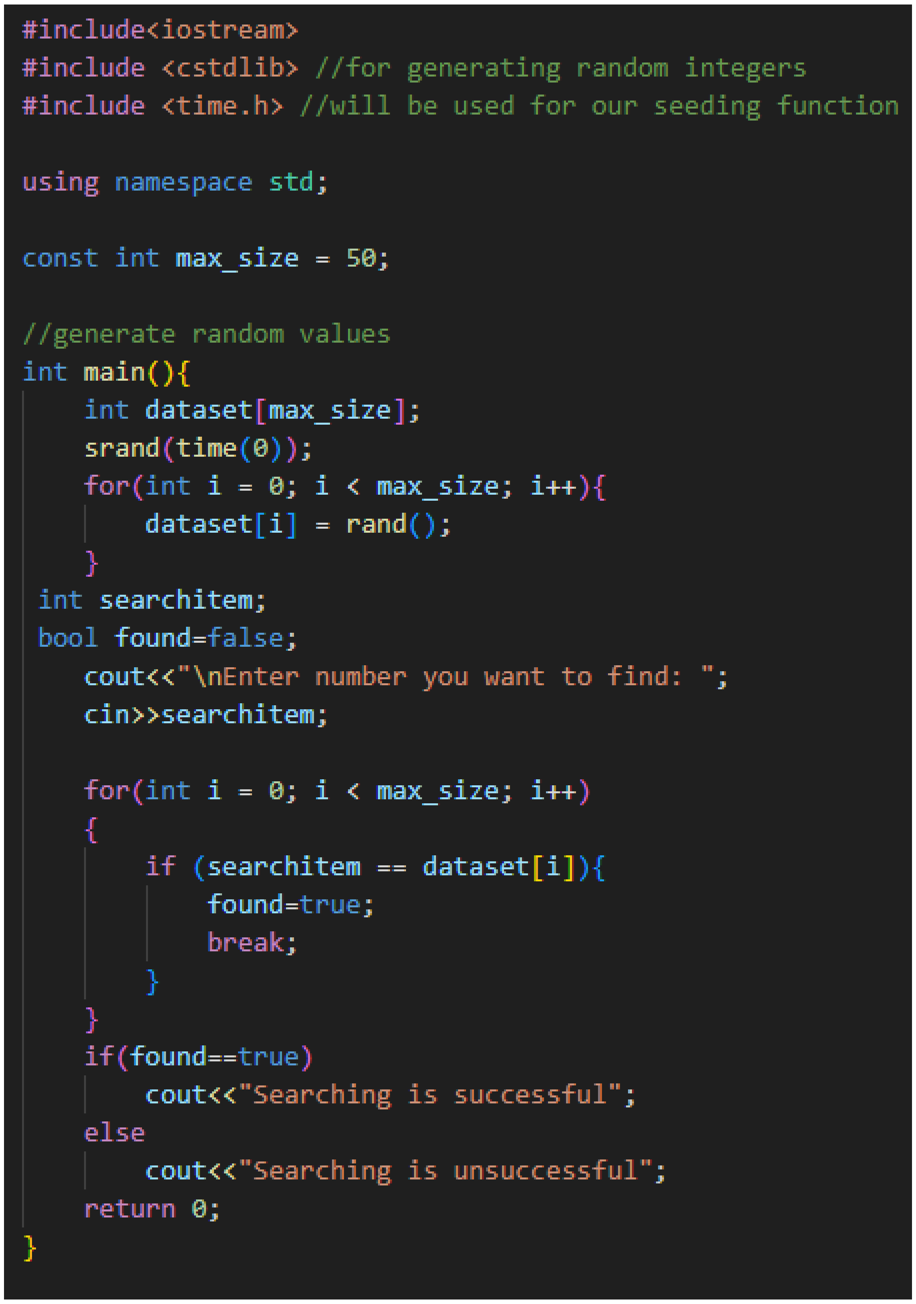
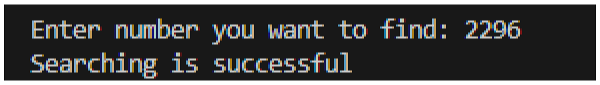
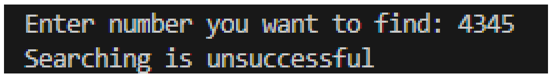
ACTIVITY NO. 6	
SEARCHING TECHNIQUES	
Course Code: CPE200	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:10/19/23**
Section:CPE21S1	**Date Submitted:10/20/23**
Name(s): Esteron, Jenel F. Fabreag, Patrick Kyel M. Banania, Ariel Jr. T Falco, Arvin Paul D.	**Instructor:** **Mr. Dennis Nava**
6. Output	

Screenshot



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

using namespace std;

const int max_size = 50;

//generate random values
int main(){
    int dataset[max_size];
    srand(time(0));
    for(int i = 0; i < max_size; i++){
        dataset[i] = rand();
    }
 int searchitem;
 bool found=false;
    cout<<"\nEnter number you want to find: ";
    cin>>searchitem;

