Southern New Hampshire University

IT 312 Final Project Guidelines and Rubric

Overview

As a software development professional, you will need to understand a variety of programming languages, and you should be able to apply industry standard best practices in program development. For the final project in this course, you will design, develop, and document a working **dice game** through command line inputs and outputs. You must choose **one game** from the following options:

1. **LCR:** Left Center Right (LCR) is a dice game using three special dice and player pieces called chips (which are not for betting, as they are in poker). In short, each player on his or her turn rolls the dice. The dice determine how many of the player's chips have to be moved around to other players. The last player holding chips wins the game. Review the complete rules for this game.

2. **Farkle:** Farkle is a game that uses six dice. The goal is to strategically decide how to score your own dice according to the point combinations laid out in the rules to be the first player to reach 10,000 points. Review the complete rules for this game.

3. **Liar's Dice:** Liar's Dice uses 5 dice per player (so 15 dice for three people, 20 dice for four people, and so on). The object of the game is to guess how many of a particular die sides are showing among all of the players, knowing only your own set. Each player's guess has to increase the number of die sides on the table. Guessing continues until a player is called out as a "liar" and all of the dice are revealed. Review the complete rules for this game.

You will be responsible for delivering a working program. The program should consist of a minimum of two classes to support an object-oriented design. Along with the program, you will submit documentation describing your program. It should include the following:

- Any special program features
- A description of your process for working through the problem, such as the way you broke down the problem and (if applicable) why your original breakdown is different than your final product
- A copy of pseudocode for your programming logic
- A short narrative on errors/bugs you encountered and how you worked to resolve them

As you progress through the course, you will be provided with building block assignments. These assignments follow a similar format to other C++ assignments you will be completing in the course. However, these assignments are designed to provide specific guidance and skills necessary to complete your final working program. You will also be using a journal to update your instructor on the progress of your work and receive feedback specific to your final project.

This project addresses the following course outcomes:

- Analyze programming problem statements for effectively designing and creating working products in alignment with problem criteria
- Implement the appropriate use of C++ language basics within programs for future software development activities

- Design and develop functional programs based on object-oriented best design principles
- Solve programming logic errors and syntax errors or exceptions to ensure acceptable program output
- Employ programming best practices to promote code readability and maintainability

Prompt

Develop a working program designed to follow the specifications in the project overview. Provide documentation describing your development process.

Specifically, the following **critical elements** must be addressed:

I. **Documentation**
The documentation accompanying your program should address the following:

A. **Problem Statement/Scenario**: Select a game to develop and analyze the scenario to determine necessary consideration for building your game program.
B. **Overall Process**: Provide a short narrative that shows your progression from problem statement to breakdown to implementation strategies. In other words, describe the process you took to work from problem statement (starting point) to the final product. Your process description should align to your end resulting program and include sufficient detail to show the step-by-step progress from problem statement analysis.
C. **Pseudocode**: Break down the problem statement into programming terms through the creation of pseudocode. The pseudocode should demonstrate your breakdown of the program from the problem statement into programming terms. If the pseudocode differs from the submitted program, document the differences and the reason for changes.
D. **Methods and Classes**: Your pseudocode reflects distinct methods and classes that will be called within the final program. If the pseudocode differs from the submitted program, document the differences and reason for changes.

II. **Program**
Your working program should include/address the following requirements. Please note that the comments within your program will count toward your instructor's assessment of the documentation aspects of your submission:

A. **Input/Output**: Your program reads input from a file and uses system input and output through the console.
B. **Control Structures**: Your program utilizes appropriate control structures for game logic.
C. **Libraries**: Your program uses standard libraries to pull in pre-defined functionality.
D. **Arrays and String Manipulation**: Your program effectively uses arrays and string manipulation.
E. **Classes Breakdown**: Your program is broken down into **at least** two appropriate classes.
F. **Methods**: Your program uses all included methods correctly within the classes.

