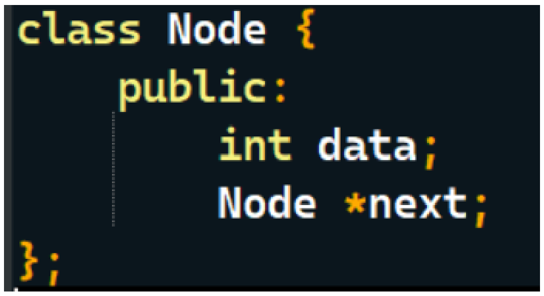
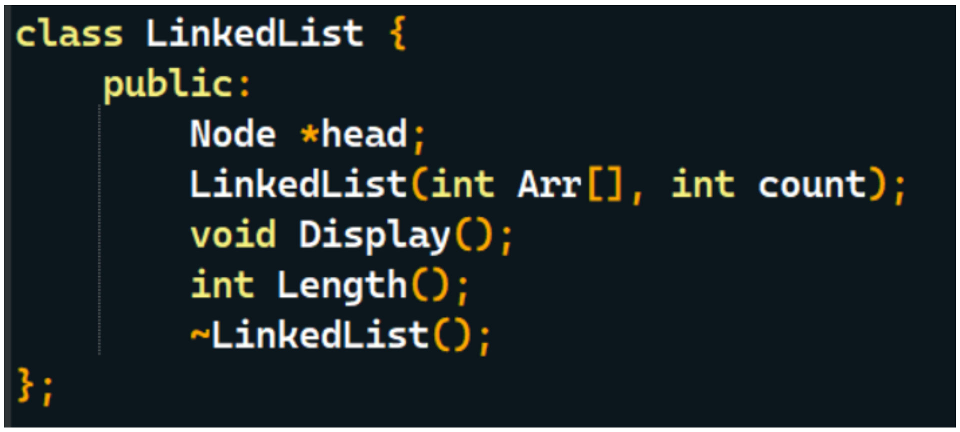
Summary

Sessions (31-01-2023)

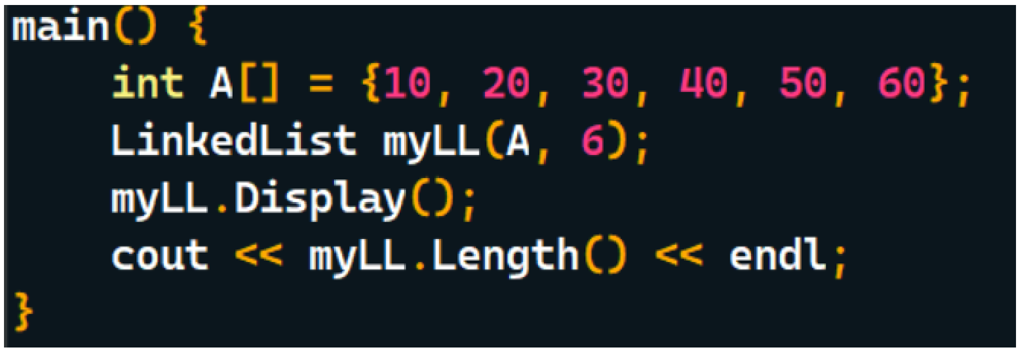
- Create Linked List using class and manage their methods.



```
class Node {
    public:
        int data;
        Node *next;
};
```



```
class LinkedList {
    public:
        Node *head;
        LinkedList(int Arr[], int count);
        void Display();
        int Length();
        ~LinkedList();
};
```



```
main() {
    int A[] = {10, 20, 30, 40, 50, 60};
    LinkedList myLL(A, 6);
    myLL.Display();
    cout << myLL.Length() << endl;
}
```

➔ In above code create head object publicly so we can use it in every method of that class.

