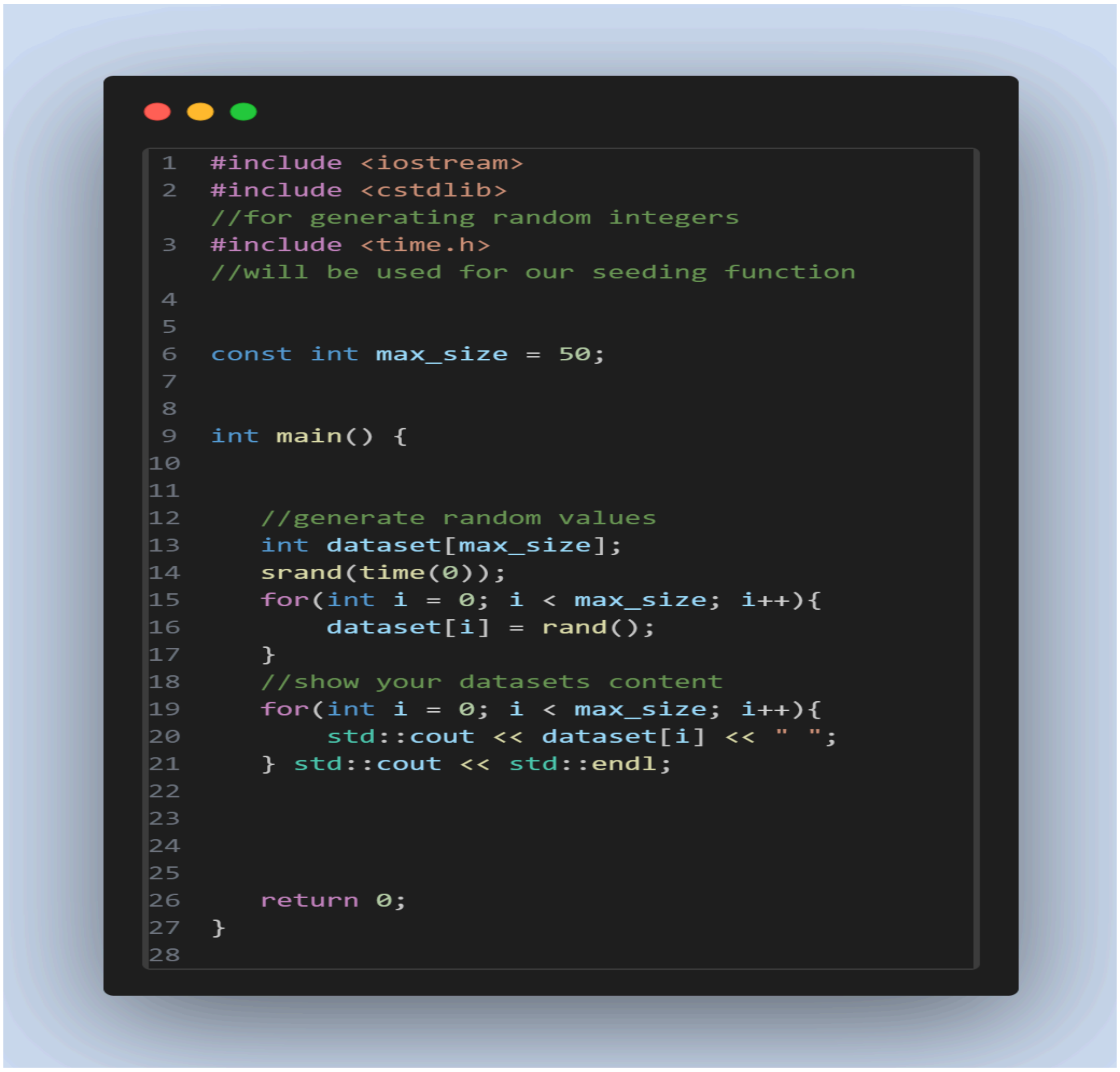
Hands-on Activity 6.1	
Searching Techniques	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed: 16/9/25**
Section: CPE21S4	**Date Submitted: 16/9/25**
Name(s): Francis Nikko I. Andoy	**Instructor:** Jimlord Quejado

