Prof. Dr. Visvanathan Ramesh
Alperen Kantarcı
Gamze Akyol

Goethe-Universität Frankfurt am Main
Institut für Informatik

# Programmierparadigmen und Compilerbau

Sommersemester 2023

## Exercise Nr. 2

Deadline Wednesday 24.05.2023, 10:00 in Moodle!

## Question 0 (0 Points)

Read the following tips!

- Group work is **not allowed!** You can talk and discuss with your friends about the exercises, but you should solve the exercises **individually**. Plagiarism is not tolerated.
- You can use ChatGPT or other web sources but you cannot ask the whole question. You should be able to explain every step of your code.
- Upload all the files you want to submit into **a single .zip file on Moodle**. Please ensure that you **only submit .pdf and .hs files.**
- Please write your **matriculation number**, **name** and **group number** to each file that you are submitting.
- Your submission file name should be in the following order:

  Name_matriculationNumber_Group_GroupNumber_Exercise_ExerciseNumber.extension.

  For example **AlperenKantarci_1111111_Group_2_Exercise_2.zip**

## Question 1 - Custom Solitaire (20 Points)

This problem involves a customized card game adapted from Prof. Dan Grossman's course. You will write a program that tracks the progress of a game. A game is played with a *card-list* and a *goal*. The player stars with a list of *held-cards*, initially empty. The player makes a move by either drawing, which means removing the first card in the card-list and adding it to the held-cards, or discarding, which means choosing one of the held-cards to remove. The game ends either when the player chooses to make no more moves or when the sum of the values of the held-cards is greater than the goal.

The objective is to end the game with the lowest score (0 is best). For scoring, the sum of the held-card values is used. If the sum is greater than the goal, the preliminary score is $3*(sum-goal)$, else, the preliminary score is $(goal-sum)$. The score is the preliminary score unless all the held-cards are the same color, in which case the score is the preliminary score divided by 2 (and rounded down as usual with integer division). The types are defined as follows and given in the main.hs file:

```
data Color = Red | Black
data Suit = Clubs | Diamonds | Hearts | Spades
data Rank = Num Int | Jack | Queen | King | Ace
data Card = Card { suit :: Suit, rank :: Rank }
data Move = Draw | Discard Card
```

Inside the main.hs file you have given the main function, and necessary functions to read inputs from the user. Your written program should be compatible with the given functions. You should also focus on understanding the concepts behind each function. You will implement the remaining functions inside the main.hs file as instructed below:

**Data Types & Pattern Matching**

1. Write a function cardColor, which takes a card and returns its color (spades and clubs are black, diamonds and hearts are red). **1 pts**

2. Write a function cardValue, which takes a card and returns its value (numbered cards have their number as the value, aces are 11, and everything else is 10). **1 pts**

3. Define a type for representing the state of the game. (What items make up the state of the game?) **2 pts**

4. Write a function convertSuit that takes a character c and returns the corresponding suit that starts with that letter. For example, if c is 'd' or 'D', it should return Diamonds. If the suit is unknown, raise an error. **1 pts**

5. Write a function convertRank that takes a character c and returns the corresponding rank. For face cards, use the first letter, for "Ace" use '1' and for 10 use 't' (or 'T'). You can use the isDigit and digitToInt functions from the Data.Char module. If the rank is unknown, raise an error. **1 pts**

6. Write a function convertCard that takes a suitname(char) and a rankname(char), and returns a card. **1 pts**

7. Write a function convertMove that takes a movename(char), a suitname(char), and a rankname(char)and returns a move. If the move name is 'd' or 'D' it should return Draw and ignore the other parameters. If the move is 'r' or 'R', it should return Discard c where c is the card created from the suit name and the rank name. **1 pts**

**Lists & Recursion**

8. Write a function removeCard, which takes a list of cards cs, and a card c. It returns a list that has all the elements of cs except c. If c is in the list more than once, remove only the first one. If c is not in the list, raise an error. **1 pts**

9. Write a function allSameColor, which takes a list of cards and returns True if all the cards in the list are the same color and False otherwise. **2 pts**

10. Write a function sumCards, which takes a list of cards and returns the sum of their values. Use a locally defined helper function that is tail recursive. **2 pts**

11. Write a function score, which takes a card list (the held-cards) and an int (the goal) and computes the score as described above. **2 pts**

12. Write a function runGame that takes a card list (the card-list), a move list (what the player "does" at each point), and an int (the goal) and returns the score at the end of the game after processing (some or all of) the moves in the move list in order. Use a locally defined recursive helper function that takes the current state of the game as a parameter: **5 pts**

    - The game starts with the held-cards being the empty list.

- The game ends if there are no more moves. (The player chose to stop since the move list is empty.)
- If the player discards some card c, play continues (i.e., make a recursive call) with the held-cards not having c and the card-list unchanged. If c is not in the held-cards, raise an error.
- The game is over if the player draws and the card-list is empty. Else if drawing causes the sum of the held- cards to exceed the goal, the game is over. Else play continues with a larger held-cards and a smaller card- list.

After implementing these functions, running the main function should give you similar outputs as below:

```
ghci> main
Enter cards:
c1
s1
c1
s1
.
Enter moves:
d
d
d
d
d
.
Enter goal:
42
Score: 3
```

Figure 1: Example of a successful run

```
ghci> main
Enter cards:
cj
s8
.
Enter moves:
d
rhj
.
Enter goal:
42
Score: *** Exception: Card not found in list
```

Figure 2: Example of an error