Sort a stack using another stack.

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

using namespace std;

void sortStack(stack<int> &inputStack) {
   stack<int> tempStack;

   while (!inputStack.empty()) {
      int temp = inputStack.top();
      inputStack.pop();

      while (!tempStack.empty() && tempStack.top() > temp) {
         inputStack.push(tempStack.top());
         tempStack.pop();
      }

      tempStack.push(temp);
   }

   while (!tempStack.empty()) {
      inputStack.push(tempStack.top());
      tempStack.pop();
   }
}

int main() {
   stack<int> inputStack;
   int n, element;
```

```
    cout << "Enter the number of elements in the stack: ";
    cin >> n;

    cout << "Enter the elements of the stack: ";
    for (int i = 0; i < n; ++i) {
        cin >> element;
        inputStack.push(element);
    }

    sortStack(inputStack);

    cout << "Sorted stack: ";
    while (!inputStack.empty()) {
        cout << inputStack.top() << " ";
        inputStack.pop();
    }
    cout << endl;

    return 0;
}
```

2)Find the minimum element in stack in O(1) time.

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

using namespace std;

class MinStack {
private:
    stack<int> mainStack;
```

```
    stack<int> minStack;

public:
    void push(int x) {
        mainStack.push(x);
        if (minStack.empty() || x <= minStack.top()) {
            minStack.push(x);
        }
    }

    void pop() {
        if (mainStack.empty()) {
            cout << "Stack is empty, cannot pop.\n";
            return;
        }
        if (mainStack.top() == minStack.top()) {
            minStack.pop();
        }
        mainStack.pop();
    }

    int top() {
        if (mainStack.empty()) {
            cout << "Stack is empty.\n";
            return INT_MIN;
        }
        return mainStack.top();
    }

    int getMin() {
        if (minStack.empty()) {
```

```
            cout << "Stack is empty.\n";
            return INT_MIN;
        }
        return minStack.top();
    }
};

int main() {
    MinStack stack;

    int choice, element;

    while (true) {
        cout << "\n1. Push\n2. Pop\n3. Get Top\n4. Get Minimum\n5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter element to push: ";
                cin >> element;
                stack.push(element);
                break;
            case 2:
                stack.pop();
                break;
            case 3:
                cout << "Top element: " << stack.top() << endl;
                break;
            case 4:
                cout << "Minimum element: " << stack.getMin() << endl;
```

```
            break;
        case 5:
            cout << "Exiting...\n";
            return 0;
        default:
            cout << "Invalid choice. Please try again.\n";
      }
   }

   return 0;
}
```

3)find the minimum element in stack in O(1) time and O(1) space.

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

using namespace std;

class MinStack {
private:
   stack<long long> s; // Using long long to handle edge cases
   long long minElement;

public:
   MinStack() {
      minElement = INT_MAX;
   }

   void push(int x) {
      if (s.empty()) {
         s.push(x);
```

```
        minElement = x;
    } else {
        long long diff = static_cast<long long>(x) - minElement;
        s.push(diff);
        if (x < minElement) {
            minElement = x;
        }
    }
}

void pop() {
    if (s.empty()) {
        cout << "Stack is empty, cannot pop.\n";
        return;
    }
    long long top = s.top();
    s.pop();
    if (top < 0) {
        minElement = minElement - top;
    }
}

int top() {
    if (s.empty()) {
        cout << "Stack is empty.\n";
        return INT_MIN;
    }
    long long top = s.top();
    if (top < 0) {
        return minElement;
    } else {
```

```
            return minElement + top;
        }
    }

    int getMin() {
        if (s.empty()) {
            cout << "Stack is empty.\n";
            return INT_MIN;
        }
        return minElement;
    }
};

int main() {
    MinStack stack;

    int choice, element;

    while (true) {
        cout << "\n1. Push\n2. Pop\n3. Get Top\n4. Get Minimum\n5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter element to push: ";
                cin >> element;
                stack.push(element);
                break;
            case 2:
                stack.pop();
```

```
            break;
        case 3:
            cout << "Top element: " << stack.top() << endl;
            break;
        case 4:
            cout << "Minimum element: " << stack.getMin() << endl;
            break;
        case 5:
            cout << "Exiting...\n";
            return 0;
        default:
            cout << "Invalid choice. Please try again.\n";
      }
   }

   return 0;
}
```

4)Implement two stacks in one array (create user defined stacks).