6. Output

SCREEN SHOTS:



```cpp
#include <iostream>
#include <cstdlib>
//for generating random integers
#include <time.h>
//will be used for our seeding function


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 your datasets content
    for(int i = 0; i < max_size; i++){
        std::cout << dataset[i] << " ";
    } std::cout << std::endl;




    return 0;
}

```

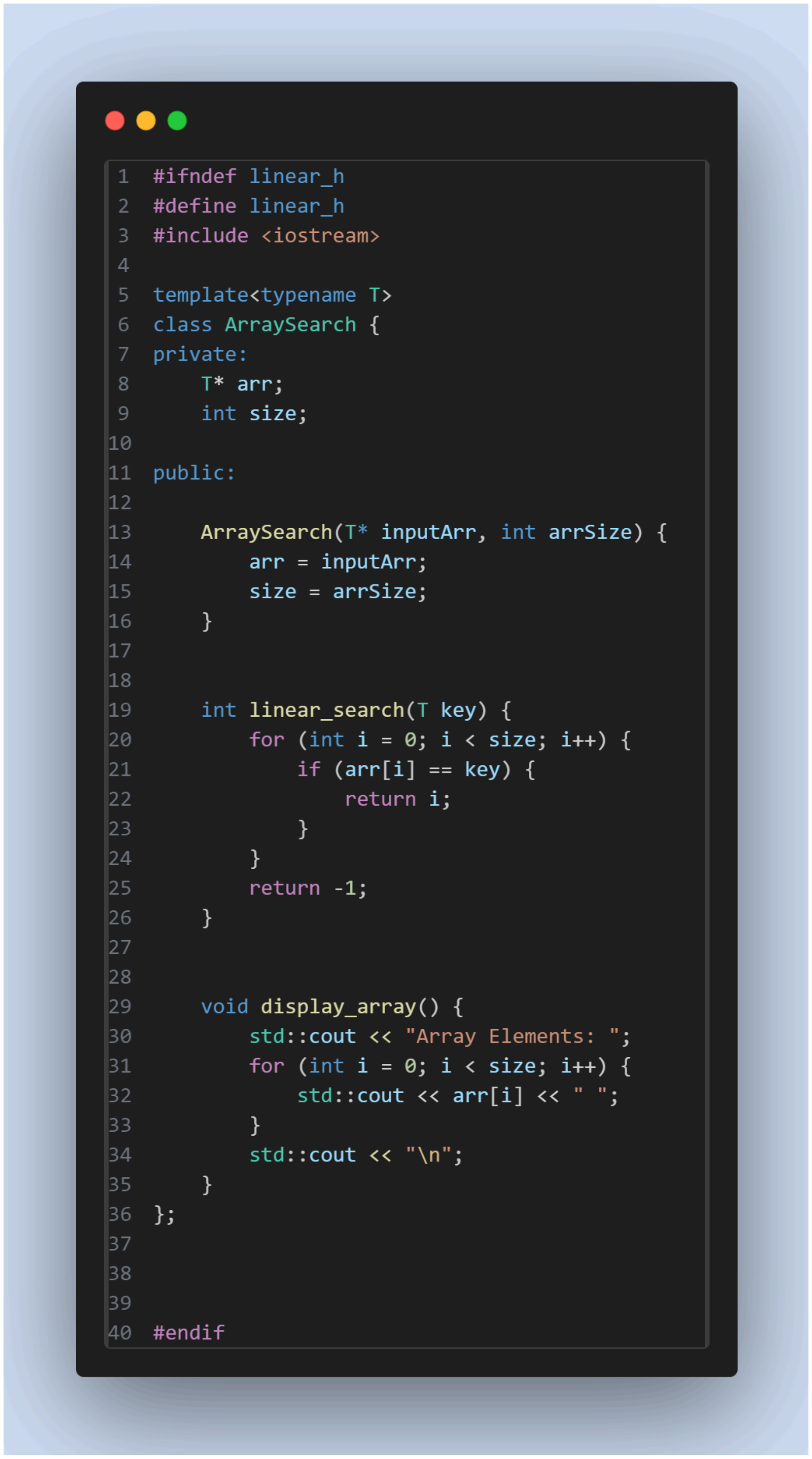
OBSERVATION:

I've observed that it generated a random numbers, using the library cstdlib. Using for loops to put the random numbers inside of the array. Although the max sizes of the array is just 50 and we can't change that aside from the fixed size of the array. We also used the const which is a type qualifier that fixed a size or a value of any variable that has it.

