



COMSATS University Islamabad, Vehari Campus

Department of Computer Science

Class: BCS-SP22

Submission Deadline: 9 Oct 2023

Subject: Data Structures and Algorithms-Lab

Instructor: Yasmeen Jana

Max Marks: 20

Reg. No: SP22-BCS-031/Mahreeba

-

Email: yasmeenjana@cuivehari.edu.pk

You can ask queries related to Lab Activities on the above email.

Activity 1:

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

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

    Node(int value) {
        data = value;
        next = nullptr;
    }
};
```

```
    }  
};
```

```
class LinkedList {
```

```
private:
```

```
    Node* head;
```

```
public:
```

```
    LinkedList() {
```

```
        head = nullptr;
```

```
    }
```

```
// Function to add a node at the end of the list
```

```
void insertAtEnd(int value) {
```

```
    Node* newNode = new Node(value);
```

```
    if (head == nullptr) {
```

```
        head = newNode;
```

```
    } else {
```

```
        Node* current = head;
```

```
        while (current->next != nullptr) {
```

```
            current = current->next;
```

```
        }
```

```
        current->next = newNode;
```

```
    }
```

```
}
```

```
// Function to add a node at the beginning of the list
```

```
void insertAtStart(int value) {
```

```
    Node* newNode = new Node(value);
```

```
newNode->next = head;

head = newNode;

}
```

// Function to add a node at a specific index

```
void insertAtIndex(int value, int index) {

    if (index < 0) {

        cout << "Invalid index. Cannot insert at a negative index." << endl;

        return;

    }

    Node* newNode = new Node(value);

    if (index == 0) {

        newNode->next = head;

        head = newNode;

    } else {

        Node* current = head;

        int currentIndex = 0;

        while (current != nullptr && currentIndex < index - 1) {

            current = current->next;

            currentIndex++;

        }

        if (current == nullptr) {

            cout << "Invalid index. Cannot insert at the specified index." << endl;

            return;

        }

        newNode->next = current->next;

        current->next = newNode;

    }

}
```

```
}
```

```
// Function to delete a node at a specific index
```

```
void deleteAtIndex(int index) {
```

```
    if (index < 0) {
```

```
        cout << "Invalid index. Cannot delete at a negative index." << endl;
```

```
        return;
```

```
    }
```

```
    if (head == nullptr) {
```

```
        cout << "List is empty. Cannot delete from an empty list." << endl;
```

```
        return;
```

```
    }
```

```
    if (index == 0) {
```

```
        Node* temp = head;
```

```
        head = head->next;
```

```
        delete temp;
```

```
    } else {
```

```
        Node* current = head;
```

```
        int currentIndex = 0;
```

```
        while (current->next != nullptr && currentIndex < index - 1) {
```

```
            current = current->next;
```

```
            currentIndex++;
```

```
        }
```

```
        if (current->next == nullptr) {
```

```
            cout << "Invalid index. Cannot delete at the specified index." << endl;
```

```
            return;
```

```
        }
```

```
    Node* temp = current->next;

    current->next = current->next->next;

    delete temp;
}
}
```

//Function to del node at first

```
void deleteAtFirst(){
    Node * temp = head;

    head = (head->next);

    delete temp;
}
```

//Function to delete at end

```
void deleteAtEnd(){
    Node * temp=head;

    Node * newlast=temp;

    while (temp->next!= nullptr)
    {
        newlast= temp;

        temp = temp->next;
    }
```

```
    newlast->next= nullptr;

    delete temp;
```

```
}
```

// Function to print the entire linked list

```

void printList() {
    cout<<"The linked list is:"<<endl;
    Node* current = head;
    while (current != nullptr) {
        cout << current->data << " ";
        current = current->next;
    }
    cout<<"\n\n-----"<<endl;
    cout<<"***Head Address:*** " <<head<<endl;

    Node* ptr =head;
    cout<<"-----"<<endl;
    cout<<"ptr address:"<<ptr<<endl;
    cout<<"ptr content:"<<ptr->next<<endl;
    while(ptr != nullptr){
        cout<<"-----"<<endl;
        cout<<"ptr: "<<ptr<<endl;
        cout<<"ptr->next:"<<ptr->next<<endl;
        cout<<"ptr->data:"<<ptr->data<<endl;
        ptr= ptr->next;
    }

