

# MCD4720 - Fundamentals of C++

## Assignment 3 - Trimester 1, 2019

### Submission guidelines

This is an individual assignment, group work is not permitted

**Deadline:** May 17, 2019, 11:55pm

**Weighting:** 25% of your final mark for the unit

#### Late submission:

- By submitting a [Special Consideration Form](#) or visit this link: <https://goo.gl/xtk6n2>
- Or, without special consideration, you lose 5% of your mark per day that you submit late (including weekends). Submissions will not be accepted **more than 5 days late**.

This means that if you got  $Y$  marks, only  $(0.95^n) \times Y$  will be counted where  $n$  is the number of days you submit late.

**Marks:** This assignment will be marked out of 50 points, and count for 10% of your total unit marks.

**Plagiarism:** It is an academic requirement that the work you submit be original. If there is any evidence of copying (including from online sources without proper attribution), collaboration, pasting from websites or textbooks, **Zero marks** may be awarded for the whole assignment, the unit or you may be suspended or excluded from your course. Monash Colleges policies on plagiarism, collusion, and cheating are [available here](#) or see this link: <https://goo.gl/bs1ndF>

**Further Note:** When you are asked to use Internet resources to answer a question, this **does not mean copy-pasting text** from websites. Write answers in your own words such that your understanding of the answer is evident. Acknowledge any sources by citing them.

## Task Details:

This assignment consists of one main programming task. The purpose of this assignment is to have you design and implement an object-oriented program in C++, as well as reflect on your approach and design choices. The assignment comprises the following components:

- A diagram with annotation that describes your object-oriented design
- The completed program
- A 300 word reflection on your program

Successful completion of the fundamentals of the task as described may obtain you up to a maximum of 80% of the total assignment marks. The last 20% of the mark will be allocated to additional functionality that you can design. The additional functionality should demonstrate advanced or more complex application of principles covered to date. It need not be large amounts of work but should demonstrate a willingness to explore new and advanced concepts. You must detail what you have done in an accompanying “readme” file.

The assignment must be created and submitted as a Visual Studio 2017 project. You may complete the exercises in your preferred IDE, however you should create a Visual Studio project in order to submit. This project must then be zipped up into one zip file for submission named “**YourFirstNameLastNameA2.zip**”. This zip file must be submitted via the Moodle assignment submission page.

- Explicit assessment criteria are provided, however please note you will be assessed on the following broad criteria:
- Meeting functional requirements as described in the assignment description
- Demonstrating a solid understanding of C++ concepts, including good practice
- Demonstrating an understanding of specific C++ concepts relating to the assignment tasks, including object-oriented design and implementation and the use of Pointers\
- Following the unit Programming Style Guide
- Creating solutions that are as efficient and extensible as possible
- Reflecting on the appropriateness of your implemented design

**NOTE!** Your submitted program **MUST** compile and run. This means you should continually compile and test your code as you do it, ensuring it compiles at every step of the way.

If you have any questions or concerns please contact your lecturer as soon as possible.

## Assignment Task: Farkle (a dice game)

### Game Overview:

In this assignment, you are to write dice game, where you roll six dice to create a range of scoring combinations. The first player to reach or exceed the target score is declared the winner! Any number of players may play, however, for the basic assignment you only need to implement one.

### Basic Game Play:

In this program, you will control a set of six dice. On your turn, you must make as many scoring combinations (as outlined below) as you wish to score points towards the winning target score. The basic game play is as follows:

- One player is randomly chosen to go first. Players take turns in clockwise order. The game is played in rounds in which each player has a turn to roll the dice and score points.
- On your turn:
  - Roll all six dice and set aside at least one die of a scoring value, (as shown below):

Dice	Points	Dice	Points
1's	100 each	3x3's	300
5's	50 each	3x4's	400
3x1's	1000	3x5's	500
3x2's	200	3x6's	600

Combinations only count when made in a single throw.

- You may now decide whether to score your points and end your turn or you can re-roll the remaining dice.

If you choose to roll again, you can keep rolling the dice, until you choose to stop or you roll no scoring values. Rolling no scoring values is called a FARKLE.

If you roll a FARKLE, you score NO POINTS for this round and your turn ends. Pass the dice to the next player.

If you set aside all 6 dice, you may re-roll all 6 dice and continue adding to your score, following all the rules above.

- The first player to score 5,000 or more points wins the game.

There are many variations to this basic game, some of which you will be able to implement as part of the extra functionality for your assignment.

