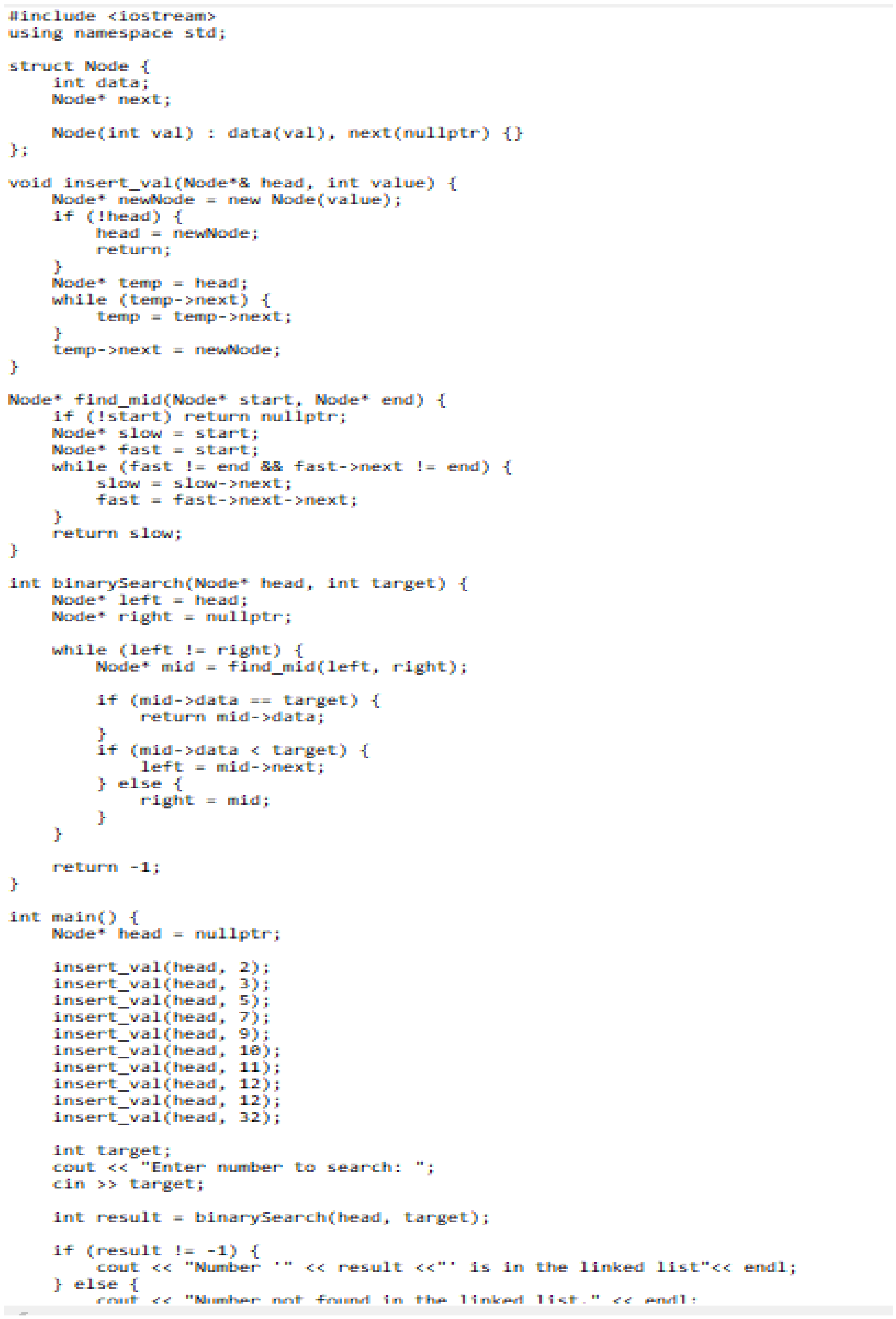


```
Original array: 10 2 3 5 12 9 32 12 11 7
Sorted array: 2 3 5 7 9 10 11 12 12 32
Enter number you want to search: 100
Number '100' is not found in array

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

Observation	For this program I created an array containing random numbers and used a sorting function in order to use the binary search. I was able to effectively use the binary search by getting a user input and then checking whether the data can be found in the sorted array.

Table 6-3a. Binary Search for Arrays

Code



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

struct Node {
    int data;
    Node* next;

    Node(int val) : data(val), next(nullptr) {}
};

void insert_val(Node*& head, int value) {
    Node* newNode = new Node(value);
    if (!head) {
        head = newNode;
        return;
    }
    Node* temp = head;
    while (temp->next) {
        temp = temp->next;
    }
    temp->next = newNode;
}

Node* find_mid(Node* start, Node* end) {
    if (!start) return nullptr;
    Node* slow = start;
    Node* fast = start;
    while (fast != end && fast->next != end) {
        slow = slow->next;
        fast = fast->next->next;
    }
    return slow;
}

int binarySearch(Node* head, int target) {
    Node* left = head;
    Node* right = nullptr;

    while (left != right) {
        Node* mid = find_mid(left, right);

        if (mid->data == target) {
            return mid->data;
        }
        if (mid->data < target) {
            left = mid->next;
        } else {
            right = mid;
        }
    }

    return -1;
}

int main() {
    Node* head = nullptr;

    insert_val(head, 2);
    insert_val(head, 3);
    insert_val(head, 5);
    insert_val(head, 7);
    insert_val(head, 9);
    insert_val(head, 10);
    insert_val(head, 11);
    insert_val(head, 12);
    insert_val(head, 12);
    insert_val(head, 32);

    int target;
    cout << "Enter number to search: ";
    cin >> target;

    int result = binarySearch(head, target);

    if (result != -1) {
        cout << "Number '" << result <<"' is in the linked list"<< endl;
    } else {
        cout << "Number not found in the linked list." << endl;
```