}

};

int main() {

```

```
LinkedList myList;
```

```
myList.insertAtStart(30);
```

```
myList.insertAtStart(20);
```

```
myList.insertAtStart(2);
```

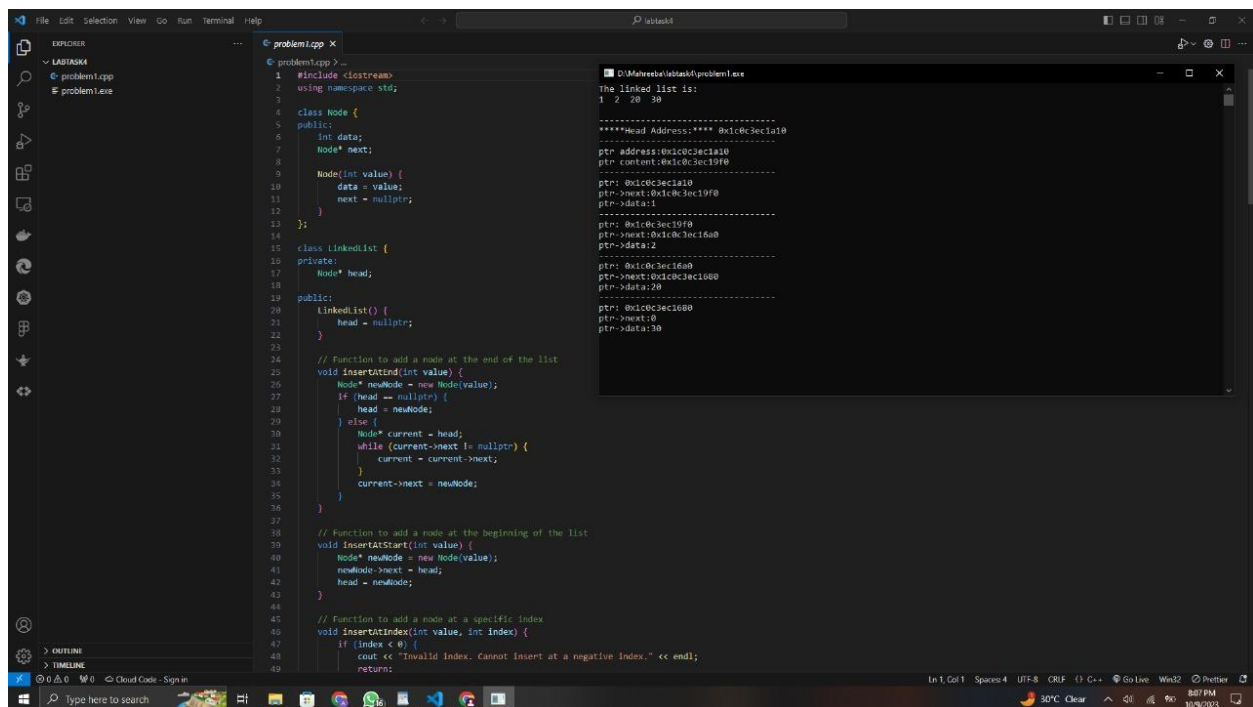
```
myList.insertAtStart(1);
```

```
myList.printList();
```

```
getchar();
```

```
return 0;
```

```
}
```



The screenshot shows a C++ IDE with a source file named `problem1.cpp` and a terminal window displaying the output of the program. The source code implements a `LinkedList` class with a `Node` structure. The `insertAtStart` function is used to add nodes with values 30, 20, 2, and 1 to the beginning of the list. The `printList` function prints the contents of the list, which are 1, 2, 20, and 30. The terminal window shows the output of the program, including the linked list contents and memory addresses for each node.

```
1 #include <iostream>
2 using namespace std;
3
4 class Node {
5 public:
6     int data;
7     Node* next;
8
9     Node(int value) {
10         data = value;
11         next = nullptr;
12     }
13 };
14
15 class LinkedList {
16 private:
17     Node* head;
18
19 public:
20     LinkedList() {
21         head = nullptr;
22     }
23
24     // Function to add a node at the end of the list
25     void insertAtEnd(int value) {
26         Node* newNode = new Node(value);
27         if (head == nullptr) {
28             head = newNode;
29         } else {
30             Node* current = head;
31             while (current->next != nullptr) {
32                 current = current->next;
33             }
34             current->next = newNode;
35         }
36     }
37
38     // Function to add a node at the beginning of the list
39     void insertAtStart(int value) {
40         Node* newNode = new Node(value);
41         newNode->next = head;
42         head = newNode;
43     }
44
45     // Function to add a node at a specific index
46     void insertAtIndex(int value, int index) {
47         if (index < 0) {
48             cout << "Invalid index. Cannot insert at a negative index." << endl;
49             return;
50         }
51     }
52 }
```

