# Lab: Stacks and Queues

Problems for exercises and homework for the ["C# Advanced" course @ SoftUni](https://softuni.bg/trainings/1633/csharp-advanced-may-2017).

You can check your solutions here: <https://judge.softuni.bg/Contests/Practice/Index/572#5>.

# Working wth Stacks

## Reverse Strings

Write program that reads:

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

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Learning Java | avaJ gninraeL |
| Stacks and Queues | seueuQ dna skcatS |

### Hints

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

## Simple Calculator

**Create a simple calculator** that can **evaluate simple expressions** that will not hold any operator different from addition and subtraction. There will not be parentheses or operator precedence.

Solve the problem **using a Stack**.

### Examples

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

### Hints

* Use an **Stack<string>**
* You can either
  + add the elements and then pop them out
  + or push them and reverse the stack

## 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 reminder into the stack
* When you are done dividing, pop all reminders from the stack, that is the binary representation

## Matching Brackets

We are given an arithmetical 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

# Working with Queues

## 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 forward. This continues repeating **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** |
| Mimi Pepi Toshko  2 | Removed Pepi  Removed Mimi  Last is Toshko |
| Gosho Pesho Misho Stefan Krasi  10 | Removed Krasi  Removed Pesho  Removed Misho  Removed Gosho  Last is Stefan |
| Gosho Pesho Misho Stefan Krasi  1 | Removed Gosho  Removed Pesho  Removed Misho  Removed Stefan  Last is Krasi |

## Math Potato

Rework the previous problem so that a **child is removed only on a prime cycle** (cycles start from 1)

If a **cycle is not prime**, just **print the child's name.**

As before, print the name of the child that is left last.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Mimi Pepi Toshko  2 | Removed Pepi  Prime Mimi  Prime Toshko  Removed Mimi  Last is Toshko |
| Gosho Pesho Misho Stefan Krasi  10 | Removed Krasi  Prime Pesho  Prime Misho  Removed Stefan  Prime Gosho  Removed Gosho  Prime Misho  Removed Pesho  Last is Misho |