## A Typical Player Turn:

During a player's turn, the logic is as follows:

- 1) First, the player must roll all 6 dice to produce 6 random numbers from 1-6.
- 2) Next, they set aside at least one die that scores points (a single 1 or 5, or 3-of-a-kind). The points for which are stored in a running total for the player's turn. These dice cannot be rolled again, until after all 6 dice are set aside (see Step 4).
- 3) The player is presented with 2 options – roll the dice or score their points.  
If they choose to score, the running total is added to their player score and their turn ends.  
If they roll again, only those dice not set aside are rolled. Steps 2 and 3 may be repeated as often as the player wishes, until they use all 6 dice (Step 4) or roll a Farkle (Step 5).
- 4) If the player sets aside all 6 dice, they start again with all 6 dice and continue scoring as before this is called a Bonus Roll. Steps 2 and 3 may be repeated as often as the player wishes.
- 5) A Farkle is a roll with no scoring values – the player cannot take a single 1 or 5, or 3-of-a-kind. When this happens, the player ends their turn immediately and does not score the points rolled during that turn. Their saved score is not changed.

Here is an example roll for a player's turn:

- On their first roll, the player rolls a 1, 2, 3, 3, 5, 5.
- They can score 100 points for the 1, or 100 points for both 5s, or 50 points for 1 of the 5s, or 200 points for the 1 and both 5s.
- They choose to set aside just the 1 for 100 points and roll the 5 remaining dice again.
- This time they roll a 2, 3, 6, 6, 6. They decide to take the 600 points for the 3-of-a-kind in 6s.
- Their running total is now 700 points. This is a good score so they opt to save their points and end their turn.

## Extra Functionality

The marking criteria indicates that you should make some individual additions to this in order to achieve the final 20% of the mark.

Following is a list of additional features you can include, with the maximum number of marks **[x]** you can earn for each one. You may implement one or more features from the list, but you will only be able to score a maximum of 20% of the marks (a total of 20 marks).

- Include multiple human players making the game playable by 1 to 4 people. At the beginning of the game the player order must be randomised. Each player must track their own score and display their name and game score when required. **[3]**
- If a player rolls three Farkles in a row (on three consecutive turns) they lose 1000 points from their saved score. Once they lose the points the counter for Farkles is reset to zero. **[3]**
- Allow the players to choose the target score and the minimum points required to roll before saving points. Targets = 5,000 points, 10,000 points or 15,000 points. Minimum roll required = 350 points, 500 points, 750 points or 1000 points. **[4]**
- Display the dice using ASCII art, showing the faces as pips not as numbers. **[4]**

- When the first player reaches or exceeds the target score, all other players (in a multi-player game) have one final turn to try and beat that score. In this case, the player at the end of the round with the highest score wins. **[5]**
- The basic game includes the minimal scoring options to make the game playable. However, you can include three or more of the following scoring combinations to make the game more interesting. **[5]**

Dice	Points	Dice	Points
3 pairs	1000	straight*	1500
2x3-of-a-kind	2000	4-of-a-kind	2x points
5-of-a-kind	3x points	6-of-a-kind	instant win

\* one of each value

- Allow the game to be played with computer players. The computer player should be able to make reasonably intelligent decisions for choosing scoring combinations and when to save their points. This AI can be a set of rules that the computer player checks before making a roll or when to save their score. **[10]**
- Implement a more sophisticated AI for the computer players. You may create different player types, each with their own strategies for playing, such as an easy player, a cautious player, an aggressive player, etc. The human player should be able to select the level of difficulty and number of their opponents at the beginning of the game. **[10]**

You certainly do not have to implement all of the above to earn marks for extra functionality. Just remember the maximum marks you can earn are given in the brackets **[x]**. It is up to you!

### **Program Design Diagram\***

In this final assignment you are able to design your program in any way you like. You must provide a diagram with annotation that shows the classes you have in your program and how they interact. It should contain any relevant notes that describe the classes.

These diagrams should include appropriate UML diagrams as covered during class, including the basic class diagram notation for public, private and protected data and functions.

### **Program Reflection\***

You must provide a 300-word written reflection of your object-oriented design and how well you believe it was to implement. You should cover the following areas:

- Discuss why you designed it the way you did.
- Discuss how well you were able to code it, highlighting any issues you found once you tried to implement your design.
- If you were to do this project again, discuss how you might change your design to make your solution easier to implement, more efficient, or better for code reuse.

## ***Suggested Development Process***

Below is a suggested process you might use to develop your assignment piece by piece so that it is not too overwhelming. Feel free to follow this if you think it will help (or not!).

