****

**Faculty of Science and Technology**

**2022/2023**

**Level 4**

**Introduction To Programming**

**Assignment 2**

**“BlackJack” Program**

**Analysis, Design, and Implementation**

**ITP Report**

**Template**

**Word Limit [1000]\***

\*Word limit does not apply to diagrams, tables and annotations or source code

1. **Self-Assessment of Performance 5%**

**Tutor : Andrew Watson**

|  |  |
| --- | --- |
| **Student ID** | **S5520809** |

**Circle the appropriate response:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Did I submit the assignment on time?** | **Yes** | **No** |  |  |
| **Did I complete the assignment?** | **Yes** | **No** |  |  |
| **If not, approx. how much did I complete?** | **%** |  |  |  |
| **How happy am I with what I submitted?** | **Very happy** | **Satisfied** | **Disappointed** | **Ashamed** |
| **What mark do I expect?** | **90%** |  |  |  |
| **Did I spend enough time on the assignment?** | **Yes** | **No** |  |  |
| **Did I get it proof-read by someone else?** | **Yes** | **No** |  |  |
| **Have I properly ‘referenced’ it?** | **Yes** | **No** |  |  |
| **Could I improve the report presentation?** | **Yes** | **No** |  |  |

**Answer the following questions:**

|  |  |
| --- | --- |
| **The best part of my performance was:** | **Producing a fun, fully playable game** |
| **The worst part of my performance was:** | **Not completely completing planning out my program before beginning to write the actual code.** |
| **One way in which I could improve the content of my assignment is:** | **Conduct more testing and prevent as many bugs as possible** |
| **One way in which I could improve the presentation of my assignment is:** | **Could make the console outputs easier to understand and follow by clearing the screen after each user input** |
| **One thing I will do to improve my performance in my next assignment is:** | **Being much more thorough when planning** |

**2. Analysis : Marks 15%**

1. **Module Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Module** | **Input** | **Process** | **Output** | **Notes** |
| generateDeck | deck | For every index in the deck (52 in total), generate a card with a suit, card name and card value. |  |  |
| shuffle | deck | Perform 1000 swaps between 2 randomly chosen indexes of deck. To do this, a temporary variable is used to store the contents of the first random index. Replace the contents of that index with the value at the second random index. Then replace the contents of the second index with the contents of the variable. |  |  |
| getRandomCardPlayer | playerHand, playerExtraPoints | Loops through random indexes in deck until a card that hasn’t already been picked, is chosen. If the chosen card is an ace, the player is asked if they want it to be worth 1 or 11 points. If the player chooses 11, 10 is added to playerExtraPoints. | “You got an ace. Do you want it to be worth 1 or 11 points?: (1/11)” |  |
| getRandomCardDealer | dealerHand, dealerExtraPoints | Loops through random indexes in deck until a card that hasn’t already been picked, is chosen. If the chosen card is an ace, 10 is added to dealerExtraPoints. |  |  |
| displayHand | playerHand or dealerHand, holeCard | Loops through every card in the hand and outputs the name and suit of each. If the hand includes a hole card, “Hole Card” is outputted. | M\_cardName, “ of ”, m\_cardSuit, “Hole Card” |  |
| sumHand | playerHand or dealerHand, includeHoleCard | Loops through every card in the hand and adds their value to a local variable, sum.  If the variable includeHoleCard is true, the first index in the hand is ignored. |  |  |
| displayStatus |  | Outputs the status of the game at the end of each turn. It also calls the displayHand function and parses the player’s then dealer’s hands | “Money pot: ”, Moneypot, “Player bet: ” playerbet, “Player’s hand: ”, “Dealer’s hand: ”, “Player’s card total: ”, “Dealer’s card total (without hole card included): ” |  |

1. **Structure Chart**
2. **Top Level Flow Chart**

Graphical user interface, text, application, Word

Description automatically generated

Diagram, schematic

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated(If menuChoice == 1 is true)

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

**3. Design : Marks 15%**

1. Deck Creation and Management

Diagram, schematic

Description automatically generated

Diagram, schematic

Description automatically generated

A picture containing diagram

Description automatically generated

Chart, box and whisker chart

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Diagram

Description automatically generated

Text

Description automatically generated

Diagram

Description automatically generated

Text

Description automatically generated

Diagram

Description automatically generated

Text

Description automatically generated

Diagram

Description automatically generated

Text

Description automatically generated

Diagram

Description automatically generated

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

Text

Description automatically generated

1. Data Input testing and validation

**menuChoice**

Diagram, schematic

Description automatically generated

**playerBet**

Diagram

Description automatically generated

**playerChoice**

Diagram

Description automatically generated

**input**

Diagram, schematic

Description automatically generated

**4. Testing and Conclusions (Marks: 5%)**

**1.Testing and Debugging**

One of the useful things I learned throughout this project, was different approaches to debugging. I found that it was much easier to spot and fix bugs if I tested the program after making each small change, as opposed to making multiple changes to different parts of the code before conducting a test. Also, when debugging, I discovered the use of adding breakpoints and stepping through a program as they save time and make it easier to see what parts of the program need to be fixed.

One of the challenges I faced, was finding the best approach to handling aces. After a few tests, I found that the program would ask the player to enter a value for each ace they draw, multiple times. After debugging, I discovered this was because the code that asked for the player’s chosen ace value was being executed each time the program called the sumHand function. In order to only ask the player once for each ace value, I changed the program to ask the player for an ace’s value in the getRandomCardPlayer function. Then if the player chooses for an ace to be worth 11 points, 10 would be added to an extra points variable using pointers, in order to keep track of the user’s inputs.

Initially, I used 2D character arrays to store the playerHand, dealerHand and the deck. However, because I was limited to only being able to use the character data type to store the name, suit and value of each card, when testing, I encountered a lot of errors when the program would carry out mathematical operations using other integer values and the character card value variables in the array. To solve this, I ended up using a 1D array of card structures which allowed the card values to be stored as integers.

**2.Conclusions**

Having completed the program, I thought about what the challenges had been what I could have done better and what went well so that I would have a better approach moving forward to the next programs I develop.

When planning the program, I only completed the IPO diagrams before starting to write the actual program code without first also finishing the flow diagrams. I soon realised the importance of completing both however, as they really help with simplifying the program’s processes and understanding its overall structure.

Also, I didn’t thoroughly plan out every aspect of my program such as how I would handle deciding the point value when the player or dealer draws an ace. This led to me having to constantly update the whole program while I was testing out different solutions.

Learning about structures and pointers helped a lot with developing the program as they allowed me to do things I didn’t even know was possible like for example, using pointers and references to parse arrays and using structures to have multiple data types exist in arrays. They also helped simplify some processes, for example using pointers as an alternative to returning multiple variables from a function.

In conclusion, I managed to produce a game that generates a random deck of cards, deal the player and dealer initial hands, asks the player whether they want to draw more cards or stick with what they have, workout what the dealer’s moves should be and calculate the outcome of the game and allow the player to choose to play again. I am happy with the final product as it implements all the required functions and feels polished with regards to the code’s execution and with the look of the consol. If I had more time to develop this project, I would have spent more time testing for bugs to ensure the user experience is as optimised as possible.

**Appendices .References**

<https://www.youtube.com/watch?v=m2P5A4nR51g>

<https://stackoverflow.com/questions/46752965/c-blackjack-stuck-trying-to-program-ace> - for inspiration to use an ace count instead of changing the ace's card structure value