ENGG1340 / COMP2113, Assignment 2

Due Date: Apr 8, 2022 23:59

If you have any questions, please post to the Moodle discussion forum on Assignment 1.

Total marks: 100 marks

- 5 marks for proper code comments, indentation and use of functions
- 95 marks for program correctness

A maximum of 5 marks will be deducted if you fail to follow the submission instructions strictly.

General Instructions

Read the instructions in this document carefully.

In this assignment you will solve 4 problems and a tester would automatically test your submitted program. So if your submitted files and program outputs do not conform to our instructions given here, your programs cannot be evaluated and you will risk losing marks totally.

Sample test cases are provided with each problem in this document. Note that the test cases may or may not cover all boundary cases for the problem. It is also part of the assessment whether you are able to design proper test cases to verify the correctness of your program. We will also use additional test cases when marking your submission.

Input and output format

Note carefully whether your C/C++ programs should read from the **standard input**, **command line input** or **file input**. Also, unless specified otherwise, your answer should be printed through the **standard output**. If you failed to follow the instructions, the tester may not be able to give a score for your program. Additionally, you should strictly follow the sample output format (including space, line breaker, etc.), otherwise, your answer might be considered as wrong.

How to use the sample test cases

Sample test cases in text file formats are made available for you to check against your work to avoid formatting errors which might fail the tester. Here's how you may use the sample test cases. Take Problem 2 test case 3 as an example. The sample input and the expected output are given in the files `input2_3.txt` and `output2_3.txt`, respectively. Suppose that your program is named "2", do the followings at the command prompt of the terminal to check if there is any difference between your output and the expected output.

```
./2 < input2_3.txt > myoutput.txt
diff myoutput.txt output2_3.txt
```

Testing against the sample test cases is important to avoid making formatting mistakes. The additional test cases for grading your work will be of the same formats as the sample test cases.

Coding environment

You must make sure that your program can compile, execute and generate the required outputs on our standard environment, namely, the gcc C/C++11 environment we have on the CS Linux servers (academy*).

For Problems 1, 2 and 3 on C++ programming, make sure the following compilation command is used to compile your programs:

```
g++ -pedantic-errors -std=c++11 [yourprogram].cpp
```

For Problem 4 on C programming, make sure the following compilation command is used to compile your program:

```
gcc -pedantic-errors -std=c11 4.c
```

As a programmer/developer, you should always ensure that your code can work perfectly as expected on a target (e.g., your client's) environment, not only on yours.

While you may develop your work on your own environment, you should always try your program (compile & execute & check results) on our standard environment before submission.

Submission

Name your C/C++ programs as in the following table and put them together into one directory. Make sure that the folder contains only these source files (`*.cpp` / `*.c`) and no other files. **Compress this directory as a `[uid].zip` file where [uid] is your university number** and check carefully that the correct zip file have been submitted. We suggest you to download your submitted file from Moodle, extract them, and check for correctness. **You will risk receiving 0 marks for this assignment if you submit incorrect files.** Resubmission after the deadline is not allowed.

Problem	Code templates provided	Files to Submit
1	-	`1.cpp`
2	`2.cpp`	`2.cpp`
3	`3.cpp`	`3.cpp`
4	-	`4.c`

Late submission

If submit within 3 days after the deadline, 50% deduction. After that, no mark.

Evaluation

Your code will be auto-graded for technical correctness. In principle, we use test cases to benchmark your solution, and you may get zero marks for not being able to pass any of the test cases. Normally partial credits will not be given for incomplete solution, as in many cases the logic of the programs are not complete and an objective assessment could be difficult. However, your work may still be considered on a case-by-case basis during the rebuttal stage.

Academic dishonesty

We will be checking your code against other submissions in the class and from the Internet for logical redundancy. Please be reminded that no matter whether it is providing your work to others, assisting others to copy, or copying others will all be considered as committing plagiarism and we will follow the departmental policy to handle such cases. Please refer to the course information notes for details.

