

Due Date: Week 4 on Saturday 24th August, 2019

Weighting: 10% of your final mark for the unit

Submission Instructions:

Your project must be submitted as a Visual Studio project, including your project files, and any appropriate text files to ensure the program compiles and runs.

You may complete the tasks in your preferred IDE, however you **MUST** create a Visual Studio project in order to submit. Your project folder must be identified by using your name and assignment number, such as ***YourNameA1***.

The entire project folder must then be zipped up into one zip file for submission. The zipped file **MUST** be named "***A1-YourFistNameLastName***.zip". This zip file must be submitted via the Moodle assignment submission page.

Explicit assessment criteria are provided, however please note you will also be assessed on the following broad criteria:

- ✓ Meeting functional requirements as described in the exercise description.
- ✓ Demonstrating a solid understanding of object-oriented design and C++ coding, including good practice.
- ✓ Following the unit Programming Style Guide.
- ✓ Creating solutions that are as efficient and extensible as possible.

NOTE! Your submitted program **MUST** compile and run. Any submission that does not compile will automatically be awarded a **ZERO**. This means it is your responsibility to continually compile and test your code as you build it.

NOTE! Your submitted files must be correctly identified and submitted (as described above). Any submission that does not comply will receive an automatic **10% penalty** (applied after marking).

Scenario:

You have just joined a new multimedia company called CyberRoo. As a new employee, you have been tasked with creating a text-based prototype for a new maths tool the company has been asked to create by a major client called *"The Tally Ho Probability Generator"*.

Your job is to demonstrate some of the basic functionality of the program, which will be further developed by the team of programmers after your finish the prototype. To do this you will complete a series of tasks, each building upon the previous task.

Task List:

- ☐ Your project must include an application file – correctly formatted including variable and function declarations.
- ☐ Create and display a menu that accesses each of the following functions of the program.
- ☐ Read data from a file and display it as an information screen about the application.
- ☐ Read some sample data from a file and store it appropriately.
- ☐ Generate a Dice Tally table and display the result using the stored data.
- ☐ Save a Dice Tally table to file, appending each new table saved, at the user's request.
- ☐ Load all saved tables from the saved file and display the contents.

Task 1: Create and Display a Menu

Your first task is to create and display a menu, just like the one shown here. In the example, option [0] has been selected, terminating the program.

Create a function called `runMenu()` and call it from the `main()` function.

From this function, the user must be able to select any of the displayed options as often as they want before ending the program.

Ensure that you validate the user's input so that only the options displayed can be chosen.

You may process the options as either numeric or single character input, such as [0] or [E] for ending the program, [1] or [I] for displaying the information, etc.

```

The Tally Ho Probability Generator
[0] End Testing the Program
[1] Display "About" Information
[2] Read and store data from files
[3] Generate a Dice Tally Table
[4] Save Tally Statistics to a file
[5] Load Tally Statistics from a file

Which option would you like (0-5): 0

Thank you for testing this application.

Press any key to continue . . .
```

Task 2: Display an Information Screen

Your next task is to display some information about the application, just like the one shown here.

Create a function called `displayText()` and call it from the `runMenu()` function, when option [1] Display "About" Information is selected.

This function must accept a filename as a parameter ("TallyAbout.txt") and successfully read and display the data.

You must also allow the user some time to read the displayed information before returning them to the menu.

You can copy **TallyAbout.txt** into the appropriate folder to read and display, instead of creating your own. However, you may need to modify the format of the data, depending on how you intend to import it into your program.

```

The Tally Ho Probability Generator

Rolling dice is included in many different
types of games - for movement, chance or
even just for luck.

Wouldn't it be nice to know what the odds
are of rolling a specific number?

Well the Tally Ho Probability Generator is
here to help you know just that!

Press any key to continue . . .
```

You may create your own information, if you wish. If you do, keep it short!

Task 3: Read and Store Data from a File

Your next task is to read and store data elements, from a text file, and store this data in appropriate variables. Create a function called `createLists()` and call it from the `runMenu()` function, when option **[2] Read and Store Data from File** is selected.

This function must accept a filename as a parameter (*.txt) and successfully read and store the data. The data is to be stored in a collection variable named `tallyData`.

In this application, the player has to actually generate a Dice Tally Table first, then save the data before the data can be loaded. Therefore, data validation and suitable error messages will be important.

You may also be required to manipulate the string data, using appropriate string methods when displaying it.

```
~~~~~
The Tally Ho Probability Generator
~~~~~
[0] End Testing the Program
[1] Display "About" Information
[2] Read and store data from files
[3] Generate a Dice Tally Table
[4] Save Tally Statistics to a file
[5] Load Tally Statistics from a file
~~~~~
Which option would you like (0-5): 2

Data loading from savedTallyData.txt
File Not Found.

