Experiment No- **Date-**

Aim – To study templates in C++

Theory –

- Templates in C++ are powerful features that allow functions and classes to operate with generic types, enabling code reusability and type safety. Templates provide a way to write generic programs that work with different data types without rewriting code for each specific type.
- Template Functions
- Template functions enable the creation of a single function that can work with different data types. The function's behavior is defined generically, allowing it to perform the same operations on any type.
- *Syntax:*

```
template <typename T>

T functionName(T parameter1, T parameter2) {

   // Function body

}
```

⍰ template <typename T> declares a template. Here, T is a placeholder for a data type that will be specified when the function is called.

⍰ The function functionName can now accept parameters of any data type, as long as the same type is used for both parameters.

Template Classes

Template classes allow the creation of classes that can work with any data type. This is useful for data structures like stacks, queues, linked lists, etc., where the data type may vary.

Syntax:

```
template <typename T>

class ClassName {

   T data; // Member of type T

public:

   ClassName(T value) : data(value) {}

   T getData() { return data; }

};
```

 template <typename T> specifies that the class is a template.

 T is used as a placeholder for the data type, which will be provided when an object of the class is created.

Advantages of Templates

1. **Code Reusability**: Write a single function or class that works with any data type.
2. **Type Safety**: Errors are caught at compile time if the wrong data type is used.
3. **Flexibility**: Allows creating generic data structures and algorithms.

Limitations of Templates

1. **Complexity**: Can make the code harder to read and understand for beginners.
2. **Compilation Time**: Increases because the compiler generates separate instances of template functions/classes for each data type used.

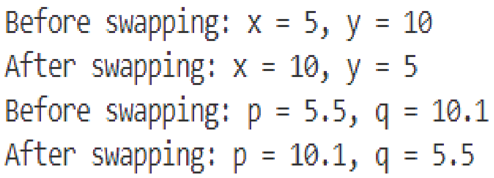
Templates are fundamental in implementing generic programming, making C++ a versatile language for different programming paradigms.

[A] Write a C++ program to implement a function template to swap two elements

Program-

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

// Function template to swap two elements
template <typename T>
void swapElements(T &a, T &b) {
    T temp = a;
    a = b;
    b = temp;
}

int main() {
    int x = 5, y = 10;
    cout << "Before swapping: x = " << x << ", y = " << y << endl;
    swapElements(x, y);
    cout << "After swapping: x = " << x << ", y = " << y << endl;

    double p = 5.5, q = 10.1;
    cout << "Before swapping: p = " << p << ", q = " << q << endl;
    swapElements(p, q);
    cout << "After swapping: p = " << p << ", q = " << q << endl;

    return 0;
}
```

OUTPUT –



```
Before swapping: x = 5, y = 10
After swapping: x = 10, y = 5
Before swapping: p = 5.5, q = 10.1
After swapping: p = 10.1, q = 5.5
```

[B] Write a C++ program to create a class template to represent a generic vector. Include the member functions to perform the following tasks

1. Create the vector

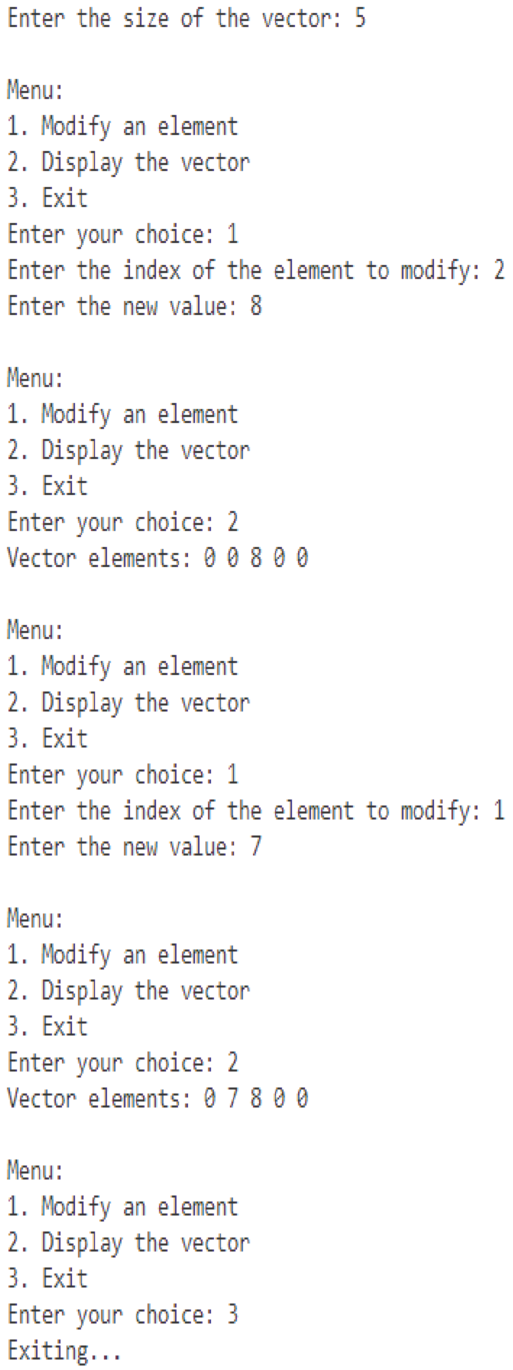
2. To modify the value of a given element

3. To display the vector elements

Program –

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

// Class template for a generic vector
template <typename T>
class Vector {
   T* arr;
   int size;
public:
   // Constructor to create the vector
   Vector(int s) {
      size = s;
      arr = new T[size];
      for (int i = 0; i < size; i++) {
         arr[i] = 0; // Initialize elements to 0
      }
   }

   // Function to modify the value of a given element
   void modifyElement(int index, T value) {
      if (index >= 0 && index < size) {
         arr[index] = value;
      } else {
            cout << "Index out of bounds!" << endl;
      }
   }

   // Function to display the vector elements
   void display() {
      for (int i = 0; i < size; i++) {
         cout << arr[i] << " ";
      }
      cout << endl;
   }
```

