SUPERIOR UNIVERSITY

Name :

TAMOOR AKRAM

Roll :

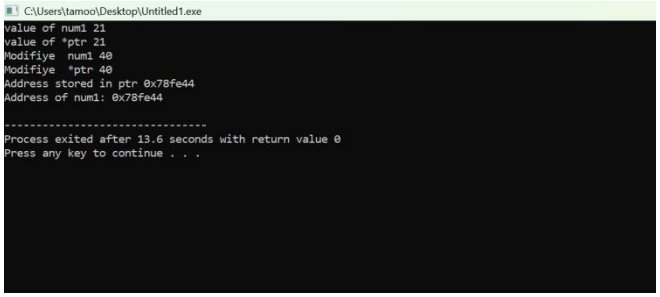
SU92-BSSEM-S24-012

Submit To:

Rasikh Ali

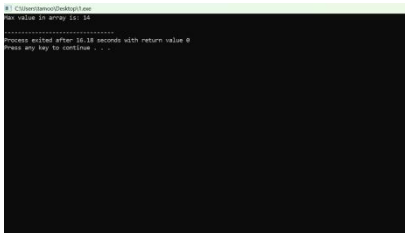
Task 1:

```
#include <iostream>
using namespace std;
int main() {
   int num1 =21;
   int* ptr = &num1;
   cout << "value of num1 " << num1<< endl;
   cout << "value of *ptr "<< *ptr << endl;
   *ptr =40;
   cout << "Modifiye  num1 " << num1 << endl;
   cout << "Modifiye  *ptr " << *ptr << endl;
   cout << "Address stored in ptr " << ptr << endl;
   cout << "Address of num1: " << &num1 << endl;
   return 0;
}
```



Task2

```
#include <iostream>
using namespace std;
int findMax(int arr[], int n) {
   int maxVal = arr[0];
   for (int i = 1; i < n; i++) {
   if (arr[i] > maxVal) {
   maxVal = arr[i];}
   }
   return maxVal;
}
int main() {
   int arr[] = {1,2,3,7,8,9,10 ,11, 12 ,13, 14,};
   int n = sizeof(arr) / sizeof(arr[0]);
   int max = findMax(arr, n);
   cout << "Max value in array is: " << max << endl;
   return 0;
}
```



Task

```
#include <iostream>
using namespace std;
struct Node {
   int data;
   Node* next;
   Node(int val) {
      data = val;
      next = nullptr;
   }
};
class SinglyLinkedList {
private:
   Node* head;
public:
   SinglyLinkedList() {
      head = nullptr;
   }
   void insertAtStart(int val) {
      Node* newNode = new Node(val);
      newNode->next = head;
      head = newNode;
      displayList();
   }
   void insertAtEnd(int val) {
      Node* newNode = new Node(val);
      if (head == nullptr) {
         head = newNode;
      } else {
         Node* temp = head;
```

```
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
    displayList();
}
void displayList() {
    if (head == nullptr) {
        cout << "The list is empty." << endl;
        return;
    }
    Node* temp = head;
    cout << "List: ";
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
};
int main() {
    SinglyLinkedList list;
    list.insertAtStart(10);
    list.insertAtEnd(20);
    list.insertAtStart(5);
    list.insertAtEnd(30);
    list.insertAtEnd(40);
    return 0;
}
```

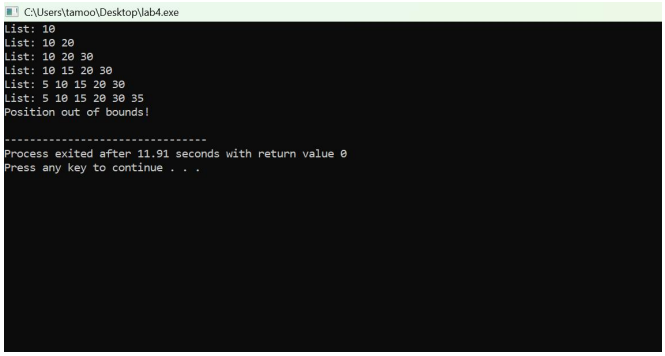
Task 4

```
#include <iostream>
using namespace std;
struct Node {
   int data;
   Node* next;
   Node(int val) {
      data = val;
      next = nullptr;
   }
};
class SinglyLinkedList {
private:
   Node* head;
public:
   SinglyLinkedList() {
      head = nullptr;
   }
   void insertAtStart(int val) {
      Node* newNode = new Node(val);
      newNode->next = head;
      head = newNode;
      displayList();
   }
```

```
    void insertAtEnd(int val) {
        Node* newNode = new Node(val);
        if (head == nullptr) {
            head = newNode;
        } else {
            Node* temp = head;
            while (temp->next != nullptr) {
                temp = temp->next;
            }
            temp->next = newNode;
        }
        displayList();
    }
    void insertAtPosition(int val, int position) {
        if (position <= 0) {
            cout << "Invalid position!" << endl;
            return;
        }
        Node* newNode = new Node(val);
        if (position == 1) {
            newNode->next = head;
            head = newNode;
        } else {
            Node* temp = head;
            int currentPosition = 1;
            while (temp != nullptr && currentPosition <
position - 1) {
                temp = temp->next;
                currentPosition++;
            }
            if (temp == nullptr) {
```

```
            cout << "Position out of bounds!" << endl;
            return;
        }
        newNode->next = temp->next;
        temp->next = newNode;
    }
    displayList();
}
void displayList() {
    if (head == nullptr) {
        cout << "The list is empty." << endl;
        return;
    }
    Node* temp = head;
    cout << "List: ";
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
};
int main() {
    SinglyLinkedList list;
    list.insertAtEnd(10);
    list.insertAtEnd(20);
    list.insertAtEnd(30);
    list.insertAtPosition(15, 2);
    list.insertAtPosition(5, 1);
    list.insertAtPosition(35, 6);
    list.insertAtPosition(100, 10);
```

```
    return 0;
}
```



