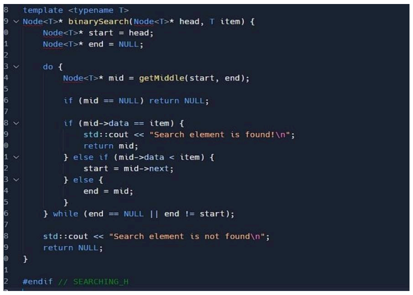
Activity No. 6

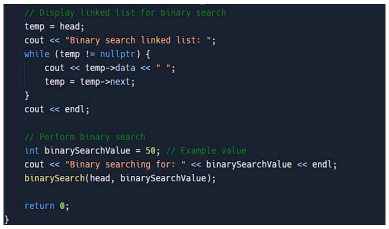
SEARCHING TECHNIQUES

Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: OCTOBER 15, 2024**
Section: CPE21S1	**Date Submitted: OCTOBER 15, 2024**
Name(s): GASPAR, AARON ROWEN O.	**Instructor: MS. MARIA RIZETTE SAYO**

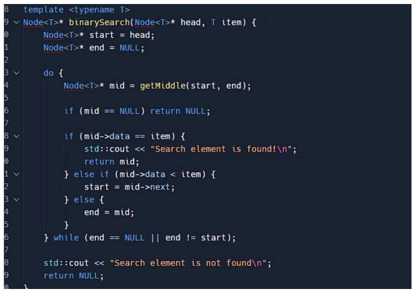
6. Output



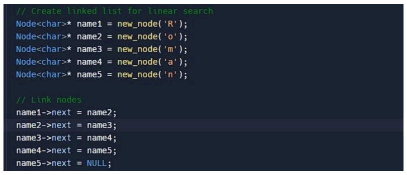
```
template <typename T>
Node<T>* binarySearch(Node<T>* head, T item) {
    Node<T>* start = head;
    Node<T>* end = NULL;

    do {
        Node<T>* mid = getMiddle(start, end);

        if (mid == NULL) return NULL;

        if (mid->data == item) {
            std::cout << "Search element is found!\n";
            return mid;
        } else if (mid->data < item) {
            start = mid->next;
        } else {
            end = mid;
        }
    } while (end == NULL || end != start);

    std::cout << "Search element is not found\n";
    return NULL;
}

#endif // SEARCHING_H
```



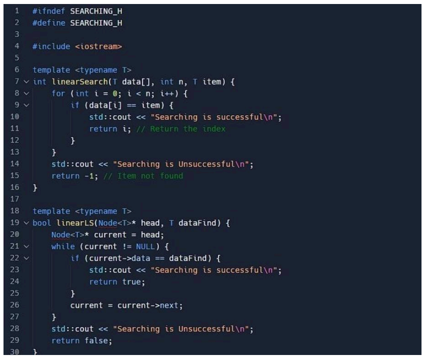
```
    // Display linked list for binary search
    temp = head;
    cout << "Binary search linked list: ";
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;

    // Perform binary search
    int binarySearchValue = 50; // Example value
    cout << "Binary searching for: " << binarySearchValue << endl;
    binarySearch(head, binarySearchValue);

    return 0;
}
```



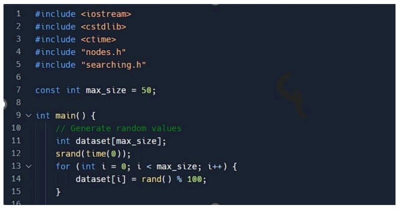
```
template <typename T>
Node<T>* binarySearch(Node<T>* head, T item) {
    Node<T>* start = head;
    Node<T>* end = NULL;

    do {
        Node<T>* mid = getMiddle(start, end);

        if (mid == NULL) return NULL;

        if (mid->data == item) {
            std::cout << "Search element is found!\n";
            return mid;
        } else if (mid->data < item) {
            start = mid->next;
        } else {
            end = mid;
        }
    } while (end == NULL || end != start);

    std::cout << "Search element is not found\n";
    return NULL;
```



```
    // Create linked list for linear search
    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 nodes
    name1->next = name2;
    name2->next = name3;
    name3->next = name4;
    name4->next = name5;
    name5->next = NULL;
```



