KATHMANDU UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DHULIKHEL, KAVRE

A

LAB REPORT

ON

“LINKED LIST”

Subject Code No: COMP 202

SUBMITTED BY:	**SUBMITTED TO:**
Name: Sanjib Dahal	Dr. Rajani Chulyadyo
Roll No: 18	Department of Computer
Group: CE	Science and Engineering
Level: II/I	

Date of Submission: - 2024/05/06

Lab Report 1

Linked list

Linked lists are linear data structures consisting of a sequence of elements called nodes, each node holding some information and a pointer to another node in the list. Unlike arrays, where elements are stored in contiguous memory locations, linked list elements are dynamically allocated and connected via pointers. This report aims to demonstrate the functionality and applications of linked lists through code demonstrations.

Advantages of Linked list

- Linked lists can grow or shrink in size dynamically. Unlike arrays, you don't need to pre-allocate memory for a fixed number of elements.
- Linked lists don’t suffer from the overhead of fixed size allocation like arrays.
- Linked lists don't suffer from "wasted space" like arrays do, where you might have allocated space for a large array but only use a small portion of it. Linked lists only use as much memory as needed.
- Insertions and deletions are efficient in linked lists.

Operations

The major operations performed in singly linked list are as follows:

1. isEmpty(): Returns true if the list is empty, and false otherwise
2. addToHead(data): Inserts an element to the beginning of the list
3. addToTail(data): Inserts an element to the end of the list
4. add(data, predecessor): Inserts an element after the given predecessor node
5. removeFromHead(): Removes the first node in the list
6. removeFromTail(): Removes the last node in the list
7. remove(data): Removes the node with the given data
8. retrieve(data, outputNodePointer): Returns the pointer to the node with the requested data
9. search(data): Returns true if the data exists in the list, and false otherwise
10. traverse(): Displays the contents of the list

Github Link to Code

The GitHub link to the repository for the implementation of singly linked list is:

https://github.com/sanjibdahal/ce3rdsem/tree/main/linkedlist

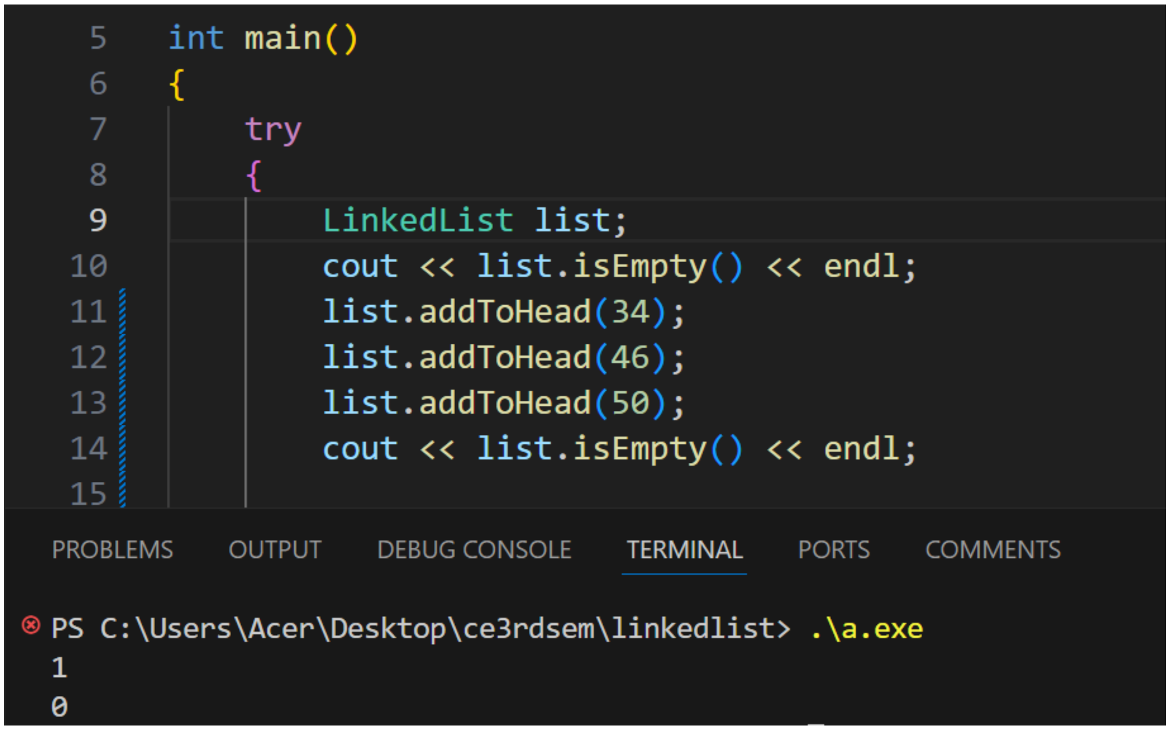
Functions and Outputs:

The outputs for each operation performed on the singly linked list are displayed below:

