Programming Project: TigerGame

CPSC 1020 – Spring 2024 – version 3/23/24

School of COMPUTING Clemson University

Introduction

For this project, you will implement a new card game called "TigerGame". In TigerGame, there are three types of cards: purple, orange, black.

Each color has a varying number of cards ranked from 1 to 15, for a total of 30 cards in one deck:

- 15 purple cards ranked 1-15
- 10 orange cards ranked 1-10
- 5 black cards ranked 6-10

Each card has a value:

The value of purple cards is equal to their rank
(e.g., purple:2 is worth 2 points)

The value of orange cards is equal to twice their rank
(e.g., orange:2 is worth 4 points)

The value of black cards is equal triple their rank
(e.g., black:7 is worth 21 points).

TigerGame is a computer game in which a player competes against the computer. The game is played as follows:

- There are two players, *computer* and *human*.
- Each player draws a hand of 6 cards from the full deck, which has been shuffled before cards are drawn.
- There are SIX rounds. Each round is played as follows:
  1. *Computer* moves first by selecting a card from its hand. For simplicity, this is implemented as a random draw. *Human* sees the card *computer* has played.
  2. *Human* selects a card from their hand to play against the *computer*'s card.
  3. The player whose card has the larger value wins the round. Any card can be played against any other card. Only their value determines who wins the round.

4. The player who wins the round is awarded points equal to the sum of the values of the two cards. For example, if *computer* plays purple:2 and *human* plays orange:4, *human* wins the round and receives 10 points.
5. If there is a tie, no player receives points.

- After six rounds, the player with the most points wins.

Template Files

A complete set of template files is uploaded to Canvas, including a main.cpp file and a Makefile. While most of the code is missing from these files (your job is to implement the missing code), the program can already be compiled with `make`. Start with these template files and **incrementally** add code. Frequently compile your program with `make` to check that your code compiles without errors or warnings!

Class Specifications

You must implement the following classes to build TigerGame. Build them in the order in which they are listed.

Card

Each card in TigerGame is represented as an object of class `Card`. This class has three private member variables: `rank`, `color`, and `value`, which is calculated as explained above. Use in-class initialization to initialize `rank` to 0, `color` to purple, and `value` to 0. Note that `color` is an enumerated type of type `Color`, which is added as a public member to `Card`.

`Card` has a parameterized constructor, which you will use to create an instance of each card in TigerGame (e.g., an orange card with rank 2). This constructor should initialize `rank` and `color` with the arguments provided *and* set `value` to its correct value given `rank` and `color`.

`Card` also has a function `strCard()` which must return a string in the following format:
`color:rank`. For example, for an orange card of rank 4 , `strCard()` should return `orange:4`.

The remaining three functions are getter functions to retrieve the values of the private member variables.

Card

- rank: int - color: Color - value: int + enum Color {purple, orange, black}
+ Card() + Card(rank: int, color: Color) + strCard(): string + getRank(): int + getColor(): Color + getValue(): int

Deck

A TigerGame consists of a deck of 30 cards: 15 purple cards ranked 1-15 and 10 orange cards ranked 1-10 and 5 black cards ranked 6-10. You will use class `Deck` to represent this deck of cards.

`Deck` has a single private variable `deck`, which is a vector of type `Card`. This vector holds all 30 cards used in TigerGame.

`Deck` has a default constructor that you need to implement. The purpose of the constructor is to create vector `deck` with all 30 cards when an object of `Deck` is instantiated. Once implemented, you can create the deck of cards for a TigerGame with `Deck deckOfCards;` at the beginning of your main program.

`Deck` has two member functions that allow manipulating the deck of cards in the game. Function `shuffle()` randomizes the order of the `Card` objects in vector `deck`. Once implemented, you can use `deckOfCards.shuffle();` in your main program to shuffle the deck. FOR TESTING PURPUSES USE A SEED VALUE OF 0 (zero).

Function `drawCard()` is used to draw a card from the deck. Implement `drawCard()` so that it returns the first card from the deck (its return type is `card`) and then removes that card from the deck.

Function `getDeckSize()` returns the current number of cards in the deck.

Deck
- deck: vector of type `Card`

+ Deck() + shuffle(): void + drawCard(): Card + getDeckSize(): int

.

Hand

Each player's hand of cards is implemented as an object of class `Hand` in TigerGame. `Hand` has a single private variable `hand`, which is a vector of type `Card`. This vector holds all N cards a player draws from the deck.

`Hand` has a parameterized constructor, which you will use to deal a hand of cards from the deck. Implement this constructor so that it draws N cards from the deck when an instance of `Hand` is instantiated.

`Hand` has three member functions, `strHand`, `dealCard`, and `getHandSize`. Use `strHand()` to show *human* their current hand. This function must return a string formatted as follows:

```
[1] purple:5 [2] orange:7 [3] orange:2 [4] purple:10 [5] orange:9 [6] black:6
```

where the numbers in brackets represent the position of each card in the hand (i.e., in vector
`hand`).

The second member function in Hand is `dealCard`, which has a parameter `num` that represents the position of a card in the hand. Use this function to let *human* deal a card from their hand. When a card is dealt, it should be removed from `hand`. For the above example, `dealCard(2)` would deal card `orange:7` and remove it from the hand.

Function `getHandSize()` returns the current number of cards in the hand.

