哈爾濱工業大學(深圳)
HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

数据结构
Data Structures

Chapter 6 Stack and Queue

Prof. Yitian Shao
School of Computer Science and Technology

Stack and Queue

Course Overview

- Stack
  - Concept
  - C++ Syntax
  - Usage and Limitations
- Queue
  - Concept
  - C++ Syntax
  - Usage and Limitations
- Deque

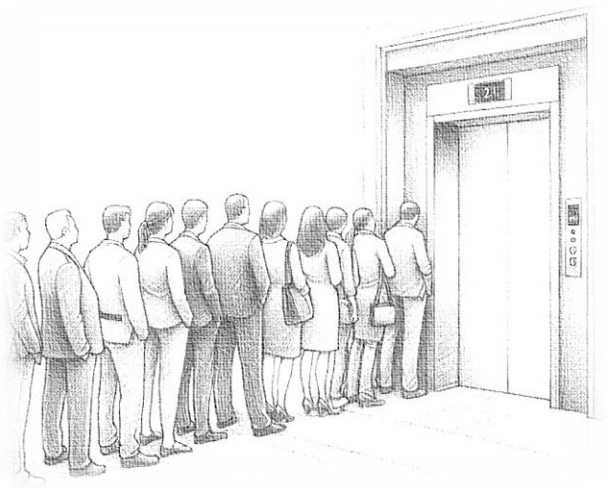
Abstract Data Type (ADT)

- **Array/Vector**: A collection of elements stored in contiguous memory locations.
- **Matrix**: 2-dimensional vector (a vector of vectors).
- **Linked List**: A collection of nodes, each containing an element and a reference to the next node (and also previous node if doubly-linked).
- Array/Vector/Matrix allow for **inserting/deleting** element anywhere in the storage, but each operation requires shifting all subsequent elements in an O(n) time complexity. **Indexing** is fast.
- Linked List: allow for **inserting/deleting** element anywhere **without** shifting subsequent elements. However, **indexing** is slow.

A new ADT: the Stack

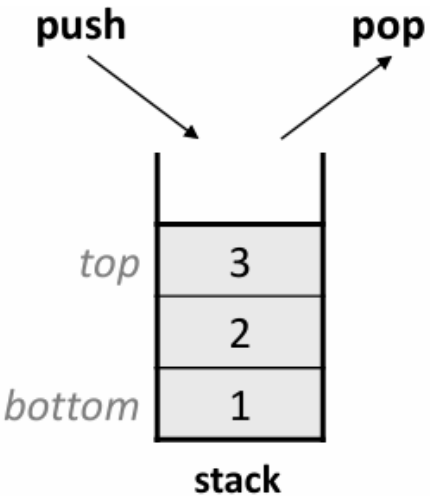
- For certain tasks, we **need not** to insert/delete element in the middle
- Sometimes, we only need to access the last element added to the storage

- **Last in first out** (LIFO)



A new ADT: the Stack

- A specialized data structure that only allows a user to add, access, and remove the **top element** - LIFO
- Super fast (O(1)) for these operations
  - Built directly into the hardware
- Main operations:
  - **push(value)**: add an element to the top of the stack
  - **pop()**: remove and return the top element in the stack
  - **peek()**: return (but do not remove) the top element in the stack



Stack examples

- Real life
  - Coins
  - Clothes
  - Plates
- In computer science
  - Function calls
  - Keeping track of edits
  - Pages visited on a website to go back to

Stack Syntax

Stack functions	Time complexity	Usage
empty()	O(1)	returns true if stack has no elements
top()	O(1)	returns **top** element without removing it; throws an error if stack is empty
pop()	O(1)	removes **top** element; throws an error if stack is empty
push(value)	O(1)	places given element on **top** of stack
size()	O(1)	returns number of elements in stack

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

stack<int> nums;
nums.push(1);
nums.push(3);
nums.push(5);
// {1, 3, 5}
cout << nums.top() << endl; // 5
nums.pop(); // nums = {1, 3}
```

Stack limitations

- You **cannot** access a stack's elements by index
- Instead, you pull elements out of the stack one at a time
- common pattern: Pop each element until the stack is empty

