JackBlack

Background Information

Hot off his statistics-based flop *School of R*, Jack Black decided he needed to cool off. He hit up the casinos for a few games of cards. Failing to realize how little he learned from his movie, he lost game after game, but Jack was anything if not tenacious. He decided to take matters into his own hands. Struck by inspiration, Jack Black created his own card game. He called it: **JackBlack**. Your task will be to write a program that simulates the game he ultimately created. *Disclaimer: Any similarity to the actor known as Jack Black and his oeuvre of masterpieces is without question, 100%, definitely coincidental. For sure. Yes.*

Basic Information

In each round of JackBlack, multiple players play against one another. Each player's goal is to get the value of their collection of cards—called their **hand**—as close to the value **31** as possible. If a player's hand value goes over 31, they **bust**, meaning they immediately lose the round. The player with the highest hand value not exceeding 31 wins. Players holding hands with the same highest value tie the round.

JackBlack does not use a standard deck of 52 playing cards; instead, each deck varies in the number of cards it holds and contains three different kinds of cards. **Number cards** display a single number in the range 1 to 15. Each number card's value is as-is; in other words, a "5" card has a value of 5. **Word cards** display a single lowercase word with only letters; no whitespace, punctuation, or numbers. Each word card's value is determined by its consonants and vowels—each consonant is worth 1 and each vowel is worth 2. The total value is the sum of these consonants and vowels modulo 16, plus 1. In other words, a word card's value is at least 1 and at most 16. The word card "apples" has a value of 9 and "bureaucracies" has a value of 5. Note that 'y' is a consonant. **Symbol cards** display a single symbol and have associated low and high values. Whether the low or high value is "chosen" depends on the player's other cards. The value that a symbol card contributes to a hand is calculated to maximize the player's points without going bust. Moreover, all of the symbol cards in a hand must either use their high values or their low values. The program will be responsible for choosing the best value configuration per player hand. Note that if no possible configuration yields a value of 31 or less, then choose the low values configuration and consider the player busted.

At the beginning of each round, starting with whomever is considered the first player, each player is dealt one card from the top of the deck. Then, starting with the first player, each player decides whether to **hit**—take another card from the deck—or **stand**—keep their current hand for the round. Each player has a certain level of risk that they are willing to live with. In other words, each player has a **maximum hit limit**, between 1 and 30 inclusive, that defines the maximum value they can have in their hand that still allows them to hit again. A player keeps hitting until they bust or until they exceed their maximum hit value. Each player must complete this flow before the next player is reached. The round is over when all the players have been cycled through once. Note that a tie is not considered a win for the purposes of statistics.

Let's run through a single round for a single player, Brandonathan. Brandonathan has a maximum hit limit of 27. He starts with a 5 number card. He is below his maximum hit limit, so he hits, and receives a ♥ symbol card with a low of 6 and a high of 11. Here are the possible hand values:

```
5 + 11 = 16.
5 +  6 = 11.
```

The high value of the ♥ symbol card must automatically be used, because it gives Brandonathan a higher hand value, 16, that does not bust him. He is still below his maximum hit limit, so he hits again, and receives a "park" word card, with a value of 6. Here are the possible hand values now:

```
5 + 11 + 6 = 22.
5 +  6 + 6 = 17.
```

The high value of the ♥ symbol card must still automatically be used, because it gives Brandonathan a higher hand value, 22, that does not bust him. He is still below his maximum hit limit, so he hits again, and receives a ∏ symbol card with a low of 8 and a high of 12. Here are the possible hand values now:

```
5 + 11 + 6 + 12 = 34.
5 +  6 + 6 +  8 = 25.
```

The low values of the ♥ and ∏ symbol cards must automatically be used, because using the high values busts Brandonathan and using the low values gives Brandonathan a hand value, 25, that does not bust him. Notice that using a low value for the ♥ symbol card and the high value for the ∏ symbol card gives Brandonathan a great hand!

```
5 + 6 + 6 + 12 = 29.
```

However, in JackBlack, it is not allowed to mix and match the low/high values of the symbol cards in a hand. They must either all use their low values or all use their high values. He is still below his maximum hit limit, so he hits again, and receives an 8 number card. Here are the possible hand values now:

```
5 + 11 + 6 + 12 + 8 = 42.
5 +  6 + 6 +  8 + 8 = 33.
```

The low values of the ♥ and ∏ symbol cards must automatically be used, because using the high values busts Brandonathan and using the low values busts Brandonathan. Using the lows rather than the highs is simply a default.

The simulation ends in the round when the deck runs out of cards. If the deck runs out of cards in the middle of a round (when more cards are still needed), simply abort the round and end the simulation. At this point, the program shows interesting analytics about the game's players and rounds. For each player, you will show the number of hands won, lost, and tied. Afterwards, you will also show the average number of hits per player per round, the average size of a winning hand, and the average size of a losing hand. Where necessary, all decimal values should be rounded to two decimal places.

