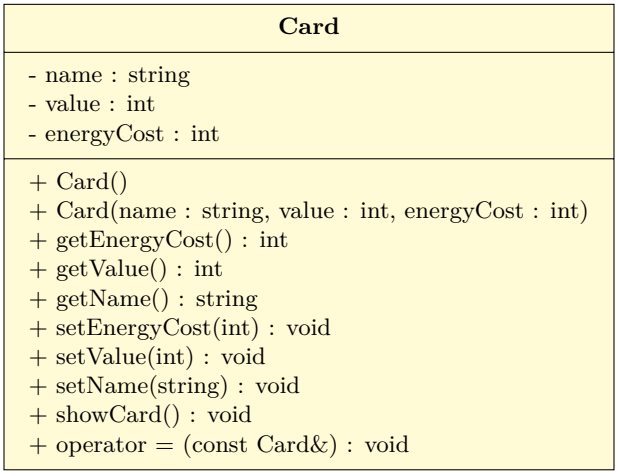
Assignment 1
CS 135 Review, Classes & Data Abstraction

CS 202 - Summer 2022

Introduction

In this assignment we will be reviewing concepts learned in CS 135 and implementing a class with a default constructor, parameterized constructor, accessors and mutators. We will be implementing a simple card class which contains the name, cost and energy value of the card. We will also be implementing a function to sort these cards.

UML Diagram



Functions and Class Details

All functions, classes and important variables used in the assignment are described below. Variables are in green. Functions that are to be implemented as part of this assignment are given in red., Functions that have been implemented for you are in blue.

Class : SortOrder

This is a global enumerator to define a sort order for the sortCards function. There are only two values defined here:

- **VALUE** - Used for sort by value.
- **ENERGYCOST** - Used for sort by energy cost.

Class : Card

This is the main class that defines a Card. The variables and functions in this class are given below.

- **string name** - Name of the card.
- **int value** - Purchase cost of the card.
- **int energyCost** - Energy required to play the card.
- **Card()** - Default constructor. This has been implemented for you and will set name to an empty string, the value to 0 and the energyCost to 0.
- **Card(string name, int value, int energyCost)** - Parameterized constructor. This function validates and then initializes the name, value and energyCost of a Card object. The name of the card must be less than 20 characters long. If the provided parameter is greater than 20 characters, then this variable is initialized to an empty string. The value of the card must be greater than 0 and less than 8. For parameter greater than 8, initialize value to 8, and for parameter less than 0, initialize value to 0. The energyCost of the Card must be between 0 and 5. For parameter less than 0, initialize energyCost to 0, and for parameter greater than 5, initialize energyCost to 5.
- **int getEnergyCost()** - Inline accessor (getter). It returns the energyCost of the object.
- **int getValue()** - Inline accessor (getter). It returns the value of the object.
- **string getName()** - Inline accessor (getter). It returns the name of the object.
- **void setEnergyCost(int)** - Setter for energyCost. Checks if provided parameter is between 0 and 5 and then sets the value of energyCost. Values less than 0 are set to 0, and values greater than 5 are set to 5.

- **void setValue(int)** - Setter for value. Checks if provided parameter is between 0 and 8 and then sets the value of value. Values less than 0 are set to 0, and values greater than 8 are set to 8.
- **void setName(string)** - Setter for name. Checks if the provided parameter is less than 20 characters long. If it is, then sets the name to the provided value. Otherwise, it is set to an empty string, and an error is printed to the `cerr` stream: "Name can't be longer than 20 characters.\n". Note that no error must be printed for the parameterized constructor because constructors are not supposed to print anything.
- **void showCard()** - Prints the card to the terminal.
- **void operator = (const Card&)** - Overloads the assignment (=) operator to allow deep copy of card objects from R-value to L-value. This has been provided to make swapping objects easier when implementing the sortCards function. *Note: Function and operator overloading will be covered in a future class.*

Global Functions

- **void sortCards(Card *cards, int size, SortBy order)** - This function sorts an array of cards in the order defined by `SortOrder`. Here, `cards` is the array to be sorted and `size` is the size of the array.
- **int main(int argc, char* argv[])** - Partially implemented. Code for reading the `cards.txt` file needs to be written. Note that the input file must be given as an argument to the program. No hard-coding is allowed. Program must return -1 for errors.

Cards File

The provided `cards.txt` file needs to be read into the `cards` array in the main function. The format for the given file is as follows:

```
CardName Value EnergyCost
```

Each line contains a new card. For example, the first two lines in the given file are:

```
Strike 0 1
Defend 0 1
```

Here, `Strike` is the name of the card with 0 and 1 being the value and energy cost respectively. The next line contains the values for another card called `Defend`. All lines in the file must be read by the program. You may assume that the file provided to the program will be formatted correctly with no errors and will always contain at most 10 cards.

Sample Output

This is a sample run of the program with the given `cards.txt` file.

```
piyush@Piyush-Desktop:/.../ast1/solution$ g++ main.cpp
piyush@Piyush-Desktop:/.../ast1/solution$ ./a.out
ERROR: Incorrect usage.
./a.out <filename>
piyush@Piyush-Desktop:/.../ast1/solution$ ./a.out fakefile.txt
ERROR: Could not open file: fakefile.txt
piyush@Piyush-Desktop:/.../ast1/solution$ ./a.out cards.txt
Strike               :: VAL 0  ENG 1
Defend               :: VAL 0  ENG 1
Poison               :: VAL 2  ENG 2
DoubleStrike         :: VAL 2  ENG 3
Ennervate            :: VAL 1  ENG 2
Finisher             :: VAL 2  ENG 5
Chip                 :: VAL 1  ENG 1

Sorting by Value...
Strike               :: VAL 0  ENG 1
Defend               :: VAL 0  ENG 1
Ennervate            :: VAL 1  ENG 2
Chip                 :: VAL 1  ENG 1
Poison               :: VAL 2  ENG 2
DoubleStrike         :: VAL 2  ENG 3
Finisher             :: VAL 2  ENG 5

Sorting by Energy Cost...
Strike               :: VAL 0  ENG 1
Defend               :: VAL 0  ENG 1
Chip                 :: VAL 1  ENG 1
Ennervate            :: VAL 1  ENG 2
Poison               :: VAL 2  ENG 2
DoubleStrike         :: VAL 2  ENG 3
Finisher             :: VAL 2  ENG 5
piyush@Piyush-Desktop:/.../ast1/solution$
```