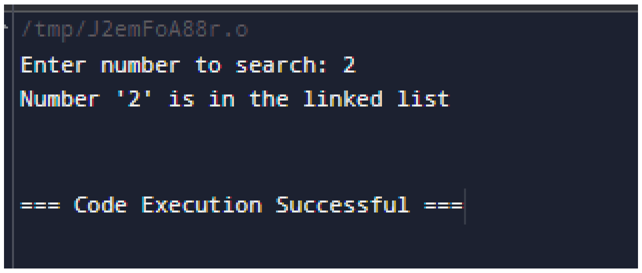
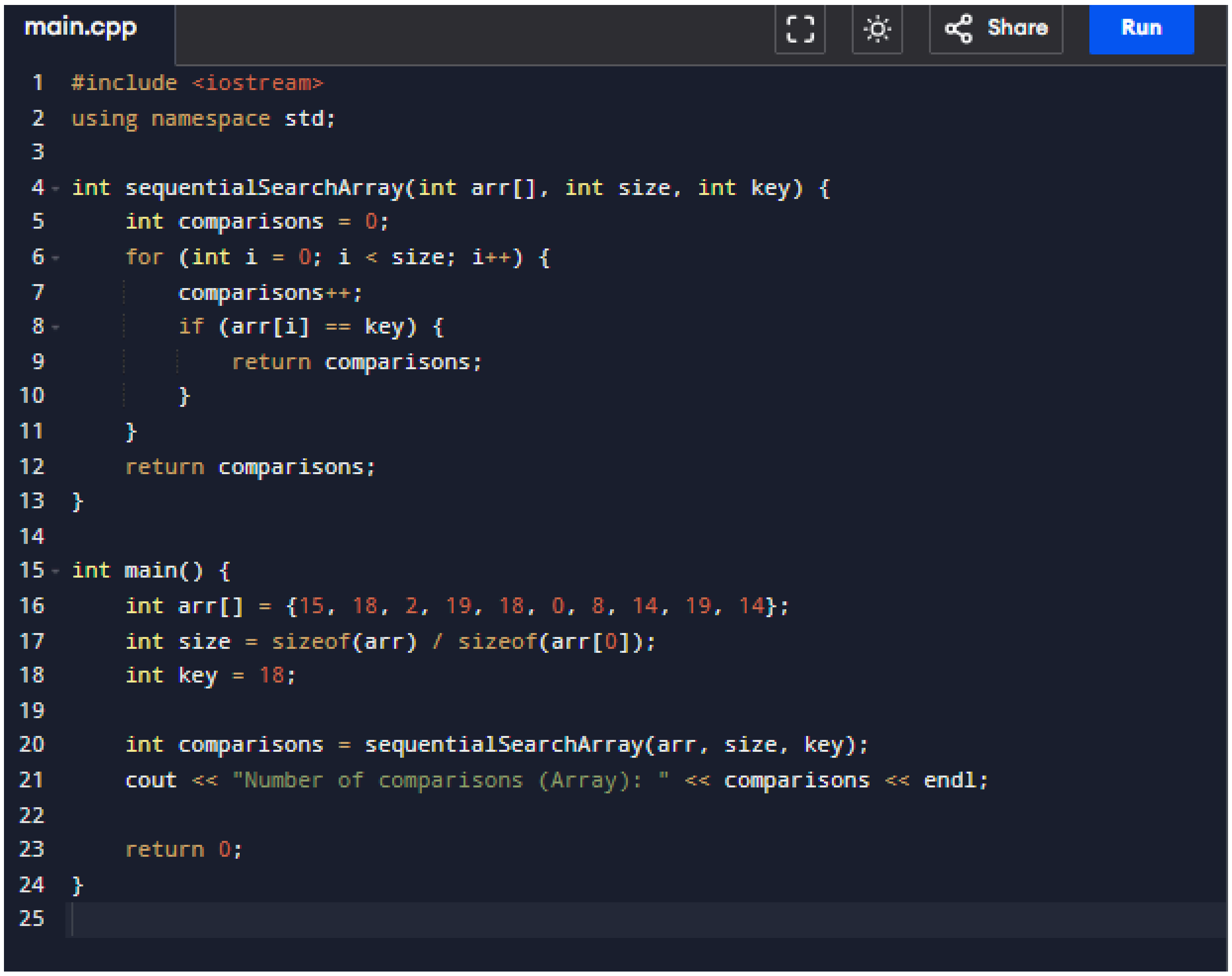
Output	 `/tmp/J2emFoA88r.o` `Enter number to search: 2` `Number '2' is in the linked list` `=== Code Execution Successful ===`
Observation	In using the binary search for a linked list I first check for the middle node of the list. This is done by using two pointers to check the midpoint of the list. After that, I used that middle node in order to implement the binary search and then get a user input to check if the value can be found in the list.