    for(int i = 0; i < max_size; i++)
    {
        if (searchitem == dataset[i]){
            found=true;
            break;
        }
    }
    if(found==true)
        cout<<"Searching is successful";
    else
        cout<<"Searching is unsuccessful";
    return 0;
}
```

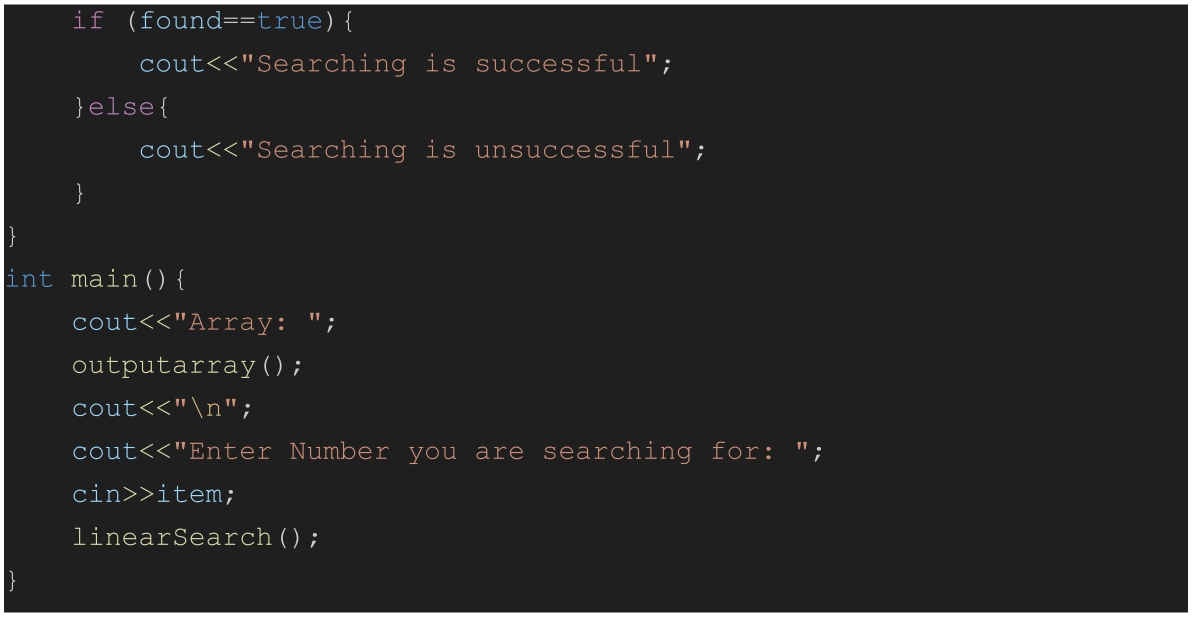
	 `Enter number you want to find: 2296`
	`Searching is successful`
	 `Enter number you want to find: 4345`
	`Searching is unsuccessful`

Observation	The nature of the combination of codes creates a random set of numbers as an array where it is outputted using a for loop. After that linear searching is created to finish the program, Where the program generates a random set of numbers then lets you input a number that you want to search, the program then searches for the number you have imputed if or if it is not in the array. It will output successfully if it is in the array and unsuccessful if the condition is not met.

Table 6-1. Data Generated and Observations.

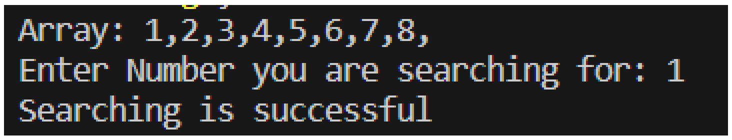
Code

```
#include<iostream>

using namespace std;

int item;
void outputarray(){
    int dataset[9]={1,2,3,4,5,6,7,8};
    int i = 0;
    while (i<8){
        if (i>8){
            break;
        }
        cout<<dataset[i]<<",";
        i++;
    }

}
void linearSearch(){
    int dataset[9]={1,2,3,4,5,6,7,8};
    int i = 0;
    bool found=false;

    while (i<=9){
        if (item == dataset[i]){
            found=true;

           break;
        }
        i++;
    }
```

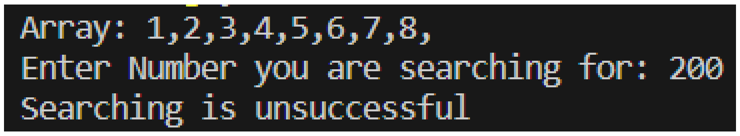


```
    if (found==true){
        cout<<"Searching is successful";
    }else{
        cout<<"Searching is unsuccessful";
    }
}
int main(){
    cout<<"Array: ";
    outputarray();
    cout<<"\n";
    cout<<"Enter Number you are searching for: ";
    cin>>item;
    linearSearch();
}
```

Output



```
Array: 1,2,3,4,5,6,7,8,
Enter Number you are searching for: 1
Searching is successful
```

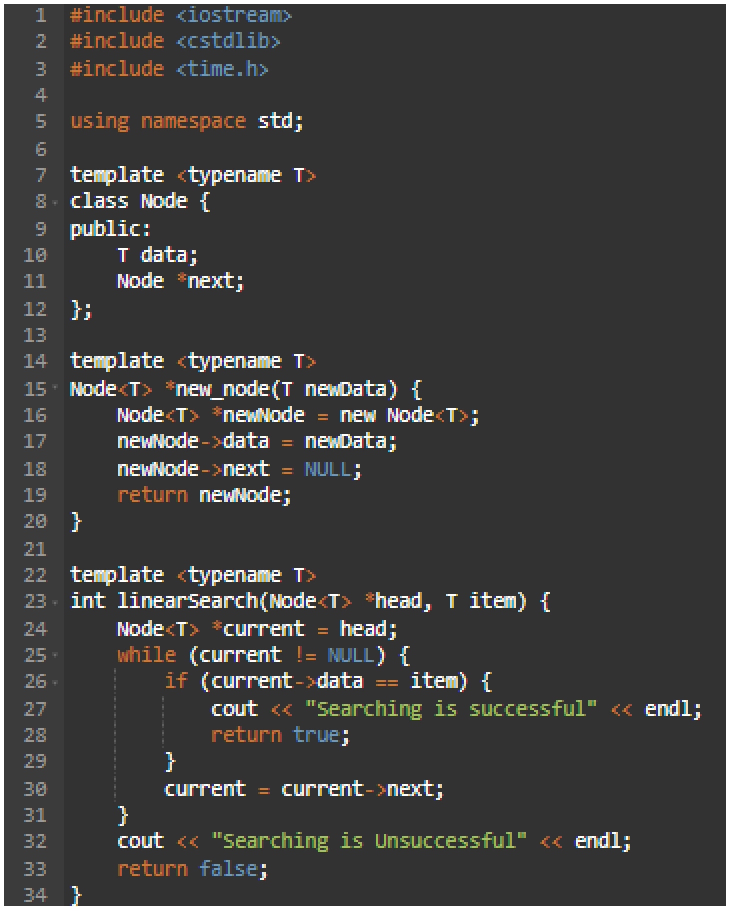