1. **isEmpty():**

   The isEmpty() function checks if the created linked list is empty or not. The function returns true value if the list is empty and returns false if it is not empty.

```
bool LinkedList::isEmpty() {
return HEAD == NULL && TAIL == NULL;
}
```



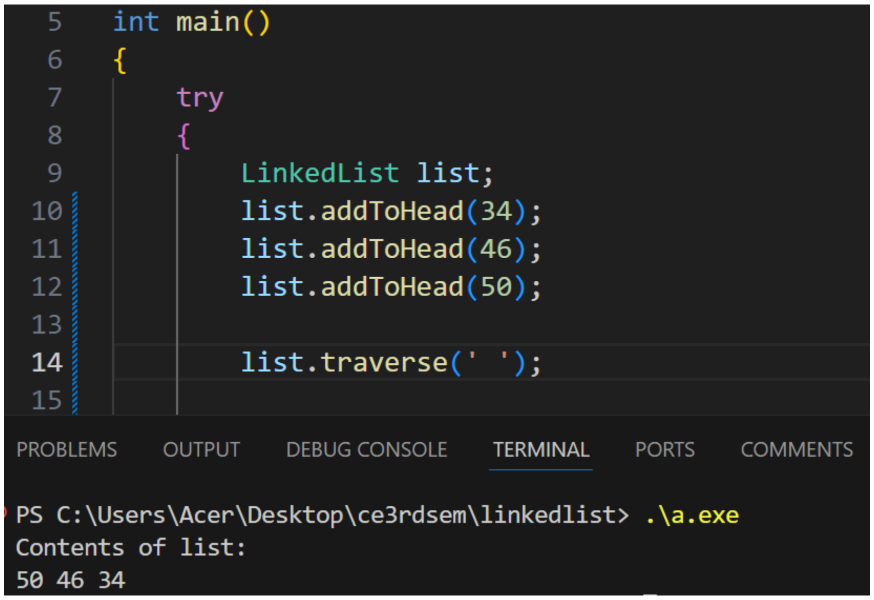
```
int main()
{
    try
    {
        LinkedList list;
        cout << list.isEmpty() << endl;
        list.addToHead(34);
        list.addToHead(46);
        list.addToHead(50);
        cout << list.isEmpty() << endl;

```

```
PS C:\Users\Acer\Desktop\ce3rdsem\linkedlist> .\a.exe
1
0
```

2. **addToHead(data):**

   The addToHead(data) function inserts a new node in the beginning of the list with its value as data.

```
void LinkedList::addToHead(int data)

{

        Node *newNode = new Node(data, HEAD);

        if (this->isEmpty())

            TAIL = newNode;

        HEAD = newNode;

}
```



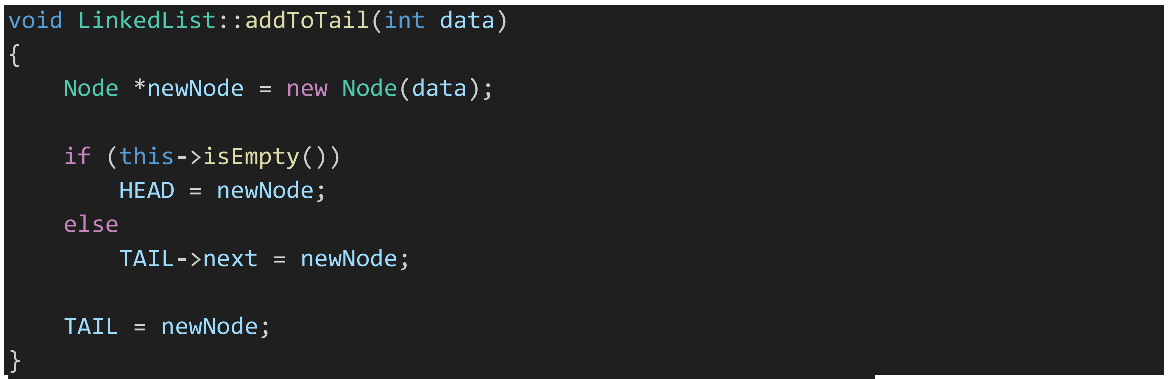
```
int main()
{
    try
    {
        LinkedList list;
        list.addToHead(34);
        list.addToHead(46);
        list.addToHead(50);

        list.traverse(' ');

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

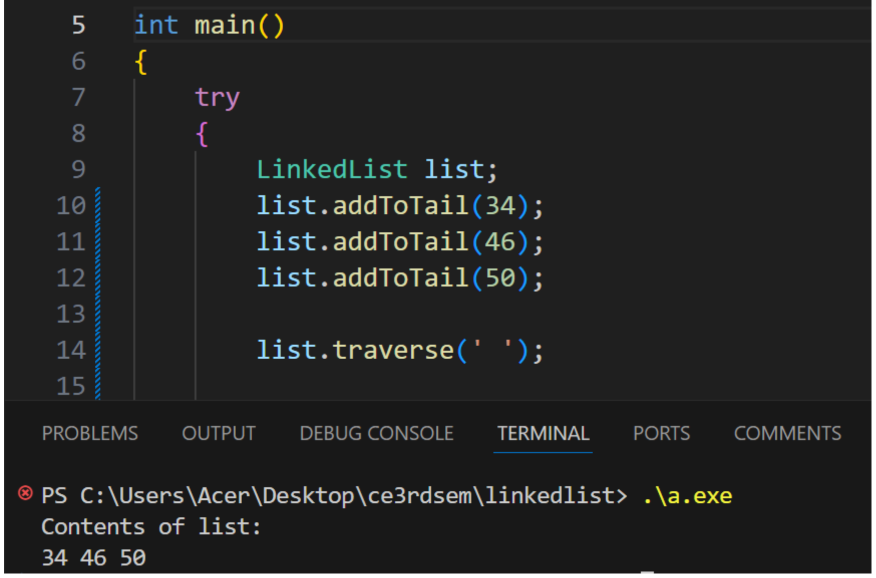
```
PS C:\Users\Acer\Desktop\ce3rdsem\linkedlist> .\a.exe
Contents of list:
50 46 34
```

3. **addToTail(data):**

The addToTail(data) function inserts a new node at the end of the list with its value as data.