Output –



```
Enter the size of the vector: 5

Menu:
1. Modify an element
2. Display the vector
3. Exit
Enter your choice: 1
Enter the index of the element to modify: 2
Enter the new value: 8

Menu:
1. Modify an element
2. Display the vector
3. Exit
Enter your choice: 2
Vector elements: 0 0 8 0 0

Menu:
1. Modify an element
2. Display the vector
3. Exit
Enter your choice: 1
Enter the index of the element to modify: 1
Enter the new value: 7

Menu:
1. Modify an element
2. Display the vector
3. Exit
Enter your choice: 2
Vector elements: 0 7 8 0 0

Menu:
1. Modify an element
2. Display the vector
3. Exit
Enter your choice: 3
Exiting...
```

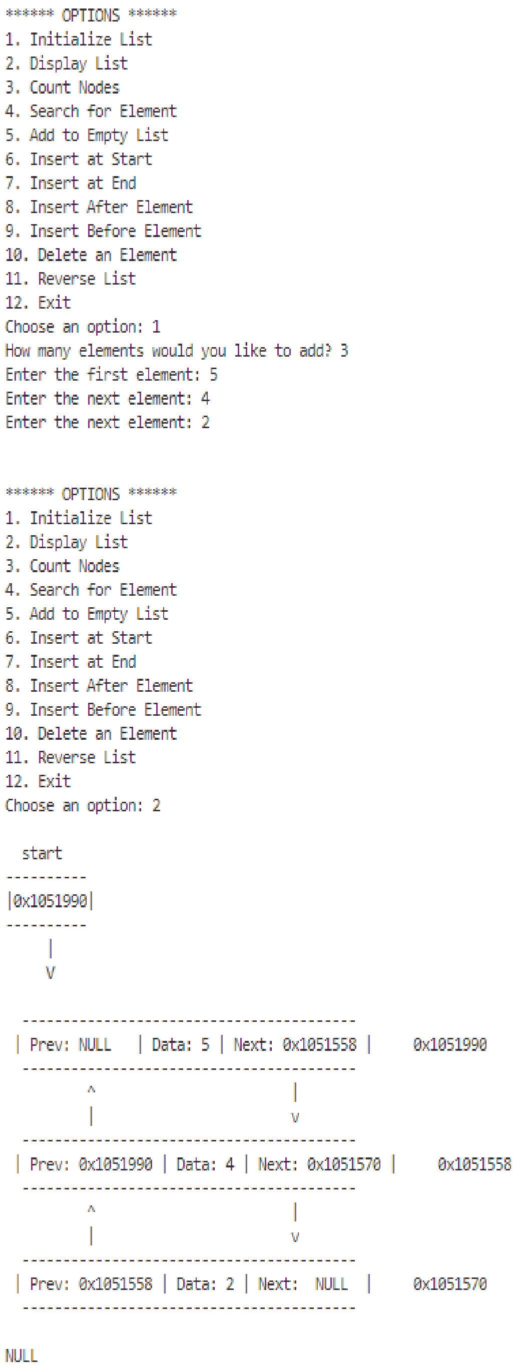
```
    // Destructor to free the allocated memory
    ~Vector() {
        delete[] arr;
    }
};

int main() {
    int n;
    cout << "Enter the size of the vector: ";
    cin >> n;

    // Create a vector of integers
    Vector<int> vec(n);

    int choice;
    do {
        // Display the menu
        cout << "\nMenu:\n";
        cout << "1. Modify an element\n";
        cout << "2. Display the vector\n";
        cout << "3. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1: {
                int index;
                int value;
                cout << "Enter the index of the element to modify: ";
                cin >> index;
                cout << "Enter the new value: ";
                cin >> value;
                vec.modifyElement(index, value);
                break;
            }
            case 2:
                cout << "Vector elements: ";
                vec.display();
                break;
            case 3:
                cout << "Exiting...\n";
                break;
            default:
                cout << "Invalid choice! Please choose again.\n";
        }
    } while (choice != 3);

    return 0;
}
```