```
Array: 1,2,3,4,5,6,7,8,
Enter Number you are searching for: 200
Searching is unsuccessful
```

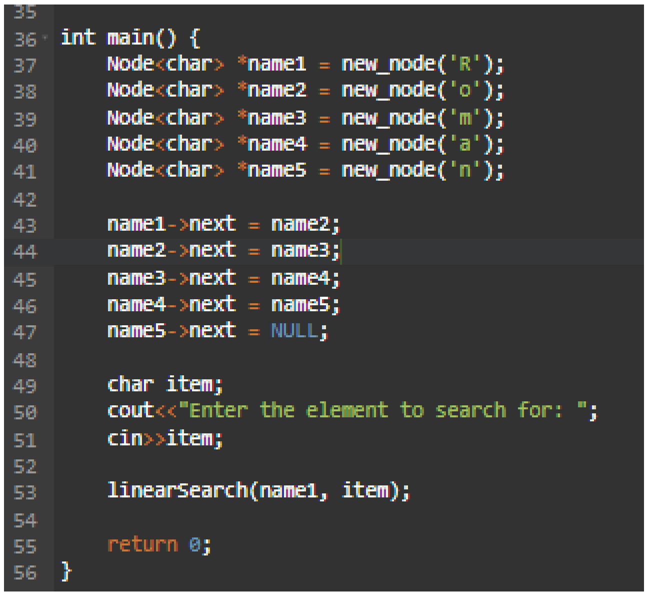
Observation	The program is a linear search of an array where there is a function created that is for the linear searching of an array where the function is created using a while loop that loops an increment of the index until it satisfies the condition which then it will output the wanted message which is either “Searching is successful” or “Searching is unsuccessful”.

Table 6-2a. Linear Search for Arrays

Code

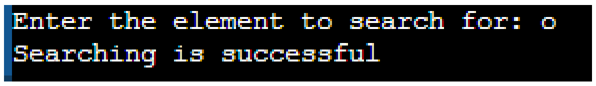


```cpp
#include <iostream>
#include <cstdlib>
#include <time.h>

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;
}

template <typename T>
int linearSearch(Node<T> *head, T item) {
    Node<T> *current = head;
    while (current != NULL) {
        if (current->data == item) {
            cout << "Searching is successful" << endl;
            return true;
        }
        current = current->next;
    }
    cout << "Searching is Unsuccessful" << endl;
    return false;
}
```

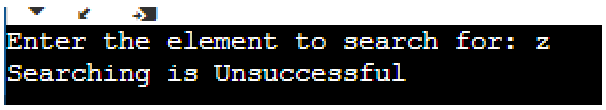


```cpp
int main() {
    Node<char> *name1 = new_node('R');
    Node<char> *name2 = new_node('o');
    Node<char> *name3 = new_node('m');
    Node<char> *name4 = new_node('a');
    Node<char> *name5 = new_node('n');

    name1->next = name2;
    name2->next = name3;
    name3->next = name4;
    name4->next = name5;
    name5->next = NULL;

    char item;
    cout<<"Enter the element to search for: ";
    cin>>item;

    linearSearch(name1, item);

    return 0;
}
```

Output



```
Enter the element to search for: o
Searching is successful
```



```
Enter the element to search for: z
Searching is Unsuccessful
```

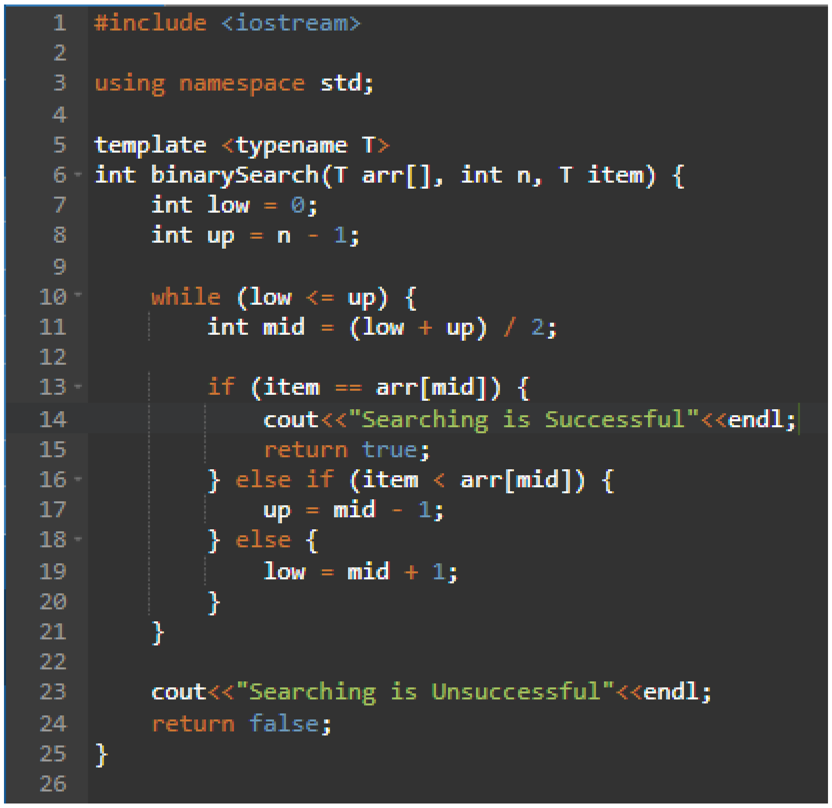
Observation

The linear search for linked list searches for the element ‘n’ that is present in the linked list where it it goes through the list elements and when it finds the element it says “searching is successful” and when it doesn’t it prints “searching is unsuccessful” where in it indicates that the element is not there

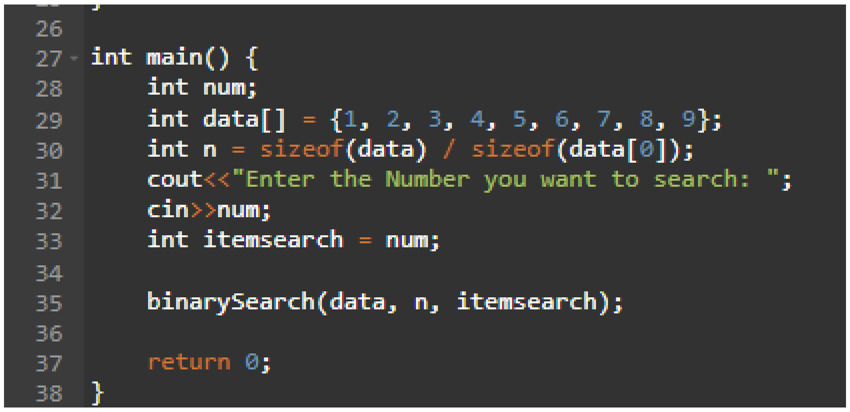
	or not present

Table 6-2b. Linear Search for Linked List

Code

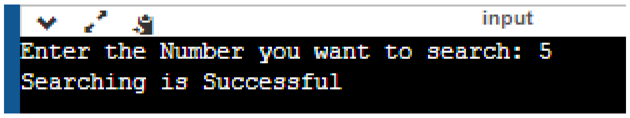


