Assignment 3: Strings and Coins

Due: 20:00, Thu 18 Oct 2018 Full marks: 100

Introduction

In this assignment, you will use functions to implement a game called *Strings and Coins*. At the beginning of the game, there is a network of coins, each connected by four strings. Two players take turns cutting a string in the network. When a cut leaves a coin with no strings connected, the player scores one point *and* takes an extra turn. Note that the player can get more than two consecutive turns as long as s/he scores again in every turn. The game ends when all strings are cut, and the player with more points wins. It is a draw if two players get the same points. Figure 1 shows an example game network. We use the symbol $ to denote coins, -- and | to denote the strings, and # to denote the "wall" (that is, the network boundary). In Figure 1(b), the top-left coin is already disconnected (assumed to be by Player 1), and the bottom-right coin is one string from being disconnected.

```
##########
#  |  |  #
#--$--$--#
#  |  |  #
#--$--$--#
#  |  |  # Player 1 score: 0
########## Player 2 score: 0
```
(a)

After some turns → … →

```
##########
#        #
#  $  $--#
#     |  #
#--$  $  #
#  |     # Player 1 score: 1
########## Player 2 score: 0
```
(b)

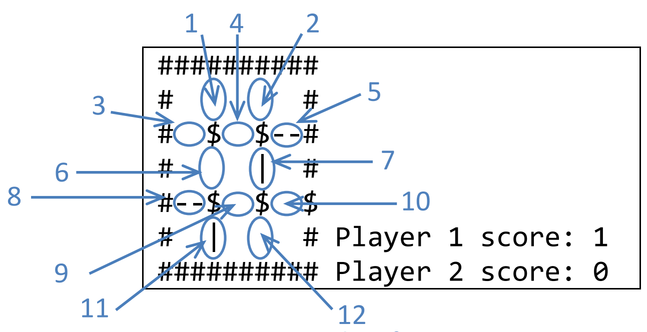
Figure 1: (a) Initial network of coins; (b) Network after some turns

Program Specification

This section describes the game network representation, some necessary functions, and program flow.

Game Network Representation

There are 12 strings initially in the network. Therefore, we use integers 1 to 12 to denote these string positions, as illustrated in Figure 2. The positions are basically ordered top-to-bottom, left-to-right.



```
##########
#        #
#  $  $--#
#     |  #
#--$  $  #
#  |     # Player 1 score: 1
########## Player 2 score: 0
```

Figure 2: Numbers for String Positions

To encode the whole network and the players' scores in a game, we use a 14-digit integer $d_1d_2d_3d_4d_5d_6d_7d_8d_9d_{10}d_{11}d_{12}d_{13}d_{14}$. The first 12 digits $d_1 \ldots d_{12}$ are either 0 or 1, denoting whether the string in the corresponding positions 1 to 12 are cut or not. (That is, $d_i = 0$ means the string in position i is already cut; $d_i = 1$ means position i remains uncut.) The last two digits d_{13} and

d_{14} range in 0–4 to denote the scores of Players 1 and 2 respectively. For example, the network in Figure 1(b) is encoded by 00001011001010, which also stores the scores of Player 1 (1) and Player 2 (0). Using this representation, the initial full network (with all strings uncut) is encoded as the integer 11111111111100. The network at game end (with all strings cut) is encoded as 000000000000*xy*, where *x* and *y* depend on the scores of Players 1 and 2. For example, 00000000000013 is a game won by Player 2. (Note: in C++, integer constants should *NOT contain leading zeroes*, otherwise, they will be treated as octal numbers. Therefore, it is actually 13 rather than 00000000000013.)

The data type `int` in C++ is typically 32-bit and is not big enough to store a 14-digit integer. In your program, you have to use a bigger integer type called `long long`. In Visual Studio, `long long` is a 64-bit signed integer type, whose range is -9,223,372,036,854,775,808 … 9,223,372,036,854,775,807.

Provided and Required Functions

Your program must contain the following functions. *Two of them are written for you already (Provided) and you should not modify their contents.* The others will be written by you (Required). These functions must be called *somewhere* in your program. You can design extra functions if you find necessary.

(Provided) `bool stringStatus(long long network, int pos)`

Returns `true` if position `pos` of the game `network` still has a string (not cut yet); returns `false` otherwise (that is, string is already cut).

(Provided) `void displayNetwork(long long network)`

Prints the `network` to the screen using the format in Figure 1.

(Required) `int playerScore(long long network, int p)`

Returns the score of Player p in `network`. (Either the 13th or 14th digits in `network`.)

(Required) `void updateNetwork(long long &network, int pos, int p)`

Performs the task of Player p cutting a string in position `pos` of `network`. The *reference parameter* network should get updated, and if any coins are disconnected, the score of Player p should be incremented, to reflect the new network configuration. For example, suppose `network` is 10010000010110. Then calling `updateNetwork(network, 4, 1)` means Player 1 cutting the string in position 4, disconnecting a coin (in the top-right if drawn out). Thus, the function call should update `network` to 10000000010120. (Position 4 is updated to 0, and score of Player 1 is increased to 2.) Note that cutting one string can sometimes disconnect at most two coins. To determine if a new coin is disconnected, calling the `stringStatus(…)` function is useful. Besides, you should *not print anything* using `cout` in this function.

