COMSATS University Islamabad

Vehari Campus

Course Title :	**Data Structure**
Registration No :	**SP22-BCS-012**
Assignment No :	**02**
Submitted By :	**Aqsa Asghar**
Submitted To :	**Mam Yasmeen Jana**
Section :	**BCS-22A**
Submission Date :	**09th Oct, 2023**

Activity: 1

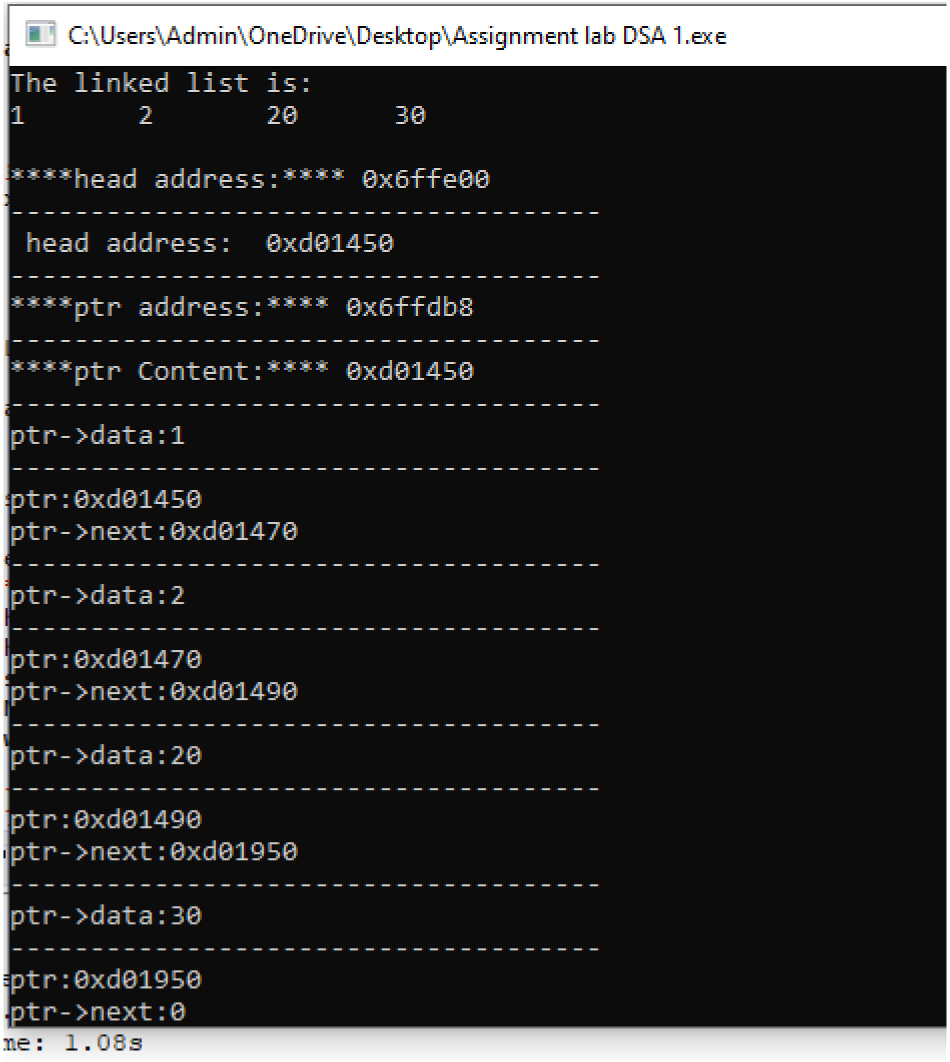
Code:

```
#include <iostream>

using namespace std; // Add this line to include the std namespace
class Node {
public:
    int data;
    Node* next;
    Node(int value) : data(value), next(NULL) {}
};
class LinkedList {
private:
    Node* head;
public:
    LinkedList() : head(NULL) {}
    void insert(int value) {
        Node* newNode = new Node(value);
        if (head == NULL) {
            head = newNode;
        } else {
            Node* current = head;
            while (current->next != NULL) {
                current = current->next;
            }
            current->next = newNode;
        }
    }

    void display() {
```

```
// Check if the linked list is empty.
if (head == NULL) {
   cout << "The linked list is empty." << endl;
   return;
}
// Create a temporary pointer to traverse the linked list.
Node* ptr = head;
cout << "The linked list is:" << endl;
while (ptr != NULL) {
   cout << ptr->data << "\t";
   ptr = ptr->next;
}
cout<<endl;
              cout<<endl;

ptr = head;



cout << "****head address:**** " << &head << endl;
cout<<"------------------------------------"<<endl;
cout << " head address:  " << head << endl;
cout<<"------------------------------------"<<endl;


              cout << "****ptr address:**** " << &ptr << endl;
                    cout<<"------------------------------------"<<endl;


cout << "****ptr Content:**** " << ptr << endl;
```

```
        while (ptr != NULL) {
                          cout<<"------------------------------------"<<endl;
                              cout << "ptr->data:" << ptr->data << endl;
                              cout<<"------------------------------------"<<endl;
            cout << "ptr:" << ptr << endl;
            cout << "ptr->next:" << ptr->next << endl;

            ptr = ptr->next;
        }
    }
    ~LinkedList() {
        Node* current = head;
        while (current != NULL) {
            Node* next = current->next;
            delete current;
            current = next;
        }
    }
};
int main() {
    LinkedList list;
    list.insert(1);
    list.insert(2);
    list.insert(20);
             list.insert(30);
    list.display();
    return 0;
}
```