```
void LinkedList::addToTail(int data)
{
    Node *newNode = new Node(data);

    if (this->isEmpty())
        HEAD = newNode;
    else
        TAIL->next = newNode;

    TAIL = newNode;
}
```



```
int main()
{
    try
    {
        LinkedList list;
        list.addToTail(34);
        list.addToTail(46);
        list.addToTail(50);

        list.traverse(' ');

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

```
PS C:\Users\Acer\Desktop\ce3rdsem\linkedlist> .\a.exe
Contents of list:
34 46 50
```

4. add(data, predecessor):

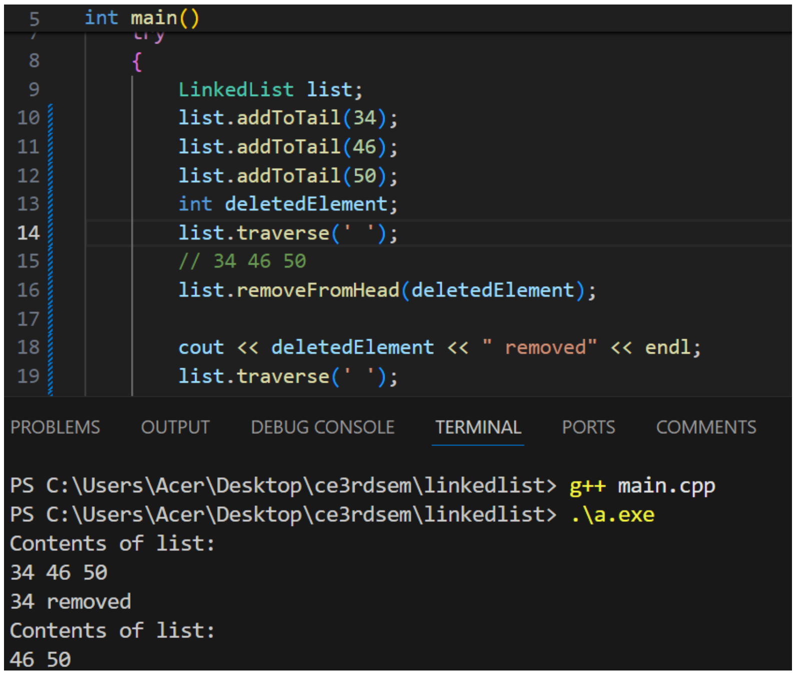
The add(data, predecessor) function inserts an element after the given predecessor node.

```
void LinkedList::add(Node *pred, int data)
{
    if (pred == NULL)
    {
        cout << "Predecessor node is NULL" << endl;
        return;
    }

    Node *newNode = new Node(data, pred->next);
    pred->next = newNode;
}
```

5. removeFromHead():

The removeFromHead() function removes the first node of the list.

```
bool LinkedList::removeFromHead(int &data)
{
    if(!this->isEmpty())
    {
        Node *nodeToDelete = HEAD;
        data = nodeToDelete->info;
        HEAD = HEAD->next;
        if (HEAD == NULL)
        {
            TAIL = NULL;
        }
        delete nodeToDelete;
        return 1;
    }
    throw runtime_error("No data to delete");
    return -1;
}
```