[C] Write an interactive program for creating a doubly linked list .The program must support insertion and deletion of a node ,The doubly Linked List class must be of template type

Program –

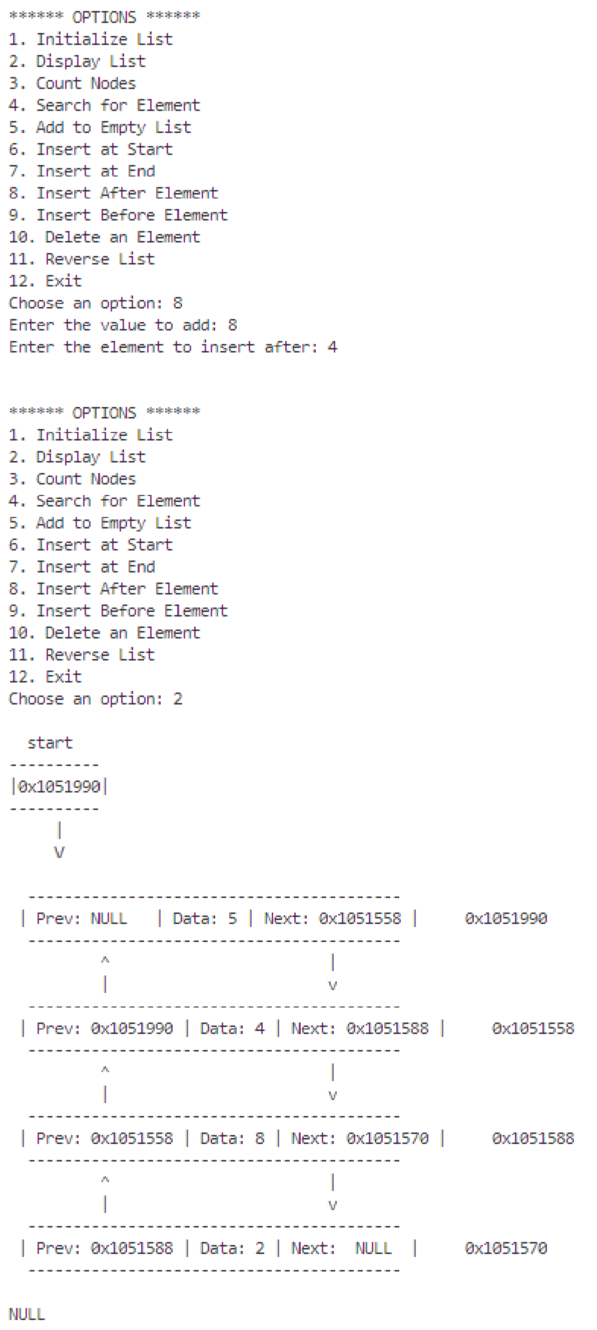
```
#include <iostream>

using namespace std;

template <typename T>
struct Node {
   T value;
   Node<T>* previous;
   Node<T>* next;
};

template <typename T>
class DoublyLinkedList {
public:
   DoublyLinkedList() : start(nullptr) {}

   void initList(T data) {
      start = new Node<T>();
      start->value = data;
      start->previous = nullptr;
      start->next = nullptr;
   }

   void insertAtBeginning(T data) {
      Node<T>* newNode = new Node<T>();
      newNode->value = data;
      newNode->previous = nullptr;
      newNode->next = start;
      if (start) {
         start->previous = newNode;
      }
      start = newNode;
   }

   void insertAtEnd(T data) {
      Node<T>* newNode = new Node<T>();
      Node<T>* tempNode = start;
      while (tempNode->next != nullptr) {
         tempNode = tempNode->next;
      }
      newNode->value = data;
      newNode->previous = tempNode;
      newNode->next = nullptr;
      tempNode->next = newNode;
   }

   void insertAfter(T data, T item) {
      Node<T>* newNode = new Node<T>();
      Node<T>* tempNode = start;
      while (tempNode != nullptr) {
         if (tempNode->value == item) {
            newNode->value = data;
            newNode->previous = tempNode;
            newNode->next = tempNode->next;
```