Output Screen Short:



```
The linked list is:
1       2       20      30

****head address:**** 0x6ffe00
-------------------------------------
 head address:  0xd01450
-------------------------------------
****ptr address:**** 0x6ffdb8
-------------------------------------
****ptr Content:**** 0xd01450
-------------------------------------
ptr->data:1
-------------------------------------
ptr:0xd01450
ptr->next:0xd01470
-------------------------------------
ptr->data:2
-------------------------------------
ptr:0xd01470
ptr->next:0xd01490
-------------------------------------
ptr->data:20
-------------------------------------
ptr:0xd01490
ptr->next:0xd01950
-------------------------------------
ptr->data:30
-------------------------------------
ptr:0xd01950
ptr->next:0
me: 1.08s
```

Activity: 2

Code:

```
#include <iostream>

using namespace std;


// Node structure for a linked list

struct Node {

   int data;

   Node* next;

   Node* prev; // For doubly linked list

};


class LinkedList {

private:

   Node* head; // Points to the head of the list

   Node* tail; // Points to the tail of the list (for circular and doubly linked lists)


public:

   // Constructor

   LinkedList() {

      head = NULL;

      tail = NULL;

   }
```

```
// Function to insert a node at the beginning
void insertAtBeginning(int value) {
    Node* newNode = new Node;
    newNode->data = value;
    newNode->next = head;

    // For doubly linked list
    if (head != NULL)
        head->prev = newNode;

    head = newNode;

    // For circular linked list
    if (tail == NULL)
        tail = head;

    // Update tail's next to point to the head
    tail->next = head;

    cout << "Insertion at the beginning successful\n";
}

// Function to insert a node at the end
void insertAtEnd(int value) {
    Node* newNode = new Node;
    newNode->data = value;
    newNode->next = NULL;

    // For doubly linked list
    newNode->prev = tail;

    if (tail != NULL)
        tail->next = newNode;

    tail = newNode;

    // For circular linked list
    if (head == NULL)
        head = tail;

    // Update tail's next to point to the head
    tail->next = head;

    cout << "Insertion at the end successful\n";
}

// Function to insert a node at a specific position
void insertAtPosition(int value, int position) {
```

```
Node* newNode = new Node;
newNode->data = value;

if (position == 1) {
    newNode->next = head;
    head = newNode;
} else {
    Node* temp = head;
    for (int i = 1; i < position - 1 && temp != NULL; ++i) {
        temp = temp->next;
    }

    if (temp == NULL) {
        cout << "\nInvalid position\n";
        return;
    }

    newNode->next = temp->next;
    temp->next = newNode;

    // For doubly linked list
    newNode->prev = temp;

    // For circular linked list
```

```
        if (newNode->next == NULL)
            tail = newNode;
    }

    cout << "Insertion at position " << position << " successful\n";
}

// Function to delete a node
void deleteNode(int value) {
    Node* temp = head;
    Node* prev = NULL;

    // Find the node to be deleted
    while (temp != NULL && temp->data != value) {
        prev = temp;
        temp = temp->next;
    }

    if (temp == NULL) {
        cout << "\nNode not found\n";
        return;
    }

    // Update links to skip the node to be deleted
    if (prev != NULL)
        prev->next = temp->next;
    else
        head = temp->next;

    // For doubly linked list
    if (temp->next != NULL)
        temp->next->prev = prev;

    // For circular linked list
    if (temp->next == NULL)
        tail = prev;

    delete temp;

    cout << "Deletion of node with value " << value << " successful\n";
}

// Function to display the linked list
void display() {
    Node* temp = head;
    cout << "\nLinked List: ";
    if (head != NULL) {
        do {
            cout << temp->data << " ";
            temp = temp->next;
        } while (temp != head);
    }
    cout << endl;
}

// Function to reverse the linked list
void reverse() {
    if (head == NULL || head->next == head)
        return;

    Node* current = head;
    Node* prev = NULL;
    Node* next = NULL;

    do {
        next = current->next;
        current->next = prev;
        current->prev = next; // For doubly linked list
        prev = current;
        current = next;
    } while (current != head);
```

```
        // Update tail and head after reversing
        head = prev;
        tail = head->next;

        cout << "Reversal successful\n";
    }

    // Function to search for a node
    bool seek(int value) {
        Node* temp = head;

        if (head != NULL) {
            do {
                if (temp->data == value)
                    return true;
                temp = temp->next;
            } while (temp != head);
        }

        return false;
    }
};

int main() {
```

