ENGG1003
Introduction to Procedural Programming
Programming Assignment 2
Due Friday, Nov 19 at 23:59 pm

(Course Value is 12.5%)

PREPARATION

Preparation for this assignment mostly consists of making sure that your Visual Studio 2015 Solution and Project Spaces for the assignment are set up correctly so that you can easily submit the required files. You will need to set up a new solution called ***c*studentnumber**PA2**. Within the solution have a single project called **Gutsy**.

OUTCOMES

This assignment aims at developing and assessing your skills in:

- Create arrays of data
- Pass arrays as arguments to functions
- Allow functions to modify the contents of arrays
- Use pointer for arrays

GUTSY

GUTSY is a table game for two or more players. It aims at getting 101 points by throwing a dice while overcoming the one dot rule, and not being so greedy. Each player, at a turn, can throw the dice and accumulate the obtained points, but if s(he) gets one dot, the accumulated points are lost (zero points).
The rules are:

- At each player turn, the player can throw the dice as many times as s(he) wants and stops when s(he) decides.
- Each time the dice is thrown, the points are accumulated.
- Once a player decides to stop, his/her accumulated points count towards the goal of 101 points. The turn goes to the next player.
- **One dot rule:** If the player gets one dot, s(he) does not get any point (zero) to be accumulated towards the goal, and the turn goes to the next player.
- The player who gets 101 or more points wins.

PROBLEM STATEMENT

Your Gutsy solution must play this game automatically and its statistics. We will need to collect statistics as follows:

- Probability of getting one dot.
- Probability of getting two dots.
- Probability of getting three dots.

- Probability of getting four dots.
- Probability of getting five dots.
- Probability of getting six dots.
- Average number of times the dice is thrown to get a "defined number" (risk factor).
- Average number of times the dice is thrown to get 101 or more (to win).

Your program should make use functions for input, and statistics calculations. The following methods are suggestions, but you can use your own functions:

- The ***main()*** function that controls all of the others.
- A function, ***playGutsy***, that prompts the user for the number of **Gutsy** players and the number of games they will play. This function stores the results of the games and required statistics in one or more appropriately-sized array(s). The computer will play the given number of games without human intervention, but before, it will ask the user for information regarding each player:
  1. Name of player,
  2. Minimum number to stop a turn on the dice ("defined number" – risk factor).
- A function that throws a dice several times and accumulates values in order to reach/pass the "defined number", and returns: the accumulated value or zero (in case a one dot is obtained), number of times the dice was thrown, and counters required for the statistics. Suggested function signature: ***void diceThrown(int dNumber, int *accumValue, int *numTimes, int *count1, int *count2, int *count3, int *count4, int *count5, int *count6)*.**
- A function that plays a turn for a player by using the previous function and accumulates the value towards the goal and statistics. This function returns if the player has won. Suggested function signature: ***int playerTurn(int player, int accumValue, int *playerTotal, int *numTimes101)*.**
- A function that prints out the player name, defined number, accumulated value and total points. Suggested function signature: ***void playerPrint(int player, char playerName[], int dNumber, int accumValue, int playerTotal)*.**
- A function, ***displayTable***, that displays (prints out) all totals per player at the end of a round turn (scores table). This table will be displayed at the end of each game, as well as at the final results.
- A function, ***statistics***, that stores statistical values accordingly in order to calculate them at the end of the games.
- A function, ***statisticsTotal***, that calculates and displays (prints out) final statistics for the following questions:
  1. Probability of getting one dot, P(1). ***P(1) = # times one dot / # times thrown dice***
  2. Probability of getting two dots.
  3. Probability of getting three dots.
  4. Probability of getting four dots.
  5. Probability of getting five dots.
  6. Probability of getting six dots.
     The sum of these probabilities should total 1.
  7. Average number of times the dice is thrown to get a defined accumulated value (per player).
  8. Average number of times the dice is thrown to get 101 or more (to win).

GUTSY GUI

NOTE: This GUI is a sample using fictional inputs and results.

A message with instructions must be displayed at the beginning of the program. After this, the computer will ask:

```
Please type in the number of players: 2
Please type in the number of games: 2
```

Take into account that these inputs must be valid as they must be positive values and greater than zero. Then the computer will ask:

```
Player 1:
     Name: Amy
     Minimum number to stop in a turn: 40
Player 2:
     Name: Ben
     Minimum number to stop in a turn: 45
```

The user can type in lower case or upper case for inputs, but the program will always convert to upper case. Once information from the user is gathered, the computer will start the automatic game.

```
                    ***** GAME STARTING *****
```