```
// process (and empty!) an entire stack
while (!nums.isEmpty()) {
    int x = nums.pop(); // store stack data in x
}
```

Sentence Reversal

- Goal: print the words of a sentence in reverse order
  - "Hello my name is Inigo Montoya" → "Montoya Inigo is name my Hello“
  - "Inconceivable" → "Inconceivable"
- Assume characters are only letters and spaces
- How could we use a Stack?

Sentence Reversal

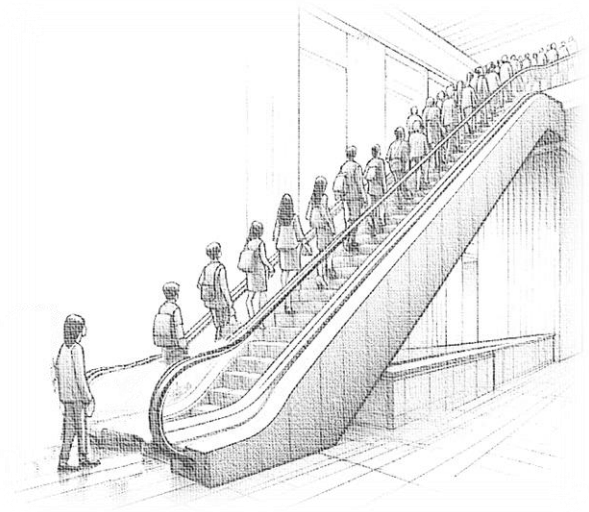
```
void printSentenceReverse(const string &sentence) {
        Stack<string> wordStack;
        string word = "";
        for (char c : sentence) {
                if (c == SPACE) {          // Space suggests the end of a word
                        wordStack.push(word);
                        word = "";         // reset the word to an empty string
                }
                else {
                        word += c;         // append characters to construct the word
                }
        }
        if (word != "") {
                wordStack.push(word);
        }
        cout << " New sentence: ";
        while (!wordStack.isEmpty()) {
                word = wordStack.pop();
                cout << word << " ";
        }
}
```

Homework

- Complete **exercise 6.1 - 6.5** (at the end of the lecture slides)

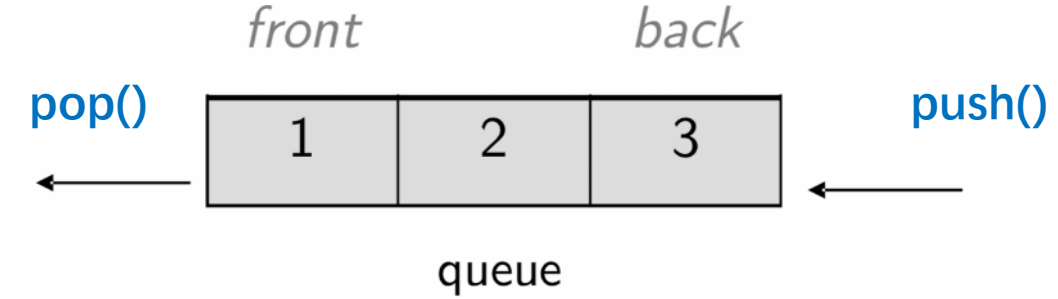
Queue

- What if we want to remove from the bottom instead of the top?
- **First In First Out** – FIFO
- Real World
  - Escalator
  - Anything first-come first-serve
- Computers
  - Sending jobs to a printer
  - Call services (being put on hold)



Queue

- Queue: ADT that retrieves elements in the order they were added.
  - There are **no indexes** (just like a stack)
  - Can **only add to the end** of the queue and **remove from the front**.
- Operations
  - **enqueue**: add an element to the back
  - **dequeue**: remove the front element
  - **examine front or back** (but do **NOT** remove) the front element



Queue Syntax

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

int main() {

  queue<int> q;

  q.push(3);
  q.push(4);
  q.push(5);

  while (!q.empty()) {
      cout << q.front() << " ";
      q.pop();
  }
  return 0;
}
```