For 0.25 points of extra credit each, you can also print the following analytics information:

- The average number of number, word, and symbol cards appearing in winning hands.

- A list of the words appearing in winning hands, in the order that they appear.

Input File Formats

Your program will take in a file as input to the program that represents the game deck. The deck is comprised of any number of number, word, and symbol cards. Assume that the deck will always have enough cards to last more than one round. An example input deck file appears below. The first line in the file contains the total number of cards in the deck file. Each subsequent line starts with the type of card (N = number card, W = word card, and S = symbol card) and continues with that card's necessary information (the number, word, and low/high values respectively). Note that the top of the file is considered the bottom of the deck.

```
17
N 5
N 11
N 8
W school
W shallow
W tenacious
W demolition
S ♠ 6 16
S ♥ 7 17
S ♣ 8 18
S ♦ 9 19
N 15
W fidelity
W destiny
N 2
S ∑ 10 20
S ∏ 6 9
```

Your program will also take in a file as input to the program that represents the players configuration. Assume that there will be a minimum of two players and a maximum of ten players. An example player configuration file appears below. The first line is the number of players. Each subsequent line represents a player, containing a one word name and a maximum hit limit. The dealer is not considered special in JackBlack. The dealer is considered a regular player, and may or may not even appear in the configuration.

```
5
Dealer 20
Simon 30
Mary 26
James 22
Samantha 18
```

Program Startup

The program on startup takes up to two input files as arguments:

```
$ ./JackBlack <deck file name> <player file name>
```

For instance:

```
$ ./JackBlack deck1.data players1.data
Running simulation with players:
Dealer (Max Hit: 20)
JackBlack (Max Hit: 25)
Ann (Max Hit: 21)
Charles (Max Hit: 17)
(...normal program flow)
```

Two files will be provided by default: *defaultDeck.data* and *defaultPlayers.data*. If the program is supplied no deck file and/or no players file, use these default data files instead.

```
$ ./JackBlack deck1.data
No players file provided. Using default players file instead.
Running simulation with players:
Dealer (Max Hit: 20)
JackBlack (Max Hit: 25)
(...normal program flow)

$ ./JackBlack
No deck file provided. Using default deck file instead.
No players file provided. Using default players file instead.
Running simulation with players:
Dealer (Max Hit: 25)
JackBlack (Max Hit: 27)
(...normal program flow)
```

You may **not** assume that the files provided as arguments to your application exist. If a user supplies a file that does not exist, then you must print "`File does not exist: <invalid file name>`." You must **not** abort the program, but instead you should proceed with the respective default file.

```
$ ./JackBlack invalidDeck.data invalidPlayers.data
File does not exist: invalidDeck.data.
No deck file provided. Using default deck file instead.
File does not exist: invalidPlayers.data.
No players file provided. Using default players file instead.
Running simulation with players:
Dealer (Max Hit: 25)
JackBlack (Max Hit: 27)
(...normal program flow)
```

Output Format

Your program should output diagnostic information to the screen, including the introductory setup, the round by round results, and the final analysis. Note that the players always appear in the order specified in the players configuration file, and that the players' cards are shown in the order that they were received from the deck. Your output should conform to the following format:

```
Running simulation with players:
Dealer (Max Hit: 24)
Simon (Max Hit: 25)
Michael (Max Hit: 23)

ROUND 1
Dealer: 5, 15, 4, 3. TOTAL: 27.
Simon: apples (8), ♠ (6/16 --> 16), 8. TOTAL: 32 (BUST).
Michael: 7, 10, 5, 5. TOTAL: 28.
WINNER: Michael.

ROUND 2
Dealer: 5, 15, 4, 3. TOTAL: 27.
Simon: apples (8), ♠ (6/16 --> 16), 6. TOTAL: 30.
Michael: 7, 10, 5, 5. TOTAL: 28.
WINNER: Simon.

ROUND 3
Dealer: 5, 15, 4, 4. TOTAL: 28.
Simon: apples (8), ♠ (6/16 --> 16), 8. TOTAL: 32 (BUST).
Michael: 7, 10, 5, 5. TOTAL: 28.
TIED: Dealer, Michael.

FINAL TALLY
Dealer. WINS: 0. LOSSES: 2. TIES: 1.
Simon. WINS: 1. LOSSES: 2. TIES: 0.
Michael. WINS: 1. LOSSES: 1. TIES: 1.

GAME ANALYSIS
Average number of hits per player per round: 2.56.
Average number of cards in winning hands: 4.33.
Average number of cards in losing hands: 5.17.

Average number of number cards in winning hands: 1.43.
Average number of word cards in winning hands: 2.17.
Average number of symbol cards in winning hands: 3.10.
Words from winning hands: tropic, community, nacho, king.
```

Assignment Completion Phases

This assignment will be completed in two phases.