Press any key to continue . . .
```

Task 4: Generate and Display a Dice Tally Table

Your next task is generate a random Dice Table, and display the final result as a concatenated or formatted string.

Create a function called `generateTable()` and call it from the `runMenu()` function, when **[3] Generate a Dice Tally Table** is selected.

This function must return a concatenated or formatted string using input from the player (answering two questions to set the parameters) and using the answers to generate contents of the table. **Note:** the number of rolls is multiplied by 10.

You will need to apply a number of different string manipulation techniques to achieve the desired result. Think through the process carefully before you start to write your code. **Hint:** work it out using pen and paper first using one die.

Ensure that the collection variable (`tallyResults`) has content before trying to select each element, to stop the program from crashing.

This variable must be reusable, so also think of ways in which to reset the data so that any subsequent Tables generated are not contaminated by old data.

Note: when rolling 2 dice the 1 value is impossible to roll so ensure your output display demonstrates this.

You can break this function into smaller sub-functions, if you find this will make the coding easier for you. Your marks will not be effected either way.

See screen shot for an example of the required output.

```
~~~~~
The Tally Ho Probability Generator
~~~~~
[0] End Testing the Program
[1] Display "About" Information
[2] Read and store data from files
[3] Generate a Dice Tally Table
[4] Save Tally Statistics to a file
[5] Load Tally Statistics from a file
~~~~~
Which option would you like (0-5): 3

How many dice to roll (1-2): 2
How many rolls required (1-10): 10

You rolled 2 dice 100 times.
~~~~~
2: *****
3: *****
4: *****
5: *****
6: *****
7: *****
8: *****
9: *****
10: *****
11: *****
12: *****

Press any key to continue . . .
```

Task 5: Save a Dice Table to a File

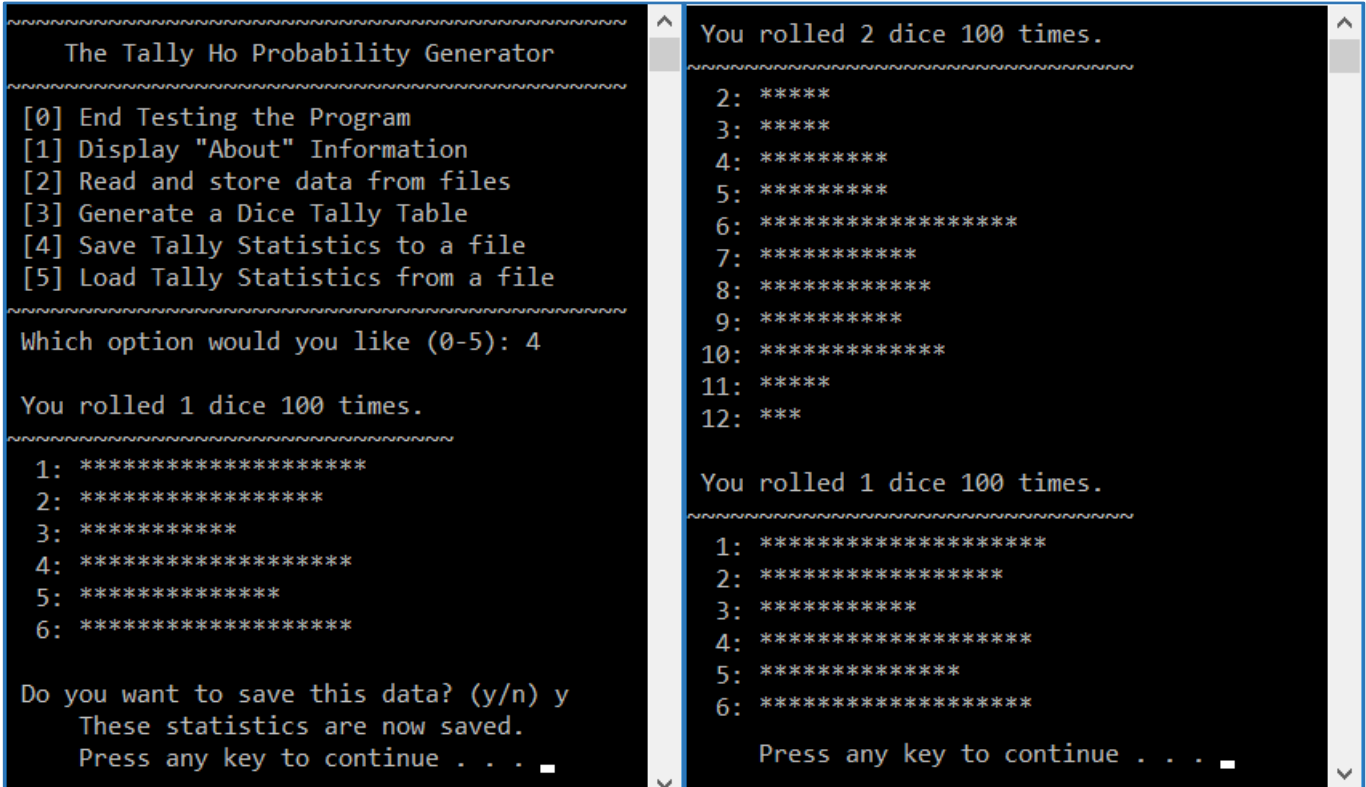
Your next task is to save a Dice Table to a file, called "`savedTallyData.txt`". In this task you must:

- ♦ display the last generated Dice Table
- ♦ ask the user if they want to save it

- if so, add it to any existing data in the file
- if not, do nothing
- ♦ return to the menu when the task is complete

Use the screen shot as a guide for your display and input request.

You should also inform the user when the data is saved, before returning to the menu.