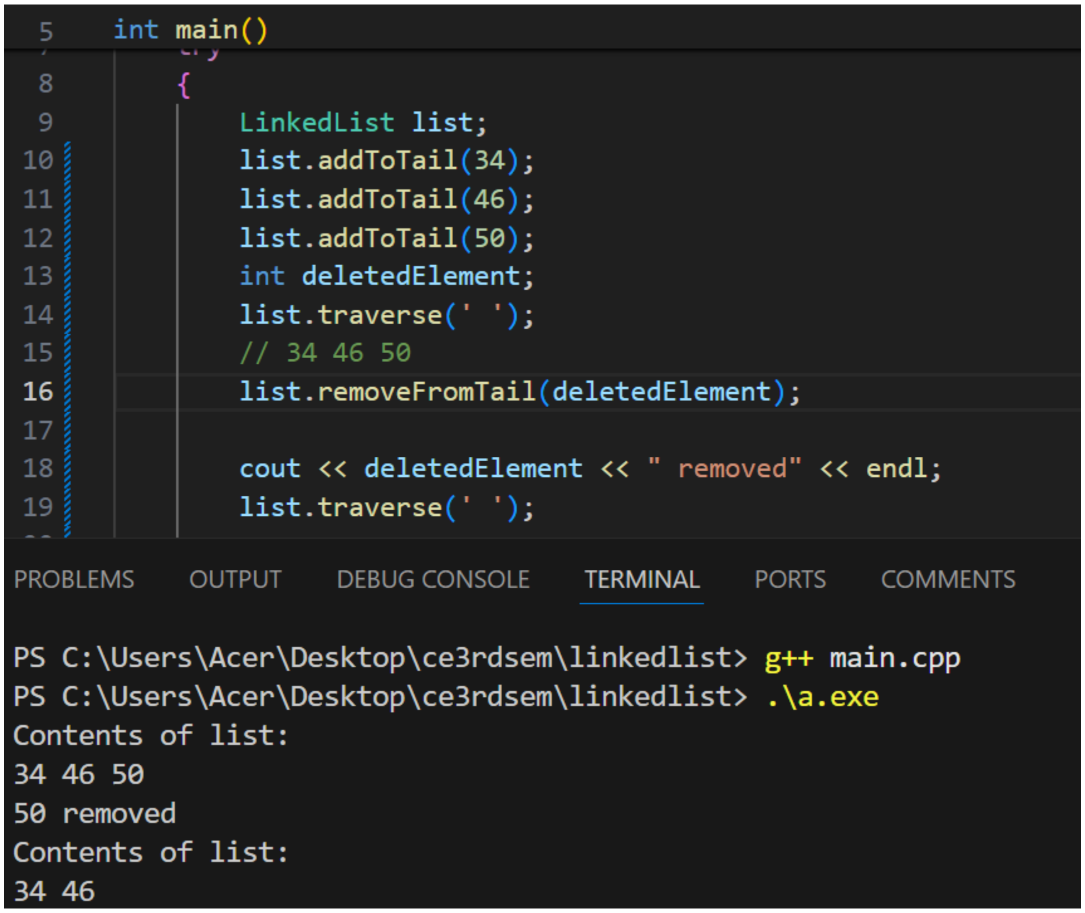
```cpp
int main()
    try
    {
        LinkedList list;
        list.addToTail(34);
        list.addToTail(46);
        list.addToTail(50);
        int deletedElement;
        list.traverse(' ');
        // 34 46 50
        list.removeFromHead(deletedElement);

        cout << deletedElement << " removed" << endl;
        list.traverse(' ');
```

```
PS C:\Users\Acer\Desktop\ce3rdsem\linkedlist> g++ main.cpp
PS C:\Users\Acer\Desktop\ce3rdsem\linkedlist> .\a.exe
Contents of list:
34 46 50
34 removed
Contents of list:
46 50
```

6. removeFromTail():

The removeFromTail() function removes the last node of the list.

```cpp
bool LinkedList::removeFromTail(int &data)
{
    if(!this->isEmpty()) {
        Node *nodeToDelete = TAIL;
        data = nodeToDelete->info;
        if (HEAD==TAIL)
        {
            HEAD = NULL;
            TAIL = NULL;
            delete nodeToDelete;
            return 1;
        }
        Node *temp = HEAD;
        while(temp->next != TAIL)
        {
            temp = temp->next;
        }
        temp->next = NULL;
        TAIL = temp;
        delete nodeToDelete;
        return 1;
    }
    throw runtime_error("No data to delete");
    return -1;
}
```



