Singly Linked List

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

class Node {
public:
    int number;
    Node* next;
};

class LinkedList {
public:
    int Length;
    Node* head;

    LinkedList();
    ~LinkedList();
    void insert(int number);
    void display();
    void deletion();
    void refreshing();
    void searching(int data);
    void searchxdelete(int data);
};

LinkedList::LinkedList() {
    this->Length = 0;
    this->head = NULL;
}

LinkedList::~LinkedList() {
    cout << "Linked List Deleted!" << endl;
}

void LinkedList::insert(int data) {
    Node* node2 = new Node();
    node2->number = data;
    node2->next = this->head;
    this->head = node2;
    this->Length++;
}

void LinkedList::display() {
    Node* temp = this->head;
    cout << "[" << Length << "] ";
    while (temp) {
        cout << temp->number << " ";
        temp = temp->next;
    }
    cout << endl;
}
```

```
void LinkedList::deletion() {
    if (Length <= 0) {
        cout << "THE LIST IS ALREADY EMPTY" << endl;
        return;
    }
    Node* temp = head;
    head = head->next;
    delete temp;
    Length--;
}

void LinkedList::refreshing() {
    while (head) {
        Node* temp = this->head;
        head = head->next;
        delete temp;
    }
    Length = 0;
}

void LinkedList::searching(int data) {
    if (Length <= 0) {
        cout << "VALUE NOT FOUND" << endl;
        return;
    }
    Node* temp = this->head;
    int counter = 0;
    while (temp) {
        if (temp->number == data) {
            counter++;
        }
        temp = temp->next;
    }
    if (counter > 0) {
        cout << "VALUE FOUND" << endl;
    }
    else {
        cout << "VALUE NOT FOUND" << endl;
    }
}

void LinkedList::searchxdelete(int data) {
    Node* current = head;
    Node* prev = nullptr;

    while (current) {
        if (current->number == data) {
            if (prev)
                prev->next = current->next;
            } else {
                head = current->next;
            }
            delete current;
```

```
            Length--;
             display();
            return;
        }
        prev = current;
        current = current->next;
    }

    cout << "VALUE NOT FOUND" << endl;
}


int main() {
    LinkedList* list = new LinkedList();
    char letter;
    int number;
    int counter = 0;

    while (cin >> letter >> number) {
        if (letter == 'i') {
            counter++;
            list->insert(number);
            list->display();
        }
        else if (letter == 'd') {
            counter--;
            if (counter < 0) {
                cout << "THE LIST IS ALREADY EMPTY" << endl;
            }
            else {
                list->deletion();
                list->display();
            }
        }
        else if (letter == 'r') {
            counter = 0;
            list->refreshing();
            list->display();
        }
        else if (letter == 's') {
            list->searching(number);
        }
        else if (letter == 'x') {
            list->searchxdelete(number);
        }
        else {
            cout << "INVALID COMMAND" << endl;
        }
    }