```
~~~~~
The Tally Ho Probability Generator
~~~~~
[0] End Testing the Program
[1] Display "About" Information
[2] Read and store data from files
[3] Generate a Dice Tally Table
[4] Save Tally Statistics to a file
[5] Load Tally Statistics from a file
~~~~~
Which option would you like (0-5): 4

~~~~~
You rolled 1 dice 100 times.
~~~~~
1: *****
2: *****
3: *****
4: *****
5: *****
6: *****

~~~~~
Do you want to save this data? (y/n) y
These statistics are now saved.
Press any key to continue . . . █

~~~~~
You rolled 2 dice 100 times.
~~~~~
2: *****
3: *****
4: *****
5: *****
6: *****
7: *****
8: *****
9: *****
10: *****
11: *****
12: *****

~~~~~
You rolled 1 dice 100 times.
~~~~~
1: *****
2: *****
3: *****
4: *****
5: *****
6: *****

~~~~~
Press any key to continue . . . █
```

Task 6: Load Saved Dice Tables from a File

Your final task is loading the saved Dice Tables from the “**savedTallyData.txt**” and displaying them appropriately.

How you achieve this is dependent upon how you saved the data. Ideally, the display should be the same as the way you displayed the Dice Table when you generated it.

Use the screen shot as a guide for your display – these Dice Tables are being displayed in the same way as they were originally generated.

You must also load this data into the `tallyData` collection variable first, then display the saved data.

Things you may need to consider:

- ♦ Do the collection lists need to be initialised before you display the saved data?
- ♦ How have you saved the data?
 - Did you save the random numbers for each element type?
 - Did you save the completed strings?
 - Did you use another method?
- ♦ How will you display the data after it is read from the saved data file?
 - Which method is the most efficient way to display it?
- ♦ How will you ensure that the user has time to read the output before returning to the menu?

Assignment Notes:

It is your responsibility to know what is expected of you. If you are unsure about anything, ask your tutor sooner rather than later. Write any questions and/or notes here to help you clarify what you have to do.

Assignment 1: Marking Criteria [50 marks in total]

NOTE: Your submitted project must be correctly identified and submitted, otherwise it will receive an automatic **10% penalty** (applied after marking).

Does the program compile and run? Yes or No

NOTE! Your submitted program **MUST** compile and run. Submissions that do not compile will receive **ZERO**.

Application Design [10]

- ♦ Application File [10]
 - The *main()* function has appropriate function calls to keep it uncluttered [2]
 - Has all the correct *#include* statements [2]
 - Has the required data members and member functions using meaningful names [2]
 - Has created all function prototypes correctly [2]
 - Has created any additional functions to streamline their code [2]

Functionality [30]

- ♦ Task 1: Menu [6]
 - The *runMenu()* function correctly initialises any required variables [1]
 - The menu options are clearly displayed [1]
 - Appropriate use of loop and decision structures to process user input [2]
 - Correct use of function calls when responding to user input [1]
 - The user controls when to return to the menu [1]
- ♦ Task 2: About Information [4]
 - The *displayInfo()* function uses the correct argument [1]
 - Appropriate validation checks have been made [1]
 - Appropriate use of loop and decision structures to read and display data [2]
- ♦ Task 3: Reading and Storing Data [5]
 - The *createLists()* function uses the correct argument [1]
 - The collection variables (arrays or vectors) are correctly initialised [1]
 - Appropriate method(s) used to read the data from the file [1]
 - Appropriate use of loop and decision structures to read and store data [2]
- ♦ Task 4: Generate and Display Tally Data [4]
 - The *generateTable()* function returns a string [1]
 - Validation checks that collection lists have been created [1]
 - One random data element is selected from each collection [1]
 - The returned string is correctly concatenated and/or formatted [1]
- ♦ Task 5: Save Data to a File [6]
 - The *saveData()* function uses the correct arguments [2]
 - The current tally data is displayed appropriately [1]
 - The user is asked whether to save or not with an appropriate process of user input [1]
 - Appropriate validation for writing the file is in place [1]
 - The data is correctly written to the specified file [1]
- ♦ Task 6: Load Data from a File [5]
 - The *loadData()* function uses the correct argument(s) [1]
 - Appropriate validation checks have been made [2]
 - Appropriate use of loop and decision structures to read and display data [2]

Quality of Solution and Code [10]

- ♦ Does the program perform the functionality in an efficient and extensible manner? [2]
- ♦ Has a well-designed program been implemented? [4]
 - Appropriate additional functions created to streamline code [2]
 - Demonstrates the use of logical procedures [2]
- ♦ Has the Programming Style Guide been followed appropriately? [4]
 - All functions and variables have meaningful names and use correct camel notation [2]
 - Appropriate use of comments throughout the code [2]