Table 6-1. Data Generated and Observations.

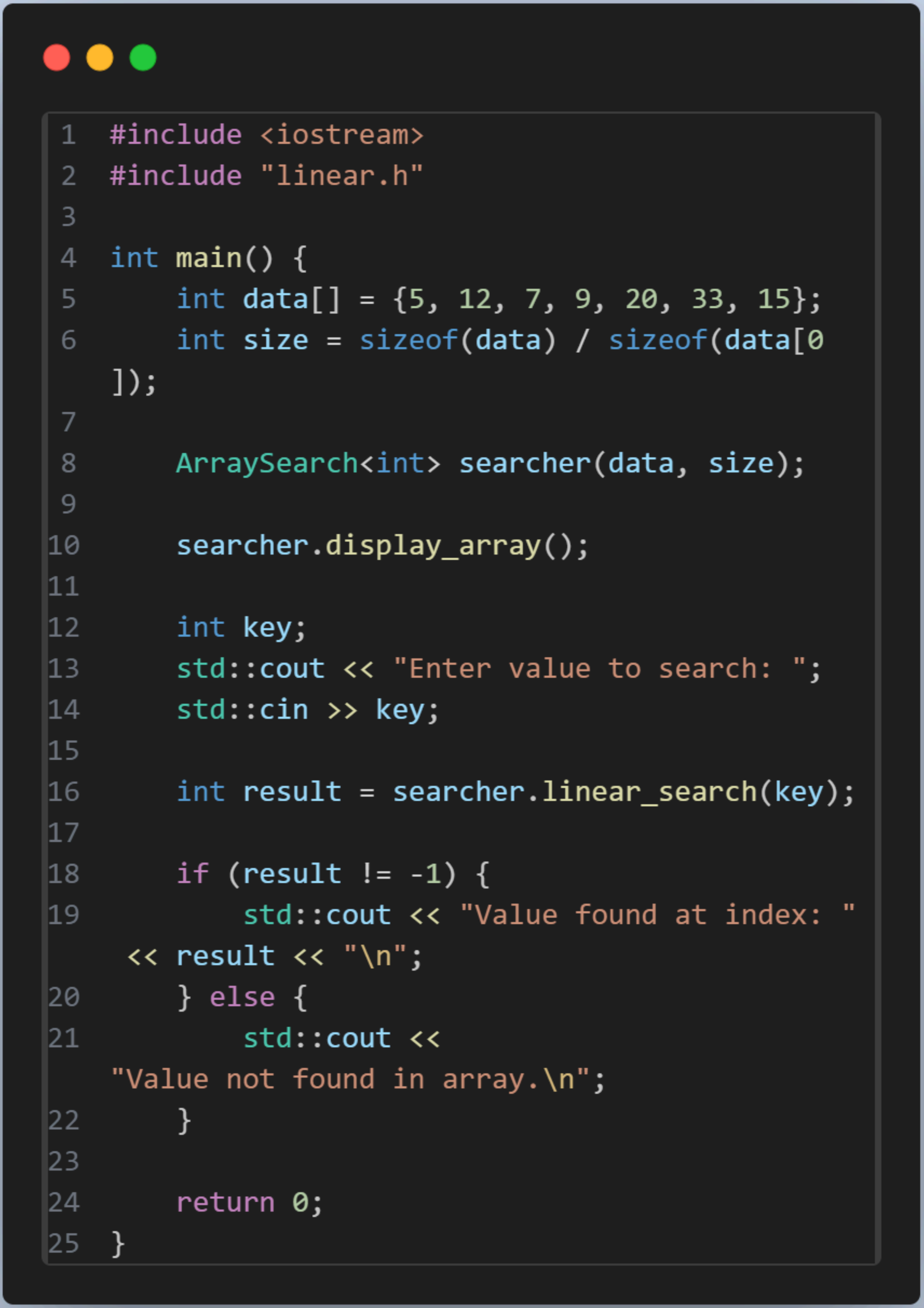
CODE:

HEADER FILE:



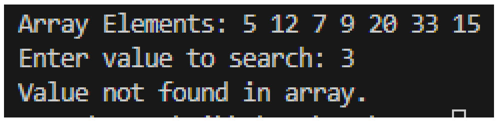
```cpp
#ifndef linear_h
#define linear_h
#include <iostream>

template<typename T>
class ArraySearch {
private:
    T* arr;
    int size;

public:

    ArraySearch(T* inputArr, int arrSize) {
        arr = inputArr;
        size = arrSize;
    }


    int linear_search(T key) {
        for (int i = 0; i < size; i++) {
            if (arr[i] == key) {
                return i;
            }
        }
        return -1;
    }


    void display_array() {
        std::cout << "Array Elements: ";
        for (int i = 0; i < size; i++) {
            std::cout << arr[i] << " ";
        }
        std::cout << "\n";
    }
};



#endif
```

MAIN:

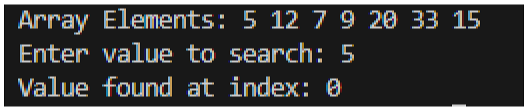


```cpp
#include <iostream>
#include "linear.h"

int main() {
    int data[] = {5, 12, 7, 9, 20, 33, 15};
    int size = sizeof(data) / sizeof(data[0
]);

    ArraySearch<int> searcher(data, size);

    searcher.display_array();

    int key;
    std::cout << "Enter value to search: ";
    std::cin >> key;

    int result = searcher.linear_search(key);

    if (result != -1) {
        std::cout << "Value found at index: "
 << result << "\n";
    } else {
        std::cout <<
"Value not found in array.\n";
    }

    return 0;
}
```

OUTPUT:



```
Array Elements: 5 12 7 9 20 33 15
Enter value to search: 3
Value not found in array.
```



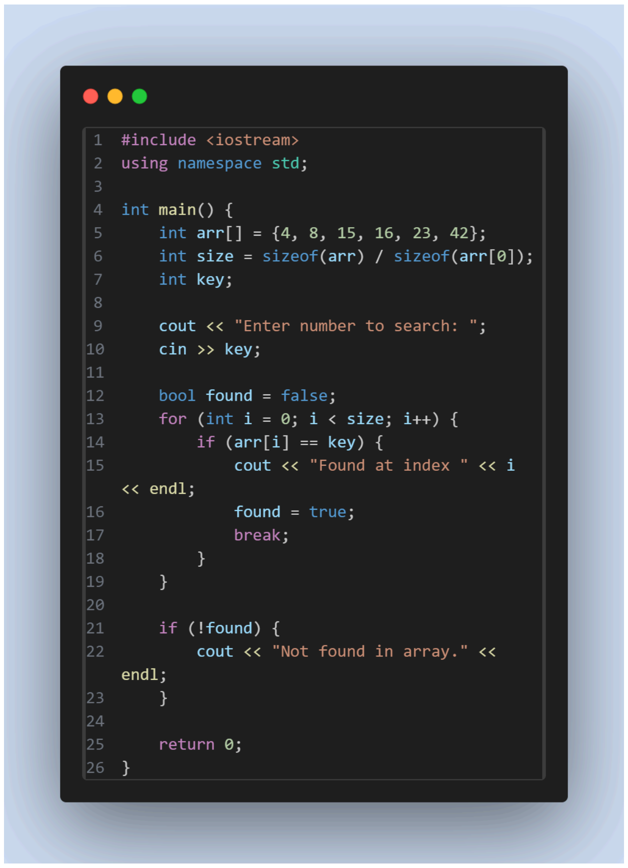
```
Array Elements: 5 12 7 9 20 33 15
Enter value to search: 5
Value found at index: 0
```

OBSERVATION:

Here I first implemented a template to make the code generic, then made a class that has a T pointer and size, since we are working with a linked list with linear search. I also made a constructor that will take 2 parameters. First is the T pointer which is the input for the user and the int for the arr size. The T pointer which is the inputArr will take a value that since I called the class in the main, with the int data type the input will also take an int value. since I declared it with a pointer operator the value will also have a pointer ready, at the constructor while running the inputarr will initialize the arr and also the size. After that I used the linear search algorithm that we used in the module. Then I also made a function that will display all the elements in the array. At the main that is when I created the array rather implement it and also I called the function to first display it then the key is for the user to be able to input the value. then I used the formula that is in the module.