Queue functions	Time complexity	Usage
empty()	O(1)	returns true if queue has no elements
front()	O(1)	returns **front** element without removing it; throws an error if queue is empty
back()	O(1)	returns **back** element without removing it; throws an error if queue is empty
pop()	O(1)	removes **front** element; throws an error if queue is empty
push(value)	O(1)	places given value on **back** of queue
size()	O(1)	returns number of elements in queue

Queue Question

```
queue<int> q;
for (int i = 1; i <= 6; i++) {
  q.push(i);
}
for (int i = 0; i < q.size(); i++) {
  cout << q.front() <<" ";
  q.pop();
}
cout << " size " << q.size() << endl;
```

Result?

A. 1 2 3 4 5 6 size 0

B. 1 2 3 size 3

C. 1 2 3 4 5 6 size 6

D. none of the above

What is stored inside q after the program completed?

In-class Exercise A

- Write a function **Repeat** that accepts a queue of integers and replaces every element with two copies of itself. For example: {1, 2, 3} becomes {1, 1, 2, 2, 3, 3}

```
void Repeat(queue<int>& q) {
  int q_size = q.size();
  for (int i = 0; i < q_size; i++) {
    int n = q.front();
    q.pop();
    q.push(n);
    q.push(n);
  }
}
```

Queue Tips

- You cannot access a queue's elements by index. Instead, you pop elements out of the queue one at a time.

```
// Process (and empty) an entire queue
while (!q.isEmpty()) {
    // do something with q.pop();
}
```

- Be careful iterating over a queue if you are changing it.

```
// Save the size before changing the queue
int q_size = q.size();
for (int i = 0; i < q_size; i++) {
    // do something with q.pop();
}
```

In-class Exercise B

- Mixed Usage of Stacks and Queues:

Reverse the order of elements in a queue

```
int main(){
    queue<int> q;
    q.push(2);
    q.push(3);
    q.push(4);
    q.push(5);

    //Reverse the order
    //of elements in q



     PrintQueue(q);
}
```

```
// function to print the queue (of any data type)
template<typename T> void PrintQueue(queue<T> q)
{
  while (!q.empty()) {
    cout << " " << q.front();
    q.pop();
  }
  cout << endl;
}
```

In-class Exercise B

Reverse the order of elements in a queue

```
int main(){
    queue<int> q;
    q.push(2);
    q.push(3);
    q.push(4);
    q.push(5);
    //Reverse the order of elements in q
    while (!q.empty()) {
       s.push(q.front());
       q.pop();
    }
    while (!s.empty()) {
       q.push(s.top());
       s.pop();
    }
     PrintQueue(q);
}
```

In-class Exercise C

- Write a function **Mirror** that accepts a **queue of strings** and appends the queue's contents to itself in reverse order. For example: {"a", "b", "c"} becomes {"a", "b", "c", "c ", "b", "a"}

```
#include <iostream>
#include <queue>
#include <stack>
#include <string>
int main()
{
    queue<string> q;
    q.push("a");
    q.push("b");
    q.push("c");

    Mirror(q);
    PrintQueue(q);
}
```

```
void Mirror(queue<string>& q) {

}
```

```
template<typename T> void
PrintQueue(queue<T> q)
{
  while (!q.empty()) {
    cout << " " << q.front();
    q.pop();
  }
  cout << endl;
}
```

In-class Exercise C: Solution

- Write a function **Mirror** that accepts a **queue of strings** and appends the queue's contents to itself in reverse order. For example: {"a", "b", "c"} becomes {"a", "b", "c", "c ", "b", "a"}

```
#include <iostream>
#include <queue>
#include <stack>
#include <string>
int main()
{
    queue<string> q;
    q.push("a");
    q.push("b");
    q.push("c");

    Mirror(q);
    PrintQueue(q);
}
```

```
void Mirror(queue<string>& q) {
    stack<string> s;
    int q_size = q.size();
    for (int i = 0; i < q_size; i++) {
        string str = q.front();
        q.pop();
        s.push(str);
        q.push(str);
    }
    while (!s.empty()) {
        q.push(s.top());
        s.pop();
    }
}
```

```
template<typename T> void
PrintQueue(queue<T> q)
{
  while (!q.empty()) {
    cout << " " << q.front();
    q.pop();
  }
  cout << endl;
}
```

Deque

- Deque ("deck"): **double-ended queue** is an **indexed sequence** container that allows **fast insertion** and **deletion** at both its **front** and its **end**.
- Basic Operations
  - push_front(), push_back()
  - pop_front(), pop_back()
  - front(), back()
  - [ ] // (indexing)
- Get **queue** and **stack** functionality in one data structure!