Table 6-3b. Binary Search for Linked List

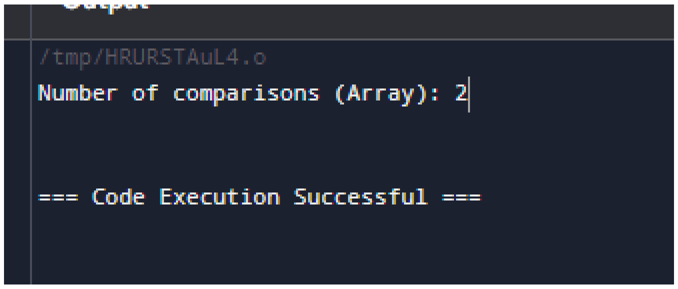
7. Supplementary Activity

Code



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

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

int main() {
    int arr[] = {15, 18, 2, 19, 18, 0, 8, 14, 19, 14};
    int size = sizeof(arr) / sizeof(arr[0]);
    int key = 18;

    int comparisons = sequentialSearchArray(arr, size, key);
    cout << "Number of comparisons (Array): " << comparisons << endl;

    return 0;
}
```

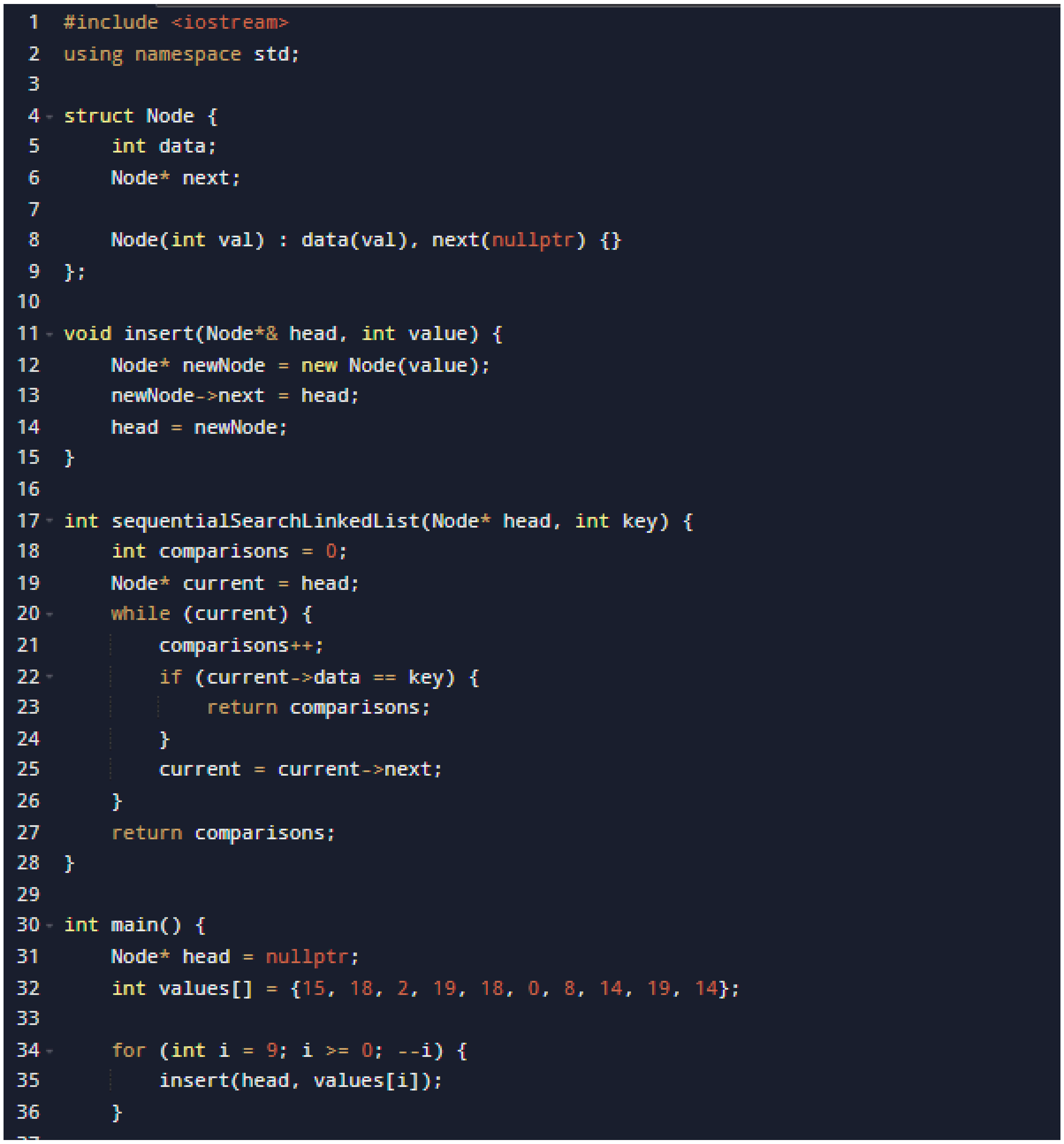
Output



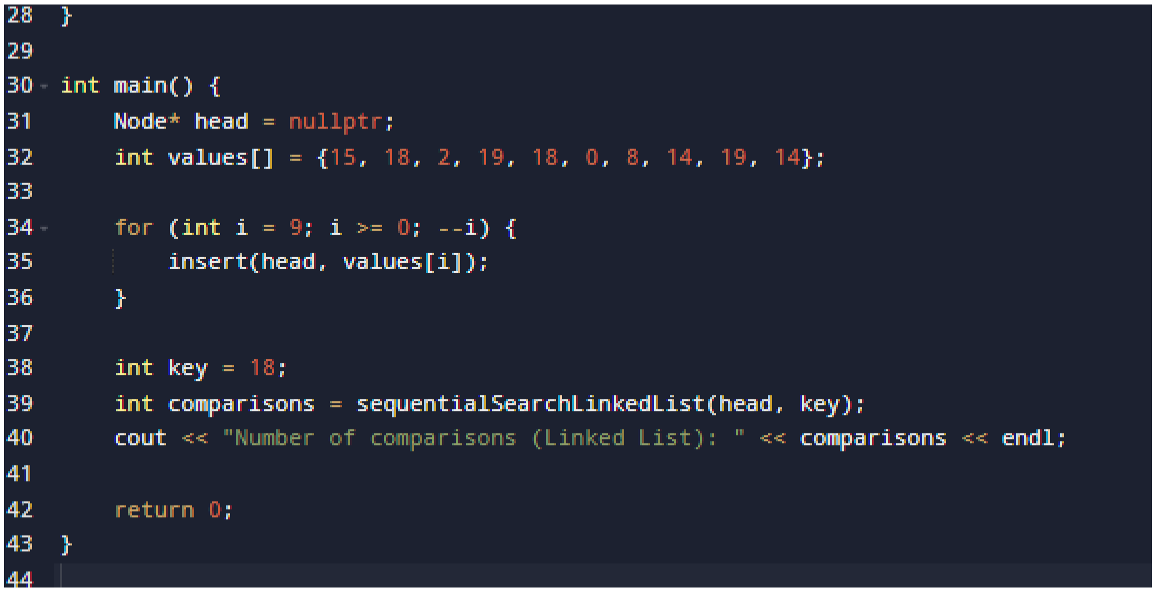
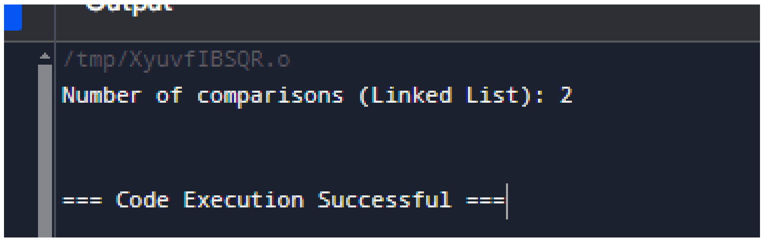
```
/tmp/HRURSTAuL4.o
Number of comparisons (Array): 2