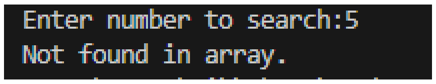
Table 6-2b. Linear Search for Linked List

CODE:

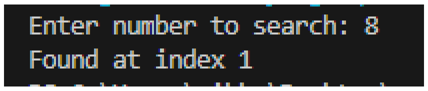


```cpp
#include <iostream>
using namespace std;

int main() {
    int arr[] = {4, 8, 15, 16, 23, 42};
    int size = sizeof(arr) / sizeof(arr[0]);
    int key;

    cout << "Enter number to search: ";
    cin >> key;

    bool found = false;
    for (int i = 0; i < size; i++) {
        if (arr[i] == key) {
            cout << "Found at index " << i << endl;
            found = true;
            break;
        }
    }

    if (!found) {
        cout << "Not found in array." << endl;
    }

    return 0;
}
```

OUTPUPT:



```
Enter number to search:5
Not found in array.
```



```
Enter number to search: 8
Found at index 1
```

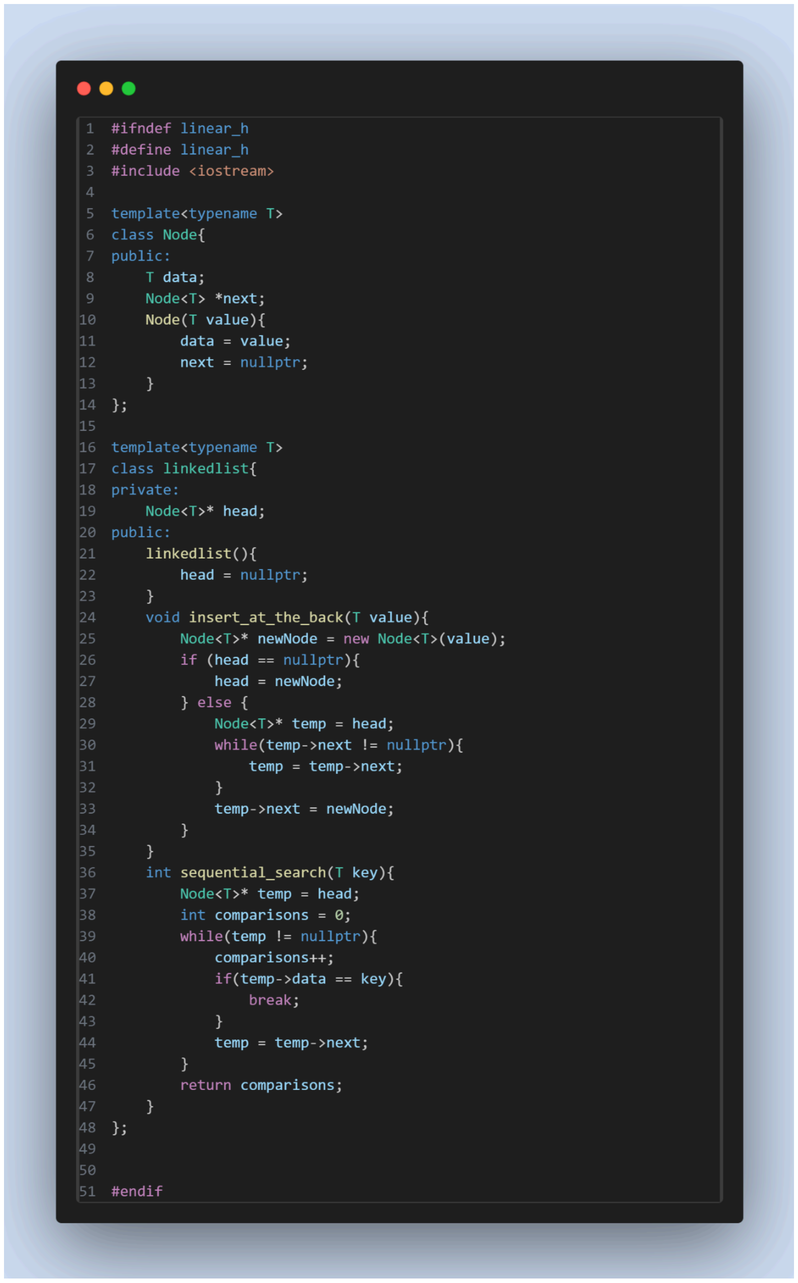
OBSERVATION:

In here, I just used the code earlier. I did change a bit of the code like the bool this will return a value for us when the variable found did find the number or the value the user input. That is set to false because I assumed that the user will just put a random number, and since I didn't display the array. as for the formula I just used the one at the top.

Table 6-2a. Linear Search for Arrays

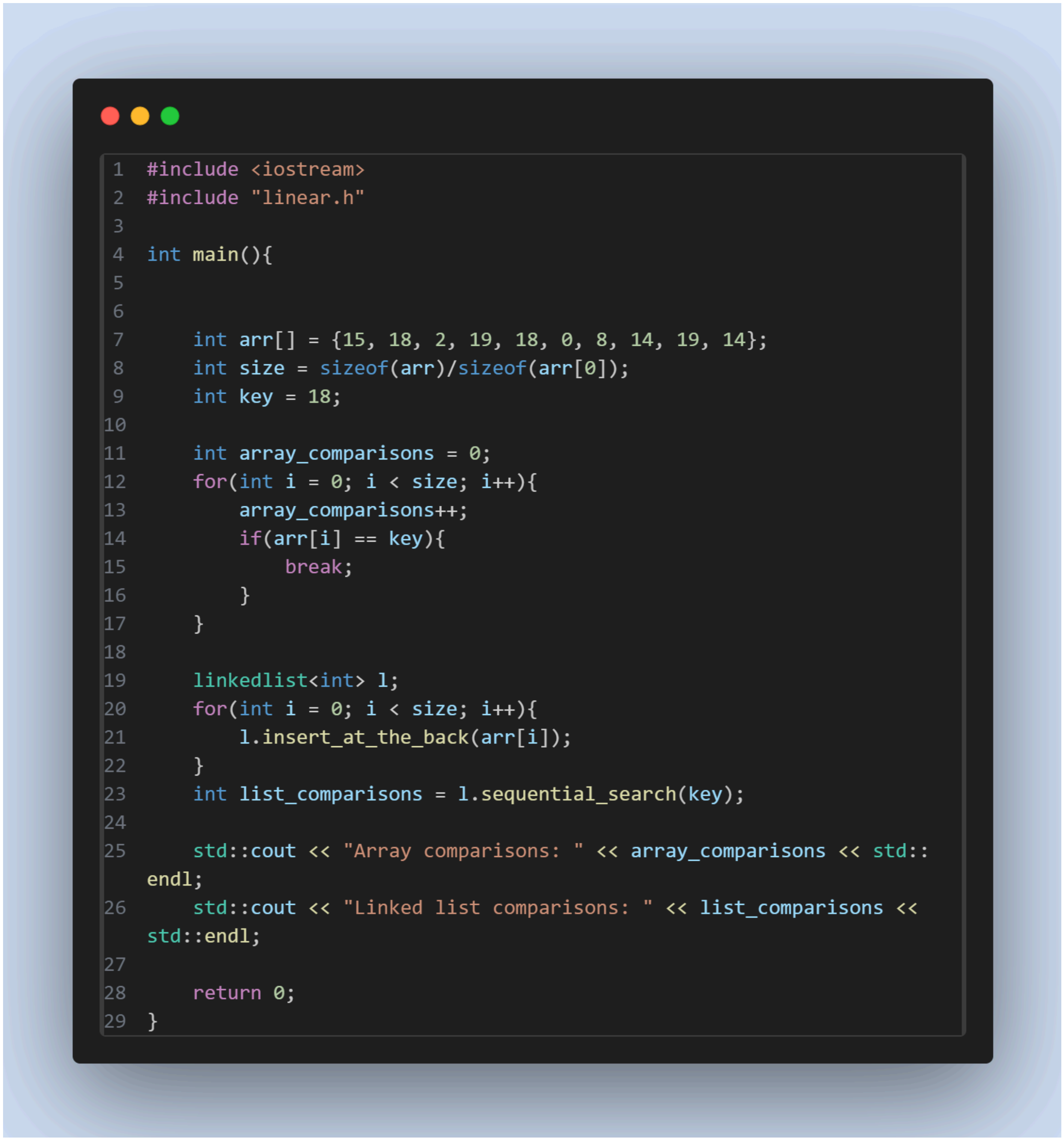
7. Supplementary Activity

CODE:
HEADER:



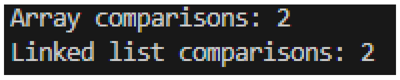
```cpp
#ifndef linear_h
#define linear_h
#include <iostream>

template<typename T>
class Node{
public:
    T data;
    Node<T> *next;
    Node(T value){
        data = value;
        next = nullptr;
    }
};

template<typename T>
class linkedlist{
private:
    Node<T>* head;
public:
    linkedlist(){
        head = nullptr;
    }
    void insert_at_the_back(T value){
        Node<T>* newNode = new Node<T>(value);
        if (head == nullptr){
            head = newNode;
        } else {
            Node<T>* temp = head;
            while(temp->next != nullptr){
                temp = temp->next;
            }
            temp->next = newNode;
        }
    }
    int sequential_search(T key){
        Node<T>* temp = head;
        int comparisons = 0;
        while(temp != nullptr){
            comparisons++;
            if(temp->data == key){
                break;
            }
            temp = temp->next;
        }
        return comparisons;
    }
};


#endif
```

MAIN:



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

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 array_comparisons = 0;
    for(int i = 0; i < size; i++){
        array_comparisons++;
        if(arr[i] == key){
            break;
        }
    }

    linkedlist<int> l;
    for(int i = 0; i < size; i++){
        l.insert_at_the_back(arr[i]);
    }
    int list_comparisons = l.sequential_search(key);

    std::cout << "Array comparisons: " << array_comparisons << std::endl;
    std::cout << "Linked list comparisons: " << list_comparisons << std::endl;

    return 0;
}
```

OUTPUT:



```
Array comparisons: 2
Linked list comparisons: 2
```

OBSERVATION:

In here i just created a class that contains all the necessary data for the Node like the data and the pointer to the next. Then I created a constructor that takes 1 parameter as the value when we later call it to the linked list class.this will also initialize the data and next in run time. Then I created another class for the actual linkedlist, then set the Node to private, since we won't create another class. I also created a constructor to initialize the head to point to the null so when I run it. It will automatically point to the null. I created a function that will insert the new nodee at the back. I used a pointer to the node since the newNode need to have a pointer to be able to point at another set of node. then set the value at the Node class Node constructor. I used if because when the I run the program the list is empty so to catch that I'll set the newNodee to be the head first. Then I created a temp to traverse all the list so that the head will not be lost, since the head is very crucial when using the linked list. then set the temp to the temp->next this line will traverse all the list. then the value of the temp->next will be put at the newNode. Then the sequential search function will focus on finding the 18 on our array at the main. We need to create a temp variable to traverse all the list. At the main I just used the formula fo the linear search. then call the class I created and set it to l so that I can call it more easily.

8. Conclusion

In conclusion, I noticed that linear, binary, and linked list are essential tools in programming. These are the fundamentals in data structure and how we can create a well structured program. Linear search is a method that is the simplest among the 2 searching algorithms. When using a linear algo we just need to start at the first index when using an array, then compare the value to the next using an if statement and the condition. This checks all the elements or list to find the va;ie that we are looking for, this also cheks 1 by 1 that is why linear search is good when the elements or the list is small. But inefficient when the data is big. That is when the binary search is useful. Although, we need it to be sorted first, since we will start at the middle then divide it when after comparing the number at the middle and the right and left side of the list or the element, that is why we need a sorting algorithm for it to work first if the data is unsorted.

9. Assessment Rubric