# Assessment Brief: Advanced Programming

| **Module Leader:** Michael Meredith | | **Level:** 5 |
| --- | --- | --- |
| **Module Name:** Advanced Programming | | **Module Code:** 55-508225 |
| **Assignment Title:** Advanced Programming – Yahtzee Case Study | | |
| **Individual Task** | **Weighting:** 50% | **Magnitude:** ~30 *hours (notionally[[1]](#footnote-1))* |
| **Submission date/time:**  25th November 2021 at 3pm | **Blackboard submission:** Yes **Turnitin submission:** No | **Format:** ZIP of Visual Studio Project |
| **Planned feedback date:**  16th December 2021 | **Mode of feedback:** Blackboard | **In-module retrieval available:** Yes |
|  | | |
| **Module Learning Outcomes**   * Describe, identify and apply good-practice object-oriented principles and concepts in a modern programming language, such as C++, to implement object-oriented designs and encapsulate functionality associated with classic computer science algorithms and data-structures * Identify and apply appropriate facilities provided in standard libraries and design patterns in the development of an application, including event-based and reactive programming. * Identify and apply memory management principles and demonstrate the impact of resource allocation within a program | | |

## Assessment Brief

This assessment is designed to allow you to demonstrate your fundamental programming skills in C/C++ using good object-oriented practice though the implementation of a basic Yahtzee[[2]](#footnote-2) game and player management system (see the accompanying video demonstration of the application in operation). It is important to read this document and understand the techniques and competencies you are being asked to demonstrate in parts of this application to ensure you meet the grading scheme set out in the Assessment Criteria section of this document (at the end). Where no specific guidance is given, make your own decisions based on good practice coding.

Consider and think about your implementation and perhaps undertake refinement passes, depending on your development strategy, to ensure your code is of good quality. Do not focus purely on implementing the game in isolation of the guidelines; the assessment brief ensures there are suitable opportunities for you to demonstrate your understanding in key areas while also allowing you to demonstrate your creativity and thinking. While this assessment specification does ask for specific techniques to be used in certain places, there is more left to you to think about the best way of tackling the problem.

There is no single right solution, and this assessment is as much about you being able to deploy key programming concepts as it is about you being able to think about and problem solve your way through the creation of an application using your knowledge, thinking, experience and general programming skills.

## Yahtzee Game

When the application starts, the program loads a list of existing players from disk into a Standard Template Library (STL) vector. This information includes name, encrypted password, and highest score for each player. Player’s game history should not be loaded at this stage and there may be zero players to load. The password must be stored using some form of basic encryption[[3]](#footnote-3) in the file (but held in memory unencrypted). The rest of the player information can be stored plain text within the file. Use C-Strings to store player name and passwords using your own heap memory management. Player information should only be saved back to disk once the user closes the application via the Exit option.

The launch screen starts with player management / picker, as illustrated in Figure 1. On this screen, the user can sort players in alphabetical order or by highest score. Sorting must be done using a suitable STL algorithm(s), and the player information is redisplayed to reflect the update. There is no automatic sorting when adding or removing players and they maintain their order unless the user chooses to sort the list using the menu options.

The user can add and/or remove players and choose a player to start the game from the player management / picker. Player names must be unique within the application; duplicates are not allowed (case insensitive). When a new player is added or one is removed, the required user interaction occurs, and the player management / picker screen is redisplayed with the updated player list. ***Nothing is saved back to disk until the user exits the program by choosing the Exit item on the launch screen and all operations are done to data in memory up to that point.***

Welcome to Champion Yahtzee

---------------------------

Player Name Highest Score

----------- -------------

Iron Man 56

Hulk 32

Thor 72

Black Widow 45

Hawkeye 32

Captain America 16

1) Choose player

2) Add player

3) Remove player

4) Sort players alphabetically

5) Sort by highest score

9) Exit

Please choose an option: 1

Please enter player name: hawkeye

Please enter player password: arrows

Figure 1: Example of the player management / picker screen / menu

To start playing, the user enters the name of the player along with its password, which is validated before progressing. If the name and password do not match, a suitable error is shown, and the user stays on the player management / picker menu. When the name and password match, the player history is loaded from disk[[4]](#footnote-4) and stored in memory, and the player information screen is shown, as illustrated in Figure 2. The game history should be stored in memory using your own heap array that you resize as needed (i.e., do not use an STL vector for this and resize as needed within your application). The game history must include total games played and total points accumulated (across all games played) along with a time-stamped scorecard for each game.