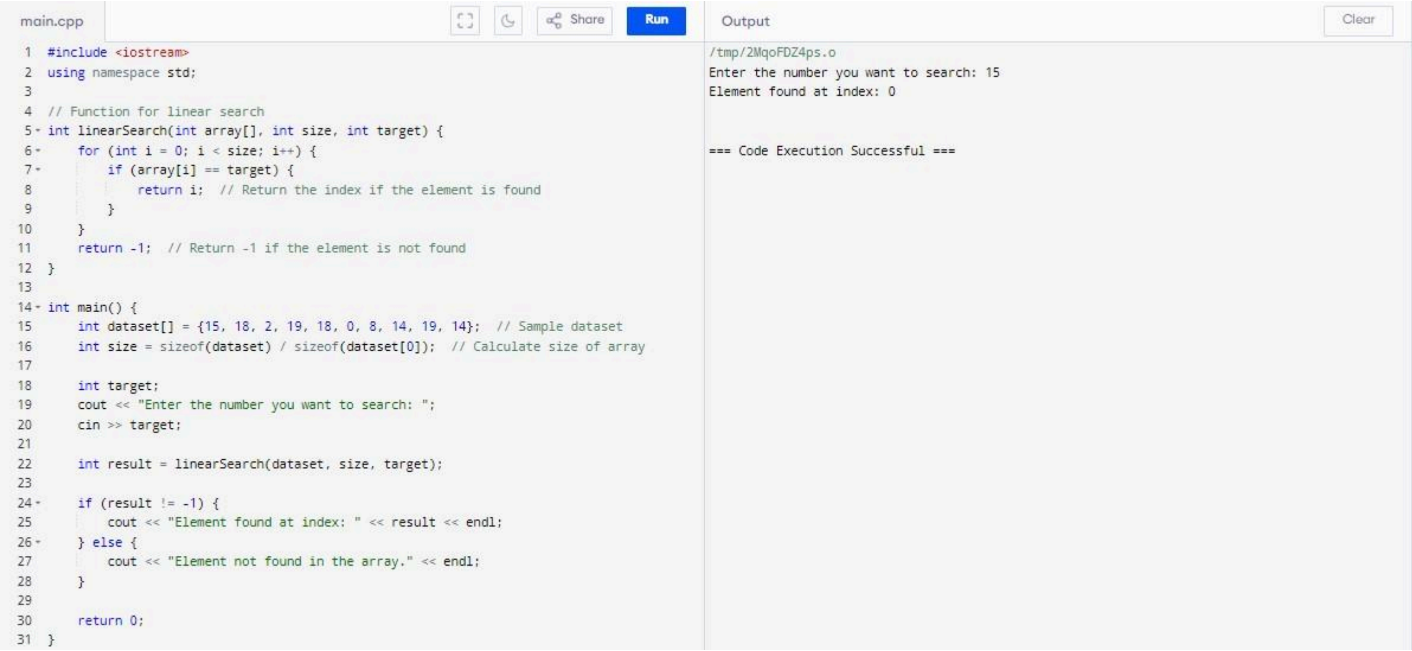
```
#ifndef SEARCHING_H
#define SEARCHING_H

#include <iostream>

template <typename T>
int linearSearch(T data[], int n, T item) {
    for (int i = 0; i < n; i++) {
        if (data[i] == item) {
            std::cout << "Searching is successful\n";
            return i; // Return the index
        }
    }
    std::cout << "Searching is Unsuccessful\n";
    return -1; // Item not found
}

template <typename T>
bool linearLS(Node<T>* head, T dataFind) {
    Node<T>* current = head;
    while (current != NULL) {
        if (current->data == dataFind) {
            std::cout << "Searching is successful\n";
            return true;
        }
        current = current->next;
    }
    std::cout << "Searching is Unsuccessful\n";
    return false;
}
```



```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "nodes.h"
#include "searching.h"

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() % 100;
    }
```

