B1.

#include<iostream>

using namespace std;

int main(){

    int n,a[1000],dem=0;

    cin>>n;

    for(int i=0;i<n;i++)

    {

        cin>>a[i];

    }

    for(int i=0;i<n;i++)

    {

        for(int j=i+1;j<n;j++)

        {

            if(a[i]==a[j])

                dem++;

        }

    }

    cout<<dem;

}

B2.

#include <iostream>

using namespace std;

struct *Node* {

    int data;

*Node*\* next;

    Node(int *val*) : data(*val*), next(nullptr) {}

};

class *LinkedList* {

private:

*Node*\* head;

public:

    LinkedList() : head(nullptr) {}

    void insert(int *p*, int *x*) {

*Node*\* newNode = **new** *Node*(*x*);

        if (*p* == 0) {

            newNode->next = head;

            head = newNode;

        } else {

*Node*\* current = head;

            for (int i = 0; i < *p* - 1 && current != nullptr; i++) {

                current = current->next;

            }

            if (current == nullptr) {

                cout << "Invalid position" << endl;

            } else {

                newNode->next = current->next;

                current->next = newNode;

            }

        }

    }

    void remove(int *p*) {

        if (*p* == 0) {

            if (head != nullptr) {

*Node*\* temp = head;

                head = head->next;

**delete** temp;

            } else {

                cout << "List is empty" << endl;

            }

        } else {

*Node*\* current = head;

            for (int i = 0; i < *p* - 2 && current != nullptr; i++) {

                current = current->next;

            }

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

                cout << "Invalid position" << endl;

            } else {

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

                current->next = temp->next;

**delete** temp;

            }

        }

    }

    void display() {

*Node*\* current = head;

        while (current != nullptr) {

            cout << current->data << " ";

            current = current->next;

        }

        cout << endl;

    }

};

int main() {

    int n;

    cin >> n;

*LinkedList* linkedList;

    for (int i = 0; i < n; ++i) {

*string* operation;

        cin >> operation;

        if (operation == "insert") {

            int p, x;

            std::cin >> p >> x;

            linkedList.insert(p, x);

        } else if (operation == "delete") {

            int p;

            cin >> p;

            linkedList.remove(p);

        }

    }

    linkedList.display();

    return 0;

}

B3.

#include <iostream>

using namespace std;

struct *Node* {

    int data;

*Node*\* prev;

*Node*\* next;

    Node(int *val*)

    {

        data=*val*;

        prev=nullptr;

        next=nullptr;

    }

};

class *DoublyLinkedList* {

private:

*Node*\* head;

*Node*\* tail;

public:

    DoublyLinkedList() {

        head=nullptr;

        tail=nullptr;

    }

    void insert(int *val*) {

*Node*\* newNode = **new** *Node*(*val*);

        if (head == nullptr) {

            head = newNode;

            tail = newNode;

        } else {

            tail->next = newNode;

            newNode->prev = tail;

            tail = newNode;

        }

    }

    int count\_triplets() {

        int count = 0;

*Node*\* current = head;

        if (head == nullptr || head->next == nullptr || head->next->next == nullptr) {

            return 0; // Less than 3 elements in the list

        }

        current = current->next;

        while (current->next != nullptr) {

            int sum = current->data + current->prev->data + current->next->data;

            if (sum == 0) {

                count++;

            }

            current = current->next;

        }

        return count;

    }

};

int main() {

    int n;

    cin >> n;

*DoublyLinkedList* linkedList;

    for (int i = 0; i < n; ++i) {

        int num;

        cin >> num;

        linkedList.insert(num);

    }

    int result = linkedList.count\_triplets();

    cout << result;

    return 0;

}

B4

#include <iostream>

#include <queue>

using namespace std;

int main(){

    int n;

    cin>>n;

    queue<int> q;

    for(int i=0;i<n;i++)

    {

*string* operation;

        cin>>operation;

        if(operation == "enqueue")

        {

            int x;cin>>x;

            q.push(x);

        }

        else if(operation == "dequeue"){

            if(!q.empty())

            {

                q.pop();

            }

        }

    }

    while(!q.empty())

    {

        cout<<q.front()<<" ";

        q.pop();

    }

    return 0;

}

B5

#include <iostream>

#include <stack>

using namespace std;

int main() {

    int n;

    cin >> n;

    stack<int> stack;

    for (int i = 0; i < n; ++i) {

*string* operation;

        cin >> operation;

        if (operation == "push") {

            int x;

            cin >> x;

            stack.push(x);

        } else if (operation == "pop") {

            if (!stack.empty()) {

                stack.pop();

            }

        }

    }

    std::stack<int> tempStack;

    while (!stack.empty()) {

        tempStack.push(stack.top());

        stack.pop();

    }

    while (!tempStack.empty()) {

        cout << tempStack.top() << " ";

        tempStack.pop();

    }

    return 0;

}