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

Problem 1 Array

Code



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

struct Node {
    int data;
    Node* next;

    Node(int val) : data(val), next(nullptr) {}
};

void insert(Node*& head, int value) {
    Node* newNode = new Node(value);
    newNode->next = head;
    head = newNode;
}

int sequentialSearchLinkedList(Node* head, int key) {
    int comparisons = 0;
    Node* current = head;
    while (current) {
        comparisons++;
        if (current->data == key) {
            return comparisons;
        }
        current = current->next;
    }
    return comparisons;
}

int main() {
    Node* head = nullptr;
    int values[] = {15, 18, 2, 19, 18, 0, 8, 14, 19, 14};

    for (int i = 9; i >= 0; --i) {
        insert(head, values[i]);
    }
```

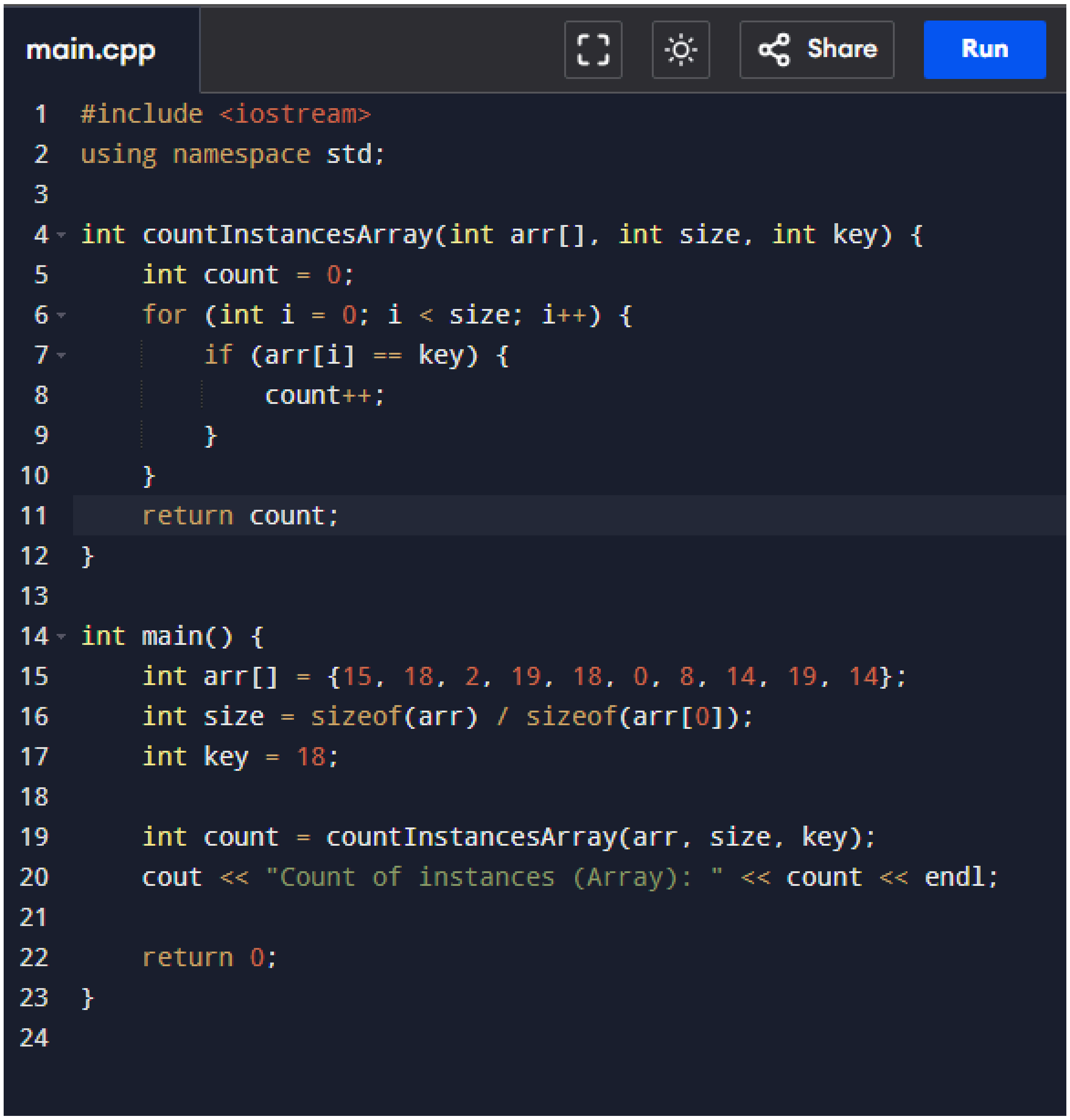
	
Output	

```
}

int main() {
    Node* head = nullptr;
    int values[] = {15, 18, 2, 19, 18, 0, 8, 14, 19, 14};

    for (int i = 9; i >= 0; --i) {
        insert(head, values[i]);
    }

    int key = 18;
    int comparisons = sequentialSearchLinkedList(head, key);
    cout << "Number of comparisons (Linked List): " << comparisons << endl;

    return 0;
}
```

```
/tmp/XyuvfIBSQR.o
Number of comparisons (Linked List): 2


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

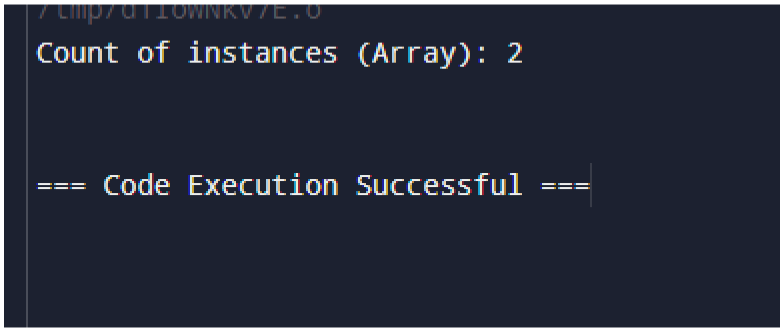
Problem 1 Linked List

Code



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

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

int main() {
    int arr[] = {15, 18, 2, 19, 18, 0, 8, 14, 19, 14};
    int size = sizeof(arr) / sizeof(arr[0]);
    int key = 18;

    int count = countInstancesArray(arr, size, key);
    cout << "Count of instances (Array): " << count << endl;