7. Supplementary Activity

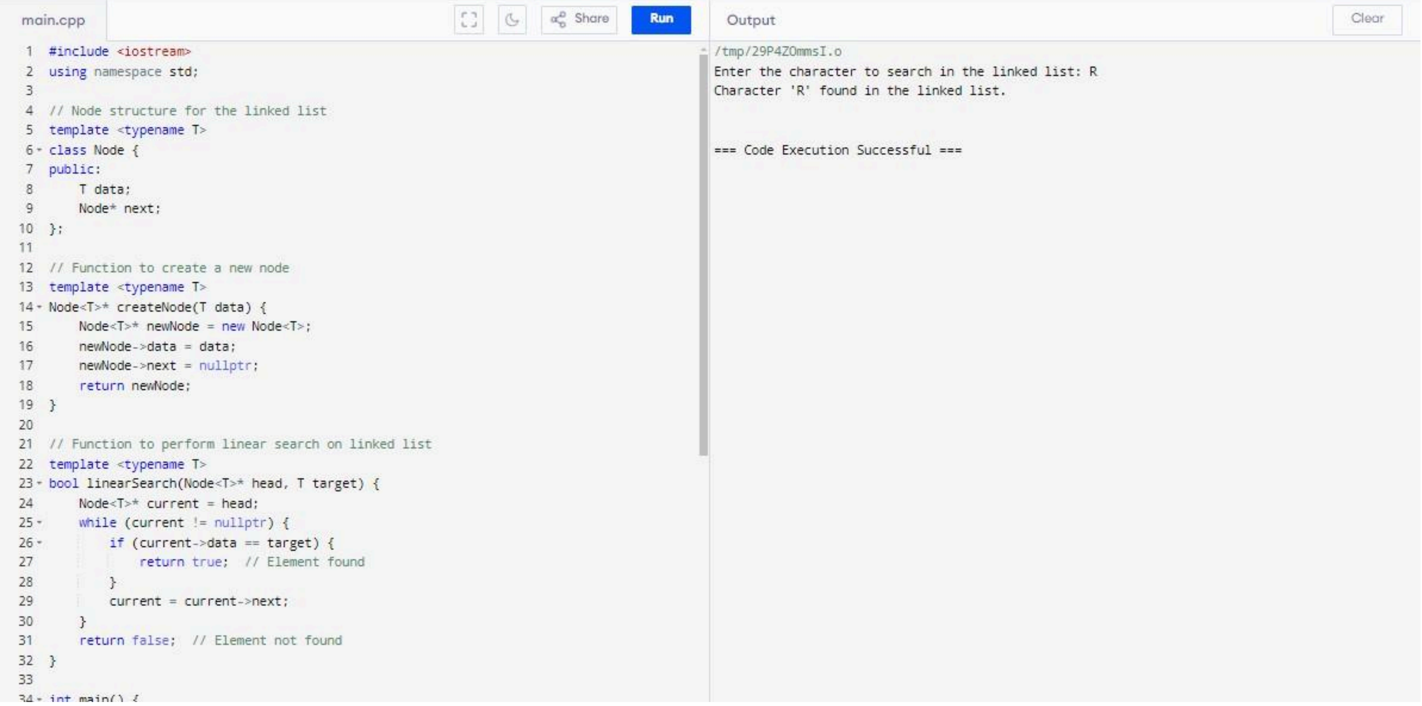
1.



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

// Function for linear search
int linearSearch(int array[], int size, int target) {
    for (int i = 0; i < size; i++) {
        if (array[i] == target) {
            return i;  // Return the index if the element is found
        }
    }
    return -1;  // Return -1 if the element is not found
}

int main() {
    int dataset[] = {15, 18, 2, 19, 18, 0, 8, 14, 19, 14};  // Sample dataset
    int size = sizeof(dataset) / sizeof(dataset[0]);  // Calculate size of array

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

    int result = linearSearch(dataset, size, target);

    if (result != -1) {
        cout << "Element found at index: " << result << endl;
    } else {
        cout << "Element not found in the array." << endl;
    }

    return 0;
}
```

```
/tmp/2MqoFDZ4ps.o
Enter the number you want to search: 15
Element found at index: 0