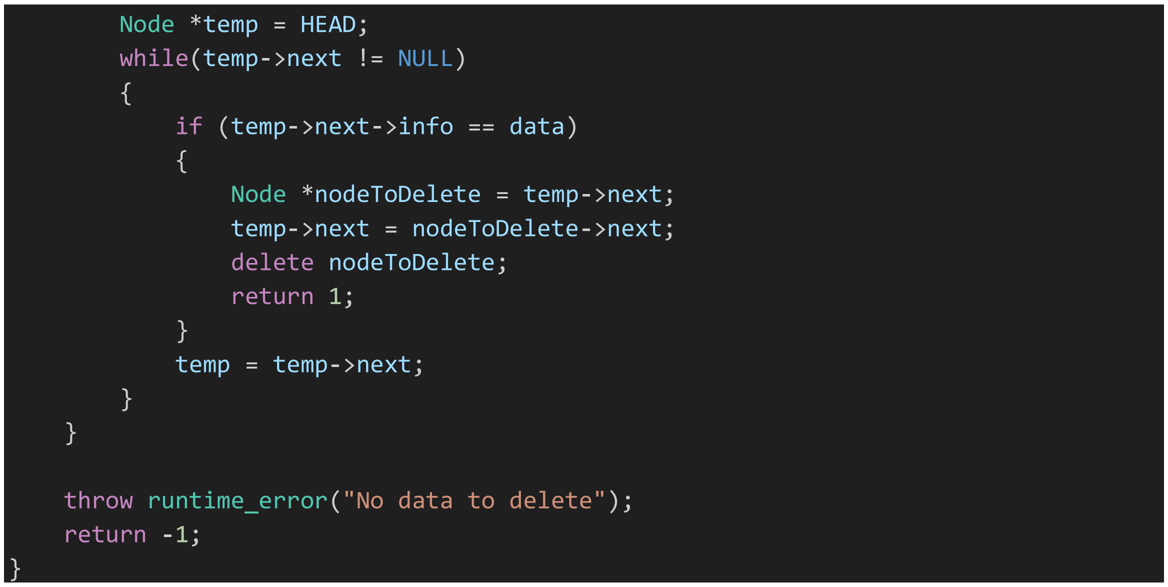
```
int main()
    {
        LinkedList list;
        list.addToTail(34);
        list.addToTail(46);
        list.addToTail(50);
        int deletedElement;
        list.traverse(' ');
        // 34 46 50
        list.removeFromTail(deletedElement);

        cout << deletedElement << " removed" << endl;
        list.traverse(' ');
```

```
PS C:\Users\Acer\Desktop\ce3rdsem\linkedlist> g++ main.cpp
PS C:\Users\Acer\Desktop\ce3rdsem\linkedlist> .\a.exe
Contents of list:
34 46 50
50 removed
Contents of list:
34 46
```