```cpp
#include <iostream>

using namespace std;

template <typename T>
int binarySearch(T arr[], int n, T item) {
    int low = 0;
    int up = n - 1;

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

        if (item == arr[mid]) {
            cout<<"Searching is Successful"<<endl;
            return true;
        } else if (item < arr[mid]) {
            up = mid - 1;
        } else {
            low = mid + 1;
        }
    }

    cout<<"Searching is Unsuccessful"<<endl;
    return false;
}

```

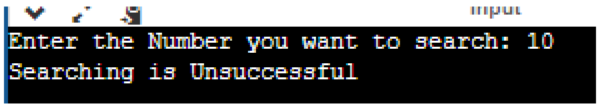


```cpp

int main() {
    int num;
    int data[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
    int n = sizeof(data) / sizeof(data[0]);
    cout<<"Enter the Number you want to search: ";
    cin>>num;
    int itemsearch = num;

    binarySearch(data, n, itemsearch);

    return 0;
}
```

Output



```
Enter the Number you want to search: 5
Searching is Successful
```



```
Enter the Number you want to search: 10
Searching is Unsuccessful
```

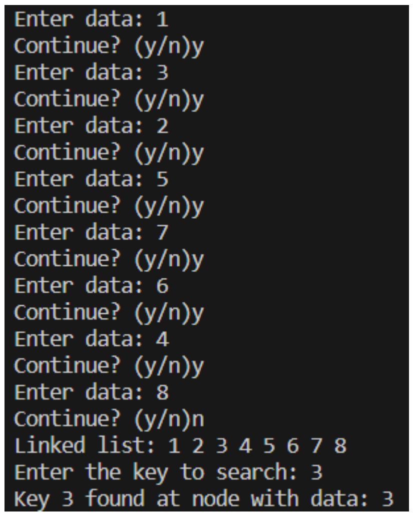
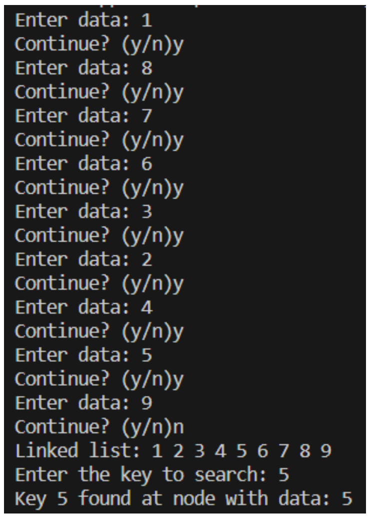
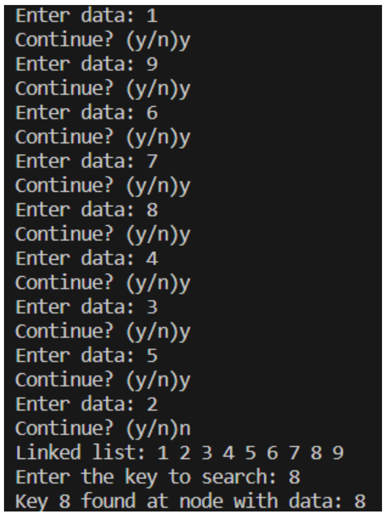
Observation

in this code we can see that we use binary searching where in the program checks the array that is present on the program if the given input that was given to be searched is present in the array and if it is present it will say that “Searching is Successful’ otherwise of its not present then it will say “Searching is Unsuccessful”

Table 6-3a. Binary Search for Arrays

Code

```cpp
#include <iostream>

using namespace std;

template <typename T>
struct Node {
    T data;
    Node<T>* next;
};

template <typename T>
Node<T>* new_node(T data) {
    Node<T>* node = new Node<T>;
    node->data = data;
    node->next = nullptr;
    return node;
}
template <typename T>
void insert_ordered(Node<T>*& head, T data) {
    Node<T>* newnode = new_node(data);
    if (head == nullptr || head->data >= data) {
        newnode->next = head;
        head = newnode;
        return;
    }

    Node<T>* curr = head;
    while (curr->next != nullptr && curr->next->data < data) {
        curr = curr->next;
    }

    newnode->next = curr->next;
    curr->next = newnode;
}
template <typename T>
void create_linked_list(Node<T>*& head) {
    char choice = 'y';
    T newData;

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

        insert_ordered(head, newData);

        cout << "Continue? (y/n)";
        cin >> choice;
    }
}
template <typename T>
void display_list(Node<T>* head) {
    while (head != nullptr) {
        cout << head->data << " ";
        head = head->next;
    }
}
template <typename T>
Node<T>* getMiddle(Node<T>* head, Node<T>* last) {
    if (head == nullptr) return nullptr;