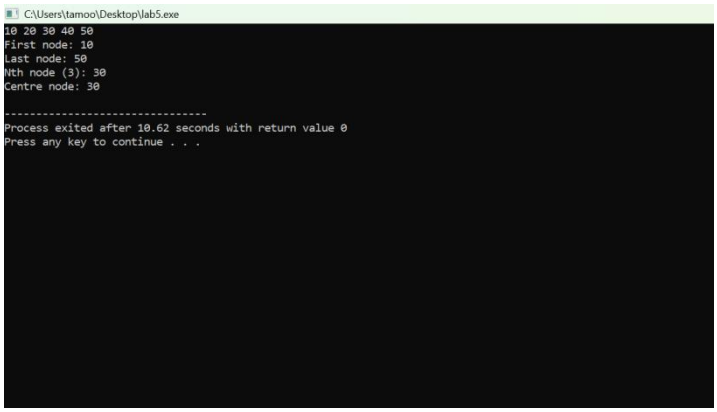
Task 5

```
#include <iostream>
using namespace std;
struct Node {
   int data;
   Node* next;
   Node(int val) {
      data = val;
      next = nullptr;
   }
};
class SinglyLinkedList {
private:
   Node* head;
public:
   SinglyLinkedList() {
      head = nullptr;
   }
   void insertAtEnd(int val) {
      Node* newNode = new Node(val);
      if (head == nullptr) {
         head = newNode;
      } else {
```

```
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}
void displayList() {
    if (head == nullptr) {
        cout << "The list is empty." << endl;
        return;
    }
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
void displayFirstNode() {
    if (head == nullptr) {
        cout << "The list is empty." << endl;
    } else {
        cout << "First node: " << head->data << endl;
    }
}
void displayLastNode() {
    if (head == nullptr) {
        cout << "The list is empty." << endl;
        return;
    }
```

```
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        cout << "Last node: " << temp->data << endl;
    }
    void displayNthNode(int n) {
        if (n <= 0) {
            cout << "Invalid position!" << endl;
            return;
        }
        Node* temp = head;
        int count = 1;
        while (temp != nullptr && count < n) {
            temp = temp->next;
            count++;
        }
        if (temp != nullptr) {
            cout << "Nth node (" << n << "): " << temp->data
<< endl;
        } else {
            cout << "Position out of bounds!" << endl;
        }
    }
    void displayCentreNode() {
        if (head == nullptr) {
            cout << "The list is empty." << endl;
            return;
        }
        Node* slow = head;
        Node* fast = head;
```

```
        while (fast != nullptr && fast->next != nullptr) {
            slow = slow->next;
            fast = fast->next->next;
        }
        cout << "Centre node: " << slow->data << endl;
    }
};
int main() {
    SinglyLinkedList list;
    list.insertAtEnd(10);
    list.insertAtEnd(20);
    list.insertAtEnd(30);
    list.insertAtEnd(40);
    list.insertAtEnd(50);
    list.displayList();
    list.displayFirstNode();
    list.displayLastNode();
    list.displayNthNode(3);
    list.displayCentreNode();
    return 0;
}
```