Hand
- hand: vector of type `Card`
+ Hand() + Hand(deck: Deck, N: int) + strHand(): string
+ dealCard(num: int): Card + getHandSize(): int

Player

Each player, *computer* and *human*, will be represented by an object of class `Player` in TigerGame. This class has two public member variables: an object of type `Hand` and an integer `score`. Member `hand` is an object of class `Hand` because each player has a hand of cards. Use `score` to keep track of a player's current score. Initialize this variable to 0 using in-class initialization.

`Player` has a parameterized constructor, which you will use to initialize `hand`. When called, this constructor will create an instance of `Player` with N cards in their hand. Note that this constructor is similar to the `Hand` constructor.

Player
+ hand: Hand + score: int
+ Player() + Player(deck: Deck, N: int)

Note: Class Player only needs a header file with in-line functions. The class is simple enough to implement everything in Player.h instead of using a separate .cpp file.

Main Program

Once you have implemented all of the above classes, you can create TigerGame. Proceed as follows to build your game:

1. Create a deck of cards and shuffle it.
2. Create two players, each one with cards in their hand.
3. Play five rounds. In each round:
    - Have *computer* deal a card from their hand.
    - Show *human* their hand of cards so that they can make a selection.
    - Have *human* deal their card.
    - Determine who won the round and update points accordingly.

Make sure to implement **input validation** where necessary. For example, when *human* only has 3 cards left on their hand, selecting card number 4 should prompt the user to select a valid card.

Sample Run

Here is a sample run of TigerGames. **Your implementation can look different**. That means you can be creative in how you design your game, but you must follow the above class specifications, e.g. `strCard()` must return a string format as `color:rank`.

```
Welcome to TigerGame!
The deck was shuffled and each player has drawn 6 cards.

Round 1
-------

The computer plays: purple:8
Your hand: [1] orange:8 [2] orange:1 [3] purple:3 [4] orange:4 [5] purple:8 [6] black:7
Which card do you want to play? 4
You played: orange:4
Tie!

Current scores:
Human: 0
Computer: 0

Round 2
-------

The computer plays: orange:6
Your hand: [1] orange:8 [2] orange:1 [3] purple:3 [4] purple:8 [5] black:7
Which card do you want to play? 1
You played: orange:8
You win this round!

Current scores:
Human: 28
Computer: 0

Round 3
-------

The computer plays: orange:2
Your hand: [1] orange:1 [2] purple:3 [3] purple:8 [4] black:7
Which card do you want to play? 3
You played: purple:8
You win this round!

Current scores:
Human: 40
Computer: 0

Round 4
-------

The computer plays: purple:5
Your hand: [1] orange:1 [2] purple:3 [3] black:7
Which card do you want to play? 1
You played: orange:1
The computer wins this round!

Current scores:
Human: 40
Computer: 7

Round 5
-------

The computer plays: purple:7

Your hand: [1] purple:3 [2] black:7
Which card do you want to play? 1
You played: purple:3
The computer wins this round!

Current scores:
Human: 40
Computer: 17

Round 6
-------

The computer plays: black:9
```

```
Your hand: [1] black:7
Which card do you want to play? 1
You played: black:7
The computer wins this round!

Current scores:
Human: 40
Computer: 65


FINAL SCORE:
Human: 40
Computer: 65

Computer has won!
```

Extra Credits

Extra credits will be awarded for implementing the following features (from easiest to hardest):

- Instead of playing a fixed number of 6 rounds, ask the user how many cards each player should draw from the deck (which determines how many rounds will be played). Make sure the player's don't pick more cards than are on the deck.
- TigerGame is an addictive game! Ask the player if they would like to play again at the end of a game so that they can play another round of TigerGame. Once the player exits the game, print a summary of all games that includes how many rounds were played, how many rounds each player won, and how many points each player received in each round and in total.
- Add a special card "Tiger" with rank 10 to the deck. If a player plays the Tiger card, the game finishes immediately. The player who played the Tiger card receives an additional 50 points; no other points are awarded for the round.

Submission Instructions

Submit your complete and correct program to Gradescope. Your submission must include the following 9 files:

1. main.cpp
2. Card.h
3. Card.cpp
4. Deck.h
5. Deck.cpp
6. Hand.h
7. Hand.cpp
8. Player.h
9. Makefile

- Make sure your program compiles and runs on the SoC Linux machines; the autograder automatically assigns 0 points to programs that do not compile.
- Code style (proper indentation, consistency, readability, use of comments, etc.) will be considered when grading your submission.

Collaboration Policy: This project is an individual programming assignment. All work you submit must be your own individual work. You may identify errors or ask about ambiguous specifications in a programming assignment on MS Teams, but your are not allowed to complete an assignment with the help from peers or outside tutors, and you

are not allowed to share code with others or copy someone else's code or code from the internet. Please make sure to carefully read the Academic Integrity Statement in the syllabus, which explains what is considered cheating in this class.

Late Policy: Work submitted late will be subject to a 10 percentage point deduction on the 0-100 scale (i.e., one letter grade) per day.

Gradescope Submission: Code must be submitted via Gradescope. Please note that the Gradescope "autograder" is designed to give you feedback on the correctness on your submission. This includes checking if all necessary files are submitted, if the unit tests used for autograding compile, and if the core functionality of your submission is correct (e.g., default constructor, argument constructor, some of the member functions). The points displayed by the autograder when you submit your program do NOT correspond to your final grade for this assignment. Additional tests might be added to the autograder after the submission deadline, and parts of the program will be manually graded.