    Node<T>* slow = head;
    Node<T>* fast = head;

    while (fast != last) {
        fast = fast->next;
        if (fast != last) {
            slow = slow->next;
            fast = fast->next;
        }
    }

    return slow;
}
template <typename T>
Node<T>* binarySearch(Node<T>* head, T key) {
    if (head == nullptr) return nullptr;

    Node<T>* start = head;
    Node<T>* last = nullptr;

    while (start != last) {
        Node<T>* middle = getMiddle(start, last);

        if (middle->data == key) {
            return middle;
        } else if (middle->data > key) {
            last = middle;
        } else {
            start = middle->next;
        }
    }

    return nullptr;
}

int main() {
    Node<int>* head = nullptr;
    create_linked_list(head);
    cout << "Linked list: ";
    display_list(head);
    int key;
    cout << "\nEnter the key to search: ";
    cin >> key;
    Node<int>* foundNode = binarySearch(head, key);
    if (foundNode) {
        cout << "Key " << key << " found at node with data: " <<
foundNode->data << endl;
    } else {
        cout << "Key " << key << " not found in the linked list" <<
endl;
    }

    return 0;
}
```

Output	  
Observation	The program consists of functions that creates a linked list, displays the list, gets the middle of the linked list and a binary search. It is interesting to see that creating a binary search traversal on a linked list is harder than using it on an array.

```
Enter data: 1
Continue? (y/n)y
Enter data: 3
Continue? (y/n)y
Enter data: 2
Continue? (y/n)y
Enter data: 5
Continue? (y/n)y
Enter data: 7
Continue? (y/n)y
Enter data: 6
Continue? (y/n)y
Enter data: 4
Continue? (y/n)y
Enter data: 8
Continue? (y/n)n
Linked list: 1 2 3 4 5 6 7 8
Enter the key to search: 3
Key 3 found at node with data: 3
```

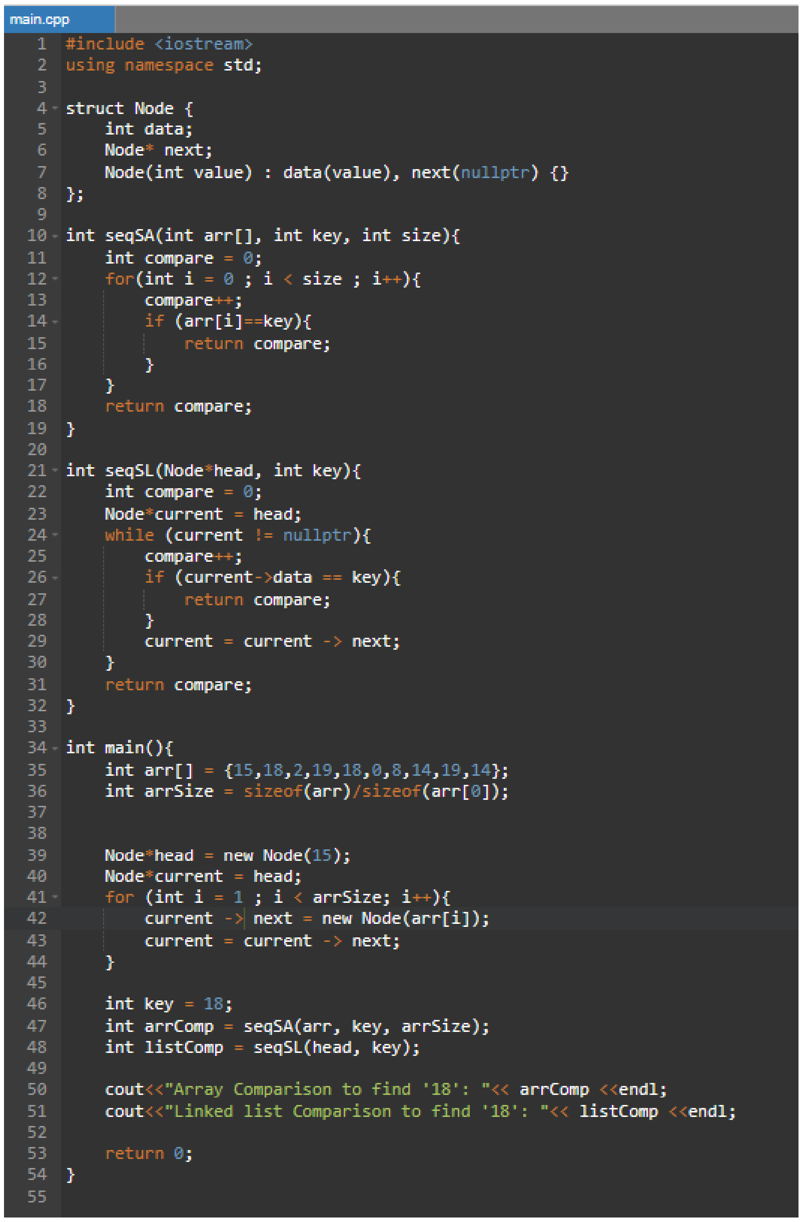
```
Enter data: 1
Continue? (y/n)y
Enter data: 8
Continue? (y/n)y
Enter data: 7
Continue? (y/n)y
Enter data: 6
Continue? (y/n)y
Enter data: 3
Continue? (y/n)y
Enter data: 2
Continue? (y/n)y
Enter data: 4
Continue? (y/n)y
Enter data: 5
Continue? (y/n)y
Enter data: 9
Continue? (y/n)n
Linked list: 1 2 3 4 5 6 7 8 9
Enter the key to search: 5
Key 5 found at node with data: 5
```

