
1 Assignment 2 Introduction

This assignment is due *before* 11:59pm on the listed date. Which means, submissions made *on or after* 11:59pm will be counted as a late submission. Late submissions *before* 11:59 pm on the following day will receive a 25% point deduction as a penalty. Submissions made *on or after* 11:59pm on the same day will *not* be graded. We strongly recommend completing the assignment before then.

Assignments due on Mondays generally involve materials covered in lectures from the previous week, whereas assignments due on Thursdays involve materials covered in lectures from the running week. So, be sure to watch the lectures and go over the reading materials before attempting the assignments.

Good luck!

2 No Strings Attached (40 points)

You are tasked with creating a simple ‘Rock Paper Scissors’ game, but with a little twist! You are not allowed to use `string` (other than with `std::cout`) or `char` in your code.

To help you with this problem, you are given a `RandNumGen` function that randomly generates (and returns) 1, 2, or 3 when called.

```
#include <iostream>
#include <random>

std::random_device rd;
std::mt19937 gen(rd());

int RandNumGen() {
    std::uniform_int_distribution<> rng(1, 3);
    return rng(gen);
}

int main() {
    //Your code here
    return 0;
}
```

Some test cases are provided below:

```
Enter 1 (Rock), 2 (Paper), or 3 (Scissors): 2
You picked: Paper
The computer picked: Rock
You win!
```

```
Enter 1 (Rock), 2 (Paper), or 3 (Scissors): 1
You picked: Rock
The computer picked: Paper
You lose!
```

```
Enter 1 (Rock), 2 (Paper), or 3 (Scissors): 3
You picked: Scissors
The computer picked: Scissors
It's a draw!
```

General Instructions:

- When writing your program, you may only use concepts that you have learned in the course thus far. The only exceptions are **string** and **char**, which you are not allowed to use to solve this problem.
- Your program should compile and run. You can assume that the input will be 1, 2, or 3.
- The formatting must match exactly as shown in the example above, including whitespaces. The blue texts in the test cases indicate user input.

3 Tic-tac-toe Validator (60 points)

Everyone knows how to play tic-tac-toe. At the end of a tic-tac-toe match, we are left with a board with Xs and Os. In this problem, we will refer to the final state of a tic-tac-toe board as a *configuration*. Using 0 (resp., 1) to characterize 0 (resp., X), and 2 to characterize empty squares, we can represent the configuration of a tic-tac-toe board with an integer. For example:

X 0 0		0 0 X		0 0 0		
0 X X	=	100011110		X X	=	112210
X X 0		X		X 0		

Note that not all configurations have 9 digits, due to some having leading 0's. Your task is to write a function `ValidateConfig` that takes a configuration as input, and returns the string `X wins!`, `0 wins!`, `It's a draw!`, or `Invalid configuration!`, depending on the configuration. An invalid configuration is a configuration that is impossible to achieve if the rules are followed. For example, `111111111` is an invalid configuration, since it is impossible for a tic-tac-toe board to have 9 Xs.

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

std::string ValidateConfig(int config) {
    //Your code here
}

int main() {
    int config{0};
    std::cout << "Enter configuration: ";
    std::cin >> config;
    std::cout << "Result: " << ValidateConfig(config) << std::endl;
    return 0;
}
```

Some test cases are provided below:

Enter configuration: 101012001

Result: X wins!

Enter configuration: 001021101

Result: X wins!

Enter configuration: 101000112

Result: O wins!

Enter configuration: 112210

Result: O wins!

Enter configuration: 110011100

Result: It's a draw!

Enter configuration: 100101111

Result: Invalid configuration!

Enter configuration: 011112212

Result: Invalid configuration!

General Instructions:

- When writing your program, you may only use concepts that you have learned in the course thus far.
- Your program should compile and run. You can assume that the input will always be an integer that represents a configuration. For example, you do not need to consider inputs like 1010101010, 9, and so on.
- The formatting should match exactly as shown in the example above, including white-spaces. The blue texts in the test cases indicate user input.

4 Assignment 2 Submission Process

It is imperative that you meticulously follow the submission process outlined below. Incorrectly structured submissions will receive a 10% point deduction as a penalty.

- Create a folder, name it `your_msu_id2`. For example, if your MSU email is `johndoe@msu.edu`, then you should name the folder `johndoe2`.
- For each programming task, create a sub-folder inside your `your_msu_id2` folder, and name it as the number that corresponds to the programming task number. For this assignment, there should be two sub-folders named '2' and '3'.
- Inside each sub-folder, put the `main.cpp` for the appropriate solution.
- Compress/Zip `your_msu_id2` folder and name it `your_msu_id2.zip`. For example, if the name of your folder is `johndoe2`, then you need to create a zip file named `johndoe2.zip`. Zip file guide: <https://copyrightservice.co.uk/reg/creating-zip-files>.
- Submit the zip file through D2L.