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

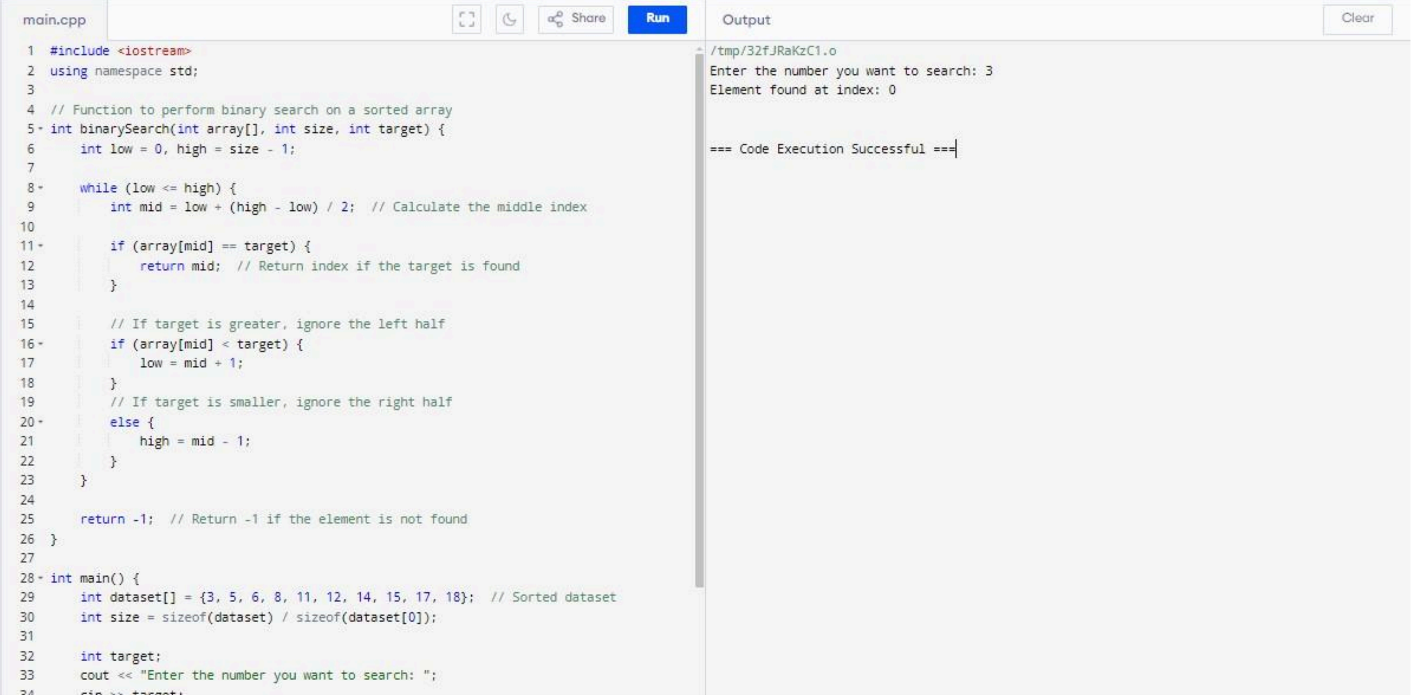
2.



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

// Node structure for the linked list
template <typename T>
class Node {
public:
    T data;
    Node* next;
};

// Function to create a new node
template <typename T>
Node<T>* createNode(T data) {
    Node<T>* newNode = new Node<T>;
    newNode->data = data;
    newNode->next = nullptr;
    return newNode;
}

// Function to perform linear search on linked list
template <typename T>
bool linearSearch(Node<T>* head, T target) {
    Node<T>* current = head;
    while (current != nullptr) {
        if (current->data == target) {
            return true;  // Element found
        }
        current = current->next;
    }
    return false;  // Element not found
}

int main() {
```

```
/tmp/29P4Z0mmsI.o
Enter the character to search in the linked list: R
Character 'R' found in the linked list.