G. **Error Free**: Your program has been debugged to minimize errors in the final product. Your instructor will run your program to determine functionality.
H. **Error Documentation**: Accurately document major errors that you encountered while developing your program.
I. **Solution Documentation**: Document how you solved the errors and what you learned from them.
J. **Formatting Best Practices**: Your program code is easy to read, following formatting best practices as defined by the industry, such as with indentation.
K. **Documentation Best Practices**: Your program contains comments where needed, in appropriate detail for communicating purpose, function, and necessary information to IT professionals.
L. **Coding Best Practices**: Your program supports clean code through descriptive variable names.

Building Blocks

Building Block One: *Programming (Rolling Dice)*

In **Module Four**, you will submit source code that demonstrates the dice rolling functionality for your game. **This submission will be graded with the Programming Rubric**.

Building Block Two: *Pseudocode (Building the Final Project)*
In **Module Five**, you will draft pseudocode for your final project. **This submission will be graded with the Pseudocode Rubric**.

Building Block Three: *Fill in the Blank (Making a Player Class)*
In **Module Five**, you will submit source code that correctly implements the player class functionality for your game. **This submission will be graded with the Fill in the Blank Rubric**.

Building Block Four: *Programming (Reading From a File)*
In **Module Six**, you will submit source code that correctly reads and displays the rules for the game from a text file. **This submission will be graded with the Programming Rubric**.

Final Project: *Dice Game*
In **Module Seven**, you will submit your final project. It should be a complete, polished artifact containing **all** of the critical elements of the final product. It should reflect the incorporation of feedback gained throughout the course. **This submission will be graded with the Final Project Rubric.**

Southern New Hampshire University

Deliverables

Building Block	Deliverable	Module Due	Grading
One	Pseudocode	Four	Graded separately; Building Block One Rubric
Two	Programming (Rolling a Die)	Five	Graded separately; Building Block Two Rubric
Three	Fill in the Blank	Five	Graded separately; Building Block Three Rubric
Four	Programming (Reading From a File)	Six	Graded separately; Building Block Four Rubric
	Final Project: Dice Game	Seven	Graded separately; Final Project Rubric

Final Project Rubric

Guidelines for Submission: Your program and documentation will be submitted at the same time and graded together as a single unit. Your program should follow best practices and your documentation should be in the form of comments within the program code (for reuse and for other IT professionals to modify) as well as in the form of a more comprehensive documentation brief (submitted as a Word document).

Critical Elements	Exemplary	Proficient	Needs Improvement	Not Evident	Value
Documentation: Problem Statement/Scenario	Meets "Proficient" criteria and shows keen insight into analysis of problem statements for informing game building (100%)	Accurately analyzes the selected scenario to determine necessary considerations for building the game (85%)	Analyzes the selected scenario to determine necessary considerations for building the game, but with gaps in accuracy or necessary considerations (55%)	Does not analyze the selected scenario to determine necessary considerations for building the game (0%)	6.33
Documentation: Overall Process	Meets "Proficient" criteria and description shows keen insight into the process for progressing from problem statements to end products (100%)	Clearly and accurately describes the process taken from problem statement analysis to end product (85%)	Describes the process taken from problem statement analysis to end product, but with gaps in accuracy or clarity (55%)	Does not describe the process taken from problem statement analysis to end product (0%)	6.33
Documentation: Pseudo-code	Meets "Proficient" criteria and shows keen insight into using pseudocode to guide development from problem statements to end products (100%)	Accurately breaks down the problem statement into programming terms through creation of accurate pseudocode (85%)	Breaks down the problem statement into programming terms through creation of pseudocode, but with gaps in accuracy (55%)	Does not break down the problem statement into programming terms through creation of pseudocode (0%)	6.33