Welcome to Champion Yahtzee Hawkeye

-----------------------------------

Highest Score: 32

Total Games: 11

Accumulated Score: 279

Average Score: 25.3

1) Play game

2) View historic scorecards

9) Return to player chooser screen

Please choose an option:

Figure 2: Example of the main player information screen

From the player information screen, the user can view their game history (see Figure 3 for an example), start a new game (see Figure 4 for an example), and return to the top-level player management / picker screen to choose another player or exit the application (Figure 1). When the user returns to the launch screen, the player’s game history is written back to disk and released from heap memory.

Hawkeye Yahtzee History

-----------------------

Game: 10 of 11

Date: 5th Sept 2021 – 14:32

----- SCORECARD -----

Aces: 2

Twos: 6

Threes: 3

Fours: 8

Fives: 0

Sixes: 12

---------------------

Total: 31

---------------------

1) View the previous scorecard

2) View the next scorecard

9) Return to player statistics screen

Please choose an option:

Figure 3: Example of player history screen

Within the game itself, you can implement just the top-half of the Yahtzee score card if you wish (i.e., just the sum of 1s, 2s, 3s, 4s, 5s and 6s) as demonstrated in Figure 4. Upon completing a game, the player is shown their final scorecard with their total, the player’s statistics are updated, and the scorecard is added to the player’s history in heap memory. ***Saving this game to disk is not done at this state, and only once the user chooses to exit the player screen and return to the player management / picker screen.*** The application returns to the player information screen (Figure 2) upon completing a game.

Dice Roll Set #2 of 6

---------------------

---- SCORECARD ----

Aces:

Twos:

Threes:

Fours: 8

Fives:

Sixes:

------------------

Total: 8

------------------

Dice 1 Dice 2 Dice 3 Dice 4 Dice 5

######### \*\*\*\*\*\*\*\*\* ######### \*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*

# 22222 # \* 22222 \* # 22222 # \* 1 \* \* 33333 \*

# 2 # \* 2 \* # 2 # \* 1 \* \* 3 \*

# 22222 # \* 22222 \* # 22222 # \* 1 \* \* 333 \*

# 2 # \* 2 \* # 2 # \* 1 \* \* 3 \*

# 22222 # \* 22222 \* # 22222 # \* 1 \* \* 33333 \*

######### \*\*\*\*\*\*\*\*\* ######### \*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*\*

Please choose how to score your dice

1: Aces

2: Twos

3: Threes

5: Fives

6: Sixes

Score against:

Figure 4: Example of Yahtzee game (# dice outlines show dice held from previous throws before being scored)

## General Tips and Comments

* All loading from disk should be done using **operator overloading** (assisting encapsulation). There are other sensible places that operator overloading can be use – think about where and deploy additional examples of operator overloading in your application
* You are being asked to load data into memory and only write it back out at certain points to ensure you are successfully **managing heap memory** (as well as trying to help direct you towards a sensible control flow). This will also need you to think about your class and data structures to effectively hold that information in memory – remember to use appropriate data types for variables, objects, and full encapsulation principles.
* Ensure your application gives sensible feedback to the user and handles input errors gracefully, including recovery were possible (i.e. don’t just exit or crash when an error occurs). You can sensibly limit allowed input characters to do this but remember to tell the user and give useful messages.
* The amount of heap memory available should be your only limit on the number of players your application can remember along with their history, but don’t be wasteful with heap memory either
* Consider other opportunities to deploy the **STL** beyond those explicitly stated in this document
* Write **good** **object-oriented** code where algorithms that operate on data are put into the same class and data members have as restrictive access as possible. A poor design is one where you have a class of data and just get and set methods and the operations on those bits of data is done outside of the class
* **Think about your code, use appropriate C/C++ constructs,** and keep the implementation as simple as it allows – code that is unnecessarily complex is harder to maintain, debug, and likely inefficient
* **DO NOT USE ANY FORM OF RECURSION TO DEAL WITH THE MENU SYSTEM NAVIGATION – THIS WILL LEAD TO SPAGHETTI CALL STACKS – USE LOOPS AND METHODS TO STRUCTURE THIS BEHAVIOUR.** I have tried to help lead you down the right path with some of the constraints and behaviours I have asked for in this specification

## Submission

Your assignment should be submitted electronically through the module Blackboard site as a **single ZIP** file that contains your **project** and **source code** files (but without the build and intellisense files and folders). Your last on‑time attempt will be viewed and graded (as per university regulations).