Program Design. You are responsible for looking through this specification and creating a design proposal outlining the classes you expect to have, the general functionality that these classes provide, the data members/structures that these classes require, and how your classes interact with one another. Be sure to note whether any classes have inheritance relationships. Your design explanation should feature little to no code. It should instead focus on the macro-elements of your proposed program. You may include UML diagrams. After I create your personal assignment repository, I will open up a **GitHub Issue** in the repository asking you to define the design of your project. You will give your design proposal there.

Program Implementation. After you submit your design proposal, I will give you feedback on your design choices. Moreover, I will give you a specific design to implement. You will not be implementing your proposed design. Keep in mind that the design given to you is perhaps not the "perfect" solution. Your design proposal may have been equally valid. Nevertheless, you should be able to implement any design given to you. You are encouraged to ask questions about why particular design decisions were made and how they compare to the decisions you made in your design proposal.

Provided Files

For this assignment, you are not given any starter code. You will be provided some sample deck and players input files that can be used to test your program, and the default deck and players input files. You will also be given an executable file that is a working solution based on this specification. Because the executable's solution code was compiled on the Linux lab machines, the executable can only be run on the Linux machines with confidence. Running it on any other system may yield undefined behavior.

Things to Think About

From an architectural perspective, there is no one perfect way to do this assignment. Nevertheless, good design choices can make your solution modular, extensible, efficient, and understandable. Endeavor to strike a balance that makes sense. Moreover, JackBlack, as a game, poses some interesting software challenges of its own. Think about the following:

- What classes do you need to implement all of the required functionality? Is there inheritance? Which classes interact with the others, and how? Understand has-a vs. is-a relationships.
- Which data structures do you need to make use of? You have many data structure at your disposal now: vectors, lists, stacks, and queues. You can also use UniqueVector! Be sure to choose them carefully. Justify your choices in your program design proposal and, ultimately, your README.
- Look at the STL documentation! Learn the interfaces of the data structures. Learn about how to use iterators to loop through a data structure's elements. There's a huge amount that the STL allows you to do. You just need to ask, explore, and research.
- Which classes act merely as basic "data containers" and which classes are responsible for the logic of the program runtime? Remember that a single class should handle a single logical set of responsibilities. Don't make any one class do too much heavy lifting. Be modular.
- Don't forget about memory management! Be sure to decide who has the responsibility of deallocating any dynamically allocated pointers.
- Overload the istream and ostream operators! It makes stream reading and writing flexible.

- Remember that *your* output format and style should match the format and style shown above EXACTLY in order for you to receive full credit. Be sure to test your running program alongside the provided executable to make sure that yours conforms. Take note of the whitespace!
- When a player adds a symbol card to their hand, why is that insertion step the key to determining whether the low or high value(s) should be used?
- Symbol cards definitely share functionality with other cards, but they are also quite different. If/when you use inheritance, it may be necessary to use dynamic casting for symbol cards.

Submission Details

Your submission must include all of the header/source files (*.h/*.cpp) required for your program to properly compile and run. You must include a makefile that includes the sources to be compiled. The name of the generated executable must be **JackBlack**. Feel free to adapt the makefile provided in Assignment #1. Do not submit any generated executables or object files (*.o). You must also update the README file with any guidance that someone looking at your project needs to and might like to know; this could include compilation instructions, interface specifications, interactions between classes, problems overcome, etc. Confirm that your code successfully compiles and runs on the Linux lab machines. See the **Assignment Guide Using Git & GitHub** for step-by-step information about how to submit your assignment via git and GitHub.

Grading

This assignment is worth 15 points of your final grade, and is broken down as follows:

Components	Percentage	Relevant Questions
Correctness	55%	Does our program's output match the expected output based on the input files and specification?
Design	20%	Does your program design proposal separate out logical units into classes and employ data structures that make sense and/or solve your problems efficiently? Is your ultimate solution modular and efficient?
Documentation	15%	Does your README document provide users with adequate information about your program? Do you adequately comment your class interface files and class implementation files (when necessary)?
Style	10%	Is your coding style readable and consistent? Do you follow (to the best of your ability) the modified Google Style Guide?

If your program does not compile on the Linux machines, you will receive no points for the **Correctness** component. If you do not submit your design proposal before its deadline, you will receive half a point off of the **Design** component.

Due Dates

There are two due dates for this assignment. The due date for the program design proposal is **October 18th by 11:59pm**. I will not look at any design proposals submitted after this date. The earlier you submit your proposal, the earlier you will receive feedback from me. If you start coding before I give you the design to use, your implementation may suffer as a result. The due date for the implementation portion of the assignment is **November 6th by 11:59pm**. These are hard deadlines, meaning that there will be little leniency with regard to lateness. For every day that you miss the deadline for the implementation portion, I will deduct 10% from the project's final grade. A sample solution will be provided some time after the deadline.

Final Words

This assignment might seem overwhelming! It may feel like the whole Republic of CSCI 235 is conspiring against you. Don't feel alone in this endeavor, my padawans. Feel free to discuss design with your fellow students. But please don't be a Rebel; plagiarism is not acceptable. Start early. Good luck, and may the Force be with you.