Task 6

```
#include <iostream>
```

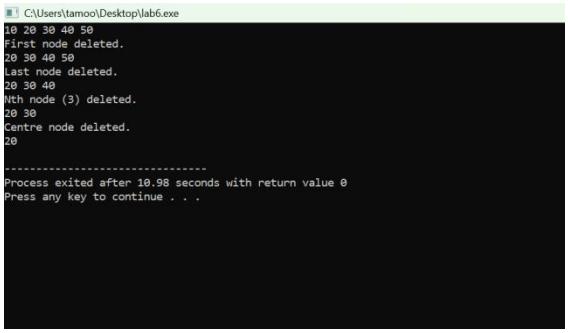
```
using namespace std;
struct Node {
   int data;
   Node* next;
   Node(int val) {
      data = val;
      next = nullptr;
   }
};
class SinglyLinkedList {
private:
   Node* head;
public:
   SinglyLinkedList() {
      head = nullptr;
   }
   void insertAtEnd(int val) {
      Node* newNode = new Node(val);
      if (head == nullptr) {
         head = newNode;
      } else {
         Node* temp = head;
         while (temp->next != nullptr) {
            temp = temp->next;
         }
         temp->next = newNode;
      }
   }
   void displayList() {
      if (head == nullptr) {
         cout << "The list is empty." << endl;
```

```
        return;
    }
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
void deleteFirstNode() {
    if (head == nullptr) {
        cout << "The list is empty." << endl;
        return;
    }
    Node* temp = head;
    head = head->next;
    delete temp;
    cout << "First node deleted." << endl;
    displayList();
}
void deleteLastNode() {
    if (head == nullptr) {
        cout << "The list is empty." << endl;
        return;
    }
    if (head->next == nullptr) {
        delete head;
        head = nullptr;
        cout << "Last node deleted." << endl;
        return;
    }
```

```
        Node* temp = head;
        while (temp->next != nullptr && temp->next->next != nullptr) {
            temp = temp->next;
        }
        delete temp->next;
        temp->next = nullptr;
        cout << "Last node deleted." << endl;
        displayList();
    }
    void deleteNthNode(int n) {
        if (n <= 0) {
            cout << "Invalid position!" << endl;
            return;
        }
        if (head == nullptr) {
            cout << "The list is empty." << endl;
            return;
        }
        if (n == 1) {
            Node* temp = head;
            head = head->next;
            delete temp;
            cout << "Nth node (" << n << ") deleted." << endl;
            displayList();
            return;
        }
        Node* temp = head;
        int count = 1;
        while (temp != nullptr && count < n - 1) {
            temp = temp->next;
```

```
        count++;
    }
    if (temp == nullptr || temp->next == nullptr) {
        cout << "Position out of bounds!" << endl;
        return;
    }
    Node* nodeToDelete = temp->next;
    temp->next = temp->next->next;
    delete nodeToDelete;
    cout << "Nth node (" << n << ") deleted." << endl;
    displayList();
}
void deleteCentreNode() {
    if (head == nullptr) {
        cout << "The list is empty." << endl;
        return;
    }
    Node* slow = head;
    Node* fast = head;
    Node* prev = nullptr;
    if (head->next == nullptr) {
        cout << "The list has only one node, cannot
delete center." << endl;
        return;
    }
    while (fast != nullptr && fast->next != nullptr) {
        prev = slow;
        slow = slow->next;
        fast = fast->next->next;
    }
    prev->next = slow->next;
```

```
        delete slow;
        cout << "Centre node deleted." << endl;
        displayList();
    }
};
int main() {
    SinglyLinkedList list;
    list.insertAtEnd(10);
    list.insertAtEnd(20);
    list.insertAtEnd(30);
    list.insertAtEnd(40);
    list.insertAtEnd(50);
    list.displayList();
    list.deleteFirstNode();
    list.deleteLastNode();
    list.deleteNthNode(3);
    list.deleteCentreNode();
    return 0;
}
```

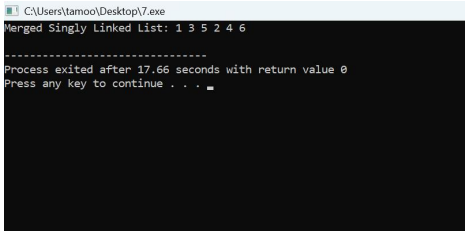


Task 7

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

struct Node {
    int data;
    Node* next;
};
```

```
Node* createList(int arr[], int n) {
    Node* head = nullptr;
    Node* tail = nullptr;
    for (int i = 0; i < n; i++) {
        Node* newNode = new Node();
        newNode->data = arr[i];
        newNode->next = nullptr;
        if (head == nullptr) {
            head = newNode;
            tail = newNode;
        } else {
            tail->next = newNode;
            tail = newNode;
        }
    }
    return head;
}