Output –



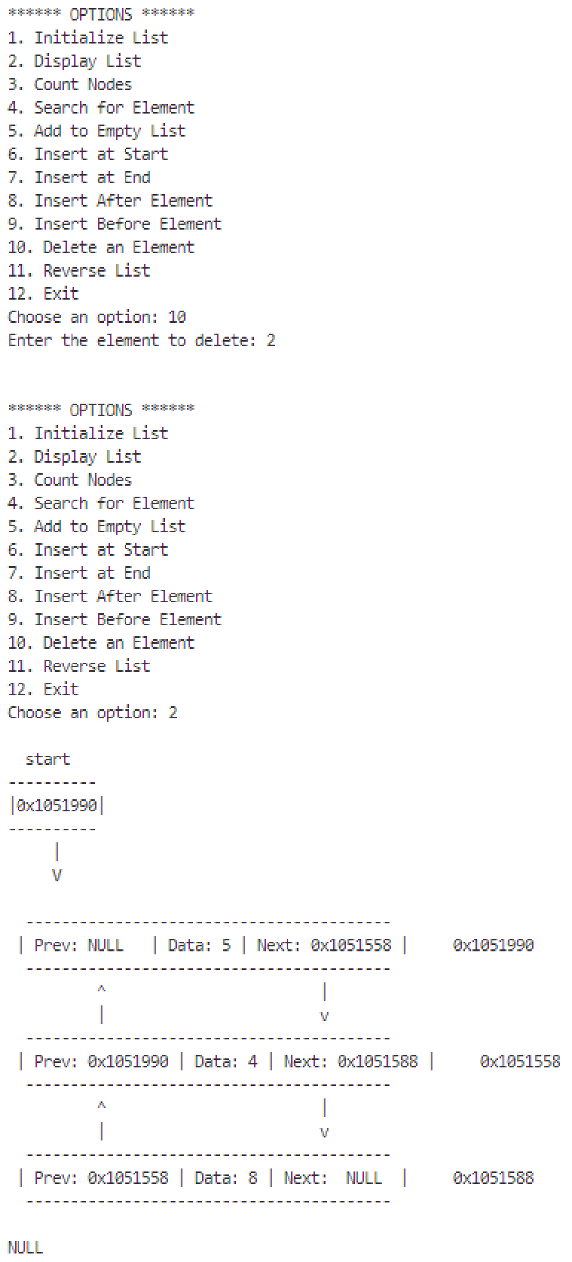
```
****** OPTIONS ******
1. Initialize List
2. Display List
3. Count Nodes
4. Search for Element
5. Add to Empty List
6. Insert at Start
7. Insert at End
8. Insert After Element
9. Insert Before Element
10. Delete an Element
11. Reverse List
12. Exit
Choose an option: 1
How many elements would you like to add? 3
Enter the first element: 5
Enter the next element: 4
Enter the next element: 2


****** OPTIONS ******
1. Initialize List
2. Display List
3. Count Nodes
4. Search for Element
5. Add to Empty List
6. Insert at Start
7. Insert at End
8. Insert After Element
9. Insert Before Element
10. Delete an Element
11. Reverse List
12. Exit
Choose an option: 2

  start
----------
|0x1051990|
----------
     |
     V

  -----------------------------------------
 | Prev: NULL    | Data: 5 | Next: 0x1051558 |     0x1051990
  -----------------------------------------
        ^                         |
        |                         v
  -----------------------------------------
 | Prev: 0x1051990 | Data: 4 | Next: 0x1051570 |     0x1051558
  -----------------------------------------
        ^                         |
        |                         v
  -----------------------------------------
 | Prev: 0x1051558 | Data: 2 | Next:  NULL   |     0x1051570
  -----------------------------------------

NULL
```

```
            if (tempNode->next != nullptr) {
                tempNode->next->previous = newNode;
            }
            tempNode->next = newNode;
            return;
        }
        tempNode = tempNode->next;
    }
    cout << "Item not found\n";
}

void insertBefore(T item, T data) {
    Node<T>* newNode = new Node<T>();
    Node<T>* tempNode = start;
    if (start == nullptr) {
        cout << "\nThe list is empty\n";
        return;
    }
    if (start->value == item) {
        newNode->value = data;
        newNode->next = start;
        start = newNode;
        return;
    }
    while (tempNode->next != nullptr) {
        if (tempNode->next->value == item) {
            newNode->value = data;
            newNode->next = tempNode->next;
            tempNode->next = newNode;
            return;
        }
        tempNode = tempNode->next;
    }
    cout << "\nItem not found\n";
}

void createList() {
    int numElements;
    T data;
    cout << "How many elements would you like to add? ";
    cin >> numElements;
    if (numElements == 0) {
        return;
    }
    cout << "Enter the first element: ";
    cin >> data;
    initList(data);
    for (int i = 1; i < numElements; i++) {
        cout << "Enter the next element: ";
        cin >> data;
        insertAtEnd(data);
    }
}

void printList() const {
    Node<T>* temp = start;

    // Print start pointer in a box pointing to the first node
    cout << "\n  start\n";
    cout << "----------\n";
```