```
Enter data: 1
Continue? (y/n)y
Enter data: 9
Continue? (y/n)y
Enter data: 6
Continue? (y/n)y
Enter data: 7
Continue? (y/n)y
Enter data: 8
Continue? (y/n)y
Enter data: 4
Continue? (y/n)y
Enter data: 3
Continue? (y/n)y
Enter data: 5
Continue? (y/n)y
Enter data: 2
Continue? (y/n)n
Linked list: 1 2 3 4 5 6 7 8 9
Enter the key to search: 8
Key 8 found at node with data: 8
```

Table 6-3b. Binary Search for Linked List

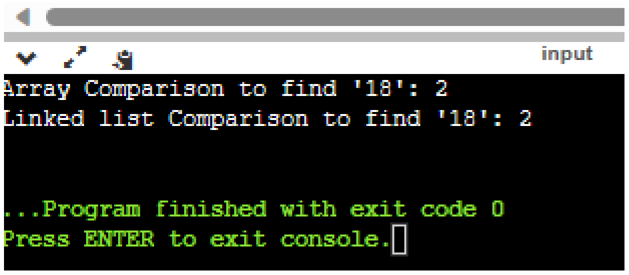
7. Supplementary Activity

Problem 1:



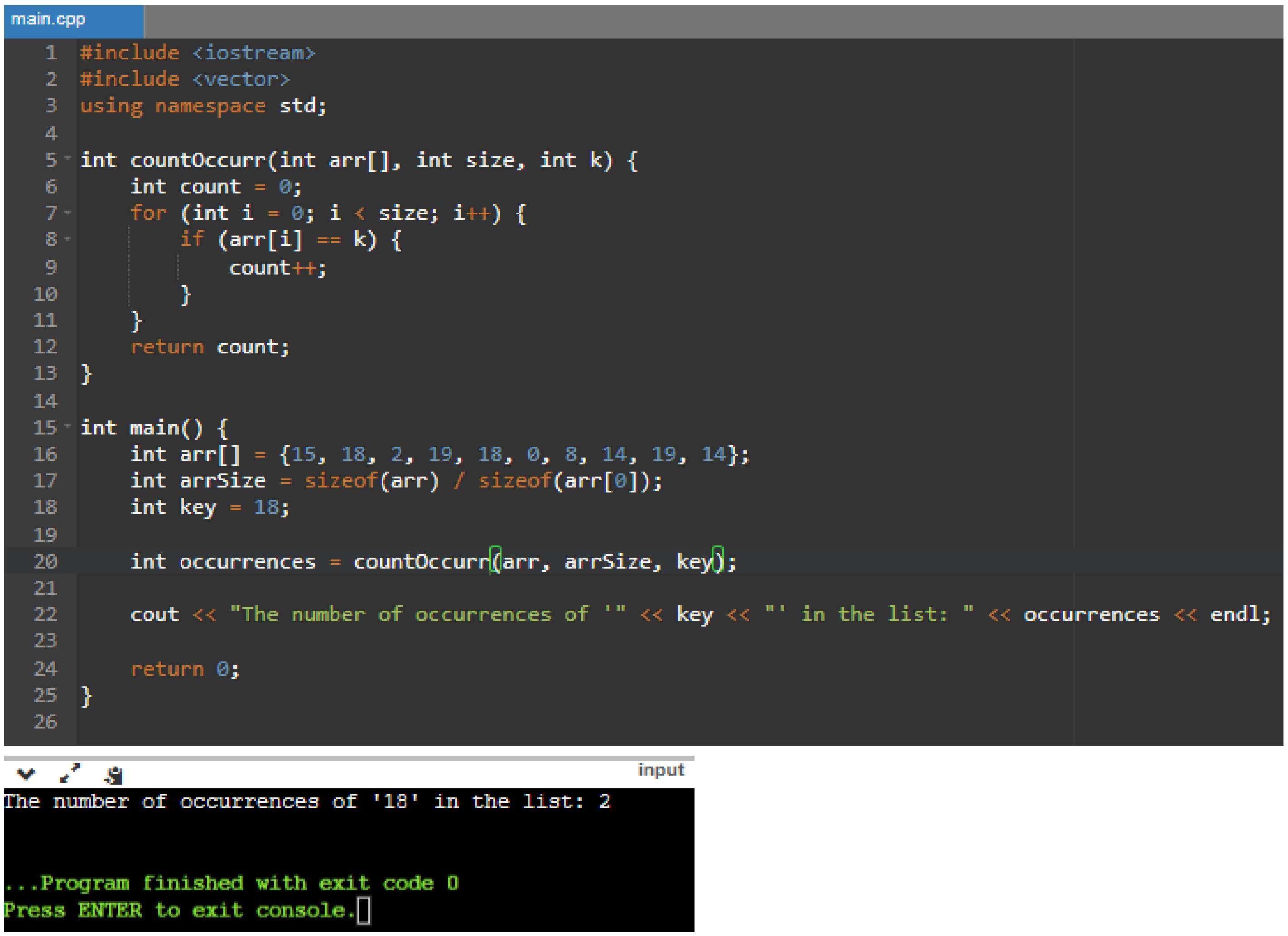
main.cpp

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

struct Node {
    int data;
    Node* next;
    Node(int value) : data(value), next(nullptr) {}
};

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

int seqSL(Node*head, int key){
    int compare = 0;
    Node*current = head;
    while (current != nullptr){
        compare++;
        if (current->data == key){
            return compare;
        }
        current = current -> next;
    }
    return compare;
}

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


    Node*head = new Node(15);
    Node*current = head;
    for (int i = 1 ; i < arrSize; i++){
        current -> next = new Node(arr[i]);
        current = current -> next;
    }

    int key = 18;
    int arrComp = seqSA(arr, key, arrSize);
    int listComp = seqSL(head, key);

    cout<<"Array Comparison to find '18': "<< arrComp <<endl;
    cout<<"Linked list Comparison to find '18': "<< listComp <<endl;

    return 0;
}
```



```
Array Comparison to find '18': 2
Linked list Comparison to find '18': 2


...Program finished with exit code 0
Press ENTER to exit console.
```

Problem 2:

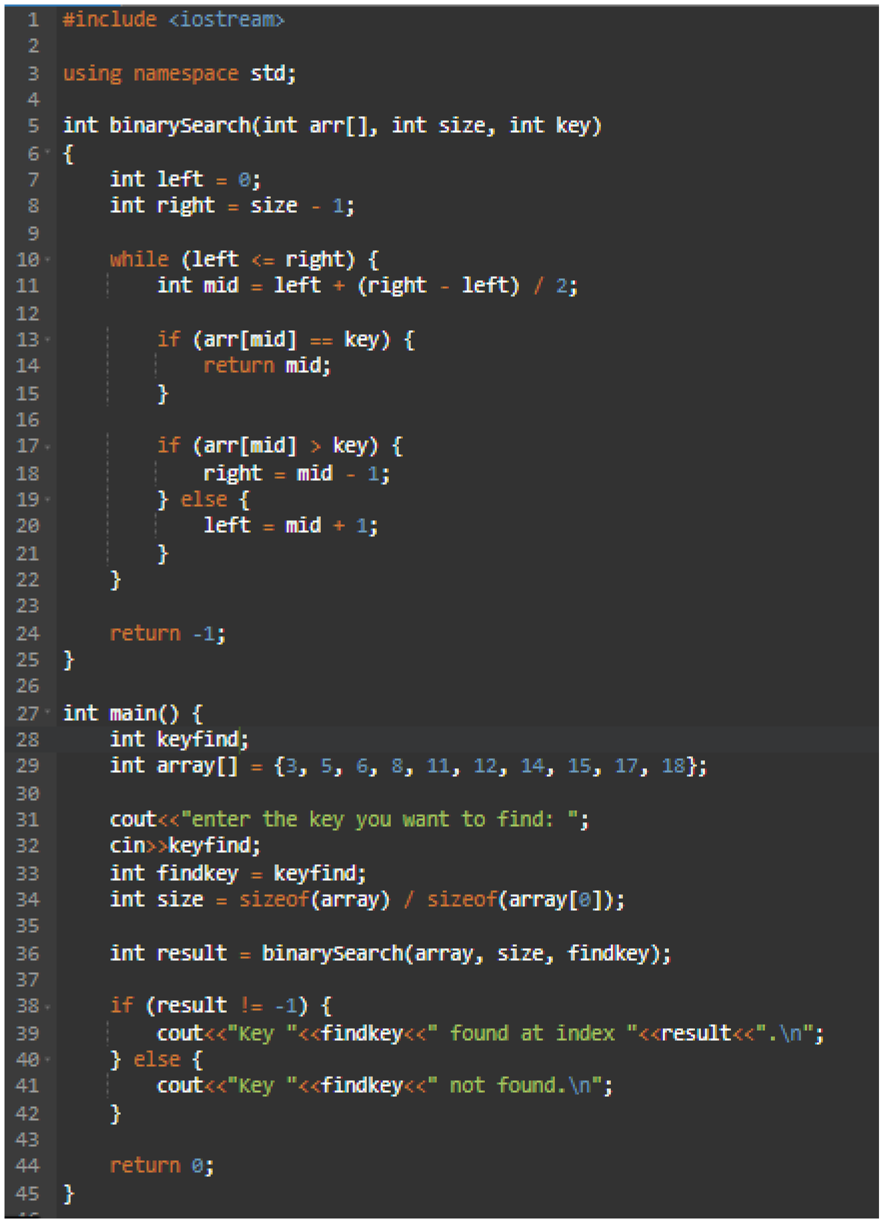


main.cpp

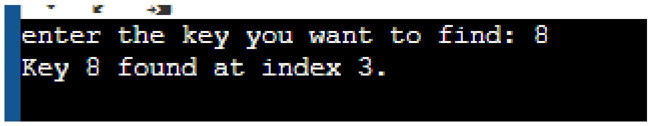
```cpp
#include <iostream>
#include <vector>
using namespace std;

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

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

    int occurrences = countOccurr(arr, arrSize, key);

    cout << "The number of occurrences of '" << key << "' in the list: " << occurrences << endl;

    return 0;
}
```

```
The number of occurrences of '18' in the list: 2


...Program finished with exit code 0
Press ENTER to exit console.
```

Problem 3:

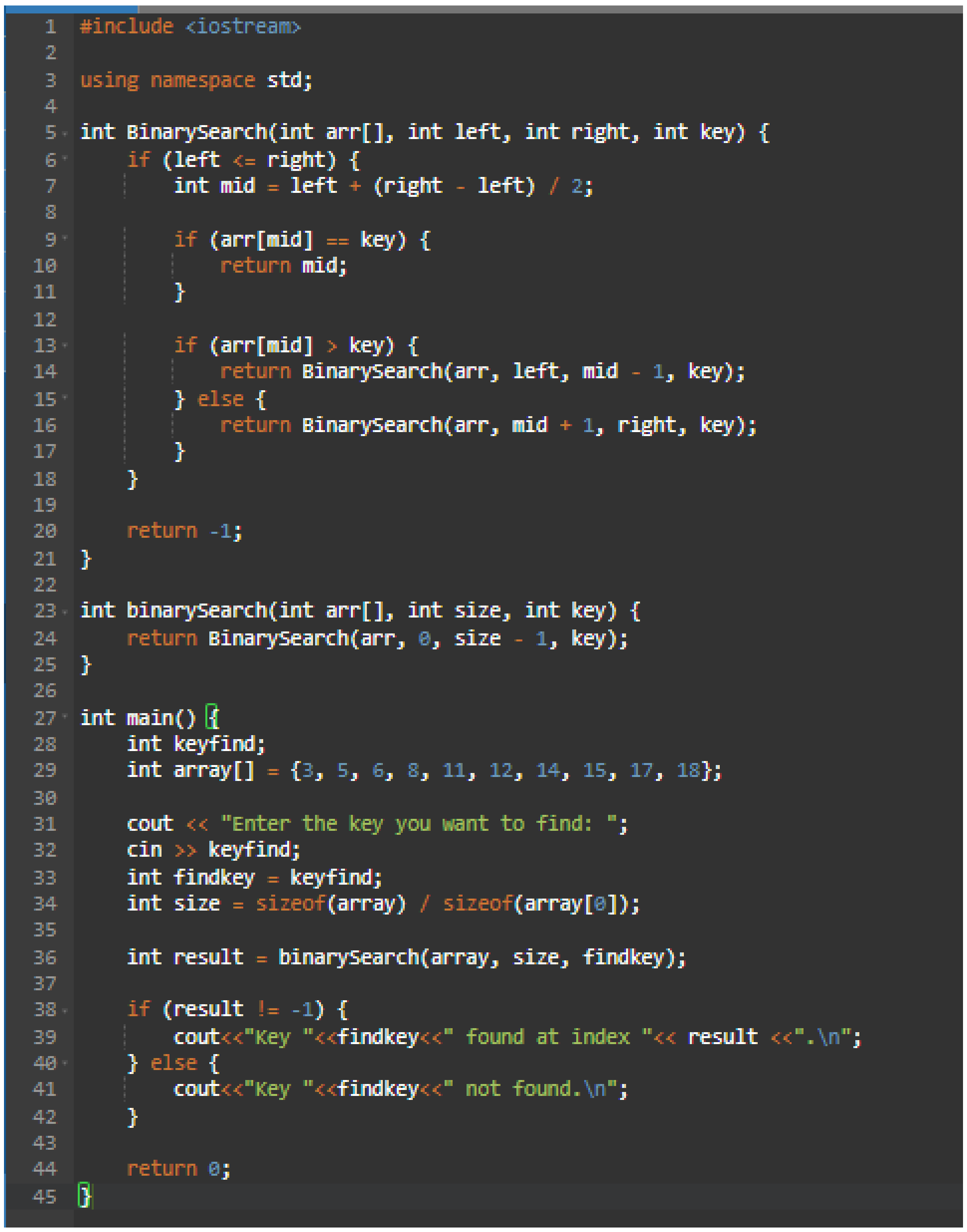


```cpp
#include <iostream>

using namespace std;

int binarySearch(int arr[], int size, int key)
{
    int left = 0;
    int right = size - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;

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

        if (arr[mid] > key) {
            right = mid - 1;
        } else {
            left = mid + 1;
        }
    }

    return -1;
}

int main() {
    int keyfind;
    int array[] = {3, 5, 6, 8, 11, 12, 14, 15, 17, 18};

    cout<<"enter the key you want to find: ";
    cin>>keyfind;
    int findkey = keyfind;
    int size = sizeof(array) / sizeof(array[0]);

    int result = binarySearch(array, size, findkey);

    if (result != -1) {
        cout<<"Key "<<findkey<<" found at index "<<result<<".\n";
    } else {
        cout<<"Key "<<findkey<<" not found.\n";
    }

    return 0;
}
```

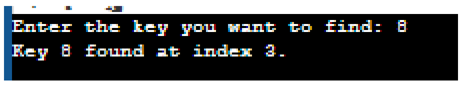


```
enter the key you want to find: 8
Key 8 found at index 3.
```

Problem 4:



```cpp
#include <iostream>

using namespace std;

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

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

        if (arr[mid] > key) {
            return BinarySearch(arr, left, mid - 1, key);
        } else {
            return BinarySearch(arr, mid + 1, right, key);
        }
    }

    return -1;
}

int binarySearch(int arr[], int size, int key) {
    return BinarySearch(arr, 0, size - 1, key);
}

int main() {
    int keyfind;
    int array[] = {3, 5, 6, 8, 11, 12, 14, 15, 17, 18};

    cout << "Enter the key you want to find: ";
    cin >> keyfind;
    int findkey = keyfind;
    int size = sizeof(array) / sizeof(array[0]);

    int result = binarySearch(array, size, findkey);

    if (result != -1) {
        cout<<"Key "<<findkey<<" found at index "<< result <<".\n";
    } else {
        cout<<"Key "<<findkey<<" not found.\n";
    }

    return 0;
}
```



```
Enter the key you want to find: 8
Key 8 found at index 3.
```

8. Conclusion

In Searching Techniques, searching is the process of finding the position of an element within a given list which is considered a success if the element is within the list, and failure if otherwise. There are two examples of Searching Techniques in which one is Linear Searching which is the simplest and easiest method of searching in which the element must be found as the list is sequentially searched. Next is, Binary Search, which searches a sorted array by repeatedly dividing the search interval in half. Begin with an interval covering the whole array.

Data Generating uses a combination of codes that creates a random set of numbers as an array where it has become the output using a for loop. After that it inputs and searches for it in the array. the outputs become successful if it's in the array and unsuccessful if it's not.

Linear Searching for Arrays uses a function that creates an array where its function is to use a while loop that loops an increment of the index until it satisfies the condition in which the output message will either be "search successful" or "search unsuccessful."

Linear Search for Linked List uses an element 'n' that is present within the linked list where it goes through the list elements and if it finds the elements, it would say "searching successful" and if not, the element won't print "searching unsuccessful."

Binary Search for Arrays is where the program checks the array that is present on the program and if it given input, it will find the search where when it is present it would say "searching is successful" and if not "searching is unsuccessful."

Supplementary Activity Problem 1, where the array comparison states that to find 18 in the array where it is placed in array 2 and inputs 2 and the output was 18 is indeed placed in 2. Problem 2, where in the array was to find the number of occurrences in the list and yet again 18 was placed in array 2 and inputs 2 and the output was 18 is in array 2. Problem 3, describes entering the key you want to find in which the input was to find 8 and the output was 3 because the array starts in 0 and the placement would be 0,1,2,3 and the output was 8 is found at index 3. Problem 4, this has the same input and output as problem 3 so the outcome was 8 was found at index 3.

In conclusion, our group did the activity with great analysis and observation in each problem and we can improve our areas of expertise because this activity helps us in representing multiple values by making use of a single variable. Overall, this is a fun activity for us all.

9. Assessment Rubric