Getting help

You are not alone! If you find yourself stuck on something, post your question to the course forum. We want this assignment to be rewarding and instructional, not frustrating and demoralizing. But we don't know when or how to help unless you ask.

Please be careful not to post spoilers. Please don't post any code that is directly related to the assignments to the discussion forums or share your work to any public domain. However you are welcome and encouraged to discuss general ideas on the discussion forums.

Problem 1: (C++) Pokémon in Order

Write a C++ program which reads in some Pokémon names from user input and outputs them in a way as described below.

Input:

- Each line contains a single word which is the name of a Pokémon.
- Last line is always "???" indicating end of input.
- You may assume the number of input names are no more than 30.

Output:

- Names of the input Pokémon, one on each line, in the following order:
    - In descending order of length of Pokémon name.
    - If two Pokémon names are of the same length, then the two names will be printed in lexicographical order (**case insensitive**).

Requirement:

- Your program MUST implement the **selection sort** algorithm you learned in Module 6. You will need to adapt the sorting algorithm to satisfy the output requirement. In other words, you are not allowed to use external libraries to handle the sorting.
- You can ONLY use the data types `char`, `bool`, `int`, `double`, `strings` and arrays.
- You are NOT allowed to use data structures from other external libraries such as STL containers (e.g., vectors), etc.

Sample Test Cases

User inputs are shown in blue.

1_1

```
Azelf
Lurantis
Drapion
Sudowoodo
Fletchinder
???
Fletchinder
Sudowoodo
Lurantis
Drapion
Azelf
```

1_2

```
chinchou
Tirtouga
Venusaur
Cobalion
Ribombee
cacturne
Vaporeon
???
cacturne
chinchou
Cobalion
Ribombee
Tirtouga
Vaporeon
Venusaur
```

1_3

```
Liepard
rhyperior
Lairon
petilil
Pignite
druddigon
Aron
???
druddigon
rhyperior
Liepard
petilil
Pignite
Lairon
Aron
```

Problem 2: Sudoku

Sudoku is a puzzle game whose goal is to enter the digits 1 to 9 into a game board with 9 x 9 cells, such that:

- Each digit appears exactly once in each row
- Each digit appears exactly once in each column
- Each digit appears exactly once in each of the nine 3 x 3 sub-grid.

An example game solution is shown below.

	0	1	2	3	4	5	6	7	8
0	5	3	9	2	8	7	1	4	6
1	8	1	7	6	3	4	5	9	2
2	4	2	6	1	9	5	7	3	8
3	7	6	5	3	1	2	9	8	4
4	9	4	1	5	6	8	2	7	3
5	3	8	2	4	7	9	6	5	1
6	1	9	4	7	2	3	8	6	5
7	6	5	8	9	4	1	3	2	7
8	2	7	3	8	5	6	4	1	9

Given an incomplete game board, we may determine the **allowed digits** for an empty cell, which are the digits that can be entered into the cell without violating the above rules for the game board. For example, in the following game board, the allowed digits for cell at position (6, 2) are 3, 5, 6 and 7.

	0	1	2	3	4	5	6	7	8
0	3	9		7	8				
1			4	9		6	8		7
2						3		6	
3			8	2	4				
4	7	3	1				6	2	4
5					6	7	9		
6		1		4					
7	2		9	6		8	3		
8					1	9		4	8

A very basic Sudoku solving technique, named **Naked Single**, is to determine the allowed digits of an empty cell and if the empty cell has only one allowed digits, we fill the empty cell with this allowed digit. In the above game board, the cell at position (4, 5) can be filled with the digit 5.

We may then apply the Naked Single technique to the empty cells to solve a Sudoku game as much as possible. Note that

- Once an empty cell is filled, the allowed digits for other empty cells may change

so we will need to redetermine them.
- We can apply the Naked Single technique to the empty cells repeatedly, until we find that no empty cells can be filled after visiting all empty cells once.
- For some very simple Sudoku game board, all empty cells can be solved eventually by using the Naked Single technique alone, but in general we may still end up with an incomplete board.

Write a C++ program that uses the Naked Single technique alone to solve a Sudoku game as much as possible as described above.

