# Lab: Stacks and Queues

Problems for exercises and homework for the ["CSharp Advanced" course @ Software University](https://softuni.bg/courses/csharp-advanced).

You can check your solutions here: <https://judge.softuni.bg/Contests/1445/Stacks-and-Queues-Lab>

1. Working with Stacks

## Reverse Strings

Write program that:

* **Reads** an **input string**
* **Reverses** it **using a** Stack<T>
* **Prints** the result back at the terminal

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| I Love C# | #C evoL I |
| Stacks and Queues | seueuQ dna skcatS |

### Hints

* Use a Stack<string>
* Use the methods **Push()**, **Pop()**

## Stack Sum

Write program that:

* **Reads** an **input of integer numbers** and **adds** them to a **stack**
* **Reads commands** until **"end"** is received
* **Prints** the **sum** of the remaining elements of the **stack**

### Input

* On the **first** **line** you will receive **an array of integers**
* On the **next** **lines**, until the "**end**" command is given, you will receive **commands** – a **single** **command** and **one** or **two** numbers after the **command, depending** on what **command** you are given
* If the **command** is "**add**", you will **always** receive **exactly two** numbers after the command which you need to **add** in the **stack**
* If the **command** is "**remove**", you will **always** receive **exactly** **one** number after the command which represents the **count** of the numbers you need to **remove** from the **stack.** If there are **not enough elements** skip the command.

### Output

* When the **command** "**end**" is received, you need to **print the sum** of the **remaining** elements in the **stack**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 2 3 4  adD 5 6  REmove 3  end | Sum: 6 |
| 3 5 8 4 1 9  add 19 32  remove 10  add 89 22  remove 4  remove 3  end | Sum: 16 |

### Hints

* Use a Stack<int>
* Use the methods **Push()**, **Pop()**
* Commands **may** be given in **mixed case**

## Decimal to Binary Converter

Create a simple program that **can convert a decimal number to its binary representation**. Implement an elegant solution **using a Stack**.

**Print the binary representation** back at the terminal.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 10 | 1010 |
| 1024 | 10000000000 |

### Hints

* If the given number is 0, just print 0
* Else, while the number is greater than zero, divide it by 2 and push the remainder into the stack
* When you are done dividing, pop all remainders from the stack – that is the binary representation

## Simple Calculator

**Create a simple calculator** that can **evaluate simple expressions** with only addition and subtraction. There will not be any parentheses.

Solve the problem **using a Stack**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2 + 5 + 10 - 2 - 1 | 14 |
| 2 - 2 + 5 | 5 |

### Hints

* Use a **Stack<string>**
* You can either
  + add the elements and then Pop() them out
  + or Push() them and reverse the stack

## Matching Brackets

We are given an arithmetic expression with brackets. Scan through the string and extract each sub-expression.

Print the result back at the terminal.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 + (2 - (2 + 3) \* 4 / (3 + 1)) \* 5 | (2 + 3)  (3 + 1)  (2 - (2 + 3) \* 4 / (3 + 1)) |
| (2 + 3) - (2 + 3) | (2 + 3)  (2 + 3) |

### Hints

* Scan through the expression searching for brackets
  + If you find an opening bracket, push the index into the stack
  + If you find a closing bracket pop the topmost element from the stack. This is the index of the opening bracket.
  + Use the current and the popped index to extract the sub-expression

1. Working with Queues

## Print Even Numbers

Write program that:

* **Reads** an array of **integers** and **adds** them to a **queue**
* **Prints** the **even** numbers **separated** by "**,** "

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 2 3 4 5 6 | 2, 4, 6 |
| 11 13 18 95 2 112 81 46 | 18, 2, 112, 46 |

### Hints

* Use a Queue<int>
* Use the methods **Enqueue()**, **Dequeue(), Peek()**

## Supermarket

**Reads** an **input** consisting of a **name** and **adds** it to a **queue** until "**End**" is received. If you receive "Paid", **print** every customer name and empty the queue, otherwise we receive a client and we have to add him to the queue. When we receive **"End"** we have to print the count of the remaining people in the queue in the format: "**{count} people remaining.".**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Liam  Noah  James  Paid  Oliver  Lucas  Logan  Tiana  End | Liam  Noah  James  4 people remaining. |
| Amelia  Thomas  Elias  End | 3 people remaining. |

## Hot Potato

Hot potato is a game in which **children form a circle and start passing a hot potato**. The counting starts with the fist kid. **Every nth toss the child left with the potato leaves the game**. When a kid leaves the game, it passes the potato along. This continues **until there is only one kid left**.

Create a program that simulates the game of Hot Potato. **Print every kid that is removed from the circle**. In the end, **print the kid that is left last**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Alva James William  2 | Removed James  Removed Alva  Last is William |
| Lucas Jacob Noah Logan Ethan  10 | Removed Ethan  Removed Jacob  Removed Noah  Removed Lucas  Last is Logan |
| Carter Dylan Jack Luke Gabriel  1 | Removed Carter  Removed Dylan  Removed Jack  Removed Luke  Last is Gabriel |

## Traffic Jam

Create a program that simulates the **queue** that forms during a **traffic** **jam**. During a traffic jam only **N** cars can **pass** the crossroads when the **light** **goes** **green**. Then the program reads the **vehicles** that **arrive** one by one and **adds** them to the **queue**. When the light **goes** **green** **N** number of cars **pass** the crossroads and **for** **each** a **message** "{car} passed!" is displayed. When the "**end**" command is given, **terminate** the program and **display** a **message** with the **total** **number** of cars that **passed** the crossroads.

### Input

* On the **first** **line** you will receive **N** – the number of cars that can pass during a green light
* On the **next** **lines**, until the "**end**" command is given, you will receive **commands** – a **single** **string**, either a **car** or "**green**"

### Output

* Every time the "**green**" command is given, **print** **out** a message for **every** **car** that **passes** the crossroads in the format "{car} passed!"
* When the "**end**" command is given, **print** **out** a message in the format "{number of cars} cars passed the crossroads."

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 4  Hummer H2  Audi  Lada  Tesla  Renault  Trabant  Mercedes  MAN Truck  green  green  Tesla  Renault  Trabant  end | Hummer H2 passed!  Audi passed!  Lada passed!  Tesla passed!  Renault passed!  Trabant passed!  Mercedes passed!  MAN Truck passed!  8 cars passed the crossroads. |
| 3  Enzo's car  Jade's car  Mercedes CLS  Audi  green  BMW X5  green  end | Enzo's car passed!  Jade's car passed!  Mercedes CLS passed!  Audi passed!  BMW X5 passed!  5 cars passed the crossroads. |