Documentation: Methods and Classes	Meets “Proficient” criteria and shows keen insight into the journey from pseudocode to final product in terms of necessary methods and classes for function (100%)	Pseudocode accurately reflects the necessary methods and classes that will be called upon in the final program or an accurate description of changes from pseudocode to final program is provided (85%)	Pseudocode reflects the necessary methods and classes that will be called upon in the final program or a description of changes from pseudocode to final program is provided, but with gaps in accuracy (55%)	Pseudocode does not reflect the necessary methods and classes that will be called upon in the final program or does not provide a description of changes from pseudocode to final program is provided (0%)	6.33
Program: Input/Output		Program correctly reads input from a file and uses system input and output through the console (100%)	Program reads input from a file and uses system input and output through the console, but with errors (55%)	Program does not read input from a file and use system input and output through the console (0%)	4.75
Program: Control Structures		Program utilizes appropriate structures for game logic (100%)	Program utilizes structures that are not appropriate for game logic (55%)	Program does not utilize structures (0%)	4.75
Program: Libraries		Program uses standards libraries to correctly pull in pre-defined functionality (100%)	Program uses standard libraries to pull in pre-defined functionality, but with errors (55%)	Program does not use standard libraries to pull in pre-defined functionality (0%)	4.75
Program: Arrays and String Manipulation		Program effectively utilizes arrays and string manipulation (100%)	Program utilizes arrays and string manipulation, but ineffectively or with errors (55%)	Program does not utilize arrays and string manipulation (0%)	4.75
Program: Classes Breakdown		Program is correctly broken down into two or more appropriate classes (100%)	Program is broken down into two or more classes, but with gaps in appropriateness or with errors (55%)	Program is not broken down into two or more classes (0%)	6.33
Program: Methods		Program uses all included methods correctly within the classes (100%)	Program uses all of the included methods in the classes, but with errors (55%)	Program does not use all of the included methods in the classes (0%)	6.33
Program: Error Free		Program is error-free, indicating necessary debugging has taken place (100%)	Program has minimal errors that still allow for functionality, indicating gaps in debugging practices (55%)	Program does not function, indicating that debugging has not occurred (0%)	6.33
Program: Error Documentation		Accurately documents errors that occurred during creation in detail (100%)	Documents errors that occurred during creation, but with gaps in accuracy or detail (55%)	Does not document errors that occurred during creation (0%)	6.33
Program: Solution Documentation		Documents logical solutions to major errors in terms of lessons learned (100%)	Documents the solutions to major errors, but not in terms of lessons learned or with gaps in logic (55%)	Does not document the solutions to major errors (0%)	6.33

Program: Formatting Best Practices	Meets "Proficient" criteria and code is exceptionally well written, showing keen insight into seamless application of best formatting practices (100%)	Program code is easy to read and follows formatting best practices as defined by the industry (85%)	Program code is challenging to read or there are gaps in attendance to best practices in formatting as defined by the industry (55%)	Program code is unreadable or does not follow best practices in formatting (0%)	6.33
Program: Documentation Best Practices	Meets "Proficient" criteria and program contains well-balanced documentation, showing keen insight into best practices in documentation code (100%)	Program contains appropriate documentation for communicating purpose, function, and necessary information to IT professionals (85%)	Program contains documentation for communicating purpose, function, and necessary information, but is not appropriate for IT professionals (55%)	Program does not contain documentation for communicating purpose, function, and necessary information (0%)	6.33
Program: Coding Best Practices		Program correctly supports clean code through descriptive variable names (100%)	Program supports clean code but not correctly through descriptive variable names (55%)	Program does not support clean code (0%)	6.33
Articulation of Response	Submission is free of errors related to citations, grammar, spelling, syntax, and organization and is presented in a professional and easy to read format (100%)	Submission has no major errors related to citations, grammar, spelling, syntax, or organization (85%)	Submission has major errors related to citations, grammar, spelling, syntax, or organization that negatively impact readability and articulation of main ideas (55%)	Submission has critical errors related to citations, grammar, spelling, syntax, or organization that prevent understanding of ideas (0%)	5.04
				Total	**100%**