You are provided with a template program `2.cpp`.

Input:

- Nine lines, each containing nine digits from 0 to 9, representing a Sudoku game board.
- A digit 0 represents an empty cell, while any digit from 1 to 9 represent a filled cell.
- You may assume that the input board is always consistent, which means that there must be at least one allowed digit for an empty cell at any time during the solving process.

Output:

- Your program should first display the given game board in the format as specified in the sample test cases, in which an empty cell is printed as `x`, and we have grid lines to better visualize the sub-grids.
- Your program should then display the final game board (which may be complete or incomplete) obtained by repeatedly applying the Naked Single technique ONLY until no more empty cells can be filled.

Requirement:

- You should start working from the `2.cpp` provided.
- The `main()` function has been written for you. A 2D array is also defined in `main()` for storing the game board. You do not need to change anything in the `main()` function.
- Complete the following functions in `2.cpp`. Read the code in the template

carefully to see what have been provided for you. You will find details of the function prototypes in the in-code comments too.
    - `ReadBoard()` which reads a Sudoku board from input
    - `PrintBoard()` which displays a given Sudoku board to screen
    - `SolveBoard()` which solves a given Sudoku board as much as possible using the Naked Single technique only.
- You can ONLY use the simple data types `char`, `bool`, `int`, `double`, `strings` and arrays. In other words, you are not allowed to use other data types or data structures STL containers (e.g., vectors), etc.
- You may add your own functions wherever appropriate for better program modularity.

Sample Test Cases

User inputs are shown in blue.

2_1:

```
3 9 0 7 8 0 0 0 0
0 0 4 9 0 6 8 0 7
0 0 0 0 0 3 0 6 0
0 0 8 2 4 0 0 0 0
7 3 1 0 0 0 6 2 4
0 0 0 0 6 7 9 0 0
0 1 0 4 0 0 0 0 0
2 0 9 6 0 8 3 0 0
0 0 0 0 1 9 0 4 8
Input Sudoku board:
3 9 x | 7 8 x | x x x
x x 4 | 9 x 6 | 8 x 7
x x x | x x 3 | x 6 x
------+-------+-------
x x 8 | 2 4 x | x x x
7 3 1 | x x x | 6 2 4
x x x | x 6 7 | 9 x x
------+-------+-------
x 1 x | 4 x x | x x x
2 x 9 | 6 x 8 | 3 x x
x x x | x 1 9 | x 4 8

Final Sudoku board:
3 9 6 | 7 8 4 | 1 5 2
1 2 4 | 9 5 6 | 8 3 7
5 8 7 | 1 2 3 | 4 6 9
------+-------+-------
9 6 8 | 2 4 1 | 5 7 3
7 3 1 | 8 9 5 | 6 2 4
4 5 2 | 3 6 7 | 9 8 1
------+-------+-------
8 1 5 | 4 3 2 | 7 9 6
2 4 9 | 6 7 8 | 3 1 5
6 7 3 | 5 1 9 | 2 4 8
```

2_2:

```
4 0 9 0 0 0 0 0 0
0 2 0 4 0 0 0 0 6
6 0 0 2 3 0 1 0 0
5 9 0 3 0 0 0 7 0
0 0 0 0 4 0 0 5 0
0 8 0 0 0 2 0 9 1
0 0 1 0 2 4 0 0 8
7 0 0 0 0 8 0 4 0
0 0 0 0 0 0 7 0 5
Input Sudoku board:
4 x 9 | x x x | x x x
x 2 x | 4 x x | x x 6
6 x x | 2 3 x | 1 x x
------+-------+-------
5 9 x | 3 x x | x 7 x
x x x | x 4 x | x 5 x
x 8 x | x x 2 | x 9 1
------+-------+-------
x x 1 | x 2 4 | x x 8
7 x x | x x 8 | x 4 x
x x x | x x x | 7 x 5

Final Sudoku board:
4 3 9 | x x x | 5 2 7
1 2 8 | 4 x x | 9 3 6
6 7 5 | 2 3 9 | 1 8 4
------+-------+-------
5 9 x | 3 x x | x 7 2
2 1 x | x 4 x | x 5 3
3 8 x | x x 2 | x 9 1
------+-------+-------
9 5 1 | 7 2 4 | 3 6 8
7 6 3 | x x 8 | 2 4 9
8 4 2 | x x x | 7 1 5
```