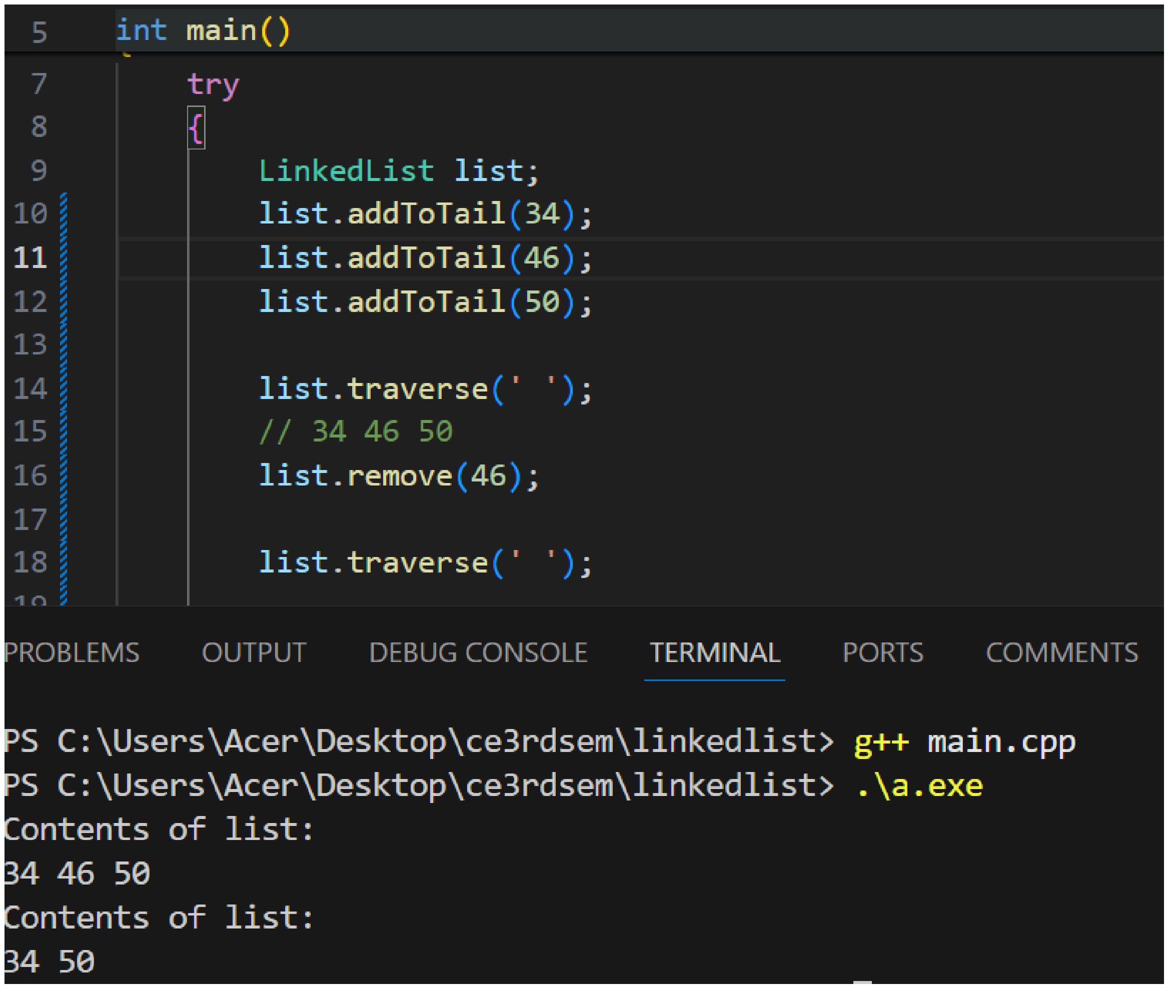
7. **remove(data):**

The remove(data) function takes data as input and removes the node with value data in list.

```
bool LinkedList::remove(int data)
{
    if(!this->isEmpty())
    {
        if (HEAD->info == data)
        {
            int temp;
            return this->removeFromHead(temp);
        }

        if (TAIL->info == data)
        {
            int temp;
            return this->removeFromTail(temp);
        }
```



```
        Node *temp = HEAD;
        while(temp->next != NULL)
        {
            if (temp->next->info == data)
            {
                Node *nodeToDelete = temp->next;
                temp->next = nodeToDelete->next;
                delete nodeToDelete;
                return 1;
            }
            temp = temp->next;
        }
    }

    throw runtime_error("No data to delete");
    return -1;
}
```



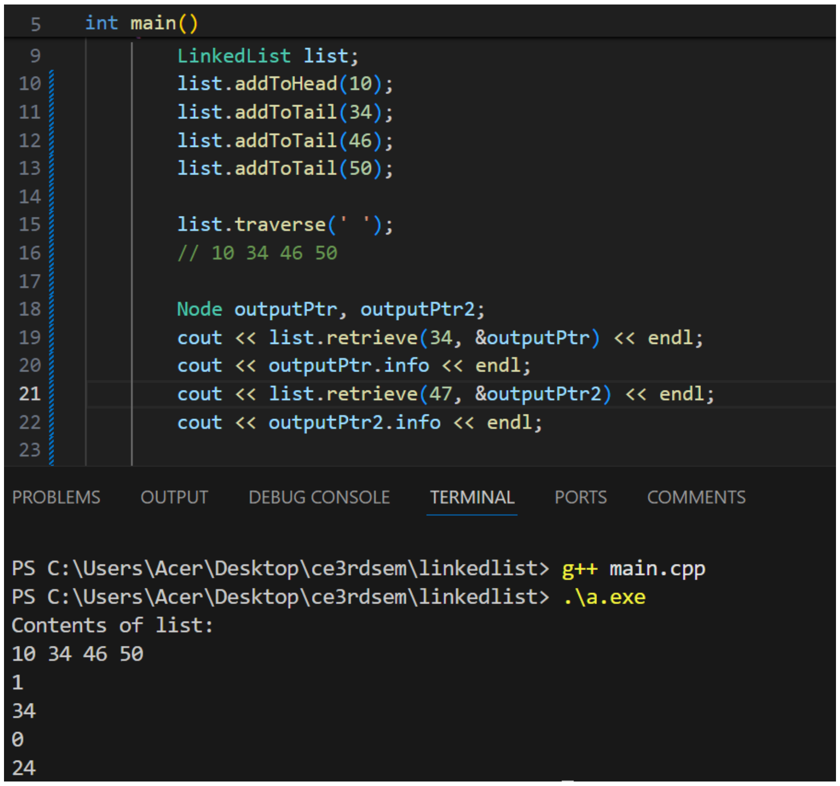
```
int main()
    try
    {
        LinkedList list;
        list.addToTail(34);
        list.addToTail(46);
        list.addToTail(50);

        list.traverse(' ');
        // 34 46 50
        list.remove(46);

        list.traverse(' ');
```

```
PS C:\Users\Acer\Desktop\ce3rdsem\linkedlist> g++ main.cpp
PS C:\Users\Acer\Desktop\ce3rdsem\linkedlist> .\a.exe
Contents of list:
34 46 50
Contents of list:
34 50
```

8. **retrieve(data, outputNodePointer):**

The retrieve(data, outputNodePointer) function helps us to retrieve the pointer of the node that the concerned data points to. When the data is retrieved, it displays the retrieved status message with the code 1 otherwise, status 0 is provided.

```
bool LinkedList::retrieve(int data, Node *outputPtr)
{
    if(!this->isEmpty())
    {
        Node *temp = HEAD;
        while(temp != NULL && temp->info != data)
        {
            temp = temp->next;
        }
        if(temp != NULL)
        {
            outputPtr->info = temp->info;
            outputPtr = temp;
            return true;
        } else {
            return false;
        }
    }
    return false;
}
```