    return 0;
}
```

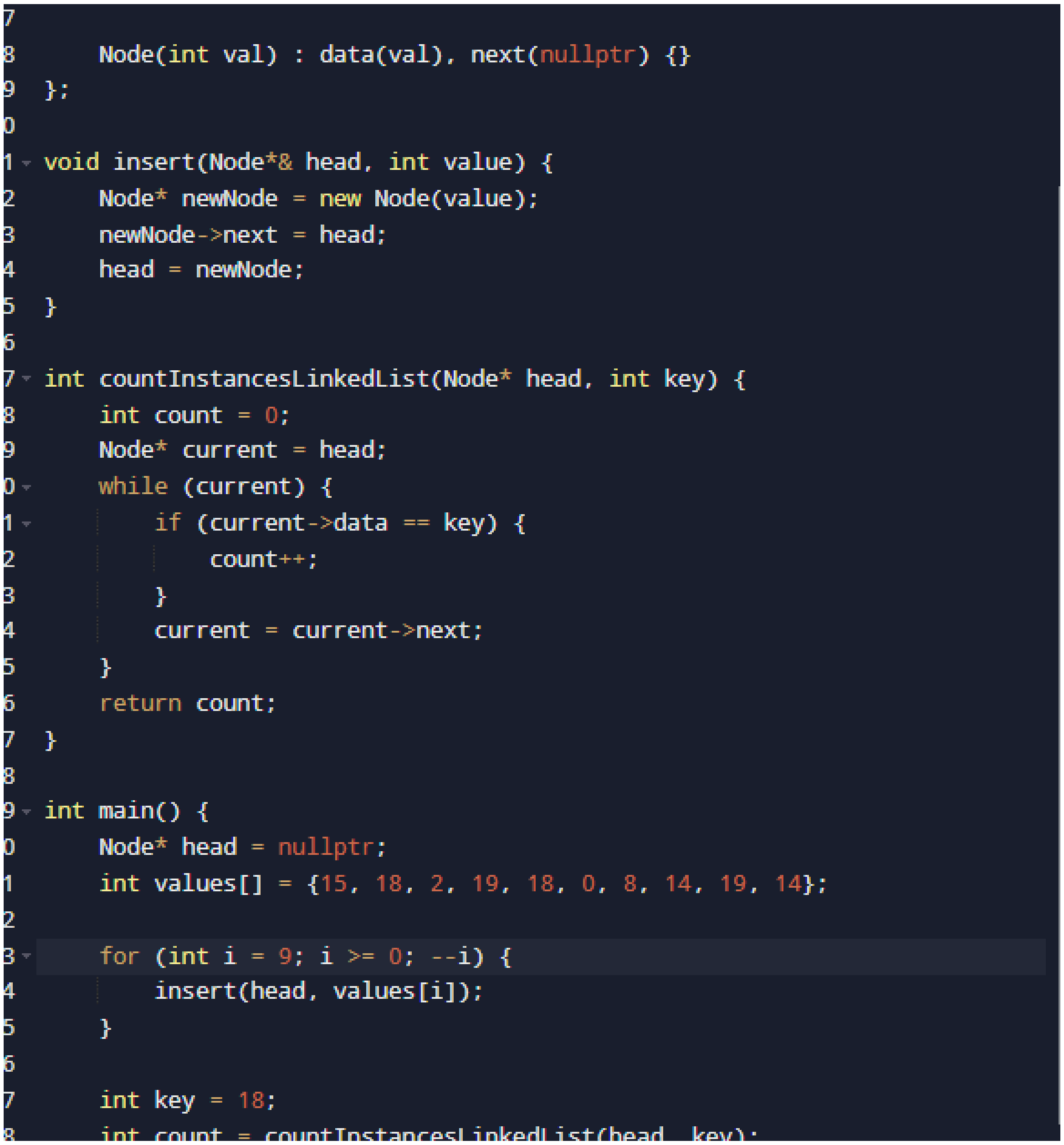
Output



```
/tmp/dTlOWNkv7E.o
Count of instances (Array): 2


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

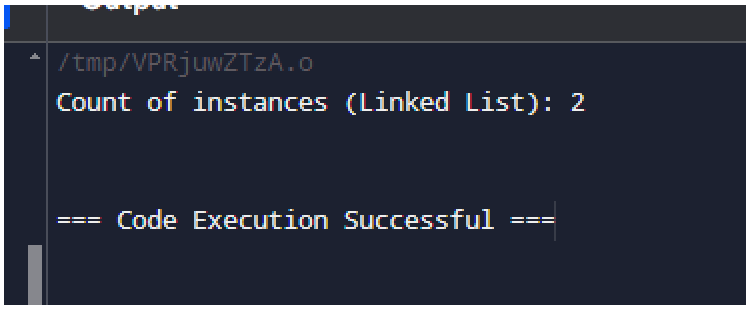
Problem 2 Array

Code



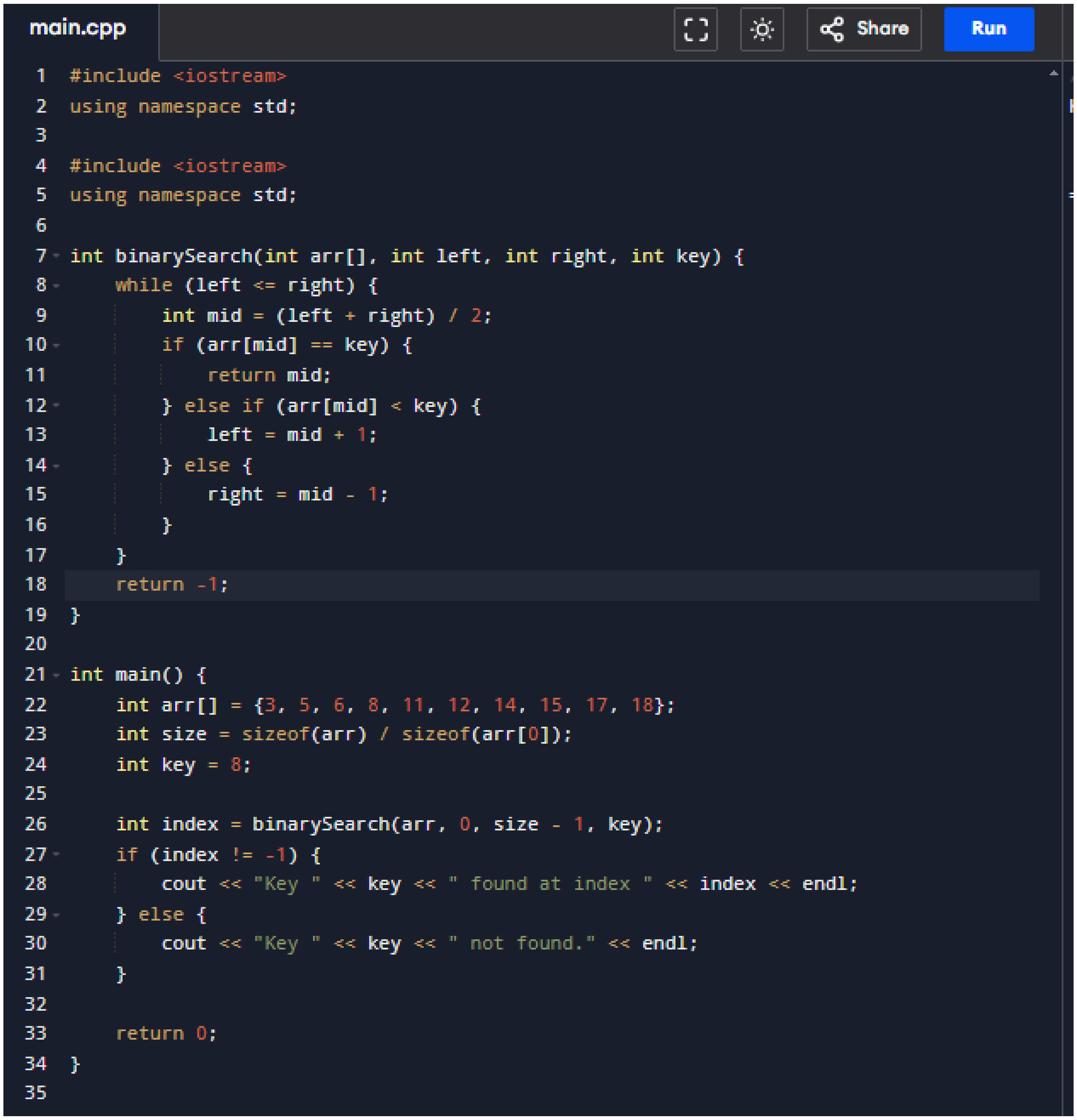
```
    Node(int val) : data(val), next(nullptr) {}
};

void insert(Node*& head, int value) {
    Node* newNode = new Node(value);