```
#include <iostream>
#include <vector>

using namespace std;

class TwoStacks {
private:
   vector<int> arr;
   int top1; // Top index of stack 1
   int top2; // Top index of stack 2
   int capacity; // Total capacity of the array

public:
```

```
TwoStacks(int size) {
    capacity = size;
    arr.resize(size);
    top1 = -1; // Initialize top of stack 1
    top2 = size; // Initialize top of stack 2
}

void push1(int value) {
    if (top1 + 1 < top2) { // Ensure there's space for stack 1
        top1++;
        arr[top1] = value;
    } else {
        cout << "Stack 1 overflow!\n";
    }
}

void push2(int value) {
    if (top2 - 1 > top1) { // Ensure there's space for stack 2
        top2--;
        arr[top2] = value;
    } else {
        cout << "Stack 2 overflow!\n";
    }
}

int pop1() {
    if (top1 >= 0) { // Check if stack 1 is not empty
        int value = arr[top1];
        top1--;
        return value;
    } else {
```

```
            cout << "Stack 1 underflow!\n";
            return -1; // Return a default value to indicate underflow
        }
    }

    int pop2() {
        if (top2 < capacity) { // Check if stack 2 is not empty
            int value = arr[top2];
            top2++;
            return value;
        } else {
            cout << "Stack 2 underflow!\n";
            return -1; // Return a default value to indicate underflow
        }
    }

    bool isEmpty1() {
        return (top1 == -1);
    }

    bool isEmpty2() {
        return (top2 == capacity);
    }
};

int main() {
    int capacity;
    cout << "Enter the capacity of the array: ";
    cin >> capacity;

    TwoStacks stacks(capacity);
```

```
    int choice, value;
    while (true) {
        cout << "\n1. Push to Stack 1\n2. Push to Stack 2\n3. Pop from Stack 1\n4. Pop from Stack 2\n5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter value to push to Stack 1: ";
                cin >> value;
                stacks.push1(value);
                break;
            case 2:
                cout << "Enter value to push to Stack 2: ";
                cin >> value;
                stacks.push2(value);
                break;
            case 3:
                if (!stacks.isEmpty1()) {
                    cout << "Popped value from Stack 1: " << stacks.pop1() << endl;
                } else {
                    cout << "Stack 1 is empty!\n";
                }
                break;
            case 4:
                if (!stacks.isEmpty2()) {
                    cout << "Popped value from Stack 2: " << stacks.pop2() << endl;
                } else {
                    cout << "Stack 2 is empty!\n";
```

```
            }
            break;
        case 5:
            cout << "Exiting...\n";
            return 0;
        default:
            cout << "Invalid choice. Please try again.\n";
      }
   }

   return 0;
}
```

5)Implement stack using queue.

```
#include <iostream>
#include <queue>

using namespace std;

class Stack {
private:
   queue<int> q1, q2;

public:
   void push(int x) {
      // Push the element to the non-empty queue
      if (!q1.empty()) {
         q1.push(x);
      } else {
         q2.push(x);
      }
```

```
}

int pop() {
    if (empty()) {
        cout << "Stack is empty!\n";
        return -1; // Return a default value to indicate underflow
    }

    int poppedElement;
    if (!q1.empty()) {
        // Move all elements from q1 to q2 except the last one
        while (q1.size() > 1) {
            q2.push(q1.front());
            q1.pop();
        }
        // Pop the last element from q1
        poppedElement = q1.front();
        q1.pop();
    } else {
        // Move all elements from q2 to q1 except the last one
        while (q2.size() > 1) {
            q1.push(q2.front());
            q2.pop();
        }
        // Pop the last element from q2
        poppedElement = q2.front();
        q2.pop();
    }

    return poppedElement;
}
```

```
int top() {
    if (empty()) {
        cout << "Stack is empty!\n";
        return -1; // Return a default value to indicate underflow
    }

    int topElement;
    if (!q1.empty()) {
        // Move all elements from q1 to q2 except the last one
        while (q1.size() > 1) {
            q2.push(q1.front());
            q1.pop();
        }
        // Get the last element from q1
        topElement = q1.front();
        q2.push(q1.front());
        q1.pop();
    } else {
        // Move all elements from q2 to q1 except the last one
        while (q2.size() > 1) {
            q1.push(q2.front());
            q2.pop();
        }
        // Get the last element from q2
        topElement = q2.front();
        q1.push(q2.front());
        q2.pop();
    }