Node* mergeLists(Node* list1, Node* list2) {
    if (list1 == nullptr) return list2;
    if (list2 == nullptr) return list1;
    Node* temp = list1;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->next = list2;
    return list1;
}

void displayList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    int arr1[] = {1, 3, 5};
    int arr2[] = {2, 4, 6};
    Node* list1 = createList(arr1, 3);
    Node* list2 = createList(arr2, 3);
    Node* mergedList = mergeLists(list1, list2);
    cout << "Merged Singly Linked List: ";
    displayList(mergedList);
    return 0;
```

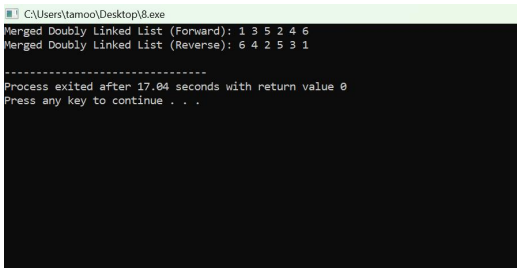
```
}
```



Task 8

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

struct Node {
   int data;
   Node* prev;
   Node* next;
};

Node* createList(int arr[], int n) {
   Node* head = nullptr;
   Node* tail = nullptr;
   for (int i = 0; i < n; i++) {
      Node* newNode = new Node();
      newNode->data = arr[i];
      newNode->prev = tail;
      newNode->next = nullptr;
      if (head == nullptr) {
         head = newNode;
         tail = newNode;
      } else {
         tail->next = newNode;
         tail = newNode;
      }
   }
   return head;
}

Node* mergeLists(Node* list1, Node* list2) {
   if (list1 == nullptr) return list2;
   if (list2 == nullptr) return list1;
   Node* temp = list1;
   while (temp->next != nullptr) {
      temp = temp->next;
   }
   temp->next = list2;
   list2->prev = temp;
   return list1;
```

```
}

void displayList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

void displayReverse(Node* head) {
    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->prev;
    }
    cout << endl;
}

int main() {
    int arr1[] = {1, 3, 5};
    int arr2[] = {2, 4, 6};
    Node* list1 = createList(arr1, 3);
    Node* list2 = createList(arr2, 3);
    Node* mergedList = mergeLists(list1, list2);
    cout << "Merged Doubly Linked List (Forward): ";
    displayList(mergedList);
    cout << "Merged Doubly Linked List (Reverse): ";
    displayReverse(mergedList);
    return 0;
}
```



Task 9

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

```
struct Node {
   int data;
   Node* next;
};

Node* head = nullptr;

void insertFirst(int value) {
   Node* newNode = new Node();
   newNode->data = value;
   if (head == nullptr) {
      head = newNode;
      newNode->next = head;
   } else {
      Node* temp = head;
      while (temp->next != head) {
         temp = temp->next;
      }
      temp->next = newNode;
      newNode->next = head;
      head = newNode;
   }
}

void insertLast(int value) {
   Node* newNode = new Node();
   newNode->data = value;
   if (head == nullptr) {
      head = newNode;
      newNode->next = head;
   } else {
      Node* temp = head;
      while (temp->next != head) {
         temp = temp->next;
      }
      temp->next = newNode;
      newNode->next = head;
   }
}

void insertNth(int value, int position) {
   if (position == 1) {
      insertFirst(value);
      return;
   }
   Node* newNode = new Node();
   newNode->data = value;
```

```
    Node* temp = head;
    for (int i = 1; i < position - 1; i++) {
        temp = temp->next;
        if (temp == head) {
            cout << "Invalid position\n";
            return;
        }
    }
    newNode->next = temp->next;
    temp->next = newNode;
}

void insertCenter(int value) {
    if (head == nullptr) {
        insertFirst(value);
        return;
    }
    Node* slow = head;
    Node* fast = head;
    while (fast->next != head && fast->next->next != head) {
        slow = slow->next;
        fast = fast->next->next;
    }
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = slow->next;
    slow->next = newNode;
}

void displayOrder() {
    if (head == nullptr) return;
    Node* temp = head;
    do {
        cout << temp->data << " ";
        temp = temp->next;
    } while (temp != head);
    cout << endl;
}

void displayReverse() {
    if (head == nullptr) return;
    Node* temp = head;
    Node* prev = nullptr;
    Node* next = nullptr;
    do {
        next = temp->next;
        temp->next = prev;
```

```
        prev = temp;
        temp = next;
    } while (temp != head);
    head->next = prev;
    head = prev;
    displayOrder();
}

int main() {
    insertFirst(10);
    insertLast(20);
    insertLast(30);
    insertNth(15, 2);
    insertCenter(25);
    cout << "List in order: ";
    displayOrder();
    cout << "List in reverse: ";
    displayReverse();
    return 0;
}
```