2_3:

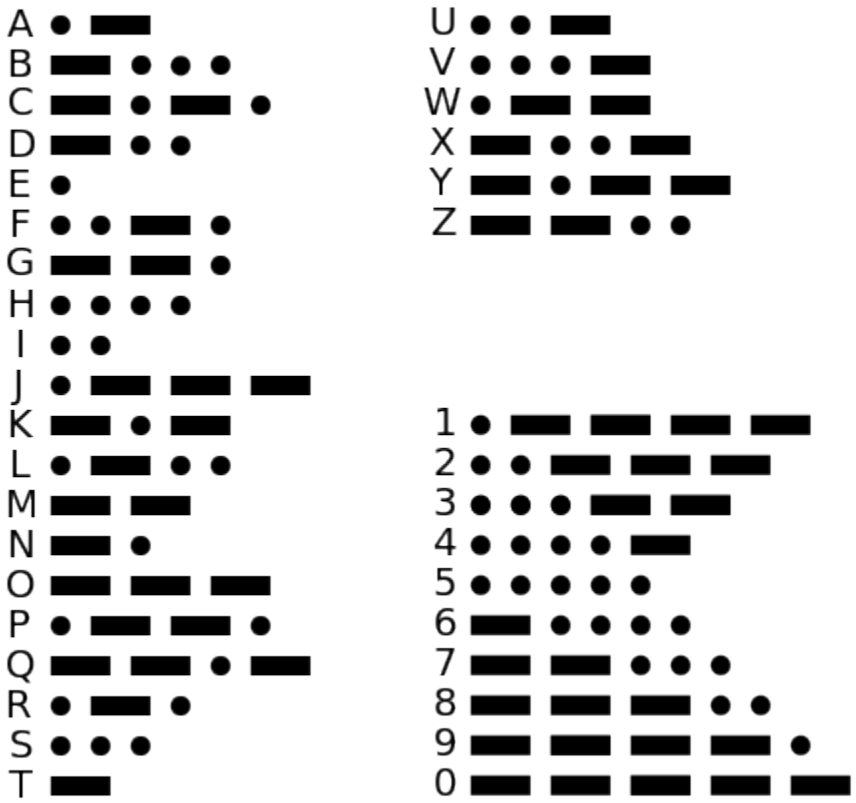
```
7 0 9 0 0 0 3 0 6
0 8 0 0 3 0 0 5 0
0 0 0 0 0 2 0 0 0
5 0 0 0 0 0 8 0 0
0 2 0 0 6 0 0 3 0
0 0 1 0 0 0 0 0 7
0 0 0 4 0 0 0 0 0
0 5 0 0 1 0 0 6 0
9 0 7 0 0 0 4 0 5
Input Sudoku board:
7 x 9 | x x x | 3 x 6
x 8 x | x 3 x | x 5 x
x x x | x x 2 | x x x
------+-------+-------
5 x x | x x x | 8 x x
x 2 x | x 6 x | x 3 x
x x 1 | x x x | x x 7
------+-------+-------
x x x | 4 x x | x x x
x 5 x | x 1 x | x 6 x
9 x 7 | x x x | 4 x 5

Final Sudoku board:
7 x 9 | x x x | 3 x 6
x 8 x | x 3 x | x 5 x
x x x | x x 2 | x x x
------+-------+-------
5 x x | x x x | 8 x x
x 2 x | x 6 x | x 3 x
x x 1 | x x x | x x 7
------+-------+-------
x x x | 4 x x | x x x
x 5 x | x 1 x | x 6 x
9 x 7 | x x x | 4 x 5
```

Problem 3: (C++) Morse Code Decoder

Morse code is a method used in telecommunication that encodes text characters with `dot` and `dash`. The following figure shows the encodings of the characters supported by our

program.