    newNode->next = head;
    head = newNode;
}

int countInstancesLinkedList(Node* head, int key) {
    int count = 0;
    Node* current = head;
    while (current) {
        if (current->data == key) {
            count++;
        }
        current = current->next;
    }
    return count;
}

int main() {
    Node* head = nullptr;
    int values[] = {15, 18, 2, 19, 18, 0, 8, 14, 19, 14};

    for (int i = 9; i >= 0; --i) {
        insert(head, values[i]);
    }

    int key = 18;
    int count = countInstancesLinkedList(head, key);
```

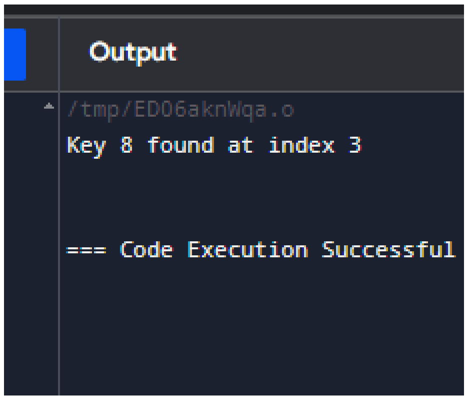
Output



```
/tmp/VPRjuwZTzA.o
Count of instances (Linked List): 2


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

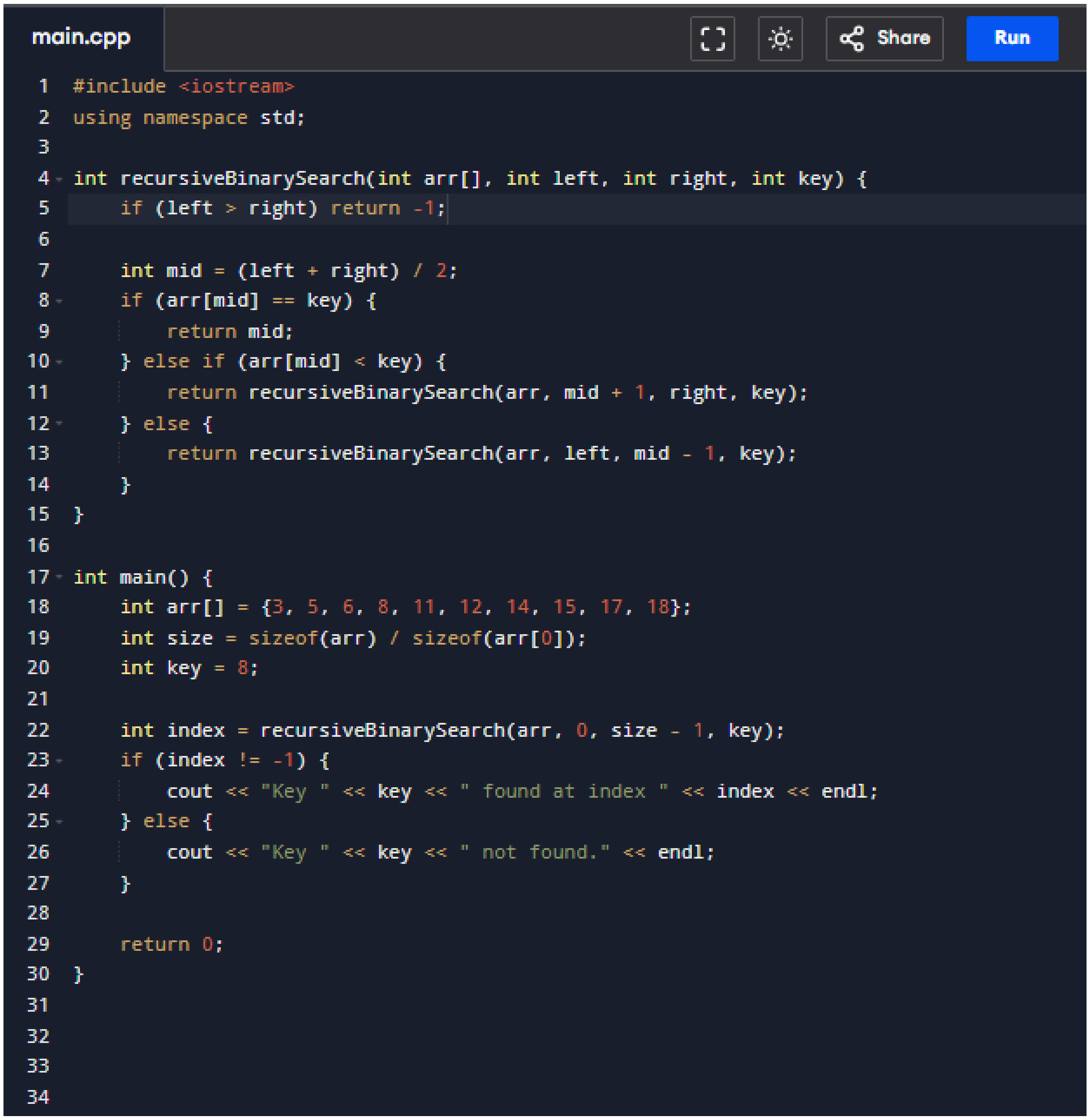
Problem 2 Linked List

Code	 main.cpp Share Run
	```cpp
	#include <iostream>
	using namespace std;