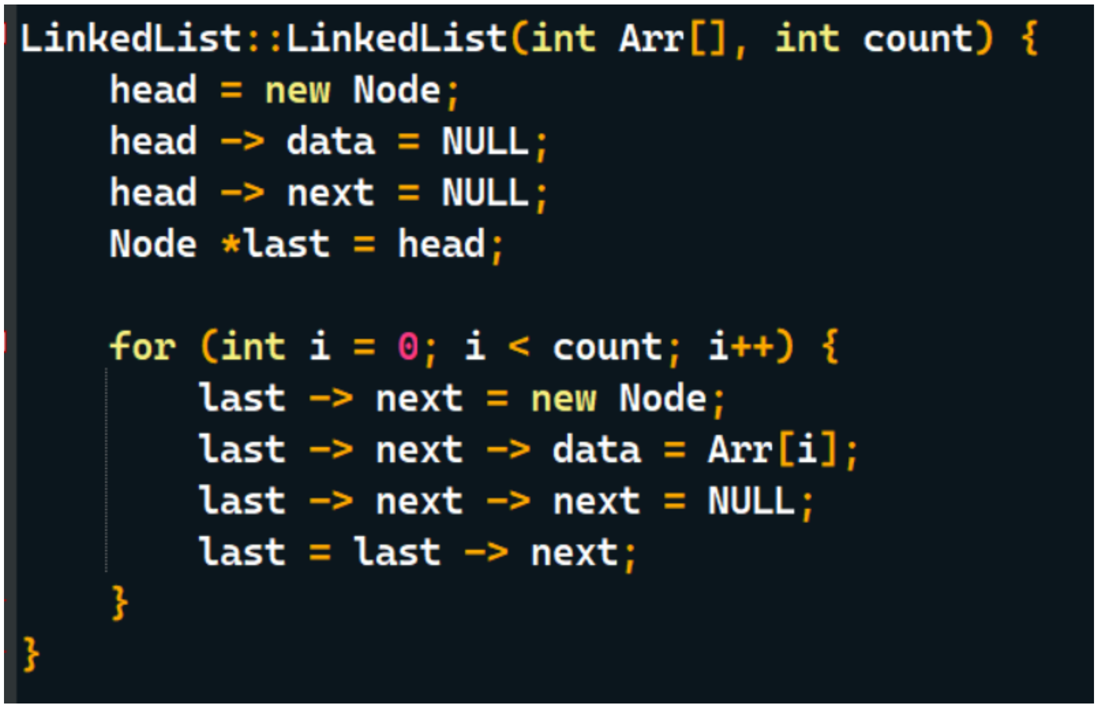
➔ Using a constructor we accept data using array and count of array elements.

➔ Create Display() method for displaying all elements in a linked list.

➔ Create Length() method for count length of linked list.

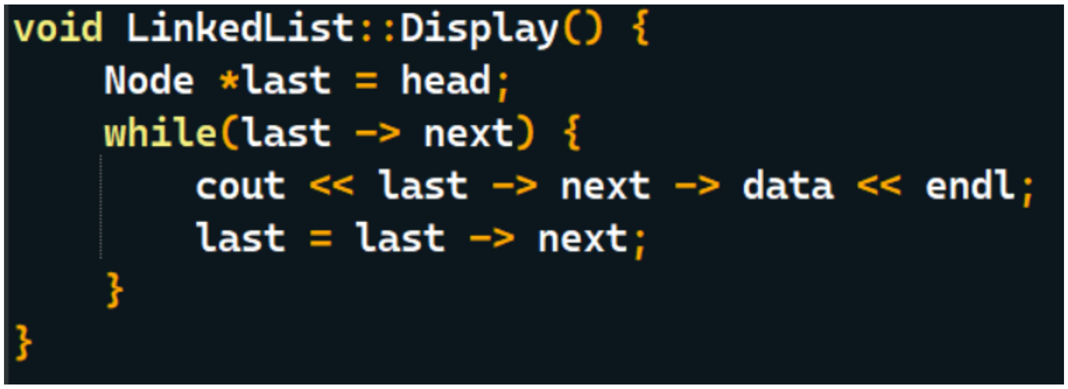
➔ Using a destructor we delete linked lists and their data which is stored in heap memory.

- **Store data using constructor** :



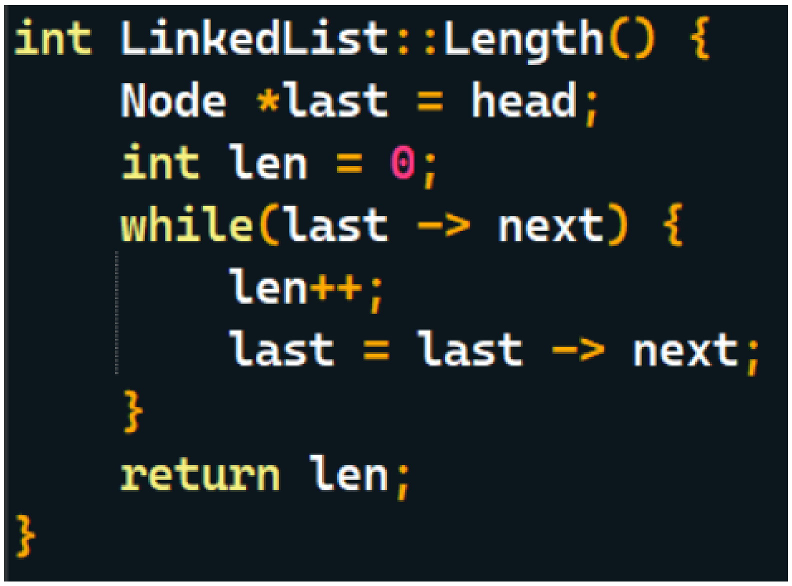
```
LinkedList::LinkedList(int Arr[], int count) {
    head = new Node;
    head -> data = NULL;
    head -> next = NULL;
    Node *last = head;

    for (int i = 0; i < count; i++) {
        last -> next = new Node;
        last -> next -> data = Arr[i];
        last -> next -> next = NULL;
        last = last -> next;
    }
}
```

- **Display Data** :