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

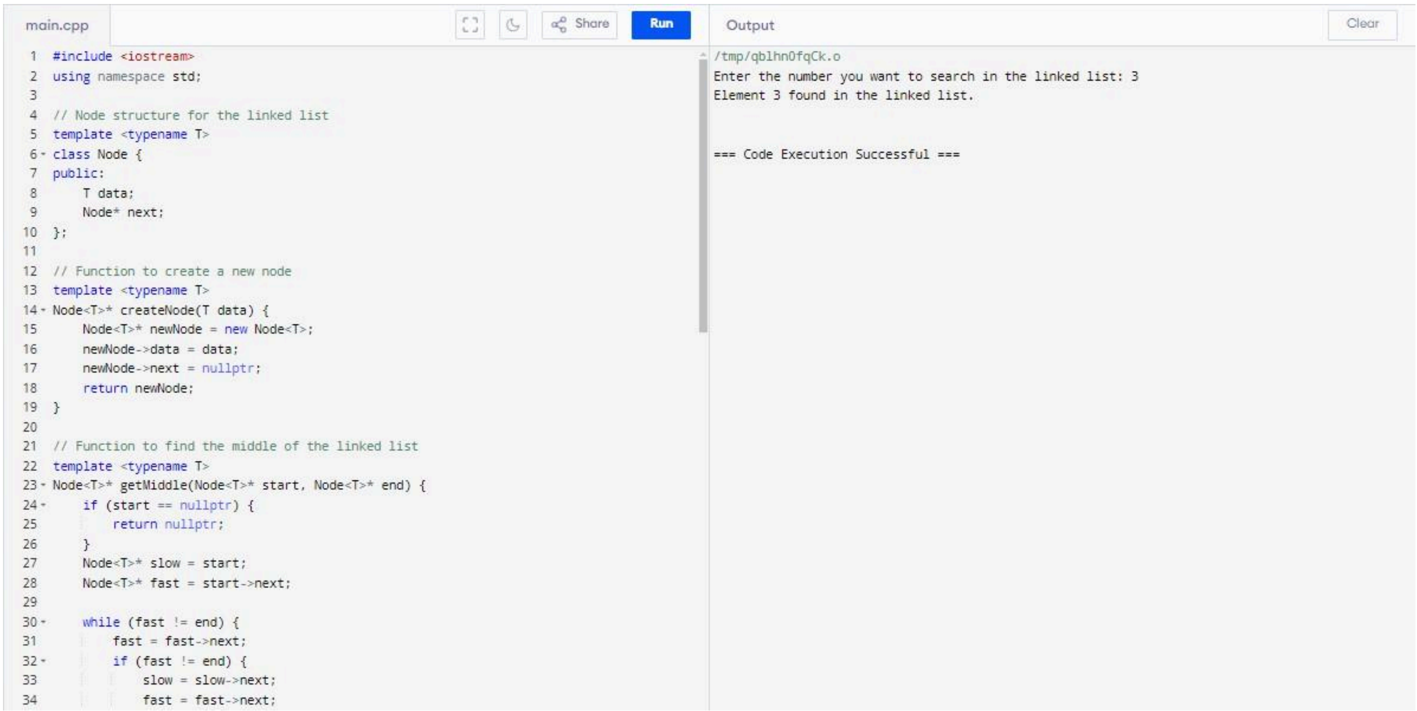
3.



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

// Function to perform binary search on a sorted array
int binarySearch(int array[], int size, int target) {
    int low = 0, high = size - 1;

    while (low <= high) {
        int mid = low + (high - low) / 2;  // Calculate the middle index

        if (array[mid] == target) {
            return mid;  // Return index if the target is found
        }

        // If target is greater, ignore the left half
        if (array[mid] < target) {
            low = mid + 1;
        }
        // If target is smaller, ignore the right half
        else {
            high = mid - 1;
        }
    }

    return -1;  // Return -1 if the element is not found
}

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

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

```
/tmp/32fJRaKzC1.o
Enter the number you want to search: 3
Element found at index: 0


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

4.



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

// Node structure for the linked list
template <typename T>
class Node {
public:
    T data;
    Node* next;
};

// Function to create a new node
template <typename T>
Node<T>* createNode(T data) {
    Node<T>* newNode = new Node<T>;
    newNode->data = data;
    newNode->next = nullptr;
    return newNode;
}

// Function to find the middle of the linked list
template <typename T>
Node<T>* getMiddle(Node<T>* start, Node<T>* end) {
    if (start == nullptr) {
        return nullptr;
    }
    Node<T>* slow = start;
    Node<T>* fast = start->next;

    while (fast != end) {
        fast = fast->next;
        if (fast != end) {
            slow = slow->next;
            fast = fast->next;
```

```
/tmp/qblhn0fqCk.o
Enter the number you want to search in the linked list: 3
Element 3 found in the linked list.


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

8. Conclusion

In this activity, I learned the implementation of two fundamental searching techniques: linear search and binary search. I also explored how these algorithms work differently on arrays and linked lists, and how the complexity of search operations changes based on the data structure. I gained experience in writing C++ code to implement both search algorithms and used pseudocode to understand the step-by-step process. Additionally, I learned how to adapt search techniques to different data structures like linked lists, which lack direct access to elements. I believe I performed well in this activity, particularly in understanding and implementing the search algorithms in C++. The pseudocode was useful, and the structure of the tasks allowed for a gradual increase in complexity. I successfully adapted the algorithms to different scenarios and handled linked lists effectively, which was a new challenge compared to arrays.

9. Assessment Rubric