	#include <iostream>
	using namespace std;

	int binarySearch(int arr[], int left, int right, int key) {
	    while (left <= right) {
	        int mid = (left + right) / 2;
	        if (arr[mid] == key) {
	            return mid;
	        } else if (arr[mid] < key) {
	            left = mid + 1;
	        } else {
	            right = mid - 1;
	        }
	    }
	    return -1;
	}

	int main() {
	    int arr[] = {3, 5, 6, 8, 11, 12, 14, 15, 17, 18};
	    int size = sizeof(arr) / sizeof(arr[0]);
	    int key = 8;

	    int index = binarySearch(arr, 0, size - 1, key);
	    if (index != -1) {
	        cout << "Key " << key << " found at index " << index << endl;
	    } else {
	        cout << "Key " << key << " not found." << endl;
	    }

	    return 0;
	}
	```

Output	 Output
	```
	/tmp/ED06aknWqa.o
	Key 8 found at index 3


	=== Code Execution Successful
	```

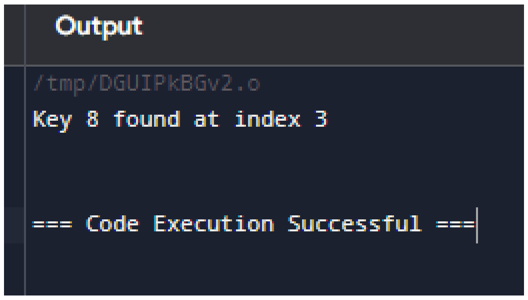
Problem 3

Code



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

int recursiveBinarySearch(int arr[], int left, int right, int key) {
    if (left > right) return -1;

    int mid = (left + right) / 2;
    if (arr[mid] == key) {
        return mid;
    } else if (arr[mid] < key) {
        return recursiveBinarySearch(arr, mid + 1, right, key);
    } else {
        return recursiveBinarySearch(arr, left, mid - 1, key);
    }
}

int main() {
    int arr[] = {3, 5, 6, 8, 11, 12, 14, 15, 17, 18};
    int size = sizeof(arr) / sizeof(arr[0]);
    int key = 8;

    int index = recursiveBinarySearch(arr, 0, size - 1, key);
    if (index != -1) {
        cout << "Key " << key << " found at index " << index << endl;
    } else {
        cout << "Key " << key << " not found." << endl;
    }

    return 0;
}



```

Output



```
/tmp/DGUIPkBGv2.o
Key 8 found at index 3


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

8. Conclusion

After completing this practical exercise, I now know a lot more about the various kinds of sorting techniques that employ arrays and linked lists. We get to use arrays and linked lists to put the various searching and sorting strategies into

practice. Before employing binary search to locate the precise necessary values in arrays and linked lists, we primarily used bubble sort. Since these sorting algorithms were new to my code, applying them was challenging. Another novel idea in traversal or searching was the binary search code in a linked list. In the end, I gained a great deal of new knowledge on how to implement arrays and linked lists using the two kinds of searching.
9. Assessment Rubric