Exercise 6.1

- Complete LeetCode 20

20. Valid Parentheses

Easy | Topics | Companies | Hint

Given a string `s` containing just the characters `'('`, `')'`, `'{'`, `'}'`, `'['` and `']'`, determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.
3. Every close bracket has a corresponding open bracket of the same type.

Exercise 6.2

- Complete LeetCode 1047 Using a Stack

1047. Remove All Adjacent Duplicates In String

Easy | Topics | Companies | Hint

You are given a string `s` consisting of lowercase English letters. A **duplicate removal** consists of choosing two **adjacent** and **equal** letters and removing them.

We repeatedly make **duplicate removals** on `s` until we no longer can.

Return *the final string after all such duplicate removals have been made*. It can be proven that the answer is **unique**.

Example 1:

```
Input: s = "abbaca"
Output: "ca"
Explanation:
For example, in "abbaca" we could remove "bb" since the letters are adjacent and equal, and this is the only
possible move.  The result of this move is that the string is "aaca", of which only "aa" is possible, so the final
string is "ca".
```

Example 2:

```
Input: s = "azxxzy"
Output: "ay"
```

Exercise 6.3

- Complete LeetCode 2000 Using a Stack

2000. Reverse Prefix of Word

Easy | Topics | Companies | Hint

Given a **0-indexed** string `word` and a character `ch`, **reverse** the segment of `word` that starts at index `0` and ends at the index of the **first occurrence** of `ch` (**inclusive**). If the character `ch` does not exist in `word`, do nothing.

- For example, if `word = "abcdefd"` and `ch = "d"`, then you should **reverse** the segment that starts at `0` and ends at `3` (**inclusive**). The resulting string will be `"dcbaefd"`.

Return *the resulting string*.

Example 1:

```
Input: word = "abcdefd", ch = "d"
Output: "dcbaefd"
Explanation: The first occurrence of "d" is at index 3.
Reverse the part of word from 0 to 3 (inclusive), the resulting string is "dcbaefd".
```

Exercise 6.4

- Complete LeetCode 682 Using a Stack

682. Baseball Game

Easy | Topics | Companies

You are keeping the scores for a baseball game with strange rules. At the beginning of the game, you start with an empty record.

You are given a list of strings `operations`, where `operations[i]` is the i^{th} operation you must apply to the record and is one of the following:

- An integer `x`.
  - Record a new score of `x`.
- `'+'`.
  - Record a new score that is the sum of the previous two scores.
- `'D'`.
  - Record a new score that is the double of the previous score.
- `'C'`.
  - Invalidate the previous score, removing it from the record.

Return *the sum of all the scores on the record after applying all the operations*.

The test cases are generated such that the answer and all intermediate calculations fit in a **32-bit** integer and that all operations are valid.

Exercise 6.5

- Complete LeetCode 1475

1475. Final Prices With a Special Discount in a Shop

Easy | Topics | Companies | Hint

You are given an integer array `prices` where `prices[i]` is the price of the i^{th} item in a shop.

There is a special discount for items in the shop. If you buy the i^{th} item, then you will receive a discount equivalent to `prices[j]` where `j` is the minimum index such that `j > i` and `prices[j] <= prices[i]`. Otherwise, you will not receive any discount at all.

Return an integer array `answer` where `answer[i]` is the final price you will pay for the i^{th} item of the shop, considering the special discount.