E.g., `A`'s morse code is *one dot and one dash*, `0`'s morse code is *five dashes*, etc.

In this problem, you are asked to write a C++ program that takes in a sequence of morse code and decodes it to text characters. The input morse code in this problem follows the rules:

- A `dot` is represented using `.`.
- A `dash` is represented using `_`.
- The space between letters is represented using `|`.
- The space between words is represented using `||`.

For example, given the input morse code:

```
...|___|...
```

The decoder should output:

```
SOS
```

Given the input morse code:

```
....|.|._..|._..|___||.__|___|._.|._..|_..
```

The decoder should output:

```
HELLO WORLD
```

You are provided with a template program `3.cpp`.

Input:

- The program **accepts morse code from a file**. The command line of the program is given by:
  `<program_name> < filename`
  E.g., if your program is named `decoder`, and the morse code is stored in a file named "message.txt"; then, the following command at the command prompt
  `./decoder < message.txt`
  will decode the morse code in "message.txt" and print out the text.

Output:

- Your program will print out the decoded text in one line.

Requirements:

- You should start working from the `3.cpp` provided.
- Complete the function
  `string morseCodeToText(string s)`
  which
  - takes the morse code as input.
  - returns the decoded text.

Note:

- The `main()` function has been completed for you, and you do not need to make any change to it.
- You may add your own functions wherever appropriate for better program

modularity.

Sample Test Cases

User inputs are shown in blue.

3_1

```
./decoder < input3_1.txt
RUN
```

3_2

```
./decoder < input3_2.txt
THIS IS THE WAY
```

3_3

```
./decoder < input3_3.txt
SHANTARAM
```

3_4

```
./decoder < input3_4.txt
PEOPLE IN THEIR RIGHT MINDS NEVER TAKE PRIDE IN THEIR TALENTS
```

Problem 4: (C) Luhn algorithm

The Luhn algorithm, also known as the "modulus 10" algorithm, is a simple checksum method to prevent simple transcription errors for a sequence of digits. It is commonly used in distinguishing valid credit card numbers. Given an input code as a string of digits, the algorithm works as follows:

1. Reverse the input string.
2. Sum the odd digits (i.e., first, third,... digits) in the reversed string to obtain a partial sum *s1*.
3. For every even digits (i.e., second, fourth, ... digits) in the reversed string, multiply the digit by 2
4. Obtain a partial sum *s2* of the products in (3) with this rule: If a product in (3) is less than 10, just add the product to *s2*; otherwise, add the sum of two digits in the product to *s2*.
5. If *s1* + *s2* is divisible by 10, then the input string of digits is a valid code.

Let's work out an example. Suppose the input string is 5255638118968609:

1. we first obtain the reversed string 9068698118365525
2. summing the odd digits, we have *s1* = 9+6+6+8+1+3+5+2=40
3. the multiples of the even digits are 0, 16, 18, 2, 16, 12, 10, 10
4. summing the multiples of the even digits, we have
   s2 = 0 + (1+6) + (1+8) + 2 + (1+6) + (1+2) + (1+0) + (1+0) = 30
5. since *s1* + *s2* = 70 which is divisible by 10, the input string is a valid code.

Write a **C program** to determine if an input string of digits is a valid code.

Input:

- An input string of digits.
- You may assume that the length of the input string is no more than 30 digits.

Output:

- the first line displays the reversed string in step 1.
- the second line displays the values s1 and s2, separated by a space.
- the third line displays the string "valid" if the input is a valid string, or "invalid" if otherwise.

Requirement:

- Use the command `gcc -pedantic-errors -std=c11 4.c -o 4` to compile your program. (Or specify `-pedantic-errors -std=c11` as the compiler flags in the Atom editor.)

Sample Test Cases

User inputs are shown in blue.

4_1

```
5255638118968609
9068698118365525
40 30
valid
```

4_2

```
5255638118968608
8068698118365525
39 30
invalid
```