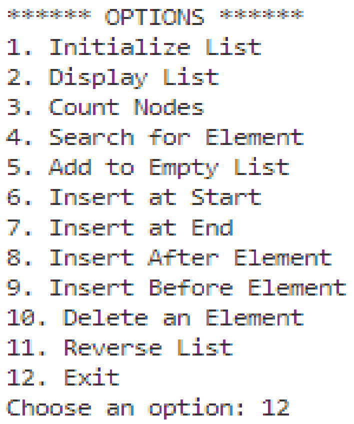


```
****** OPTIONS ******
1. Initialize List
2. Display List
3. Count Nodes
4. Search for Element
5. Add to Empty List
6. Insert at Start
7. Insert at End
8. Insert After Element
9. Insert Before Element
10. Delete an Element
11. Reverse List
12. Exit
Choose an option: 8
Enter the value to add: 8
Enter the element to insert after: 4


****** OPTIONS ******
1. Initialize List
2. Display List
3. Count Nodes
4. Search for Element
5. Add to Empty List
6. Insert at Start
7. Insert at End
8. Insert After Element
9. Insert Before Element
10. Delete an Element
11. Reverse List
12. Exit
Choose an option: 2

  start
----------
|0x1051990|
----------
     |
     V


  -----------------------------------------
 | Prev: NULL   | Data: 5 | Next: 0x1051558 |     0x1051990
  -----------------------------------------
           ^                        |
           |                        v
  -----------------------------------------
 | Prev: 0x1051990 | Data: 4 | Next: 0x1051588 |     0x1051558
  -----------------------------------------
           ^                        |
           |                        v
  -----------------------------------------
 | Prev: 0x1051558 | Data: 8 | Next: 0x1051570 |     0x1051588
  -----------------------------------------
           ^                        |
           |                        v
  -----------------------------------------
 | Prev: 0x1051588 | Data: 2 | Next:  NULL  |     0x1051570
  -----------------------------------------

NULL
```

```
    cout << "|" << start << "|\n";
    cout << "----------\n";
    cout << "     |            \n";
    if (start != nullptr) {
      cout << "     V            \n\n";
    }

    while (temp != nullptr) {
      if (temp->previous == nullptr) {
        cout << "  ----------------------------------------   \n";
        cout << " | Prev: NULL   | Data: " << temp->value <<
" | Next: " << temp->next << " |     " << temp << " \n";
        cout << "  ----------------------------------------   \n";
      } else if (temp->next == nullptr) {
        cout << "  ----------------------------------------   \n";
        cout << " | Prev: " << temp->previous << " | Data: "
<< temp->value << " | Next:  NULL  |     " << temp << " \n";
        cout << "  ----------------------------------------   \n";
      } else {
        cout << "  ----------------------------------------   \n";
        cout << " | Prev: " << temp->previous << " | Data: "
<< temp->value << " | Next: " << temp->next << " |     " <<
temp << " \n";
        cout << "  ----------------------------------------   \n";
      }

      if (temp->next != nullptr) {
        cout << "        ^                  |\n";
        cout << "        |                  v\n";
      }
      temp = temp->next;
    }
    cout << "\nNULL\n";
  }

  int countNodes() const {
    int count = 0;
     for (Node<T>* tempNode = start; tempNode != nullptr;
tempNode = tempNode->next) {
      count++;
    }
    return count;
  }

  Node<T>* search(T item) const {
    Node<T>* tempNode = start;
    while (tempNode != nullptr) {
      if (tempNode->value == item) {
        return tempNode;
      }
      tempNode = tempNode->next;
    }
    return nullptr;
  }

  void removeNode(T data) {
    Node<T>* tempNode = start;
    Node<T>* prevNode = nullptr;
    while (tempNode != nullptr) {
      if (tempNode->value == data) {
```



```
****** OPTIONS ******
1. Initialize List
2. Display List
3. Count Nodes
4. Search for Element
5. Add to Empty List
6. Insert at Start
7. Insert at End