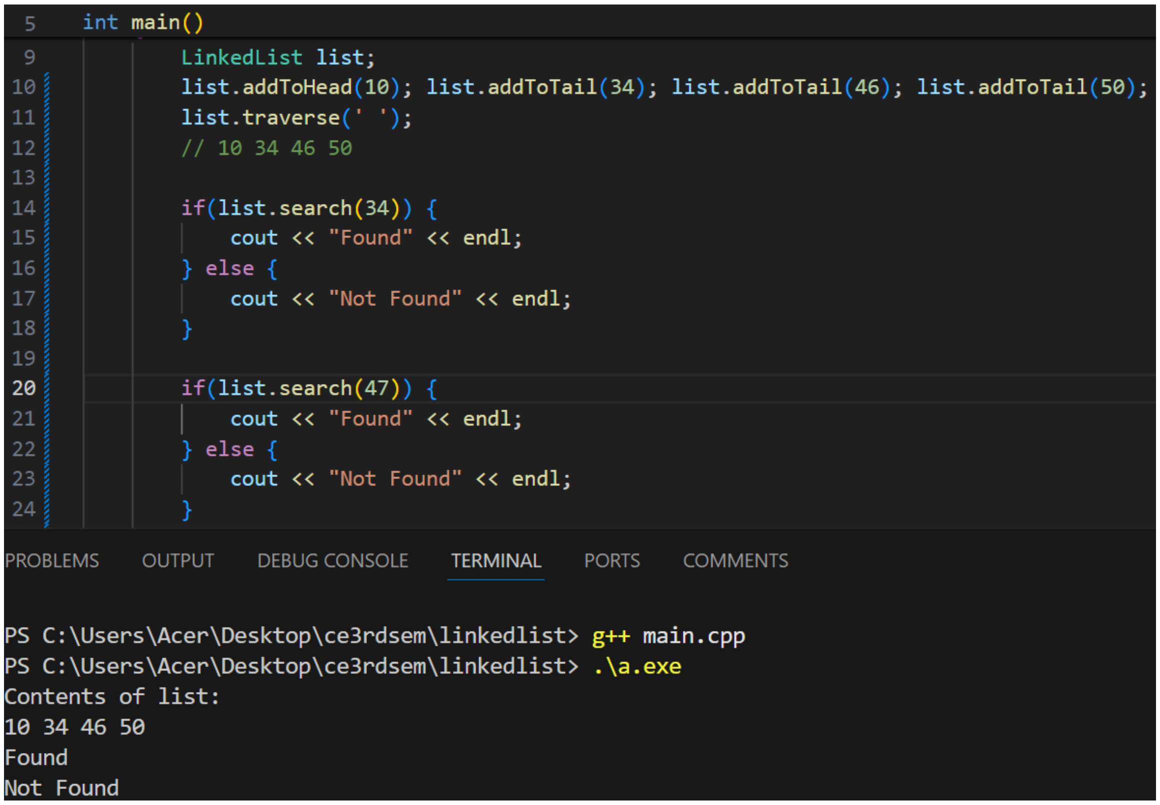
```
int main()
        LinkedList list;
        list.addToHead(10);
        list.addToTail(34);
        list.addToTail(46);
        list.addToTail(50);

        list.traverse(' ');
        // 10 34 46 50

        Node outputPtr, outputPtr2;
        cout << list.retrieve(34, &outputPtr) << endl;
        cout << outputPtr.info << endl;
        cout << list.retrieve(47, &outputPtr2) << endl;
        cout << outputPtr2.info << endl;
```

```
PS C:\Users\Acer\Desktop\ce3rdsem\linkedlist> g++ main.cpp
PS C:\Users\Acer\Desktop\ce3rdsem\linkedlist> .\a.exe
Contents of list:
10 34 46 50
1
34
0
24
```

9. search(data):

The search(data) function checks if the inputted data is in the list or not and returns the message found if it exists else not found.

```cpp
bool LinkedList::search(int data)
{
    if(!this->isEmpty())
    {
        Node *temp = HEAD;
        while(temp != NULL && temp->info != data)
        {
            temp = temp->next;
        }
        if(temp != NULL)
        {
            return true;
        } else {
            return false;
        }
    }

    throw runtime_error("No data to search");
    return false;
}
```



