Hao Huynh

Professor Mark Lehr

CSC 11 – 48982

12 December 2015

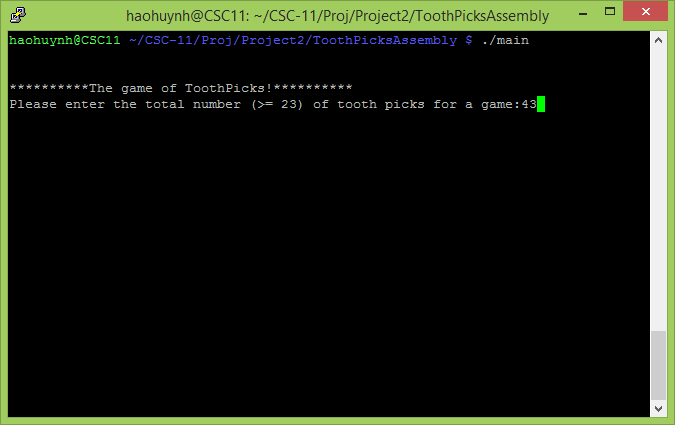
Project 2: Tooth Picks Game.

1. Introduction.

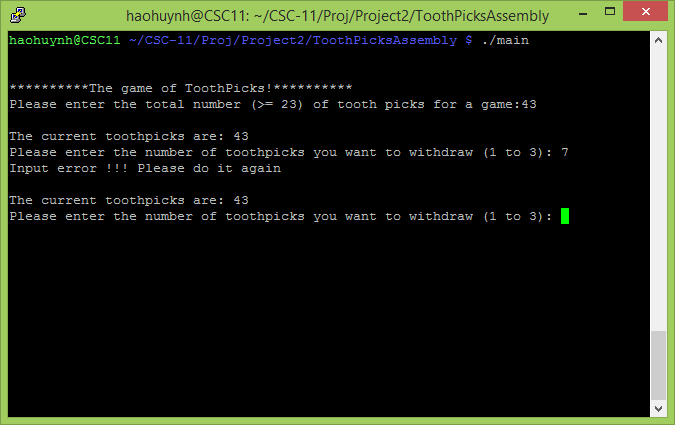
This Tooth Picks game is built upon the idea of the game of “23” being a two player game that begins with a pile of 23 toothpicks. However, in this game, users are freely to choose the total number of toothpicks at the beginning. Players, in turn, withdraw either 1, 2, or 3 toothpicks at a times. The player who withdraws the last toothpick loses the game. This program lets a human play one-on-one against the computer, and the human always moves first. In the computer turn, it will try winning the game by flowing rules being explained in the description.

I decided to implement this game because the C++ code of its program is actually not complex and long, so I could finish it within a limited period of time. However, I still could demonstrate techniques such as calling external C functions as well as internal functions from an Assembly program, applying basic computational mathematics, handling stacks and using Assembly constructs including if, if-else, while, and do-while.

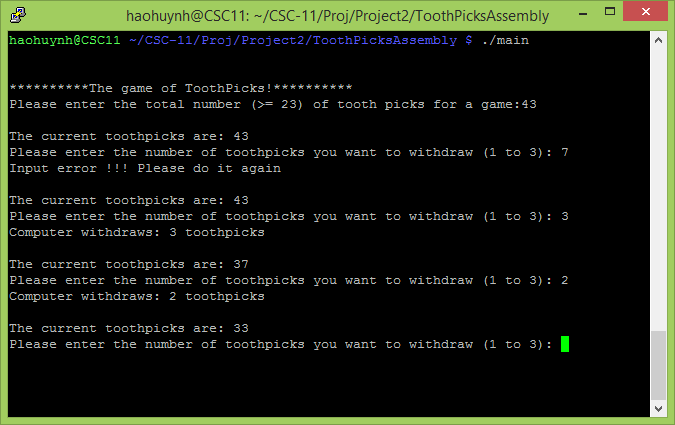
1. Summary.
   1. The program statistics:
      1. The number lines of code (LOC) are about 300 lines.
      2. There are about 20 variables.
      3. There are 2 methods.
   2. Miscellaneous: This program took me approximately 20 hours for coding and fixing errors. Moreover, I had learned about the standard input stream buffer and the way to scan a character right after reading a number.
2. Description.
   1. The program problems:
      1. Let the computer be the winner.
      2. Read a character from keyboard buffer right after scanning a number.
      3. Translate the game rules from the C++ code into the Assembly language.
   2. The program solutions:
      1. Create simple rules base on the nature of a modulus.
      2. Use the character scan pattern as “ %c” (the space before %c solves the new line input problem).
      3. Divide C++ code into small, logical pieces (i.e. constructs), then translate those pieces into Assembly.
   3. Sample Input/Output:
      1. The player enters the total number of toothpicks:



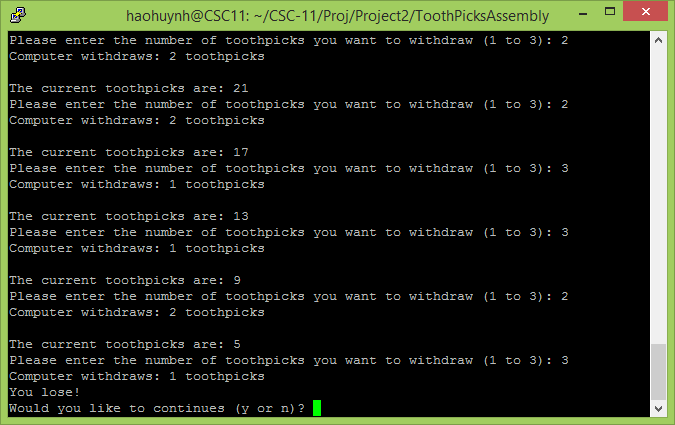
* + 1. Input validation:



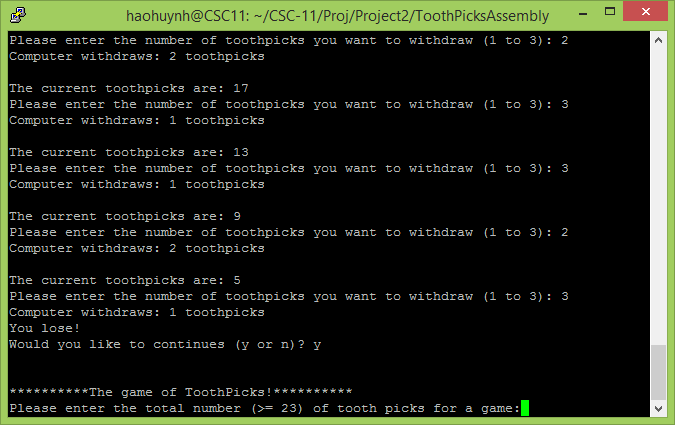
* + 1. The player and computer’s turns:



* + 1. Computer wins a game:



* + 1. The player replays the game:



* 1. Flowcharts and Pseudo codes:
     1. [Tooth Picks.](Flowcharts/toothpicks.jpg)
     2. Computer Rules:
        1. If there are more than 4 toothpicks left, find the remainder of the division between the current total number of toothpicks and number 4. Then, according that remainder, the computer should withdraw:

|  |  |
| --- | --- |
| The remainder | Toothpicks Withdrawn |
| 0 | 3 |
| 1 or 2 | 1 |
| 3 | 2 |

* + - 1. If there are 2 to 4 toothpicks left, the computer should withdraw enough toothpicks to leave the last one.
      2. If there is only 1 toothpick left, the computer loses the game.
  1. Concepts:

|  |  |
| --- | --- |
| Concept | Location |
| 1. External C functions | * main.s * calWTPForComputer.s |
| 1. Internal functions |
| 1. Constructors:  * If * If-else * While * Do-while | * All files |
| 1. Stack |
| 1. Basic computational math | * getMod.s |

1. Program Files.
   1. “[main.s](ToothPicksAssembly/main.s)”
   2. “[calWTPForComputer.s](ToothPicksAssembly/calWTPForComputer.s)”
   3. “[getMod.s](ToothPicksAssembly/getMod.s)”