- 1) Design the program on paper.
  - Decide what classes you think you need and how they will relate to each other.
  - Write down your thoughts on why you think this is the best design.
  - You need to do this for the assignment anyway!
  - Remember how you can use inheritance and pointers.
- 2) Write a simple class that you need and test it!
- 3) Write another simple class that you need and test it!
- 4) Do these two classes interact? If so, see if you can get basic interaction happening!
  - Hint: I imagine some objects are going to battle...
- 5) Repeat the process for each additional class or function
  - Either add to existing classes or write new ones if you need to.
  - Remember, just focus on the small specific task!

I hope this helps. The important thing is to focus on small sub-tasks, one at a time, and build up functionality that way.

## ***Assignment 3: Marking Criteria [up to 100 marks in total]***

Does the program compile and run? Yes or No

- **A non-compiling program will receive an automatic 50 marks penalty.**

### **1. Class Design [10]**

#### **1.1. Class Diagrams [3]**

- Correct structure used (Name, Attributes, Behaviours) [1]
- Included the correct designations for public (+) and private (-) data [1]
- All variables and functions have meaningful names [1]

#### **1.2. Rationale for Class Design [7]**

- Clear description of why the specific classes were chosen [3]
- Clear description of how these classes contribute to the overall OO design [4]

### **2. Functionality [35]**

- 2.1. Game set up: initialising the player, game variables and creating a collection of dice [4]**
- 2.2. Implementation of the basic game mechanics (roll and score) [4]**
- 2.3. Implementation of basic scoring combinations (as shown on table) [8]**
- 2.4. Implementation score tracking and updating as required [4]**
- 2.5. Appropriate screen display and uncluttered User Interface [5]**
- 2.6. Appropriate data validation to avoid program crashing [8]**
- 2.7. Appropriate end game conditions triggered [2]**

### **3. Quality of Solution and Code [25]**

#### **3.1. Has a well-designed OO program been implemented? [5]**

- Contains classes appropriate to the assignment brief [3]
- Program structures support and OO design [2]

#### **3.2. Individual Class Design [5]**

- Each class has an appropriate header file [2]
- Contains only aspects that relate to the class (has no data members or member functions that are not directly related to the class) [3]

#### **3.3. Does the program perform the functionality in an efficient and extensible manner? [10]**

- Appropriate use of pointers in the program [5]
- Appropriate use of functions and function calls [2]
- Appropriate use of good programming practices and techniques [2]
- Extraneous and redundant code removed [1]

#### **3.4. Has the Programming Style Guide been followed appropriately? [5]**

- Appropriate commenting and code documentation [3]
- Correct formatting of code within \*.h and \*.cpp files [2]

### **4. Reflection [10]**

#### **4.1. Discussion of motivations for the program design [3]**

#### **4.2. Discussion of how well the design was to implement [3]**

#### **4.3. Discussion of what they would do differently if they were to start it again [4]**

### **5. Extra Functionality [20]**

- The player can select from 1-4 human players for the game [3]
- Three Farkles in a row loses the player -1000 points from saved score [3]
- Player sets the target and minimum score required to roll before saving points [4]
- Display the dice using ASCII art [4]
- Allow other players one final turn when the target score is reached or exceeded [5]
- Include three or more additional scoring combinations [5]
- Allow the game to be played with computer players [10]
- Implement a more sophisticated AI for the computer players [10]

**Sample Screenshots to give you some ideas for your layout and interface design.**

```
C:\Users\choward\Documents\MonashUnits\FIT1048-...
*****
FARKLE :: A dice game for 1-4 players
*****
Cheryl Total: 0      Current: 0      Dice Left: 6
*****
It's your turn, Cheryl

Do you want to [q]uit, [r]oll or [s]core? r

      [3] [5] [4] [5] [2] [6]
       1  2  3  4  5  6
```

Displaying information to the player is important.

This shot shows player's saved total, running total and how many dice to roll.

Note the use of prompt messages to help the player know what to do.

```
Please select the dice you want to lock
the die. Do not include space

The dice you want are: 2_
```

This shot shows the rolled dice and the locked die the player selected on a previous roll. The dice are in [6] to make them stand out from the selection ones.

Data validation is used to ensure the player enters only the correct data.

```
C:\Users\choward\Documents\MonashUnits\FIT1048-...
*****
FARKLE :: A dice game for 1-4 players
*****
Cheryl Total: 0      Current: 50      Dice Left: 5
*****
It's your turn, Cheryl

Do you want to [q]uit, [r]oll or [s]core? r

      [6] [x] [5] [5] [3] [1]
       1  2  3  4  5  6

Please select the dice you want using the number under
the die. Do not include spaces. Press Enter to finish.

The dice you want are: 5

Please enter only scoring values.
The dice you want are: 6
```



```
C:\Users\choward\Documents\MonashUnits\FIT1048-...
*****
FARKLE :: A dice game for 1-4 players
*****
Cheryl Total: 300    Current: 0    Dice Left: 6
*****
It's your turn, Cheryl

Do you want to [q]uit, [r]oll or [s]core? r

    [3] [3] [6] [6] [3] [3]
     1  2  3  4  5  6

Please select the dice you want using the number under
the die. Do not include spaces. Press Enter to finish.

The dice you want are: 125_
```

This shot shows the player's saved score and displays their next roll.

