# CS2310 2021/22 Semester B Mid-Term Quiz

1. Total time allowed: 90 minutes

2. Three questions and do not include any additional libraries

3. Download “Mid-Term.cpp” from Canvas, and re-name it using your student ID as “YourStudentID.cpp”, e.g., if your student ID is 1234567, name it as “12345678.cpp”

4. Write your student ID in “line 12” of “YourStudentID.cpp” as well

5. Write all your solutions in “YourStudentID.cpp”

1. **Computation [30%]**

The formula $f$ has two inputs: ***n*** (date type: **int**) and ***x*** (date type: **double**). The result of $f$ is also **double** type.

$$f = n/(x^1 \times 1! + x^2 \times 2! + x^3 \times 3! + \cdots + x^n \times n!)$$

Write a program according to the following requirements:

- For the inputs of ***n*** and ***x***, we assume that they are already the **int-type** and **double-type** numbers, respectively, i.e., you do not need to check whether they are belonged to any other data types.

- For the input ***n***, check whether it is **positive**. If no, ask the user to input ***n*** again.
- Next, for the input ***x***, check whether $0 < x \leq 0.2$. If no, ask the user to input ***x*** again.

- If both ***n*** and ***x*** are valid, compute and output the value of $f$, with only **two digits** in the decimal part.

**Note 1:** Your code should **NOT** use the function **pow(x,i)**.
**Note 2:** Your code should **NOT** use **recursion** or any existing factorial function to calculate $n!$, i.e., $n! = n \times (n-1) \times (n-2) \times \cdots \times 2 \times 1$. Please use loop to calculate $n!$.
**Note 3:** We consider $n!$ is **NOT** very large, which can be stored in an **int-type** variable.
**Note 4:** You can compute the **denominator** first $f = x^1 \times 1! + x^2 \times 2! + x^3 \times 3! + \cdots + x^n \times n!$ and then $f = n/f$.

Example-1 (Inputs are underlined):

```
Please input n (positive): 3
Please input x (0<x<=0.2): 0.2
The value of f is: 9.15
```

Example-2 (Inputs are underlined):

```
Please input n (positive): 10
Please input x (0<x<=0.2): 0.1
The value of f is: 75.78
```

Example-3 (Inputs are underlined):

```
Please input n (positive): 0
Please input n (positive): -1
Please input n (positive): 10
Please input x (0<x<=0.2): 0
Please input x (0<x<=0.2): 0.3
Please input x (0<x<=0.2): 0.19
The value of f is: 11.75
```

2. **Calculation [30%]**

Please write a program to finish the following three parts:

**Part-(a)**. The program reads one number ***num1*** (**data type**: **int**) of **five** digits and another number ***num2*** (**data type**: **int**) of **one** digit. Please determine whether ***num1*** contains ***num2***.

- We assume inputs are correct, where ***num1*** and ***num2*** always have 5 digits and 1 digit, respectively. No need to check their correctness.
- **Hint**: extract the **last digit** of ***num1*** and compare it with ***num2***. If not equal, update ***num1*** as the rest digits and repeat the previous step. For example, ***num1*** = 12345 and ***num2*** is 4. In the first iteration, we get the last digit of ***num1***, which is 5 and different from 4. We update ***num1*** as the 1234 to start the second iteration. We get the last digit of this new ***num1***, which is 4 and is the same as ***num2***.

**Part-(b)**. Please continue to read one number ***num1*** (**data type**: **int**) of **five** digits and another number ***num2*** (**data type**: **int**) of **two** digits. Please determine whether ***num1*** contains ***num2***.

- We assume inputs are correct, where ***num1*** and ***num2*** always have 5 and 2 digits, respectively. No need to check their correctness.
- Hint: extend the hint of part-(a) for two digits each time.

**Part-(c)**. Please continue to read one number ***num1*** (**data type**: **int**) of **five** digit and another number ***num2*** (**data type**: **int**) with *no more than* **five** digits. Please determine whether ***num1*** contains ***num2***.

- We assume inputs are correct, where ***num1*** always has 5 digits and the number of digits in ***num2*** is 1, 2, 3, 4, or 5. No need to check the input correctness.

**Note1**: we assume that all the input digits of ***num1*** and ***num2*** are meaningful, i.e., the most significant digit is not zero, except it is 0. For example, we do not consider the inputs, like 05678, 01, 00, etc.
**Note 2:** we assume ***num1*** and ***num2*** are always **non-negative**.

Example-1 (Inputs are underlined):

```
Part-(a), please enter num1 and num2: 12345 1
12345 contains 1
Part-(b), please enter num1 and num2: 12345 13
12345 does not contain 13
Part-(c), please enter num1 and num2: 12345 1234
12345 contains 1234
```

Example-2 (Inputs are underlined):

```
Part-(a), please enter num1 and num2: 12340 0
12340 contains 0
Part-(b), please enter num1 and num2: 12305 30
12305 contains 30
Part-(c), please enter num1 and num2: 12345 12345
12345 contains 12345
```

Example-3 (Inputs are underlined):

```
Part-(a), please enter num1 and num2: 12003 0
12340 contains 0
Part-(b), please enter num1 and num2: 10000 11
10000 does not contain 11
Part-(c), please enter num1 and num2: 12321 123
12321 contains 123
```

## 3. Game [40%]

Please develop a game, where three symbols '**o**' '**x**' '#' are randomly generated on a 7x7 game board initially. In each step, you can swap two **adjacent** symbols (in the same *row* or *column*). If three consecutive symbols become identical, you convert these three symbols to - - - on the game board. There are three key steps:

- **Initialization**. Given a random number 1, 2, or 3 (**date type**: **int**) for each cell of the 7x7 game board, generate the game board in the console using **o, x,** # according to the corresponding 1, 2, and 3 value.