```

The linked list is:
1 2 20 30
-----
****Head Address:**** 0x1c03ec1a0
ptr address:0x1c03ec1a0
ptr content:0x1c03ec190
-----
ptr: 0x1c03ec1a0
ptr->next:0x1c03ec190
ptr->data:1
-----
ptr: 0x1c03ec190
ptr->next:0x1c03ec180
ptr->data:2
-----
ptr: 0x1c03ec180
ptr->next:0x1c03ec160
ptr->data:20
-----
ptr: 0x1c03ec160
ptr->next:0
ptr->data:30
-----
```

Activity 2:

Write a program that will implement single, doubly, and circular linked list operations by showing a menu to the user.

The menu should be:

Which linked list you want:

- 1: Single
- 2: Double
- 3: Circular

After the option is chosen by the user:

Which operation you want to perform:

- 1: Insertion
- 2: Deletion
- 3: Display
- 4: Reverse
- 4: Seek
- 5: Exit

Let's suppose, the user has chosen the insertion option then the following menu should be displayed:

- 1: insertion at beginning
- 2: insertion at end
- 3: insertion at the specific data node

A sample output screenshot is below:

Solution:

```
#include <iostream>

using namespace std;

// Node for linked lists

struct Node {
```



```
int data;

Node* next;

Node* prev;

};
```

```
class LinkedList {
```

```
private:
```

```
    Node* head;
```

```
    Node* tail;
```

```
public:
```

```
    LinkedList() {
```

```
        head = nullptr;
```

```
        tail = nullptr;
```

```
    }
```

```
// Function for insertion
```

```
void insertAtBeginning(int value) {
```

```
    Node* newNode = new Node;
```

```
    newNode->data = value;
```

```
    newNode->next = head;
```

```
    newNode->prev = nullptr;
```

```
    if (head != nullptr) {
```

```
        head->prev = newNode;
```

```
    }
```

```
    head = newNode;
```

```
    if (tail == nullptr) {
```

```
        tail = newNode;
    }
}
```

// Function for insertion at end

```
void insertAtEnd(int value) {
    Node* newNode = new Node;
    newNode->data = value;
    newNode->next = nullptr;

    if (tail != nullptr) {
        tail->next = newNode;
        newNode->prev = tail;
    }

    tail = newNode;

    if (head == nullptr) {
        head = newNode;
    }
}
```

// Function to insert at specific location

```
void insertAfter(int target, int value) {
    Node* newNode = new Node;
    newNode->data = value;

    Node* current = head;
```

```

while (current != nullptr && current->data != target) {
    current = current->next;
}

if (current != nullptr) {
    newNode->next = current->next;
    current->next = newNode;
    newNode->prev = current;

    if (newNode->next != nullptr) {
        newNode->next->prev = newNode;
    }

    if (current == tail) {
        tail = newNode;
    }
}
}

```

// Function for deletion of data

```

void deleteNode(int value) {
    Node* current = head;
    Node* previous = nullptr;

    while (current != nullptr && current->data != value) {
        previous = current;
        current = current->next;
    }
}

```

```

if (current != nullptr) {
    if (previous != nullptr) {
        previous->next = current->next;
    } else {
        head = current->next;
    }

    if (current == tail) {
        tail = previous;
    }

    delete current;
}
}

```

// Function to print the list

```

void display() {
    Node* current = head;

    while (current != nullptr) {
        cout << current->data << " ";
        current = current->next;
    }

    cout << endl;
}

```

// Function to reverse the linked list

```

void reverse() {

```

```

cout<<"\nReversing the list...."<<endl;

Node* current = head;

Node* prev = nullptr;

Node* next = nullptr;

while (current != nullptr) {

    next = current->next;

    current->next = prev;

    prev = current;

    current = next;

}

head = prev;

}

// Function to seek a specific data value
bool seek(int value) {

    Node* current = head;

    while (current != nullptr) {

        if (current->data == value) {

            return true;

        }

        current = current->next;

    }

    return false;

}

};

```

```

int main() {

    int choice, subChoice;

    LinkedList sList, dList, cList;

    int insertValue;

    do { //do fpr list selection operation

        cout << "Which linked list you want:\n"

            << "1: Single\n"

            << "2: Double\n"

            << "3: Circular\n"

            << "Select: ";

        cin >> choice;

        switch (choice) {

            case 1:

                do { //after selection of singly

                    cout << "Which operation you want to perform:\n"

                        << "1: Insertion\n"

                        << "2: Deletion\n"

                        << "3: Display\n"

                        << "4: Reverse\n"

                        << "5: Seek\n"

                        << "6: Exit\n"