Selecting the dice is as easy as typing in the number of the required dice then processing each character in the string to calculate the players running total, based on the scoring system you have implemented.

```
C:\Users\choward\Documents\MonashUnits\FIT1048-...
*****
FARKLE :: A dice game for 1-4 players
*****
Cheryl Total: 300    Current: 450    Dice Left: 6
*****
It's your turn, Cheryl

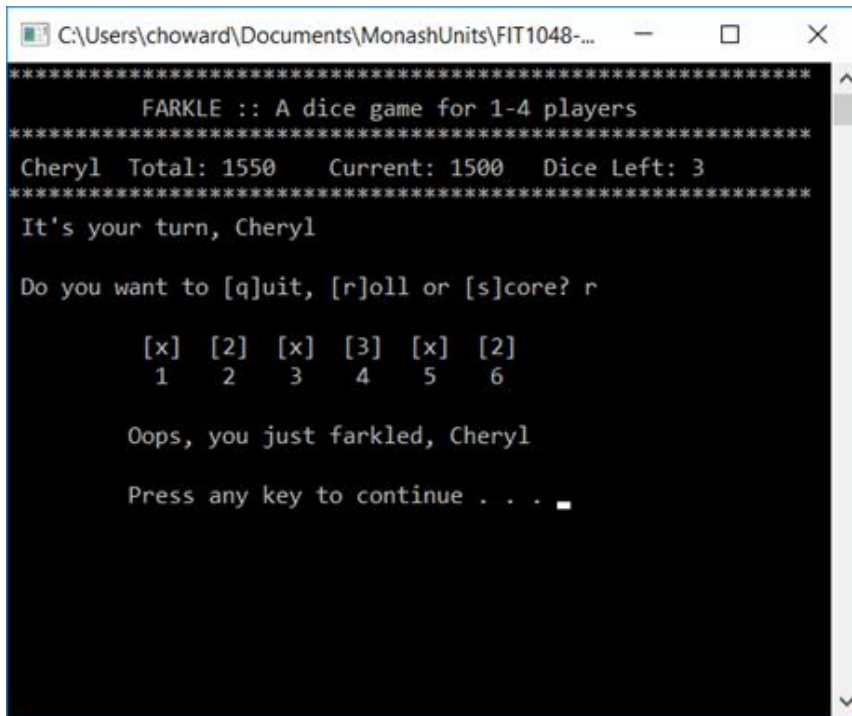
Do you want to [q]uit, [r]oll or [s]core? r

    [4] [1] [6] [4] [2] [3]
     1  2  3  4  5  6

Please select the dice you want using the number under
the die. Do not include spaces. Press Enter to finish.

The dice you want are: _
```

This shot shows the player has rolled and scored all 6 dice so has the option to continue rolling with 6 dice.



This shot shows the rolled dice, all locked dice and that the player did not roll any scoring values – they Farkled! This ends their turn and they do not score any points – they lose their current running total (in this case 1500 points ... this is what happens when you get greedy).

### Submission Instructions:

A zip file containing your Visual Studio project and any associated documentation files (readme, if required) must be compiled and uploaded to the Moodle site. Your code **MUST** be submitted as a Visual Studio project to facilitate ease of assessment and feedback.

The assignment must be created and submitted as a Visual Studio 2017 project. You may complete the exercises in your preferred IDE, however you should create a Visual Studio project in order to submit.

### Files to be submitted:

One Visual Studio Project (2017) called “***YourFirstNameLastNameID***”, containing all the required files.

You need to make sure there are no spaces in the source filenames.

Zip the folder under the same name and submit it to moodle.

**NOTE!** *Your submitted files must be correctly identified (as described above). Any submission that does not comply will receive an automatic 20 marks penalty (applied after marking).*

To learn more about Farkle please visit the following links:

1. <https://en.wikipedia.org/wiki/Farkle>
2. <https://www.youtube.com/watch?v=vQHxBRYs3qQ>
3. <https://www.youtube.com/watch?v=PtZlur9Kmb8>
4. <http://www.playonlinedicegames.com/farkle>