  Here is an example of the generated random numbers (left), and the displayed game board (right):

| | | | | | | |
|---|---|---|---|---|---|---|
| **1** | **1** | **2** | **3** | **1** | **2** | **2** |
| **2** | **3** | **1** | **1** | **1** | **2** | **1** |
| **2** | **1** | **1** | **2** | **3** | **1** | **3** |
| **1** | **2** | **3** | **3** | **3** | **2** | **3** |
| **3** | **1** | **3** | **3** | **1** | **3** | **2** |
| **2** | **2** | **3** | **2** | **1** | **2** | **1** |
| **3** | **2** | **2** | **1** | **2** | **2** | **1** |

| | | | | | | |
|---|---|---|---|---|---|---|
| o | o | x | # | o | x | x |
| x | # | o | o | o | x | o |
| x | o | o | x | # | o | # |
| o | x | # | # | # | x | # |
| # | o | # | # | o | # | x |
| x | x | # | x | o | x | o |
| # | x | x | o | x | x | o |

- **Pre-processing**. Before game starts, if there are already three consecutive symbols being identical, you convert them by three short lines -. Hint: you can check row-by-row and then column-by-column.

  Here is an example of the game board before (left) and after (right) the pre-processing.

| | | | | | | |
|---|---|---|---|---|---|---|
| o | o | x | # | o | x | x |
| x | # | o | o | o | x | o |
| x | o | o | x | # | o | # |
| o | x | # | # | # | x | # |
| # | o | # | # | o | # | x |
| x | x | # | x | o | x | o |
| # | x | x | o | x | x | o |

| | | | | | | |
|---|---|---|---|---|---|---|
| o | o | x | # | o | x | x |
| x | # | - | - | - | x | o |
| x | o | o | x | # | o | # |
| o | x | - | - | - | x | # |
| # | o | # | # | o | # | x |
| x | x | # | x | o | x | o |
| # | x | x | o | x | x | o |

- **Play.** The player needs to input the coordinates of two consecutive cells to be swapped. If the input is valid, you need to detect whether the new three-consecutive symbols appear. If so, convert them to - - -. The coordinate of each cell is the pair of its row index and column index, defined as follows:

| | | | | | | |
|---|---|---|---|---|---|---|
| **0 0** | **0 1** | **0 2** | **0 3** | **0 4** | **0 5** | **0 6** |
| **1 0** | **1 1** | **1 2** | **1 3** | **1 4** | **1 5** | **1 6** |
| **2 0** | **2 1** | **2 2** | **2 3** | **2 4** | **2 5** | **2 6** |
| **3 0** | **3 1** | **3 2** | **3 3** | **3 4** | **3 5** | **3 6** |
| **4 0** | **4 1** | **4 2** | **4 3** | **4 4** | **4 5** | **4 6** |
| **5 0** | **5 1** | **5 2** | **5 3** | **5 4** | **5 5** | **5 6** |
| **6 0** | **6 1** | **6 2** | **6 3** | **6 4** | **6 5** | **6 6** |

  **Note**: after the swapping, more than 3 consecutive identical symbols may exist along one direction (row or column). You only need to covert three of them (not all of them). For example, for the game board (left) below, after the user inputs two coordinates 4 5 4 6, four 'x' symbols appear in column 5. Only three of them are converted to - - -.

  Meanwhile, three consecutive '#' symbols appear in another column (column 6), you need to convert them as well.

| | | | | | | |
|---|---|---|---|---|---|---|
| o | o | x | # | o | x | x |
| x | # | - | - | - | x | o |
| x | o | o | x | # | o | # |
| o | x | - | - | - | x | # |
| # | o | # | # | o | # | x |
| x | x | # | x | o | x | o |
| # | x | x | o | x | x | o |

| | | | | | | |
|---|---|---|---|---|---|---|
| o | o | x | # | o | x | x |
| x | # | - | - | - | x | o |
| x | o | o | x | # | o | - |
| o | x | - | - | - | - | - |
| # | o | # | # | o | - | - |
| x | x | # | x | o | - | o |
| # | x | x | o | x | x | o |

The requirements for the entered coordinates are as follows. **If not satisfied, the user needs to input again**.

1. The two coordinates are inside the game board.
2. The two coordinates are different.
3. The symbols on these two cells are different.
4. The symbols on these two cells do not include the '-' symbol.
5. The two coordinates are adjacent.

**Notes**:

1. For the user's input, we only need to check above five requirements.
2. We assume there are always (at least) two adjacent cells satisfying above requirements each time, i.e., the user can always input two valid coordinates each time.
3. When two valid coordinates are obtained, **DO NOT** check **all** the cells on the game board to find the three consecutive identical symbols. Check a **local** range of these two entered coordinates.
4. No need to consider when the game should end.

Example:

```
The initial game board:
o x o # o # o
x o # # x # #
x x o o # o x
x # o # x # x
# # # o # x #
# o x x o o x
o x x o # x #

The pre-processed game board:
o x o # o # o
- o # # x # #
- x o o # o x
- # o # x # x
- - - o # x #
# o x x o o x
o x x o # x #

Input the coordinates of two adjacent cells you want to swap:
0 0 0 3
Input the coordinates of two adjacent cells you want to swap:
1 4 2 4
o x o # o # o
- o - - - # #
- x o o x o x
- # o # x # x
- - - o # x #
# o x x o o x
o x x o # x #

Input the coordinates of two adjacent cells you want to swap:
1 5 2 5
o x o # o # o
- o - - - o #
- x o o x # x
- # o # x # x
- - - o # x #
# o x x o o x
o x x o # x #

......
```

– End –