                        << "Select: ";

                    cin >> subChoice;

                    switch (subChoice) {

                        case 1:

```

```

int typeofinsertion;

cout<<"1. Insertion at beginning\n"

<<"2. Insertion at end\n"

<<"3. Insertion at specific data value\n"

<<"Chose your option: ";

cin>>typeofinsertion;

switch(typeofinsertion){

    case 1:

        cout << "Enter the value to insert: ";

        cin >> insertValue;

        sList.insertAtBeginning(insertValue);

        cout<<"\n Items in list: ";

        sList.display();

        break;

    case 2:

        cout << "Enter the value to insert: ";

        cin >> insertValue;

        sList.insertAtEnd(insertValue);

        cout<<"\n Items in list: ";

        sList.display();

        break;

    case 3:

        int targetval;

        cout << "Enter the value to insert: ";

        cin >> insertValue;

        cout<<"Enter the value where you want to insert: ";

        cin>>targetval;

        sList.insertAfter(targetval,insertValue);

        cout<<"\n Items in list: ";

```

```

        sList.display();

        break;

default:

    cout<<"You didn't selected appropriate option"<<endl;

    break;

}

break;

case 2:

    int deleteValue;

    cout << "Enter the value to delete: ";

    cin >> deleteValue;

    sList.deleteNode(deleteValue);

    break;

case 3:

    cout << "Single Linked List: ";

    sList.display();

    break;

case 4:

    sList.reverse();

    break;

case 5:

    int seekValue;

    cout << "Enter the value to seek: ";

    cin >> seekValue;

    if (sList.seek(seekValue)) {

        cout << "Value found in Single Linked List.\n";

    } else {

        cout << "Value not found in Single Linked List.\n";

    }

```



```

        break;
    case 6:
        cout << "Exiting Single Linked List menu.\n";
        break;
    default:
        cout << "Invalid choice. Please try again.\n";
        break;
    }
} while (subChoice != 6);
break;

```

case 2:

```

do { //after slection of doubly
    cout << "Which operation you want to perform:\n"
        << "1: Insertion\n"
        << "2: Deletion\n"
        << "3: Display\n"
        << "4: Reverse\n"
        << "5: Seek\n"
        << "6: Exit\n"
        << "Enter your choice : ";
    cin >> subChoice;
}

```

```

switch (subChoice) {

```

```

    case 1:

```

```

        int typeofinsertion;
        cout<<"1. Insertion at beigning\n"
            <<"2. Insertion at end\n"

```

```

    <<"3. Insertion at specific data value\n"
    <<"Chose your option: ";
cin>>typeofinsertion;
switch(typeofinsertion){
    case 1:
        cout << "Enter the value to insert: ";
        cin >> insertValue;
        dList.insertAtBeginning(insertValue);
        cout<<"\n Items in list: ";
        dList.display();
        break;
    case 2:
        cout << "Enter the value to insert: ";
        cin >> insertValue;
        dList.insertAtEnd(insertValue);
        cout<<"\n Items in list: ";
        dList.display();
        break;
    case 3:
        int targetval;
        cout << "Enter the value to insert: ";
        cin >> insertValue;
        cout<<"ENter the value where you want to insert: ";
        cin>>targetval;
        dList.insertAfter(targetval,insertValue);
        cout<<"\n Items in list: ";
        dList.display();
        break;
    default:

```

```

        cout<<"You didn't selected appropriate option"<<endl;
        break;
    }break;
case 2:
    int deleteValue;
    cout << "Enter the value to delete: ";
    cin >> deleteValue;
    dList.deleteNode(deleteValue);
    break;
case 3:
    cout << "Double Linked List: ";
    dList.display();
    break;
case 4:
    dList.reverse();
    break;
case 5:
    int seekValue;
    cout << "Enter the value to seek: ";
    cin >> seekValue;
    if (dList.seek(seekValue)) {
        cout << "Value found in Double Linked List.\n";
    } else {
        cout << "Value not found in Double Linked List.\n";
    }
    break;
case 6:
    cout << "Exiting Double Linked List menu.\n";
    break;

```

```

        default:
            cout << "Invalid choice. Please try again.\n";
            break;
    }
} while (subChoice != 6);
break;

```

case 3:

```

do { //after selection of circular
    cout << "Which operation you want to perform:\n"
        << "1: Insertion\n"
        << "2: Deletion\n"
        << "3: Display\n"
        << "4: Reverse\n"
        << "5: Seek\n"
        << "6: Exit\n"
        << "Enter your choice: ";
    cin >> subChoice;
}

```