    return topElement;
```

```
    }

    bool empty() {
        return q1.empty() && q2.empty();
    }
};

int main() {
    Stack stack;

    int choice, value;
    while (true) {
        cout << "\n1. Push\n2. Pop\n3. Get Top\n4. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter value to push: ";
                cin >> value;
                stack.push(value);
                break;
            case 2:
                cout << "Popped value: " << stack.pop() << endl;
                break;
            case 3:
                cout << "Top element: " << stack.top() << endl;
                break;
            case 4:
                cout << "Exiting...\n";
                return 0;
```

```
            default:
                cout << "Invalid choice. Please try again.\n";
        }
    }

    return 0;
}
```

6)Implement queue using stack.

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

using namespace std;

class Queue {
private:
    stack<int> s1; // For enqueue operation
    stack<int> s2; // For dequeue operation

public:
    void enqueue(int x) {
        // Move all elements from s2 to s1
        while (!s2.empty()) {
            s1.push(s2.top());
            s2.pop();
        }
        // Push the new element onto s1
        s1.push(x);
    }

    int dequeue() {
        if (empty()) {
```

```
        cout << "Queue is empty!\n";
        return -1; // Return a default value to indicate underflow
    }

    // Move all elements from s1 to s2
    while (!s1.empty()) {
        s2.push(s1.top());
        s1.pop();
    }
    // Pop the front element from s2
    int frontElement = s2.top();
    s2.pop();

    return frontElement;
}

int front() {
    if (empty()) {
        cout << "Queue is empty!\n";
        return -1; // Return a default value to indicate underflow
    }

    // Move all elements from s1 to s2
    while (!s1.empty()) {
        s2.push(s1.top());
        s1.pop();
    }
    // Get the front element from s2
    int frontElement = s2.top();

    return frontElement;
```

```
    }

    bool empty() {
        return s1.empty() && s2.empty();
    }
};

int main() {
    Queue queue;

    int choice, value;
    while (true) {
        cout << "\n1. Enqueue\n2. Dequeue\n3. Get Front\n4. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter value to enqueue: ";
                cin >> value;
                queue.enqueue(value);
                break;
            case 2:
                cout << "Dequeued value: " << queue.dequeue() << endl;
                break;
            case 3:
                cout << "Front element: " << queue.front() << endl;
                break;
            case 4:
                cout << "Exiting...\n";
                return 0;
```

```
            default:
                cout << "Invalid choice. Please try again.\n";
        }
    }

    return 0;
}
```

7)Reverse a linked list using stack

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

using namespace std;

// Definition for singly-linked list.
struct ListNode {
    int val;
    ListNode *next;
    ListNode(int x) : val(x), next(nullptr) {}
};

class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        if (head == nullptr || head->next == nullptr) {
            return head;
        }

        stack<ListNode*> s;
        ListNode* curr = head;

        // Push all nodes onto the stack
```

```
        while (curr != nullptr) {
            s.push(curr);
            curr = curr->next;
        }

        // Pop nodes from the stack and rebuild the linked list
        head = s.top();
        s.pop();
        curr = head;

        while (!s.empty()) {
            curr->next = s.top();
            s.pop();
            curr = curr->next;
        }

        // Mark the end of the reversed list
        curr->next = nullptr;

        return head;
    }
};

// Function to print the linked list
void printList(ListNode* head) {
    while (head != nullptr) {
        cout << head->val << " ";
        head = head->next;
    }
    cout << endl;
}
```

```
int main() {
    Solution solution;
    ListNode* head = nullptr;
    ListNode* prev = nullptr;
    int n, val;

    cout << "Enter the number of elements in the linked list: ";
    cin >> n;

    cout << "Enter the elements of the linked list: ";
    for (int i = 0; i < n; ++i) {
        cin >> val;
        ListNode* newNode = new ListNode(val);
        if (head == nullptr) {
            head = newNode;
        } else {
            prev->next = newNode;
        }
        prev = newNode;
    }

    cout << "Original list: ";
    printList(head);