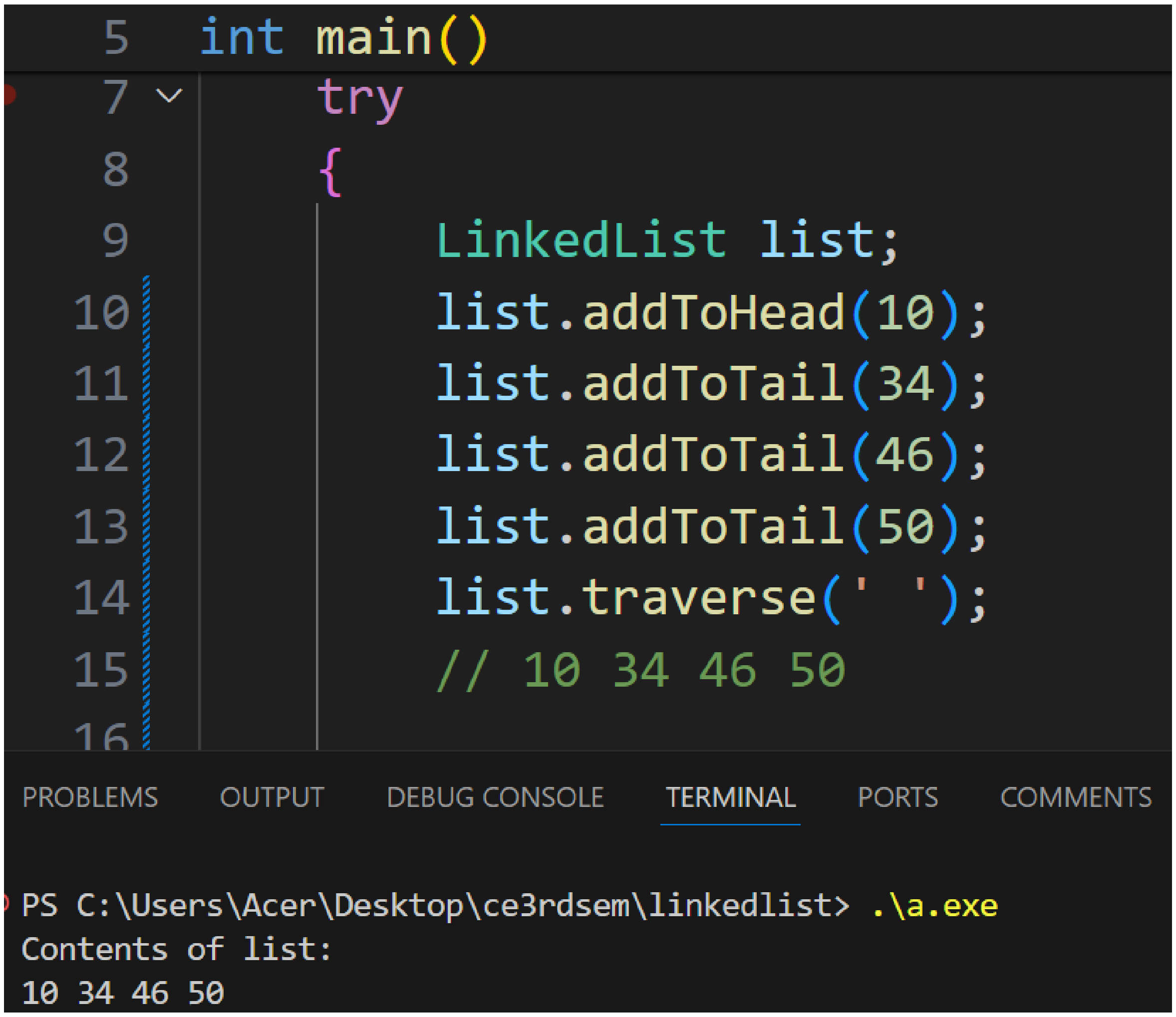
```cpp
int main()
        LinkedList list;
        list.addToHead(10); list.addToTail(34); list.addToTail(46); list.addToTail(50);
        list.traverse(' ');
        // 10 34 46 50

        if(list.search(34)) {
            cout << "Found" << endl;
        } else {
            cout << "Not Found" << endl;
        }

        if(list.search(47)) {
            cout << "Found" << endl;
        } else {
            cout << "Not Found" << endl;
        }
```

```
PS C:\Users\Acer\Desktop\ce3rdsem\linkedlist> g++ main.cpp
PS C:\Users\Acer\Desktop\ce3rdsem\linkedlist> .\a.exe
Contents of list:
10 34 46 50
Found
Not Found
```

10. traverse():

The traverse() function is used to visit every node of the linked list and print the info stored in the node.

```cpp
void LinkedList::traverse(char separator = ' ')
{
    cout << "Contents of list:" << endl;
    for (Node *temp = HEAD; temp != NULL; temp = temp->next)
    {
        cout << temp->info << separator;
    }
    cout << endl;
}
```



```cpp
int main()
    try
    {
        LinkedList list;
        list.addToHead(10);
        list.addToTail(34);
        list.addToTail(46);
        list.addToTail(50);
        list.traverse(' ');
        // 10 34 46 50
```

```
PS C:\Users\Acer\Desktop\ce3rdsem\linkedlist> .\a.exe
Contents of list:
10 34 46 50
```

Conclusion:

Overall, linked lists are efficient and flexible linear data structures with different operations to insert, delete, search elements in the list. It is important data structures as it uses dynamic memory allocation and don’t have problems like shortage of space as in arrays.

Linked list can be used to implement other data structures like stack, queue, arrays, etc.