In the above functions, you can assume that (a) the parameter `network` is always a proper encoding of a game network (14-digit integer; 1st–12th digits are 0 or 1; 13rd and 14th digits are the scores, etc.); (b) the parameter `pos` is always 1–12; and (c) the parameter p is always either 1 or 2.

Program Flow

You should call the functions above to aid your implementation of the following program flow.

1. The program starts the game with a full network (11111111111100). Player 1 takes the first turn.
2. Then, you should prompt the current player to enter an integer to denote the position where s/he wants to cut a string.

3. A user input is invalid if the input position is not 1–12, or the position was already cut. In case it is invalid, display a warning message and go back to Step 2.
4. Update the network by cutting the string in the input position.
5. If the current player has disconnected one or more coin(s), print a message and keep him/her the current player. Otherwise, swap the other player to become the current player.
6. Repeat steps 2 to 5 until all 12 strings have been cut. (That is, until game is over.)
7. Once all strings have been cut, determine the winner or a draw and display the message *"Player 1 wins!"*, *"Player 2 wins!"*, or *"Draw game!"* accordingly.

Other Requirements

You *cannot declare any global variables (variables declared outside any functions)*. You *cannot use any functions in the `<cmath>` library*. You *cannot use any arrays* in this assignment.

Program Output

The following shows some sample output of the program. The blue text is user input and the other text is the program output. You can try the provided sample program for other input. *Your program output should be exactly the same as the sample program* (i.e., same text, same symbols, same letter case, same number of spaces, etc.). Otherwise, it will be considered as *wrong*, even if you have computed the correct result.

```
##########
#  |  |  #
#--$--$--#
#  |  |  #
#--$--$--#
#  |  |  # Player 1 score: 0
########## Player 2 score: 0
Player 1, make your move (1-12): 1↵
##########
#     |  #
#--$--$--#
#  |  |  #
#--$--$--#
#  |  |  # Player 1 score: 0
########## Player 2 score: 0
Player 2, make your move (1-12): -1↵
Invalid. Try again!
Player 2, make your move (1-12): 13↵
Invalid. Try again!
Player 2, make your move (1-12): 1↵
Invalid. Try again!
Player 2, make your move (1-12): 12↵
##########
#     |  #
#--$--$--#
#  |  |  #
#--$--$--#
#  |     # Player 1 score: 0
########## Player 2 score: 0
```

```
Player 1, make your move (1-12): 2↵
##########
#        #
#--$--$--#
#  |  |  #
#--$--$--#
#  |     # Player 1 score: 0
########## Player 2 score: 0
Player 2, make your move (1-12): 11↵
##########
#        #
#--$--$--#
#  |  |  #
#--$--$--#
#        # Player 1 score: 0
########## Player 2 score: 0
Player 1, make your move (1-12): 3↵
##########
#        #
#  $--$--#
#  |  |  #
#--$--$--#
#        # Player 1 score: 0
########## Player 2 score: 0
Player 2, make your move (1-12): 10↵
##########
#        #
#  $--$--#
#  |  |  #
#--$--$  #
#        # Player 1 score: 0
########## Player 2 score: 0
Player 1, make your move (1-12): 6↵
##########
#        #
#  $--$--#
#     |  #
#--$--$  #
#        # Player 1 score: 0
########## Player 2 score: 0
Player 2, make your move (1-12): 4↵
Player 2 scores! Gets another turn.
##########
#        #
#  $  $--#
#     |  #
#--$--$  #
#        # Player 1 score: 0
########## Player 2 score: 1
Player 2, make your move (1-12): 7↵
```

```
##########
#        #
#  $  $--#
#        #
#--$--$  #
#        # Player 1 score: 0
########## Player 2 score: 1
Player 1, make your move (1-12): 9↵
Player 1 scores! Gets another turn.
##########
#        #
#  $  $--#
#        #
#--$  $  #
#        # Player 1 score: 1
########## Player 2 score: 1
Player 1, make your move (1-12): 8↵
Player 1 scores! Gets another turn.
##########
#        #
#  $  $--#
#        #
#  $  $  #
#        # Player 1 score: 2
########## Player 2 score: 1
Player 1, make your move (1-12): 5↵
##########
#        #
#  $  $  #
#        #
#  $  $  #
#        # Player 1 score: 3
########## Player 2 score: 1
Player 1 wins!
```

Update on 7 Oct:
No need to print *"Player x scores! Gets another turn."* after the last move.

Submission and Marking

- Your program file name should be `stringcoin.cpp`. Submit the file in Blackboard (https://blackboard.cuhk.edu.hk/).
- Insert your name, student ID, and e-mail address as comments at the beginning of your source file.
- You can submit your assignment multiple times. Only the latest submission counts.
- Your program should be *free of compilation errors and warnings*.
- Your program should *include suitable comments as documentation*.
- *Plagiarism is strictly monitored and heavily punished if proven.* Lending your work to others is subjected to the same penalty as the copier.