Example 1:

```
Input: prices = [8,4,6,2,3]
Output: [4,2,4,2,3]
Explanation:
For item 0 with price[0]=8 you will receive a discount equivalent to prices[1]=4, therefore, the final price you will pay is 8 - 4 = 4.
For item 1 with price[1]=4 you will receive a discount equivalent to prices[3]=2, therefore, the final price you will pay is 4 - 2 = 2.
For item 2 with price[2]=6 you will receive a discount equivalent to prices[3]=2, therefore, the final price you will pay is 6 - 2 = 4.
For items 3 and 4 you will not receive any discount at all.
```

Exercise 6.6

- Complete LeetCode 225

225. Implement Stack using Queues

Easy | Topics | Companies

Implement a last-in-first-out (LIFO) stack using only two queues. The implemented stack should support all the functions of a normal stack (`push`, `top`, `pop`, and `empty`).

Implement the `MyStack` class:

- `void push(int x)` Pushes element x to the top of the stack.
- `int pop()` Removes the element on the top of the stack and returns it.
- `int top()` Returns the element on the top of the stack.
- `boolean empty()` Returns `true` if the stack is empty, `false` otherwise.

Notes:

- You must use **only** standard operations of a queue, which means that only `push to back`, `peek/pop from front`, `size` and `is empty` operations are valid.
- Depending on your language, the queue may not be supported natively. You may simulate a queue using a list or deque (double-ended queue) as long as you use only a queue's standard operations.

Exercise 6.7

- Complete LeetCode 2073 with Queue-based Simulation

2073. Time Needed to Buy Tickets

Easy | Topics | Companies | Hint

There are `n` people in a line queuing to buy tickets, where the `0th` person is at the **front** of the line and the `(n - 1)th` person is at the **back** of the line.

You are given a **0-indexed** integer array `tickets` of length `n` where the number of tickets that the `ith` person would like to buy is `tickets[i]`.

Each person takes **exactly 1 second** to buy a ticket. A person can only buy **1 ticket at a time** and has to go back to **the end** of the line (which happens **instantaneously**) in order to buy more tickets. If a person does not have any tickets left to buy, the person will **leave** the line.

Return the **time taken** for the person **initially** at position **k** (0-indexed) to finish buying tickets.

Simulate the ticket-buying process by using a queue of indexes:

1) Each individual is **popped** from the front of the queue and receives a ticket;
2) If an individual still needs more tickets, **push** them to the back of the queue;
3) Iterating until the queue is **empty**;
4) If the k-th person has bought all their tickets, return the time.

Exercise 6.8

- Complete LeetCode 232

232. Implement Queue using Stacks

Easy | Topics | Companies

Implement a first in first out (FIFO) queue using only two stacks. The implemented queue should support all the functions of a normal queue (`push`, `peek`, `pop`, and `empty`).

Implement the `MyQueue` class:

- `void push(int x)` Pushes element x to the back of the queue.
- `int pop()` Removes the element from the front of the queue and returns it.
- `int peek()` Returns the element at the front of the queue.
- `boolean empty()` Returns `true` if the queue is empty, `false` otherwise.

Notes:

- You must use **only** standard operations of a stack, which means only `push to top`, `peek/pop from top`, `size`, and `is empty` operations are valid.
- Depending on your language, the stack may not be supported natively. You may simulate a stack using a list or deque (double-ended queue) as long as you use only a stack's standard operations.

Exercise 6.9

- Complete LeetCode 950 with Queue-based Simulation

950. Reveal Cards In Increasing Order

Medium | Topics | Companies

Simulate the revealing process using a **queue** of indices to find the order that the indices of cards be revealed

You are given an integer array `deck`. There is a deck of cards where every card has a unique integer. The integer on the `i^th` card is `deck[i]`.

You can order the deck in any order you want. Initially, all the cards start face down (unrevealed) in one deck.

You will do the following steps repeatedly until all cards are revealed:

1. Take the top card of the deck, reveal it, and take it out of the deck.
2. If there are still cards in the deck then put the next top card of the deck at the bottom of the deck.
3. If there are still unrevealed cards, go back to step 1. Otherwise, stop.

Return *an ordering of the deck that would reveal the cards in increasing order*.

Note that the first entry in the answer is considered to be the top of the deck.