The program will then take turns for each player will display:

```
Game 1

  Amy's turn: 6 6 6 5 6 5 6 =40. Accumulated total=40. Throws=7
  Ben's turn: 6 1 =0. Accumulated total=0. Throws=2

  Amy's turn: 1 =0. Accumulated total=40. Throws=1
  Ben's turn: 6 6 6 1 =0. Accumulated total=0. Throws=4

  Amy's turn: 6 6 6 5 6 6 6 =41. Accumulated total=81. Throws=7
  Ben's turn: 5 6 6 1 =0. Accumulated total=0. Throws=4

  Amy's turn: 6 6 6 6 5 6 1 =0. Accumulated total=81. Throws=7
  Ben's turn: 6 6 5 6 6 6 6 6 =47. Accumulated total=47. Throws=8

  Amy's turn: 6 6 6 6 5 5 6 =40. Accumulated total=121.Throws=7
```

When a play scores 101, the winner will be declared a table of statistics will be displayed as follows:

```
     ***** Winner: Amy *****
Game 1 - Statistics
     P(1): 0.1064
     P(2): 0.0000
     P(3): 0.0000
     P(4): 0.0000
     P(5): 0.1702
     P(6): 0.7234
     Amy - A(40) = 9
     Ben - A(45) = 18
     A(101) = 29

Final Table
     Player     Risk Factor        Times Won
     Amy            40                1
     Ben            45                0
```

Then the program will continue playing the following games and displaying as presented above. At the end will display again the final table and statistics (similar tables as presented above), but sorted alphabetically.

```
Final Statistics for 2 Games
      P(1): 0.1585
      P(2): 0.0000
      P(3): 0.0000
      P(4): 0.0000
      P(5): 0.1768
      P(6): 0.6646
      Amy - A(40) = 16
      Ben - A(45) = 20
      A(101) = 45
```

PROGRAMMING REQUIREMENTS

- No global variables are allowed.
- All statistical values must be displayed with 4 decimal places.
- You must manage information about players and statistics using arrays.
- Your program needs to display all information such that is easy to read. Your code also must be very well commented and have head of code and head of functions.
- You should use all methods you create. You can also create any additional functions/methods that you considered necessary.

Observation: While it is possible to complete this assignment using a number of parallel arrays, your design will be better and easier to work with, if you define your own structured type (it is best to use our naming convention of structname_t), and then have a single array of these objects.

Test your program and submit results in a single document together with the solution file with the following input data (create your players' names):

Players	Minimum number	Games
2	10, 15	2
2	10, 15	3
2	10, 15	5
4	10, 15, 20, 25	2
4	10, 15, 20, 25	3
4	10, 15, 20, 25	5

Your program will most probably required the **includes** of stdafx.h (provides printf_s, scanf_s, getchar), stdlib.h (provides rand, srand, and abs), and time.h (provides time, which is used to "seed" the random number generator).

Averages, probabilities or means are commonly used in engineering calculations. The *mean* number is easy to calculate. Simply add up the number of throws of XX dots and divide by the total number of throws.

INSTRUCTIONS FOR SUBMISSION – PLEASE READ THIS CAREFULLY!!

Preparation for submission:

1. Right-click and hold on the ***cstudentnumber*PA2** folder and choose Send To –> Compressed Folder.
2. This will create a file called **c*studentnumber*PA2.zip**. You will now need to add in the

Assessment Item Cover Sheet.

3. Click and hold on the zip file and choose Open with -> Windows Explorer.
4. Then drag and drop your completed Assessment Item Cover Sheet file onto this window so that it is displayed alongside the solutions directory name (do not drag the cover sheet file inside the solutions folder name). This will add the Assignment Cover Sheet to the zip file.
5. Then drag and drop your tests results file onto the same window so that it is displayed alongside the solutions directory name (do not drag the tests results file inside the solutions folder name). This will add the tests results to the zip file.
6. Close the Windows Explorer window.

Submit your updated zip file via the link for Assignment 2 submission in the Assignments tab on Blackboard.

EXTRA NOTES

Please remember that incorrectly submitted assignments will not be marked**.**

- **Remember, a completed Assessment Cover Sheet must accompany your submission.** A copy of it will be placed on Blackboard, in the Assignments section. Assignments without a completed Assessment Item Cover Sheet **will not be further marked.**
- Assignments should contain the Visual Studio project, compressed into a .zip file. Only one file per student will be allowed.
- Assignments containing a compressed Visual Studio solution or project that was created outside the correct specification (see section PREPARATION) **will be discounted according to the amount of time wasted by the marker in correcting the submission – if the submission cannot be made "runnable" within a reasonable amount of time then it will not be further marked.**
- Your program file should adhere closely to the layout and commenting standards presented in class, and your code should be properly documented with inline comments.

Remember that this assignment will take many hours to complete, and the submission itself should take less than 10 minutes. **Do not waste your effort by submitting it incorrectly.**
Late submissions are subject to the rules specified in the Course Outline.

C Sanin - 2017