The submission must be in the form of a Visual Studio 2019 project within the compressed ZIP file and contain all files that allow the project to be opened, built, and run on a campus computer. **Make sure that you upload the correct files by checking what you have submitted by downloading it again from Blackboard and opening the project from the UNZIP of the submission** - mistakes discovered after the deadline cannot be corrected; it is your responsibility to ensure that you submit the correct files by the deadline. You may be asked to provide a walkthrough of your code during which you will need to discuss all aspects of the work you submitted before a grade is awarded.

## Avoiding Plagiarism

Looking for snippets of code on the internet and searching for materials that will help solve parts of a problem is part of software development. If you do this, you must add a comment in your code stating the URL for snippets of code that have heavily influenced your solution and a short summary of what you have taken / learnt to demonstrate you understand and not just unashamedly lift it. You should not be directly copy from the internet (or elsewhere), even once referenced, but making it your own code anyway. There is no need to reference core learning that you apply to this task – for example, if you have learnt how to use lambda expression and applied it here, that is fine.

This assessment is about you demonstrating your knowledge and understanding when it comes to programming and not the internet’s, your friend’s, or Discord’s. If you try to pass off work as your own that isn’t, then an academic conduct meeting will be arranged, and you’ll likely get zero marks. A second offense can lead to withdrawal from the course, so please ask if you’re unsure.

## Grading

Please see the assessment criteria rubric at the end of this document. The marking aligns to the university generic grading descriptor against the key competencies and criteria given in the left-most column. Standard university grading is being used for this task.

## In-module retrieval

Following first-sit assessment, In-Module Retrieval (IMR) will be offered to students achieving below 40% for either (or both) of these assessment tasks.

IMR means that you will have an opportunity to resubmit a reworked version of your original assignment, following feedback, within a short space of time (usually 5-10 working days) in order to achieve a pass grade. The maximum mark available is 40%.

You will need to look out for an email from your Module Leader (to your SHU email address) shortly after the marks have been released. This will give you more details and tell you what you need to do next.

If you decide not to participate in the In-Module Retrieval attempt and subsequently do not pass the module, you will be referred which means you will have an opportunity to resubmit an assignment during the reassessment period (usually July for students on standard courses).

The time available to you to submit an IMR attempt will be less for those students who submit after the original submission deadline date (with or without an approved extension).

Further information about IMR is available on Assessment 4 Students on [shuspace](https://academic.shu.ac.uk/assessment4students/marks-and-feedback/reassessment/)

# Assessment Criteria (using the university [standard level 5 generic grade descriptor](https://students.shu.ac.uk/regulations/assessment_awards/University%20Grade%20Descriptors%20(Level%204;%20new%20Level%207).pdf))