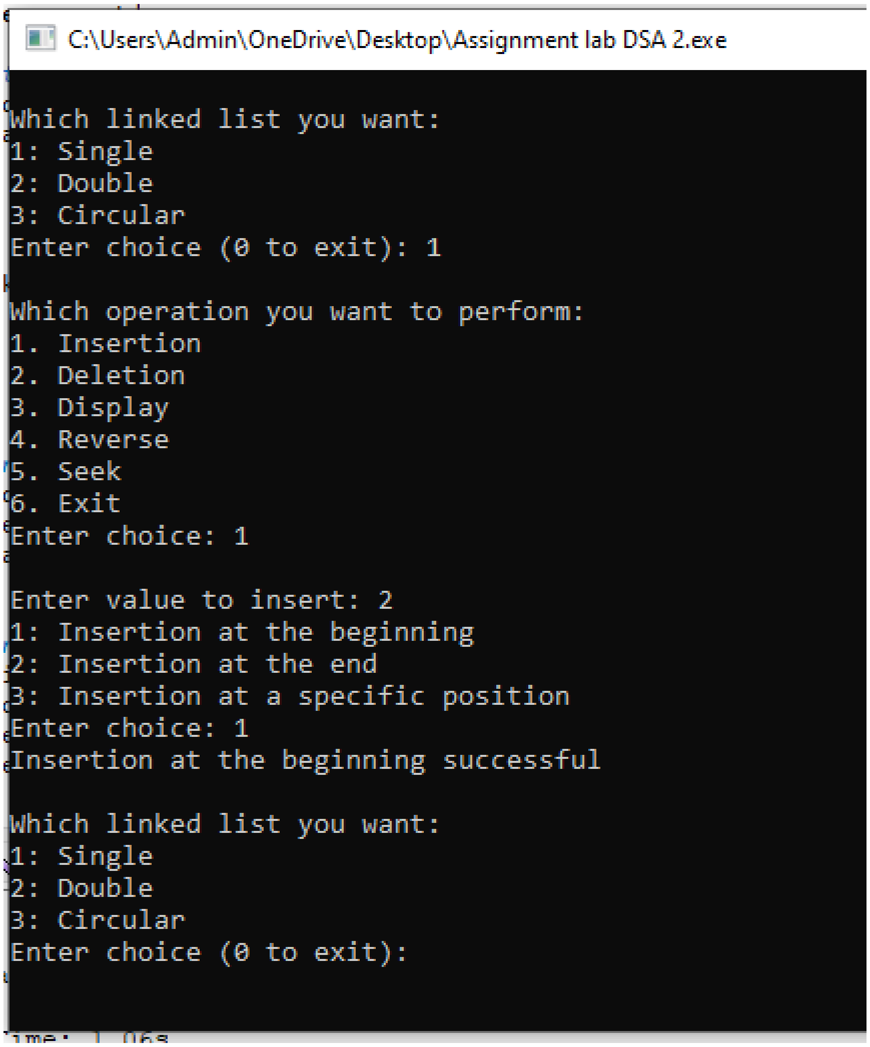
```
LinkedList singleList;
LinkedList doubleList;
LinkedList circularList;

int choice;
int listType;

do {
    cout << "\nWhich linked list you want:\n";
    cout << "1: Single\n";
    cout << "2: Double\n";
    cout << "3: Circular\n";
    cout << "Enter choice (0 to exit): ";
    cin >> listType;

    if (listType == 0)
        break;

    cout << "\nWhich operation you want to perform:\n";
    cout << "1. Insertion\n";
    cout << "2. Deletion\n";
    cout << "3. Display\n";
    cout << "4. Reverse\n";
    cout << "5. Seek\n";
cout << "6. Exit\n";

cout << "Enter choice: ";

cin >> choice;


int value, position;


switch (choice) {

    case 1:

        cout << "\nEnter value to insert: ";

        cin >> value;


        cout << "1: Insertion at the beginning\n";

        cout << "2: Insertion at the end\n";

        cout << "3: Insertion at a specific position\n";

        cout << "Enter choice: ";

        cin >> choice;


        switch (choice) {

            case 1:

                if (listType == 1)

                    singleList.insertAtBeginning(value);

                else if (listType == 2)

                    doubleList.insertAtBeginning(value);

                else if (listType == 3)
        circularList.insertAtBeginning(value);
    break;

case 2:
    if (listType == 1)
        singleList.insertAtEnd(value);
    else if (listType == 2)
        doubleList.insertAtEnd(value);
    else if (listType == 3)
        circularList.insertAtEnd(value);
    break;

case 3:
    cout << "\nEnter position to insert: ";
    cin >> position;
    if (listType == 1)
        singleList.insertAtPosition(value, position);
    else if (listType == 2)
        doubleList.insertAtPosition(value, position);
    else if (listType == 3)
