Stack

Definition 3.1

Stack is a container where elements are added and deleted according to the last-in-first-out (LIFO) order.

- Addition is called pushing
- Deleting is called popping

Example 3.1

- *Stack of papers in a copier*
- *Undo-redo features in editors*
- *Back button on Browser*

Interface of stack

Reference: https://en.cppreference.com/w/cpp/container/stack

Stack supports four interface methods

- `stack<T> s` : allocates new stack s
- `s.push(e)` : Pushes the given element e to the top of the stack.
- `s.pop()` : Removes the top element from the stack.
- `s.top()` : accesses the top element of the stack.

Some support functions

- `s.empty()` : checks whether the stack is empty
- `s.size()` : returns the number of elements

Axioms of stack

Let `s1` and `s` be stacks.

- `Assume(s1 == s); s.push(e); s.pop(); Assert(s1==s);`
- `s.push(e); Assert(s.top()==e);`

`Assume(s1 == s)` means that we assume that the content of s1 and s are the same.
`Assert(s1 == s)` means that we check that the content of s1 and s are the same.

Exercise: action on the empty stack

Exercise 3.1

Let `s` be an empty stack in C++.

- What happens when we run `s.top()`?
- What happens when we run `s.pop()`?

Ask ChatGPT.

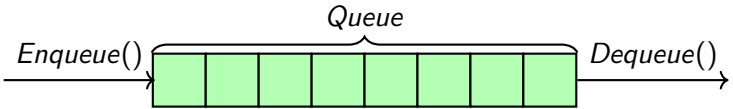
Commentary: Answer: `s.top()` will cause a segmentation fault. `s.pop()` will not cause any error and exit without any effect.

Queue

Definition 3.2

Queue is a container where elements are added and deleted according to the first-in-first-out (FIFO) order.

- Addition is called enqueue
- Deleting is called dequeue



Example 3.2

- *Entry into an airport*
- *Calling lift in a building (priority queue)*

Interface of queue

Reference: https://en.cppreference.com/w/cpp/container/queue

Queue supports four main interface methods

- `queue<T> q` : allocates new queue q
- `q.enqueue(e)` : Adds the given element e to the end of the queue. (push)
- `q.dequeue()` : Removes the first element from the queue. (pop)
- `q.front()` : access the first element .

Some support functions

- `q.empty()` : checks whether the queue is empty
- `q.size()` : returns the number of elements

Commentary: All literature uses the terms enqueue and dequeue, but unfortunately C++ library uses push for enqueue and pop uses for dequeue. Other languages such as Java uses the terms enqueue and dequeue.

Axioms of queue

1. `queue<T> q; Assert(q.empty() == true);`

2. `q.enqueue(e); Assert(q.empty() == false);`

3. `Assume(q.empty() == true);`
   `q.enqueue(e); Assert(q.front() == e);`

4. `Assume(q.empty() == false && old_q == q);`
   `q.enqueue(e); Assert(old_q.front() == q.front());`

5. `Assume(q.empty() == true && old_q == q);`
   `q.enqueue(e); q.dequeue(); Assert(old_q == q);`

6. `Assume(q.empty() == false && q == q1);`
   `q.enqueue(e);q.dequeue(); q1.dequeue();q1.enqueue(e); Assert(q == q1);`

Stack and Queue Example Problems

1. Example 1: Stack - Reverse a String

Problem: Reverse a string using a stack.

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

string reverseString(string s) {
    stack<char> st;
    for (char c : s)
        st.push(c);

    string result = "";
    while (!st.empty()) {
        result += st.top();
        st.pop();
    }
    return result;
}

int main() {
    cout << reverseString("hello") << endl;  // Output: "olleh"
    return 0;
}
```

2. Example 2: Stack - Balanced Parentheses

Problem: Check if parentheses are balanced.

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

bool isBalanced(string expr) {
    stack<char> st;
    for (char c : expr) {
        if (c == '(')
            st.push(c);
        else if (c == ')') {
            if (st.empty()) return false;
            st.pop();
        }
    }
    return st.empty();
}

int main() {
    cout << isBalanced("(())") << endl;   // Output: 1 (true)
    cout << isBalanced("(()") << endl;    // Output: 0 (false)
    return 0;
}
```

4. Example 4: Queue Using Two Stacks

Problem: Implement a queue using two stacks.

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

class QueueUsingStacks {
    stack<int> s1, s2;

public:
    void enqueue(int x) {
        s1.push(x);
    }

    int dequeue() {
        if (s2.empty()) {
            if (s1.empty()) {
                cout << "Queue is empty\n";
                return -1;
            }
            while (!s1.empty()) {
                s2.push(s1.top());
                s1.pop();
            }
        }
        int front = s2.top();
        s2.pop();
        return front;
    }
};

int main() {
    QueueUsingStacks q;
    q.enqueue(1);
    q.enqueue(2);
    cout << q.dequeue() << endl;  // Output: 1
    cout << q.dequeue() << endl;  // Output: 2
    return 0;
}
```

5. Example 5: Stack - Next Greater Element

Problem: For each element in an array, find the next greater element to the right. If no greater element exists, use -1.

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

vector<int> nextGreaterElements(vector<int>& nums) {
    stack<int> st;
    vector<int> result(nums.size(), -1);

    for (int i = nums.size() - 1; i >= 0; --i) {
        while (!st.empty() && st.top() <= nums[i])
            st.pop();

        if (!st.empty())
            result[i] = st.top();

        st.push(nums[i]);
    }

    return result;
}

int main() {
    vector<int> nums = {4, 5, 2, 25};
    vector<int> result = nextGreaterElements(nums);

    for (int x : result)
        cout << x << " ";  // Output: 5 25 25 -1
    return 0;
}
```

6. Example 6: Stack - Check for Palindrome Using Stack

Problem: Use a stack to check if a string is a palindrome (ignoring spaces and case).

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

bool isPalindrome(string s) {
    stack<char> st;
    string cleaned = "";

    for (char c : s) {
        if (isalnum(c)) {
            cleaned += tolower(c);
            st.push(tolower(c));
        }
    }

    for (char c : cleaned) {
        if (c != st.top()) return false;
        st.pop();
    }

    return true;
}

int main() {
    cout << isPalindrome("Madam") << endl;         // Output: 1 (true)
    cout << isPalindrome("Hello") << endl;         // Output: 0 (false)
    return 0;
}
```