```

switch (subChoice) {
    case 1:
        int typeofinsertion;
        cout<<"1. Insertion at beigning\n"
            <<"2. Insertion at end\n"
            <<"3. Insertion at specific data value\n"
            <<"Chose your option: ";
        cin>>typeofinsertion;
        switch(typeofinsertion){
            case 1:

```

```
cout << "Enter the value to insert: ";
```

```
cin >> insertValue;
```

```
cList.insertAtBeginning(insertValue);
```

```
cout<<"\n Items in list: ";
```

```
cList.display();
```

```
break;
```

```
case 2:
```

```
cout << "Enter the value to insert: ";
```

```
cin >> insertValue;
```

```
cList.insertAtEnd(insertValue);
```

```
cout<<"\n Items in list: ";
```

```
cList.display();
```

```
break;
```

```
case 3:
```

```
int targetval;
```

```
cout << "Enter the value to insert: ";
```

```
cin >> insertValue;
```

```
cout<<"ENter the value where you want to insert: ";
```

```
cin>>targetval;
```

```
cList.insertAfter(targetval,insertValue);
```

```
cout<<"\n Items in list: ";
```

```
cList.display();
```

```
break;
```

```
default:
```

```
cout<<"You didn't selected appropriate option"<<endl;
```

```
break;
```

```
}break;
```

```
case 2:
```

```

        int deleteValue;

        cout << "Enter the value to delete: ";

        cin >> deleteValue;

        cList.deleteNode(deleteValue);

        break;
    case 3:

        cout << "Circular Linked List: ";

        cList.display();

        break;
    case 4:

        cout << "Cannot reverse a circular linked list.\n";

        break;
    case 5:

        int seekValue;

        cout << "Enter the value to seek: ";

        cin >> seekValue;

        if (cList.seek(seekValue)) {

            cout << "Value found in Circular Linked List.\n";

        } else {

            cout << "Value not found in Circular Linked List.\n";

        }

        break;
    case 6:

        cout << "Exiting Circular Linked List menu.\n";

        break;
    default:

        cout << "Invalid choice. Please try again.\n";

        break;
}

```

```
} while (subChoice != 6);
```

```
break;
```

default:

```
cout << "Invalid choice. Please try again.\n";
```

```
break;
```

```
}
```

```
} while (choice != 4);
```

```
cout << "Exiting the program.\n";
```

```
return 0;
```

```
}
```

```
196         cin >> insertValue;
197         slist.insertAtEnd(insertValue);
198         cout << "\n Items in list: ";
199         slist.display();
200         break;
201
202     case 3:
203         int targetval;
204         cout << "Enter the value to insert: ";
205         cin >> insertValue;
206         cout << "Enter the value where you want to insert: ";
207         cin >> targetval;
208         slist.insertAfter(targetval, insertValue);
209         cout << "\n Items in list: ";
210         slist.display();
211         break;
212     default:
213         cout << "You didn't selected appropriate option" << endl;
214         break;
215 }
216 break;
217
218 case 2:
219     int deleteValue;
220     cout << "Enter the value to delete: ";
221     cin >> deleteValue;
222     slist.deleteNode(deleteValue);
223     break;
224 case 3:
225     cout << "Single linked List: ";
226     slist.display();
227     break;
228 case 4:
229     slist.reverse();
230     break;
231 case 5:
232     int seekValue;
233     cout << "Enter the value to seek: ";
234     cin >> seekValue;
235     if (slist.seek(seekValue)) {
236         cout << "Value found in Single Linked List.\n";
237     } else {
238         cout << "Value not found in Single linked List.\n";
239     }
240     break;
241 case 6:
242     cout << "Exiting Single Linked List menu.\n";
243     break;
244 default:
245     cout << "Invalid choice. Please try again.\n";
246     break;
```

Output:

```
Which linked list you want:
1: Single
2: Double
3: Circular
Select: 1
Which operation you want to perform:
1: Insertion
2: Deletion
3: Display
4: Reverse
5: Seek
6: Exit
Select: 1
1. Insertion at beigning
2. Insertion at end
3. Insertion at specific data value
Chose your option: 1
Enter the value to insert: 11
Items in list: 11
Which operation you want to perform:
1: Insertion
2: Deletion
3: Display
4: Reverse
5: Seek
6: Exit
Select:
```

You can get help from the below link:

<https://github.com/programming-debug/Data-Structure-Lab/blob/main/Lab3/single-link%20list.cpp>

In this Word file, you should place the code and its output screenshot.

After completing the activities, Upload the final pdf and cpp code files to the **“DSA_Lab”** repository.