8. Insert After Element
9. Insert Before Element
10. Delete an Element
11. Reverse List
12. Exit
Choose an option: 10
Enter the element to delete: 2


****** OPTIONS ******
1. Initialize List
2. Display List
3. Count Nodes
4. Search for Element
5. Add to Empty List
6. Insert at Start
7. Insert at End
8. Insert After Element
9. Insert Before Element
10. Delete an Element
11. Reverse List
12. Exit
Choose an option: 2

  start
----------
|0x1051990|
----------
     |
     V


  ----------------------------------------
 | Prev: NULL   | Data: 5 | Next: 0x1051558 |     0x1051990
  ----------------------------------------
        ^                  |
        |                  v
  ----------------------------------------
 | Prev: 0x1051990 | Data: 4 | Next: 0x1051588 |     0x1051558
  ----------------------------------------
        ^                  |
        |                  v
  ----------------------------------------
 | Prev: 0x1051558 | Data: 8 | Next:  NULL  |     0x1051588
  ----------------------------------------

NULL
```



```
 ****** OPTIONS ******
 1. Initialize List
 2. Display List
 3. Count Nodes
 4. Search for Element
 5. Add to Empty List
 6. Insert at Start
 7. Insert at End
 8. Insert After Element
 9. Insert Before Element
 10. Delete an Element
 11. Reverse List
 12. Exit
 Choose an option: 12
```

```
        if (prevNode != nullptr) {
          prevNode->next = tempNode->next;
        } else {
          start = tempNode->next;
        }
        if (tempNode->next != nullptr) {
          tempNode->next->previous = prevNode;
        }
        delete tempNode;
        return;
      }
      prevNode = tempNode;
      tempNode = tempNode->next;
    }
    cout << "Element not found\n";
  }

  void reverseList() {
    Node<T>* tempNode = nullptr;
    Node<T>* current = start;
    while (current != nullptr) {
      tempNode = current->previous;
      current->previous = current->next;
      current->next = tempNode;
      current = current->previous;
    }
    if (tempNode != nullptr) {
      start = tempNode->previous;
    }
  }

  void swapAlternateNodes() {
    if (start == nullptr || start->next == nullptr) {
      return;
    }

    Node<T>* tempNode1 = start;
    Node<T>* tempNode2 = start->next;
    start = tempNode2;
    Node<T>* temp;

    while (true) {
      temp = tempNode2->next;
      tempNode2->next = tempNode1;
      tempNode2->previous = tempNode1->previous;
      tempNode1->previous = tempNode2;
      tempNode1->next = temp;
      if (temp == nullptr || temp->next == nullptr) {
        break;
      }
      tempNode1->next = temp->next;
      tempNode1 = temp;
      tempNode2 = tempNode1->next;
    }
  }

private:
  Node<T>* start;
};
```

```
            cin >> item;
            list.insertAfter(data, item);
            break;
        case 9:
            cout << "Enter the value to add: ";
            cin >> data;
            cout << "Enter the element to insert before: ";
            cin >> item;
            list.insertBefore(item, data);
            break;
        case 10:
            cout << "Enter the element to delete: ";
            cin >> data;
            list.removeNode(data);
            break;
        case 11:
            list.reverseList();
            cout << "List reversed\n";
            break;
        case 12 :
             break;
        default:
            cout << "Invalid option\n";
            break;
        }
    } while (option != 12);

    return 0;
}
```

Conclusion – All the codes were successfully executed using the concepts of Templates.