    // Reverse the linked list
    head = solution.reverseList(head);

    cout << "Reversed list: ";
    printList(head);
```

```
    // Free memory
    while (head != nullptr) {
        ListNode* temp = head;
        head = head->next;
        delete temp;
    }

    return 0;
}
```

8)Reverse a string using stack.

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

using namespace std;

string reverseString(const string& str) {
    stack<char> charStack;
    string reversedStr;

    // Push each character of the string onto the stack
    for (char c : str) {
        charStack.push(c);
    }

    // Pop characters from the stack to form the reversed string
    while (!charStack.empty()) {
        reversedStr.push_back(charStack.top());
        charStack.pop();
    }
```

```
    return reversedStr;
}

int main() {
    string inputString;

    cout << "Enter a string: ";
    getline(cin, inputString);

    string reversedString = reverseString(inputString);

    cout << "Reversed string: " << reversedString << endl;

    return 0;
}
```

9)Check if an expression is balanced parenthesis or not.

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

using namespace std;

bool isBalanced(const string& expression) {
    stack<char> parenthesesStack;

    // Traverse each character in the expression
    for (char c : expression) {
        if (c == '(' || c == '[' || c == '{') {
            // If opening parenthesis, push onto stack
            parenthesesStack.push(c);
        } else if (c == ')' || c == ']' || c == '}') {
```

```
            // If closing parenthesis, check if stack is empty or top is matching opening parenthesis
            if (parenthesesStack.empty()) {
                return false; // Unmatched closing parenthesis
            } else if ((c == ')' && parenthesesStack.top() == '(') ||
                       (c == ']' && parenthesesStack.top() == '[') ||
                       (c == '}' && parenthesesStack.top() == '{')) {
                parenthesesStack.pop(); // Matched, pop from stack
            } else {
                return false; // Mismatched parentheses
            }
        }
    }

    // Check if stack is empty (all opening parentheses are matched)
    return parenthesesStack.empty();
}

int main() {
    string expression;

    cout << "Enter an expression with parentheses: ";
    getline(cin, expression);

    if (isBalanced(expression)) {
        cout << "The expression has balanced parentheses.\n";
    } else {
        cout << "The expression does not have balanced parentheses.\n";
    }

    return 0;
}
```

10)You are given a stack of elements, stack size is unknown delete middle element of the stack.

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

using namespace std;

void deleteMiddleElement(stack<int>& s, int middle) {
    if (s.size() == 0 || middle < 0 || middle >= s.size()) {
        cout << "Invalid middle index or empty stack.\n";
        return;
    }

    stack<int> tempStack;

    // Pop and store elements from the original stack until reaching the middle element
    for (int i = 0; i < middle; ++i) {
        tempStack.push(s.top());
        s.pop();
    }

    // Skip the middle element (don't push it to tempStack)
    s.pop();

    // Push back the elements from tempStack to the original stack
    while (!tempStack.empty()) {
        s.push(tempStack.top());
        tempStack.pop();
    }
}

int main() {
```

```
    stack<int> s;
    int n, element;

    cout << "Enter the number of elements in the stack: ";
    cin >> n;

    cout << "Enter the elements of the stack: ";
    for (int i = 0; i < n; ++i) {
        cin >> element;
        s.push(element);
    }

    int middle = (s.size() - 1) / 2; // Calculate the index of the middle element

    deleteMiddleElement(s, middle);

    cout << "Stack after deleting middle element: ";
    while (!s.empty()) {
        cout << s.top() << " ";
        s.pop();
    }
    cout << endl;

    return 0;
}
```

11)Reverse a stack using queue.

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

using namespace std;
```

```
void reverseStack(stack<int>& s) {
    queue<int> q;

    // Enqueue all elements of the stack into the queue
    while (!s.empty()) {
        q.push(s.top());
        s.pop();
    }

    // Dequeue all elements from the queue and push them back into the stack
    while (!q.empty()) {
        s.push(q.front());
        q.pop();
    }
}

int main() {
    stack<int> s;
    int n, element;

    cout << "Enter the number of elements in the stack: ";
    cin >> n;

    cout << "Enter the elements of the stack: ";
    for (int i = 0; i < n; ++i) {
        cin >> element;
        s.push(element);
    }

    // Reverse the stack
```

```
    reverseStack(s);

    cout << "Reversed stack: ";
    while (!s.empty()) {
        cout << s.top() << " ";
        s.pop();
    }
    cout << endl;

    return 0;
}
```