|  | **FAIL** (insufficient) | | | | **THIRD** (sufficient) | | | **LOWER SECOND** (good) | | | **UPPER SECOND** (very good) | | | **FIRST**  (excellent) | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Zero | Low  Fail | Mid Fail | Marginal Fail | Low  3rd | Mid 3rd | High 3rd | Low  2.2 | Mid  2.2 | High 2.2 | Low  2.1 | Mid 2.1 | High 2.1 | Low  1st | Mid  1st | High 1st | Exceptional 1st | Perfect 1st |
| Criteria and weighting | **<19** | | **20-39** | | **40-49** | | | **50-59** | | | **60–69** | | | **70-84** | | **85+** | | |
| **Sensible code separation (use of class and functions), encapsulation and data validation**  20% | No evidence of knowledge or understanding of the subject; no understanding of taught concepts, with facts being reproduced in a disjointed or decontextualised manner; ignores set material in development of work; fails to address the requirements of the brief; lacks basic communication skills. A general level of incompetency in practical tasks; an evident lack of practice; set tasks are not completed; few or no skills relating to tasks are evident. No adherence to rules/conventions set by the industry/employment context | | Insufficient knowledge and understanding of the subject and its underlying concepts; some ability to evaluate given reading/research however work is more generally descriptive; naively follows or may ignore set material in development of work; given brief may be only tangentially addressed or may ignore key aspects of the brief; communication shows limited clarity, poor presentation, structure may not be coherent. Practical tasks are attempted; skill displayed in some areas; there are a significant number of errors; a lack of proficiency in most areas; guidance may be needed to reproduce aspects of the task and/or apply learned skills. Tasks may be incomplete; failure to adhere to some of the rules/conventions set by the industry/employment context. | | Knowledge and understanding sufficient to deal with terminology, basic facts and concepts but fails to make meaningful synthesis; some ability to select and evaluate reading/research however work may be more generally descriptive; strong reliance on available support set sources to advance work; arguments may be weak or poorly constructed; communication/presentation is generally competent but with some weaknesses. Competence in technical/artistic skills; tasks/processes are completed with a degree of proficiency and confidence; tasks are completed with a basic level of independent thought; effective judgements have been made; basic evaluation and analysis of performance in practical tasks is evident. Errors in workflow or completion of the task; general adherence to appropriate rules/conventions set by the industry/employment context. | | | Good knowledge and understanding of the area of study balanced towards the descriptive rather than analytical; evidence of appropriate selection and evaluation of reading/research but generally reliant on set sources to advance work/direct arguments; communication shows clarity, but structure may not always be coherent. A confident approach to practical tasks; a solid grasp of the related processes, tools, technology; creativity in the completion of the task; proficiency is demonstrated by an accurate and coordinated performance; tasks are completed with a good level of independent thought; some autonomy is evident; an ability to reflect on practical work and set future goals. General adherence to standards set by the industry/employment context. | | | Very good knowledge and understanding of the area of study as the student is typically able to relate facts/concepts together with some ability to apply to known/taught contexts; evidence of appropriate selection and evaluation of reading/research, some beyond the prescribed range, may rely on set sources to advance work/direct arguments; demonstrates autonomy in approach to learning; strong communication skills. Broadly autonomous completion of practical tasks/processes; ability to adapt in response to change or unexpected experiences; technical/artistic decision making is highly developed; a clear command of the skills relevant to the task/process; ability to reflect on practical work and set future goals within the wider context of industry/workplace. Adherence to standards set by the industry/employment context. | | | Outstanding/excellent knowledge and understanding of the area of study as the student is typically able to go beyond what has been taught (particularly for a mid/high 1st); evidence of extensive and appropriate selection and critical evaluation/synthesis/analysis of reading/research beyond the prescribed range, to advance work/direct arguments; excellent communication; performance deemed beyond expectation of the level. The ability to make decisions and carry out tasks/processes with autonomy; creative flair and the ability to (re)interpret predefined rules/conventions to select and justify individual working practice; highly developed problem-solving skills; accuracy and fluency; excellent command of skills appropriate to the task; the ability to reflect critically on practical work within the wider context of industry/workplace. Broadly meets expectations set by the industry/employment context | | Exceptional breadth and depth of knowledge and understanding of the area of study, significantly beyond what has been taught in all areas; evidence of extensive and appropriate selection and critical evaluation/synthesis/analysis and of reading/research beyond the prescribed range, in both breadth and depth, to advance work/direct arguments; excellent communication; performance beyond expectation. The ability to make decisions and carry out tasks/processes with autonomy; excellent leadership skills in group contexts; creative flair; extremely well-developed problem-solving skills; the ability to carry out sustained critical reflection on practical work within the wider context of industry/workplace. Fully meets expectations set by the industry/employment context. | | |
| **Good use of OO practices in C++ including operator overloading and general language usage**  30% |
| **Use of Standard Template Library**  20% |
| **Dynamic memory management and ownership without memory leaks**  20% |
| **Clean concise code, good use of comments, spacing, constants, source files, naming and labelling and code structure without unnecessary repetition**  10% |

1. This will of course depend on your level of skill, experience, knowledge, and engagement in the module. Do not leave this until the last minute and start early so you don’t run out of time since this is not a 10-minute task and you are done. [↑](#footnote-ref-1)
2. See [Yahtzee - Wikipedia](https://en.wikipedia.org/wiki/Yahtzee) for game play and rules [↑](#footnote-ref-2)
3. Encryption can be as simple as adding / subtracting one from the ASCII value. This feature is there to guide you towards thinking about encapsulation. But go as wild as you like with your encryption algorithm… just make sure to use encapsulation [↑](#footnote-ref-3)
4. Use a different filename for each player but think about how to name these files uniquely [↑](#footnote-ref-4)