|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **School of Sciences** | UCLan Coursework Assessment Brief | | | 2023-2024 |
| Module Title: Advanced Programming with C++  Module Code: CO2402 | | | Level 5 |
| **C++ Programming Assignment**  **WheelOfFortune-ish** | This assessment is worth 50% of the overall module mark. | | |
| **THE BRIEF / INSTRUCTIONS**  **Introduction**  For this assignment, you will work towards implementing a simplified simulation of the well-known game Wheel of Fortune. This is not an interactive game, but rather a simulation of two (2) players taking turns over a set number of rounds. Play is to be automated according to a sequence of pseudo-random numbers – there is no artificial intelligence, and no user input. All output will be text based and directed towards the console – there are no graphical elements to this assignment.  I expect you to complete this project in your own time outside of scheduled labs. This is an individual project, and no group work is permitted. Do not diverge from the assignment specification. If you do not conform to the assignment specification, then you will lose marks. Ask for clarification if you are unsure!  **Avoiding Plagiarism**  You will be held responsible if someone copies your work - unless you can demonstrate that you have taken reasonable precautions against copying.  **Learning Outcomes Assessed (see module descriptor for full list)**   * Make an informed choice of implementation method for a given problem. * Implement and document a structured program to meet a given specification. * Select and apply appropriate data structures and algorithms to a given problem.   **Deliverables, Submission and Assessment**   * There are three deliverables – your **source code**, a **report**, and a **video demonstration**. * **To get a mark above 0 you must submit all 3 deliverables.** * Submission is electronic, via Blackboard,   + For the source code, upload **ONLY the .cpp and .h files**. DO NOT submit the solution file!   + Make sure you include your name as a comment on the first line of all your source code files. Also include your name at the top of your report.   + See the last sections for a summary of what to include in your report and video.   **Resources**   * This assignment has an associated set of support files.   + wheel.txt – A text file for reading in the game configuration   + random.cpp – Code to generate a sequence of pseudo-random numbers from a seed   + seed.txt – A text file containing the random number seed for testing   + code style guide   **Overview**  Although there is no graphical element to the program, it may be helpful for you to think of the game as being played on a wheel consisting of 16 slices. Each of the slices has a type and some additional information, which is contained within the data file *wheel.txt*. In your simulation, the game should have two players: *John* and *Maria*. At the start of the game, the wheel’s starting point is the slice corresponding to “500” (as shown above).  A game is played as a series of rounds (different hidden words correspond to different rounds). During a round, each player takes turns to roll the wheel of fortune. You will represent rolling the wheel by generating a random number between 1 and 16. Different things will happen according to the slice rolled by a player. Exactly what happens on each slice depends on how many marks you want...  If implemented in full, the game will be played for four (4) rounds corresponding to four (4) different hidden words. At the end of each round, the winning player (who has revealed the round’s word) will bank the money collected during the round. At the end of the game (i.e., end of round 4), the player with the most money is declared the winner.  **Program Specification**  You should implement the features described below in order. **To be eligible for a mark within any classification, you must have attempted (with ‘some success’) all the features for all the previous classifications. This means that you must have earned at least 50% of the marks awarded to each of the features of the previous classifications.**   * **Basic Scenario = bare pass (40% max)**   If you implement only what is described in this section (and provide video evidence of it running to completion), you will get 40%. No more, no less.   * + Create a class called CSlice to represent the slices on the wheel.   + Read in the data file *wheel.txt* and use it to set up an array of 16 CSlice objects.     - You do not need to use pointers or dynamic memory allocation at this level.     - Each line in the file represents one slice.       * The first number in each line represents the slice’s type.       * Following the type is the slice’s amount and a description.     - You will need to store this information as data in the CSlice objects as you create them.     - Rows beginning with ‘1’ denote non-special squares.     - The descriptions of different types of slices (as found in wheel.txt) are also provided in the table below.  |  |  |  | | --- | --- | --- | | **Type** | **Amount** | **Name/Description** | | 1 | 20 | 20EUR | | 1 | 30 | 30EUR | | 1 | 40 | 40EUR | | 1 | 50 | 50EUR | | 1 | 60 | 60EUR | | 1 | 80 | 80EUR | | 1 | 100 | 100EUR | | 1 | 150 | 150EUR | | 1 | 200 | 200EUR | | 1 | 500 | 500EUR | | 2 | 0 | LoseTurn | | 3 | 0 | Bankrupt | | 4 | 0 | Bankrupt+ | | 5 | 0 | SecondChance | | 6 | 0 | Steal | | 7 | 0 | Jackpot |  * + Read in the data file *rounds.txt* and use it to set up an array of 4 strings.     - You do not need to use pointers or dynamic memory allocation at this level.     - Each line in the file represents the target hidden word for each round.   + For each player you simply need to store their name.     - You could use variables for this.   + In your main program, simulate playing only the first round of the game as follows:     - When the game is started, a welcome message is displayed. The format of the message is: 'Welcome to WheelOfFortune-ish'     - Players (starting from John) should take turns playing as follows:       * The player rolls the wheel of fortune.         + You will represent rolling the wheel by generating a random number between 1 and 16.         + The starting point of the wheel (when the program starts) is the slice represented by the first line in wheel.txt.         + From that point onward, subsequent wheel rolls by any player should start from the last slice that was rolled.       * Output the message: '<Player> rolls <number>'       * On the next line, the name of the player and the name of the slice the player has rolled should be displayed.         + Output the message: '<Player> rolls <slice description>'     - When a player lands on any non-special slice (i.e., type = 1):       * The player chooses at random a letter from a-z (assume that all words will contain only non-capital alphabetic letters).         + By generating a random number between 1 and 26, corresponding to all 26 letters in the English alphabet.         + Output the message: '<Player> guesses <letter chosen>'       * If the letter chosen is part of the hidden word of the round (i.e., first line in rounds.txt) and is still hidden, then this should be “revealed”.         + You can use an array of Booleans to keep track of the letters revealed.         + For e.g., if the word is “programming” and the chosen letter is ‘m’, then the array would look like this:   *p r o g r a m m i n g*   |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **F** | **F** | **F** | **F** | **F** | **F** | **T** | **T** | **F** | **F** | **F** |  * + - * + Output the message:   '<Player> reveals <number of letters revealed in last play>'   * + - * + A player carries on playing if he/she has revealed a letter, i.e., they continue rolling the wheel and choosing letters at random.   Once a letter chosen is not part of the round’s word, or has already been revealed, the turn switches to the opponent player, who repeats the same process.  When this occurs output the message: '<Player> loses turn due to inappropriate letter choice'   * + - When a player lands on a special slice, i.e., types 2 - 7, the player simply loses his/her turn. Output the message: '<Player> loses turn due to rolling a special slice'     - The round ends when the entire word is revealed.       * The player who achieves this during his/her turn wins the game.     - At the end of the game, you should output the following messages:   'Game Over'  '<Player> has won the round’   * + Your code must follow the style guide precisely.   + Your code must be well commented throughout. * **Pass Mark = third classification (41 - 50%)**   + All objects should now be stored in dynamic memory.     - Slices should be accessed via an array (or vector) of pointers.   + Declare separate classes for each different type of special slices on the wheel, i.e., types 2 - 7.     - A special slice is *A Kind Of* slice, i.e., each new class should be derived from CSlice.   + The game should be played for 4 rounds (corresponding to the 4 words included in *rounds.txt*).     - At the beginning of each round output the message: 'Round <number>:<word>'     - During the 1st and 3rd round, player John is first to play.     - During the 2nd and 4th round, player Maria is first to play.     - For each player, you will need to also keep track of the total money earned during the current round, as well as the total money banked in previous rounds.     - Only the player who manages to reveal the last letter of each round’s word gets to bank the corresponding amount he/she accumulates during the round.     - When a type 1 slice is rolled, a player is awarded the amount of the slice multiplied by the number of new letters revealed.       * For e.g., if the amount is 100 and the letter chosen is revealed twice, then the player’s balance for the round is awarded 100\*2 = 200.       * Output the message: '<Player> earns <amount earned in last play>'     - The current round’s balance of the winning player should be transferred to the money banked in previous rounds upon completion of the round. Output the message:   '<Player> wins round and banks <amount earned during round>'   * + - The amount earned during the round by the loosing player is simply discarded.     - At the end of each round output the following messages:   'John’s total banked amount is <John’s banked money>  'Maria’s total banked amount is <Maria’s banked money>   * **Lower second classification (51-60%)**   + For this grade band you need to attempt a polymorphic solution.     - You will need a hierarchy of classes for the different types of slices.     - When it comes to implementing the different behaviour of the slices, the functionality is devolved to the lower levels of the hierarchy:       * You should have a collection of pointers of type CSlice\*, but the bespoke implementation code should be written in the derived classes.       * You should not be calling methods on the derived classes directly.   + You must use an STL container to store/access the CSlice pointers.   + Implement additional classes derived from CSlice for the *LoseTurn*, *Steal* and *Jackpot* slices (types 2, 6, 7).     - Ignore all other special slices (if rolled, the player still loses his/her turn).   + If a player rolls the 'LoseTurn' slice, the player loses his/her turn. Output the messages:   '<Player> rolls LoseTurn'  ‘<Player> loses turn’   * + If a player rolls the “Steal” slice and subsequently reveals a letter, the player “steals” 50% of the opponent’s banked money. This means that the money stolen is immediately credited to the current player’s banked money, while the opponent’s banked balance is halved. Output the messages:   '<Player> rolls Steal'  '<Player> steals <amount> from <Opponent Player>'   * + If a player rolls "Jackpot" and subsequently reveals a letter, the player’s banked amount is immediately doubled. Output the messages:   '<Player> rolls Jackpot'  '<Player> doubles banked money’   * **Upper second classification (61-70%)**   + For this grade, the polymorphic aspect of your solution must be fully correct, and polymorphism must be serving a genuine purpose.     - You will not be awarded marks if you just plonk the keyword "virtual" into your code, whilst the work within your code is still done in a procedural fashion. In other words, you should NOT check for the type of a slice before executing the actions that correspond to the slice. Your code should be such that the corresponding actions when rolling a slice are automatically inferred (transparently) through polymorphic practices.   + Players must be implemented as classes (not variables in main or structs).   + Derive additional classes from CSlice to implement the *Bankrupt* and *SecondChance* slices (types 3, 5).     - Ignore the *Bankrupt+* special slice (if rolled, the player still loses his/her turn).   + If a player rolls the “SecondChance” slice and subsequently reveals a letter, the player obtains a token that allows him/her to retain his/her turn in case he/she loses it in the future.     - For e.g., if players guess a letter that is not included in the round’s word, they normally would lose their turn. If, however, the player has a SecondChance token, then they do not lose their turn, but instead they lose one of their tokens.     - Each player may hold any number of SecondChance tokens at any given point in time.     - Every time a player should lose his/her turn, the player’s SecondChance tokens (if any) are reduced by 1 instead.     - Output the messages:   '<Player> rolls SecondChance’  'Number of SecondChance tokens earned: <number of tokens>'   * + - When a player uses a SecondChance token, output the messages:   '<Player> uses SecondChance token’  'Remaining SecondChance tokens: <number of tokens>’   * + If a player rolls the “Bankrupt” slice, the player loses (i) all money earned during the current round, and (ii) all SecondChance tokens collected. Additionally, the player loses his/her turn. Output the message: '<Player> rolls Bankrupt’.   + At this level there should be no global variables in your code. (Global constants are allowed). * **First classification (71-80%)**   + The implementation should make use of object-oriented methods throughout (not just the polymorphic parts). This will mean the implementation of several classes.     - One of the classes must be the game itself, which will act as a manager class.   + Derive an additional class from the class created for Bankrupt to implement the Bankrupt+ slice (type 4).     - If a player rolls the “Bankrupt+” slice, the same actions should be triggered as in the case of “Bankrupt”, however, additionally all banked money of the player should also be lost. Output the message: '<Player> rolls Bankrupt+ and loses everything’ * **High First classification (81-100%)**   + Implement the following extra functionality: in any given round, players should be allowed to choose at random only letters that have NOT BEEN CHOSEN yet during the round.     - The way this is implemented (in terms of time efficiency) will play a role in the marks to be awarded for this functionality.   + The STL syntax for vectors is ugly and cumbersome.     - Use typedef and auto to clean up the ugly syntax of vector declarations.   + Use smart pointers such that you do not directly invoke dynamic memory allocation anywhere in your code.     - Use the C++14 syntax to avoid the keyword new.     - This will require a little bit of independent research on your part, since we have not yet covered this.   + There must be no memory leaks when you run your code.     - Proof of this should be included in the document/video demonstration.   **Report Contents**  For any grade:   * Include a functionality checklist as the first page of your report. This will tell me **which grade level you are working towards**, and which sample output I should compare your output to. * Copy the entire console output of your program running from the seed I supplied. Use Consolas 10pt font for this.   For a grade above 80%   * Take a screenshot of the Visual Studio Debug Output window showing evidence that your program has no memory leaks.   **Video Demonstration Contents**  The video should be short. You can either upload an .mp4 file to Blackboard with your submission, or else upload it to YouTube (or similar) and provide a link.  It needs to begin by showing something on the screen that shows it is you logged in.  It should then show your assignment compiling without errors and running to completion.  The console output should be the same as the one copied into your report. The point of the video is to demonstrate that you haven’t faked it.  There is no need for any commentary. | | | | |
| **PREPARATION FOR THE ASSESSMENT**  Please study all topics covered to date. | | | | |
| **RELEASE DATES AND HAND IN DEADLINE**  Assessment Release date: 29/12/2023 Assessment Deadline Date and time: 19/03/2024, 23:59  Please note that this is the final time you can submit – not the time to submit!  Your feedback/feed forward and mark for this assessment will be provided on Blackboard approximately 2-3 weeks after submission. | | | | |
| **SUBMISSION DETAILS**   * This assignment must be submitted on Blackboard by **23:59 on 19th of March 2024.** | | | | |
| **HELP AND SUPPORT**   * For any questions regarding the worksheet, please contact the module leader (Andreas Pamboris) directly:   + Via email ([apamboris@uclan.ac.uk](mailto:apamboris@uclan.ac.uk))   + In class   + During office hours * You will find links to lots of useful resources in the My Library tab on Blackboard. * If you have any other query or require further support speak with us for advice on accessing all the University services as well as the Library services. Whatever your query, our expert staff will be able to help and support you. For more information, how to contact us and our opening hours visit [Student Information and Support Centre](https://www.uclan.ac.uk/students/library-it/library/the_i.php). * If you have any valid mitigating circumstances that mean you cannot meet an assessment submission deadline and you wish to request an extension, you will need to apply online prior to the deadline. | | | | |
| Disclaimer: The information provided in this assessment brief is correct at time of publication. In the unlikely event that any changes are deemed necessary, they will be communicated clearly via e-mail and a new version of this assessment brief will be circulated. | | | Version: 2 | |