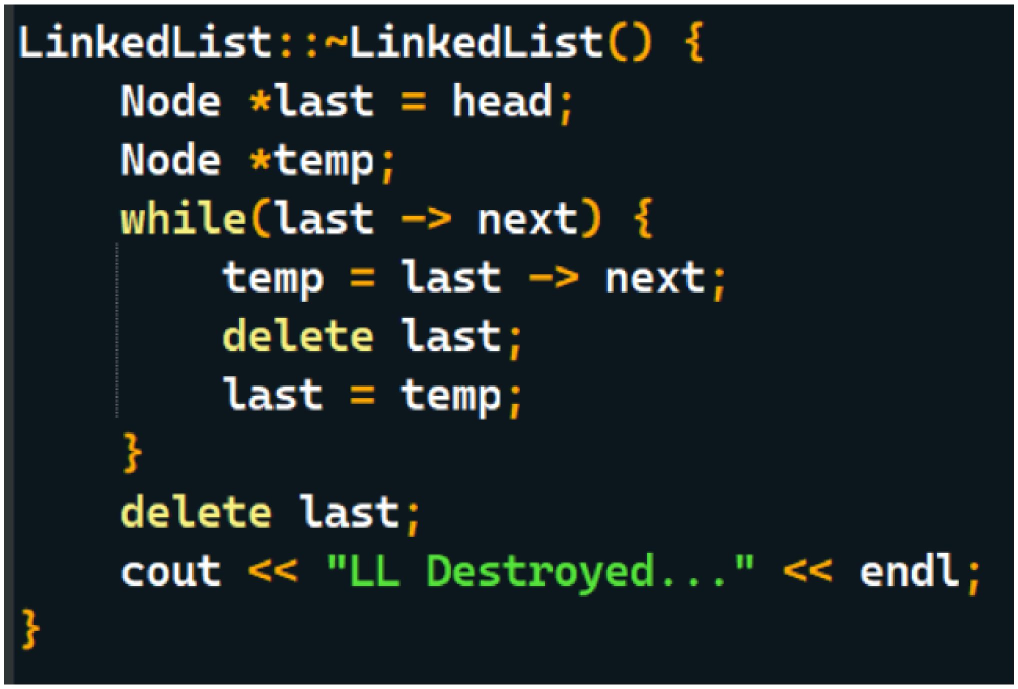
```
void LinkedList::Display() {
    Node *last = head;
    while(last -> next) {
        cout << last -> next -> data << endl;
        last = last -> next;
    }
}
```

- **Length Data** :



```
int LinkedList::Length() {
    Node *last = head;
    int len = 0;
    while(last -> next) {
        len++;
        last = last -> next;
    }
    return len;
}
```

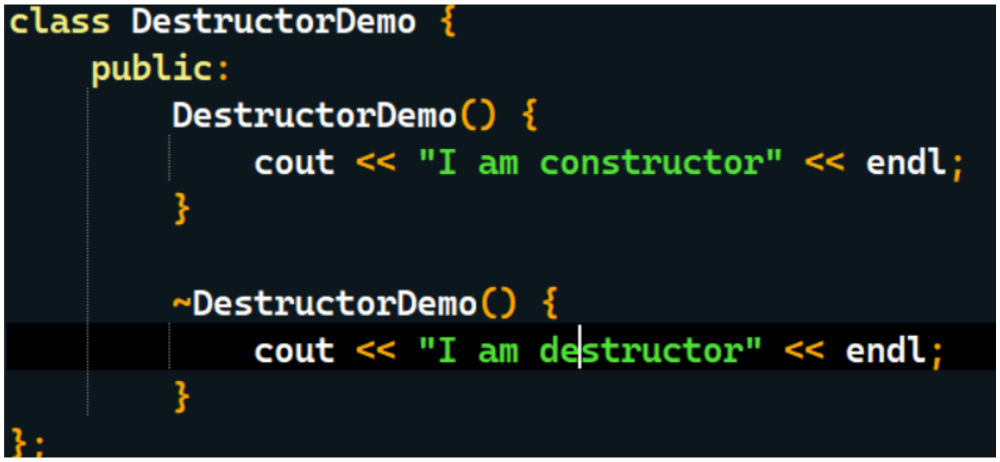
- **Delete Linked List** :



```
LinkedList::~LinkedList() {
    Node *last = head;
    Node *temp;
    while(last -> next) {
        temp = last -> next;
        delete last;
        last = temp;
    }
    delete last;
    cout << "LL Destroyed..." << endl;
}
```

- **Abstract Data type (ADT) :**
    - ➔ We can create data structures along with their operations using class and such data structures that are not in-built are known as Abstract Data Type (ADT).
    - ➔ The definition of ADT only mentions what operations are to be performed but not how these operations will be implemented. It does not specify how data will be organized in memory and what algorithms will be used for implementing the operations.
- **Destructor** :

Destructor is an instance member function which is invoked automatically whenever an object is going to be destroyed. Meaning, a destructor is the last function that is going to be called before an object is destroyed.



```
class DestructorDemo {
    public:
        DestructorDemo() {
            cout << "I am constructor" << endl;
        }

        ~DestructorDemo() {
            cout << "I am destructor" << endl;
        }
};
```