CS 247 Spring 2014
Assignment 3 Specifications

Q1) [70 marks] Design Patterns

The Design Patterns page on the course Web page includes a number of example applications of various Design Patterns. You will help to populate this page by creating a **new** example application of your choice, using of **one** of the following Design Patterns:

i. Strategy Pattern
ii. Template Pattern
iii. Adaptor Pattern
iv. Facade Pattern
v. Iterator Pattern (iterates over a simple linear collection)
vi. Simple Factory
vii. [15 bonus marks] Factory Method Pattern
viii. [15 bonus marks] Decorator Pattern
ix. [15 bonus marks] Composite Iterator Pattern (iterates over a Composite)

Your answer to this question includes the following:

a) State which Design Pattern you are implementing, and provide a short description of the problem to which you are applying the Design Pattern.

b) Provide a UML model of the design pattern as it applies to your example application. Highlight the model elements (i.e., classes, associations, attributes, operations) that are introduced to the model *because* the design pattern is being used.

c) Implement your example application in C++. Provide a simple main program that demonstrates how a client programmer would use the resulting classes in the model. Provide a Makefile that builds the executable of your example program, where the name of the executable is `exec`. The TAs will execute your program.

Correct solutions to this question might be posted in the future to the Design Patterns page on the course Web page.

Q2) [20 marks] STL

The Straights project includes a number of situations where STL containers and algorithms could be used. For each of the following situations, (1) provide the declaration of an STL container holding the relevant data, and (2) provide a code fragment that uses an STL algorithm (plus any necessary function or function-object declarations) to implement the desired functionality. Assume that the code operates on collections of the provided Card class.

a) Initialize the deck of cards so that cards are in the following order:
AC 2C 3C ... QC KC AD 2D ... QD KD AH 2H ... QH KH AS 2S ... QS KS

b) Find the first playable card in a hand of cards. Assume the existence of a nonmember function `bool isLegal(const Table&, const Card&)` that determines whether a card is a legal play given a Table of existing straights.

c) Increment the players' scores at the end of a round. Assume that the Player class has a member function `void Player::incScore()` that increments a player's score with the points from the player's discard pile.

Q3) [60 marks] Generic Programming

You will use C++ STL containers and algorithms to implement a Hangman program. Hangman is a game in which the player tries to guess what word has been selected from a pool of words. The player guesses letters that he or she thinks might be in the word, and the game displays where the guessed letters appear in the word. If the player correctly guesses the entire word before making five incorrect guesses about the word's letters, the player wins.

0. Command Line Parameter

To start the program, the user provides a file of words from which a gameword is selected; the user can also provide an optional integer argument, like this:

```
./hangman words 44
```

where `words` is a file of words. The integer argument is used to seed the pseudo-random selection of gamewords. At the beginning of your program, call the `srand48` function (see `man srand48`) with the provided argument. If there is no second argument, assume that it is 0. Games that are started with the same argument value have the same sequences of selected gamewords.

1. Pool of gamewords

The first command-line argument is a file of words separated by whitespace. Read the words into an STL container of your choice. The input file might include words that are unsuitable for the game. Remove from the container any word that

- contains a nonalphabetic character
- has fewer than 6 characters

The relative order of the words that remain in the container should be preserved. Output the contents of the modified container, one word per line, to a file named `gamewords`. The last word in the file should be followed by a newline character.

2. Game Start

The first game starts after the pool of gamewords has been processed and output to a file. Use `lrand48()` modulo the number of words in the pool of gamewords to randomly select the game's gameword. Print the following four lines:

```
Word:<sp><gameword><newline>
Letters used:<newline>
You have 5 lives left.<newline>
Next guess:<sp>
```

`<gameword>` is the gameword, but with all of its letters replaced by hyphens`-`. Thus if the gameword is 7 characters long, then 7 hyphens are printed. `<sp>` is a single blank space and `<newline>` is a newline character. The player starts with 5 lives.

3. Play

Whenever it is the player's turn, the player can either guess a letter that he or she thinks is in the gameword or can try to guess the whole word.

a) Guess a letter

If the guessed letter has not previously been guessed during this game, then print an update to the state of the game:

```
Word:<sp><gameword><newline>
Letters used:<sp><list of letters><newline>
You have <num> lives left.<newline>
Next guess:<sp>
```

The differences between the above four lines and the lines on the previous page are in **bold** font. `<gameword>` is the gameword except that each letter not yet guessed is replaced by a hyphen. `<list of letters>` are the letters that have been been guessed within this game, including the letter guessed in this turn. The letters are printed in lowercase, separated by a single space, and printed in the order in which they have been guessed. If a character other than an alphabetic letter is guessed, it is treated as a guessed letter. `<num>` is the number of lives that the player has left. The player loses a life every time he or she guesses a letter that is not in the gameword. (If the player has 1 life left, then `"1 life"` is printed rather than `"<num> lives"`.)

If the guessed letter is the last unguessed letter that completes the gameword, then print

```
You WIN!<sp><sp>The word was "<gameword>".<newline>
Do you want to play again? [Y/N]<sp>
```

where `<gameword>` is the gameword with all of its letters revealed. If the player's response starts with "`y`" or "`Y`", then start a new game with a new randomly selected gameword. Otherwise, quit.

Guesses of letters are NOT case sensitive, so if letter 'A' or 'a' is guessed, it matches all instances of 'A' and 'a' in the gameword. As an example, if the gameword is '`African`' and 'a', 'r', and 'e' have been guessed in that order, then the updated state of the game is:

```
Word:<sp>A-r--a-<newline>
Letters used:<sp>a<sp>r<sp>e<newline>
You have 4 lives left.<newline>
Next guess:<sp>
```

If the guessed letter has previously been guessed, then print an error message, followed by the (unchanged) state of the game:

```
You already guessed letter "a".<newline>
Word:<sp>A-r--a-<newline>
Letters used:<sp>a<sp>r<sp>e<newline>
You have 4 lives left.<newline>
Next guess:<sp>
```

b) Guess a word

If the guess is longer than a single character, it is treated as a guess of the entire gameword. If the guessed word exactly matches the gameword (where this guess IS case sensitive), then print

```
You WIN!<newline>
Do you want to play again? [Y/N]<sp>
```

Otherwise, print

```
You LOSE!<sp><sp>The word was "<gameword>".<newline>
Do you want to play again? [Y/N]<sp>
```

where `<gameword>` is the gameword with all of its letters revealed. If the player's response starts with "`y`" or "`Y`", then start a new game with a new randomly selected gameword. Otherwise, quit.

4) Robustness

If no command-line arguments are provided, print the error message:

```
Error: No input file specified.
```

If the program cannot find or open the file specified by the first command-line argument `<arg>`, print the error message:

```
Error: Could not open file "<arg>".
```

If the pool of gamewords is empty (either from the start or after removing all unsuitable words), print the error message:

```
Error: Pool of game words is empty.
```

5) Design Requirements

Your solution must use STL containers and algorithms to perform much of its functionality. Specifically, it must use an **STL container** of your choice to store the pool of gamewords, and it must use **STL container operations** or **STL algorithms** (possibly with predicates and function objects) to

- Read words from the input file into the STL container
- Remove unsuitable words from the container
- Output the modified contents of the container to the file named `gamewords`
- Find guessed letters in the gameword
- Test whether the guessed letter completes the gameword
- Print the gameword (with unguessed letters replaced by hyphens)

Hint: If there is a loop in your program (other than the main loop that encodes a player's turn), there is a good chance that it should be replaced with an STL algorithm.