    return 0;
}
```

Doubly Linked List with Reverse Operation

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

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

class Linkedlist {
public:
    int Length;
    Node* head;
    Node* tail;

    Linkedlist();
    ~Linkedlist();
    void insert(int number);
    void display();
    void deletion();
    void refreshing();
    void searching(int data);
    void searchxdelete(int data);
    void reverse();
};

Linkedlist::Linkedlist() {
    this->Length = 0;
    this->head = NULL;
    this->tail = NULL;
}

Linkedlist::~Linkedlist() {
    cout << "LINKED LIST DELETED" << endl;
}

void Linkedlist::display() {
    Node* temp = this->head;
    cout << "[" << Length << "] ";
    while (temp) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

void Linkedlist::insert(int data) {
    Node* node2 = new Node();
    node2->data = data;
    node2->prev = NULL;
    node2->next = this->head;
```

```
    if (Length == 0) {
        this->tail = node2;
    } else {
        this->head->prev = node2;
    }
    this->head = node2;
    this->Length++;

    display();
}

void Linkedlist::deletion() {
    Node* temp = head;
    head = head->next;
    delete temp;
    Length--;
}

void Linkedlist::reverse() {
  Node* prev = NULL;
  Node* curr = head;
  Node* next = NULL;

  while (curr != NULL) {
    next = curr->next;
    curr->next = prev;
    prev = curr;
    curr = next;
  }
  head = prev;
}

void Linkedlist::refreshing() {
    while (head) {
        Node* temp = head;
        head = head->next;
        delete temp;
    }
    Length = 0;
}

int main() {
    Linkedlist* list = new Linkedlist();
    char letter;
    int number;
    int counter = 0;

    while (cin >> letter >> number) {
        if (letter == 'i') {
            counter++;
            list->insert(number);
        }
```

```
        else if (letter == 'd') {
            counter--;
            if (counter < 0) {
                cout << "THE LIST IS ALREADY EMPTY" << endl;
            }
            else {
                list->deletion();
                list->display();
            }
        }
        else if (letter == 'r') {
            counter = 0;
            list->refreshing();
            list->display();
        }
        else if (letter == 'e') {
            list->reverse();
            list->display();
            list->reverse();
        }
        else {
            cout << "INVALID COMMAND" << endl;
        }
    }

    return 0;
}
```

Stack-Based String Manipulation

```
#include <iostream>
#include <stack>

using namespace std;

void christmas(char cGift, string G) {
    stack<char> Gift;
    stack<char> reversegift;
    bool check = false;

    for (int i = G.length()-1; 0 <= i; i--) {
        if (G[i] == cGift && !check) {
            check = true;
             while(!Gift.empty()){
           reversegift.push(Gift.top());
           Gift.pop();
          }
        }
      else{
```

```
            Gift.push(G[i]);
        }
     }
         if(!check){
            string s = "";
            while(!Gift.empty()){
                s= Gift.top() + s;
                Gift.pop();
            }
            cout << s;
         }

         while(!Gift.empty()){
            cout <<Gift.top();
            Gift.pop();
         }
         cout << " ";
          while(!reversegift.empty()){
            cout <<reversegift.top();
            reversegift.pop();
          }
    }

int main() {
    int Tsize;
    cin >> Tsize;
    char cGift;
    string G;

    for(int i = 0; i < Tsize; i++) {
        cin >> cGift >> G;

        christmas (cGift , G);
        cout <<endl;
    }


    return 0;
}
```

Bubble Sort

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

```
void christmas(char cGift, string G) {
    stack<char> leftPart;
    stack<char> reversedLeft;
    bool found = false;

    for (int i = G.length() - 1; i >= 0; i--) {
        if (G[i] == cGift && !found) {
            found = true;
            while (!leftPart.empty()) {
                reversedLeft.push(leftPart.top());
                leftPart.pop();
            }
            cout << cGift;
        } else {
            leftPart.push(G[i]);
        }
    }

    // If gift was not found, just print original string
    if (!found) {
        cout << G;
        return;
    }

    // Print reversed left part
    while (!reversedLeft.empty()) {
        cout << reversedLeft.top();
        reversedLeft.pop();
    }
}

int main() {
    int Tsize;
    cin >> Tsize;

    for (int i = 0; i < Tsize; i++) {
        char cGift;
        string G;
        cin >> cGift >> G;
        christmas(cGift, G);
        cout << endl;
    }
    return 0;
}
```

Selection Sort

```
void selectionsort(int array[], int array_size) {
    for (int i = 0; i < array_size - 1; i++) {
        int min = i;
```

```
        for (int j = i + 1; j < array_size; j++) {
            if (array[j] < array[min]) {
                min = j;
            }
        }

        // Swap AFTER finding the minimum
        int temp = array[min];
        array[min] = array[i];
        array[i] = temp;

        cout << "PASS #" << i + 1 << ": ";
        for (int k = 0; k < array_size; k++) {
            cout << array[k] << " ";
        }
        cout << endl;
    }
}
```