        circularList.insertAtPosition(value, position);
    break;

default:
```

```
            cout << "Invalid choice\n";
    }
    break;

case 2:
    cout << "\nEnter value to delete: ";
    cin >> value;
    if (listType == 1)
        singleList.deleteNode(value);
    else if (listType == 2)
        doubleList.deleteNode(value);
    else if (listType == 3)
        circularList.deleteNode(value);
    break;

case 3:
    if (listType == 1)
        singleList.display();
    else if (listType == 2)
        doubleList.display();
    else if (listType == 3)
        circularList.display();
    break;
```

```
case 4:
    if (listType == 1)
        singleList.reverse();
    else if (listType == 2)
        doubleList.reverse();
    else if (listType == 3)
        circularList.reverse();
    break;

case 5:
    cout << "\nEnter value to seek: ";
    cin >> value;
    if (listType == 1)
        cout << (singleList.seek(value) ? "Value found" : "Value not found") << endl;
    else if (listType == 2)
        cout << (doubleList.seek(value) ? "Value found" : "Value not found") << endl;
    else if (listType == 3)
        cout << (circularList.seek(value) ? "Value found" : "Value not found") << endl;
    break;

case 6:
    break;

default:
```

```
            cout << "Invalid choice\n";
        }
    } while (choice != 6);
    return 0;
}
```

Output Screen Short:



```
Which linked list you want:
1: Single
2: Double
3: Circular
Enter choice (0 to exit): 1

Which operation you want to perform:
1. Insertion
2. Deletion
3. Display
4. Reverse
5. Seek
6. Exit
Enter choice: 1

Enter value to insert: 2
1: Insertion at the beginning
2: Insertion at the end
3: Insertion at a specific position
Enter choice: 1
Insertion at the beginning successful

Which linked list you want:
1: Single
2: Double
3: Circular
Enter choice (0 to exit):
```