ENGR 151

Fall 2023

Lab 10

Exercise 1: Programming Challenges

This part of the lab will be focused on programming and coding challenges. This lab will consist of 2 challenge problems, with ranging levels of difficulty, with a bonus challenge at the end as an optional challenge.

Challenge 1 - Making Change

Your code will be required to take in monetary amounts using **`cin`** and output using **`cout`**. Your solution must be the least number of coins possible to make that change.

Background

There is not that much background knowledge needed for this challenge. You just need to know the American coin denominations. We have a quarter, worth 25 cents; a dime, worth 10 cents; a nickel, worth 5 cents; and a penny, worth 1 cent. Given a monetary amount, you must output the minimum number of coins needed to make that amount, and what those coins are. For example, we can look at an example, which is 1.17. You need a minimum of 8 coins to make this amount: 4 quarters, 1 dime, 1 nickel, and 2 pennies. Note that you can use 117 pennies, but this is not the minimum amount of coins, so it is the incorrect output.

Your Code

Use the starter code given on Canvas in the file **making_change.cpp**.

You take in input through the terminal in format `A.xx`, where A is the number of dollars and xx is the number of cents. Your code should read these in, calculate the change and output them to the terminal using **cout** in the format:

`c` `coins:` **`q`** `Quarters,` **`d`** `Dimes,` **`n`** `Nickels,` **`p`** `Pennies.`

Where c, q, d, n, and p are the total number of coins, quarters, dimes, nickels, and pennies in the solution, respectively. For example, given the input 1.17, you should output:

`8 coins: 4 Quarters, 1 Dimes, 1 Nickels, 2 Pennies.`

Note: In this task, q, d, n and p can take values of zero. For example, there might exist a combination of coins where the number of pennies is zero.

Submit **making_change.cpp** to Autograder.

- **making_change.cpp -** reads in a monetary value and outputs the amount of each coin needed to create that value.

Challenge 2 - Cracking the Code

You decide to take on a side job as a spy. In the file **"encrypted_codes.txt"** you will be given a number of codes to decrypt. Based on an algorithm explained below, you will decipher each message using string manipulation and **output the new message to the file** "**decrypted_codes.txt**". Download **decode_message_starter.h** and **decode_message_starter.cpp**.

In **decode_message_starter.h**, you should implement the functions as described. Be sure to rename this to **decode_message.h** before submitting.

In **decode_message_starter.cpp**, you will complete the main function to decode a message using the algorithm described below. Be sure to rename this to **decode_message.cpp** before submitting.

You can check your decryption with the **expected_decrypted_codes.txt** file.

Important notes!

- Each line in the input file is one code.
- The lines may contain whitespace.
- Your program should work for an input file of any size.
- When writing into **decrypted_codes.txt**, make sure the decrypted line is written into the file with a newline character at the end or **endl**.

Here's how you will decrypt each code:
If the first character of the string is "$" or "#" then you will decrypt the string by:

1) Removing the first character (index 0)
2) Calling `remove_duplicates(str)`
3) Calling `shift(str)`
4) Calling `replace_space(str)`

If the first character is **not** "$" or "#", then you will decrypt the string by:

1) Calling `remove_key(str)`
2) Calling `replace_space(str)`
3) Calling `shift(str)`

As you may have guessed, this will require you to write 4 functions with the following definitions:

```
//This function takes a string by reference and
//removes any consecutive duplicate letters from that string
void remove_duplicates(string & code);
      //For example the string "aabccebce" would become "abcebce"
```

```
//This function takes a string by reference and
//removes any instances of the "key"
//The "key" is defined as:
//the first 3 letters of the original string input
void remove_key(string & code);
      //For example, when passing the string "osugoosubluosue"
      //to the function, the "key" would be defined as "osu"
      //and the string would be "goblue" after removing the key
```

```
//This function takes a string by reference and
//shifts every letter in the string up by 3
void shift(string & code);
      //For example, the string "abc" would become "def"
```

```
//This function takes a string by value and
//Returns a string where all spaces are replaced by underscores
//(pass-by-value is used here for the sake of practicing both)
string replace_space(string code);
      //For example, if the string "hello world !" was passed in,
      //"hello_world_!" would be returned
```

Challenge 3 - Sudoku Solver (OPTIONAL)

Your code should solve a Sudoku grid, represented by a 9 by 9 array of integers and output the solved grid to the standard output if a solution exists. Your task is to write a function that returns true/false on whether the grid is solvable. The starter file, **sudoku_starter.cpp**, already contains code to display the grid on the standard output. You should implement the *SolveSudoku()* function, in the **sudoku_starter.cpp** file. This starter file can be found on canvas.

Background

Sudoku is a game where the objective is to fill a 9 by 9 grid with numbers ranging from 1 to 9. Each row and column must only have one of each number, as well as each “section”. Sections are defined as 3 by 3 areas of the grid, usually marked off with thicker lines. There are always exactly 9 sections that do not overlap.

The grid already has some numbers when you start, restricting where you can put what numbers. Thus, there is only one unique solution for the sudoku grid. Here is an example of a grid, which is grid **A** in the main function of **sudoku_starter.cpp.**

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

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

On the left it is unsolved, while on the right it is solved. As you can see, no two numbers share the same row, column, or section. In our code, the blanks will be represented by the integer 0.

Your Code

Download **sudoku_starter.cpp** off Canvas. You will be implementing the *SolveSudoku()* function, which takes an array of integers, and returns whether the array is solvable using sudoku. Blank areas of the sudoku grid are represented by 0s. Since the array is passed by reference, you will need to modify it to the sudoku solution. In the end, the code should **cout** 4 grids with no 0s, corresponding to solving all the 4 examples in the starter code. The solution for grid A is in the background section (the previous section) of this challenge.

We recommend starting out with just grid “easy” and commenting everything else out. Just don’t forget to include all the grids when you run it a final time! When you run your code, you should see that all 4 grids from the 4 examples are displayed on the standard output.