TIC1001—Introduction to Computing and Programming I
National University of Singapore
School of Continuing and Lifelong Education

# Practical Examination 1

22 September 2018

**Time allowed:** 2 hours

**Instructions (please read carefully):**

1. This is **an open-book exam**. You are allowed to bring in any course or reference materials in printed form. No electronic media or storage devices are allowed.
2. This practical exam consists of **two** questions. The time allowed for solving this test is **2 hours**.
3. The maximum score of this test is **20 marks**. Note that the number of marks awarded for each question **IS NOT** correlated with the difficulty of the question.
4. You are advised to attempt all questions. Even if you cannot solve a question correctly, you are likely to get some partial credit for a credible attempt.
5. While you are also provided with the templates `q1-template.c` and `q2-template.c` to work with, your answers should be submitted on Coursemology. Note that you can **only run the test cases on Coursemology for a limited number of tries** because they are only for checking that your code is submitted correctly. You are expected to test your own code for correctness locally on VS Code and not depend only on the provided test cases. Do ensure that you pasted your answers correctly by running the test cases at least once.
6. In case there are problems with Coursemology.org and we are not able to upload the answers to Coursemology.org, you will be required to name your file `q1-<student no>.c` and `q2-<student no>.c` where `<student no>` is your student number and leave the file in the `tic1001-pe1` folder on the Desktop. If your file is not named correctly, we will choose any .c file found at random.
7. Please note that it shall be your responsibility to ensure that your solution is submitted correctly to Coursemology and correctly left in the desktop folder at the end of the examination. Failure to do so will render you liable to receiving a grade of **ZERO** for the Practical Exam, or the parts that are not uploaded correctly.
8. Please note that while sample executions are given, it is **not sufficient to write programs that simply satisfy the given examples**. Your programs will be tested on other inputs and they should exhibit the required behaviours as specified by the problems to get full credit. There is no need for you to submit test cases.

# ALL THE BEST!

## Question 1: 7-Up! [10 marks]

A number game is played where players in a circle count off from 1, i.e. the first player says "1", and the next will say "2", and so on, going multiple times around the circle until someone loses. However, if the number is a multiple of 7 or contains the digit 7, the player has to say "7-Up" instead of the number.

To make the game harder, if the number is **BOTH** a multiple of 7 and contains the digit 7, the player has to say "Double-Up" instead.

You decide to use your newly acquired programming knowledge to write a program to help you win this game. However, it is possible that the game might use a different digit from 7, so your program has to be general enough to work with all digits from 1 to 9.

### A. Contains Digit

Implement the function `bool contains_digit(int n, int d)` that returns `true` if the positive integer $n$ contains the digit $d$, and `false` otherwise. [5 marks]

**Sample tests:**

| Test function | Output |
|---|---|
| `contains_digit(12345, 2)` | `true` |
| `contains_digit(1234567890, 0)` | `true` |
| `contains_digit(1010101, 1)` | `true` |
| `contains_digit(12345, 7)` | `false` |
| `contains_digit(987654321, 0)` | `false` |

### B. Simulating the Game

Implement the function `void play(int start, int stop, int d)` which takes in the starting and ending number, and digit of play. It should **display** (i.e. print on the screen) line-by-line, the appropriate thing to say for the numbers from start to end inclusive following these rules:

- "`Double-Up`" if $n$ is a multiple of $d$ and contains the digit $d$
- "$d$`-Up`" (where $d$ is the actual digit) if $n$ is either a multiple of $d$ or contains the digit $d$.
- the number $n$, if otherwise.

Notes:

1. You may wish to reuse the function defined in part A and can assume that is has been correctly defined on Coursemology.
2. You may assume that the inputs `start` $> 0$ and `stop` $>$ `start` and $2 \leq$ `d` $\leq 9$.
3. The maximum `int` bound will not be exceeded in the computation.

[5 marks]

**Sample tests (on next page):**

| `play(30, 45, 4);` | `play(690, 710, 7);` | `play(25,45, 3);` |
|---|---|---|
| 30 | 690 | 25 |
| 31 | 691 | 26 |
| 4-Up | 692 | 3-Up |
| 33 | 7-Up | 28 |
| 4-Up | 694 | 29 |
| 35 | 695 | Double-Up |
| 4-Up | 696 | 3-Up |
| 37 | 7-Up | 3-Up |
| 38 | 698 | Double-Up |
| 39 | 699 | 3-Up |
| Double-Up | Double-Up | 3-Up |
| 4-Up | 7-Up | Double-Up |
| 4-Up | 7-Up | 3-Up |
| 4-Up | 7-Up | 3-Up |
| Double-Up | 7-Up | Double-Up |
| 4-Up | 7-Up | 40 |
| | 7-Up | 41 |
| | Double-Up | 3-Up |
| | 7-Up | 3-Up |
| | 7-Up | 44 |
| | 7-Up | 3-Up |

## C. Bonus Question

This question gives 2 additional bonus marks.

Implement the function `contains_double(int n)` which takes in a positive integer $n$, and returns `true` if $n$ contains at least two consecutively identical digits, and `false` otherwise.

**Sample tests:**

| Test function | Output |
|---|---|
| `contains_double(12345)` | `false` |
| `contains_double(123123)` | `false` |
| `contains_double(112233)` | `true` |
| `contains_double(333333)` | `true` |
| `contains_double(120021)` | `true` |

## Question 2: Bacteria [10 marks]

Dr. Wink is currently researching a medicine to control the population of the bacteria *Ccisscari*. You are asked to tabulate the experiment results.

The *Ccisscari* bacteria **doubles** its number every **generation**. Dr. Wink's medicine, applied every `G` generations, can kill off `K`% of the bacteria.

Below is an example experiments for **10** generations, with **50** *Ccisscari* bacteria in the beginning, and the medicine was applied every 3 generations to kill off 10% of the bacteria:

| Generation | Population | Kill | New population |
|---|---|---|---|
| 1 | 50 | 0 | 100 |
| 2 | 100 | 0 | 200 |
| 3 | 200 | 20 | 360 |
| 4 | 360 | 0 | 720 |
| 5 | 720 | 0 | 1440 |
| 6 | 1440 | 144 | 2592 |
| 7 | 2592 | 0 | 5184 |
| 8 | 5184 | 0 | 10368 |
| 9 | 10368 | 1036 | 18664 |
| 10 | 18664 | 0 | 37328 |

At the end of the simulation, there were **37328** bacteria left.

Your task is to implement the function `int simulate` that takes the following inputs:

- `int start_population`: The initial population of the bacteria.
- `int end_generation`: Total number of generations to simulate.
- `int G`: The interval to apply the medicine.
- `int K`: The effect of the medicine, in percentage.

This function returns the remaining bacteria at the end of the simulation. For example, `simulate(50, 10, 3, 10);` gives the answer **37328**.

Several notes:

1. The medicine is applied before the population doubles.
2. The medicine effect should be rounded down to the nearest integer (as you cannot kill a fraction of a bacteria).
3. You can assume that the maximum `int` bound will not be exceeded in the computations.

[10 marks]

**Sample tests (on next page):**

| Test function | Output |
|---|---|
| `simulate(50, 10, 3, 10)` | `37328` |
| `simulate(1, 20, 1, 50)` | `2` |
| `simulate(100, 8, 4, 90)` | `256` |
| `simulate(10, 50, 12, 100)` | `0` |

**— E N D  O F  P A P E R —**