**CSCI3180 – Principles of Programming Languages – Spring 2018**

**Assignment 3 — Dynamic Scoping and Perl**

**Deadline: Apr 8, 2018 (Sunday) 23:59**

# 1 Introduction

The purpose of this assignment is to offer you a first experience with Perl, which supports both dynamic scoping and static scoping. Our main focus is dynamic scoping. The assignment consists of two parts.

First, you are required to implement a popular card game called "Golden Hook Fishing" in Perl. Second, you are required to implement a system called "GPU Management Server" in Perl with dynamic scoping. For both tasks, the detailed OO designs are given. In the process, you will learn both the programming flexibility and bad readability of codes written with dynamic scoping.

All your implementation in Perl should be able to run with Perl v5.18.2, Perl 5, version 18, subversion 2. Besides, you are required to add *"use warnings;"* and *"use strict;"* at the head of your program.

**IMPORTANT:** All your codes will be graded on the Linux machines in the Department. You are welcome to develop your codes on and platform, but please remember to test them on Department's machines.

**NO PLAGIARSM!!!** You are free to design your own algorithm and code your own implementation, but you should not "steal" codes from your classmates. If you use an algorithm or code snippet that is publicly available or use codes from your classmates or friends, be sure to cite it in the comments of your program. Failure to comply will be considered as plagiarism.

# 2 Task 1: Golden Hook Fishing

In this task, you need to implement the "Golden Hook Fishing" game. You should strictly follow the specified OO design and game rules for this task. For the OO design, you have to follow the prototypes of the given classes exactly, but can choose to add new member variables and methods.

## 2.1 Description

There are 52 cards excluding "Joker" in a deck. At the beginning of each game, the deck is shuffled and distributed to $N$ players evenly. In our game, we assume 52 is to be divisible by the player's number $N$. Besides, players can not see the cards assigned to them. In other words, a deck of 52 cards is divided into $N$ decks of $52/N$ cards to each player and none of them can see the cards.

Then players take turns to play cards. In each player's turn, he/her takes a card from the top of the his/her deck and adds it to the tail of current card stack. If there is another same-character card in the stack, the players can take the cards between these two cards (include these two cards) and put them at the bottom of his/her deck. **A special situation is that if the player takes a 'J' from the top of his/her deck, he/she can take all the cards on the desktop to himself/herself.** To make it possible to end the game in limited rounds, **this situation is established if and only if the "card stack" is not empty before a 'J' is taken from the deck**. In other words, if the 'J' is the only card in the "card stack", then the "J" will be left in the "card stack" as a regular card and the player will get nothing return. If one player is out when he/she has no cards in his/her deck. The game will end when only one player left.

## 2.2 Perl Classes

Please follow the classes `Deck`, `Player`, `Game` defined below in your Perl implementation. You are free to add other variables, methods or classes. We would start the program by running: `GoldenHookFishing_sample1(2).pl`

1. **Class** `Deck`

   An abstraction of a deck of cards. You have to implement it with the following components:

   - **Instance Variable(s)**

     `cards`

     - This variable is the reference to a one-dimensional array recording initial cards in the deck. Each card is represented using the character in the face of this card. Suit information can be ignored since it will not affect the value of a card.

   - **Instance Method(s)**

     `new`

     - Instantiate a `Deck` object with a deck of 52 cards without shuffling, and return this object.

     `shuffle`

     - Shuffle all cards in the deck. Please *strictly follow* our design to call this function.

     `AveDealCards(num)`

     - Return an array of num reference of divided cards.

   *(The implementation of this Class is provided to students, no need to implement it!)*

2. **Class** `Player`

   A super class representing a participant in the game. You have to implement it with the following components:

   - **Instance Variable(s)**

     `name`

     - This is a variable recording the name of the participant.

     `cards`

     - This variable is the reference to a one-dimensional array recording cards in the player's deck.

   - **Instance Method(s)**

     `new(name)`

     - Instantiate a Player object with its name, and return this object.

     `getCards(card)`

     - Put the taken cards to the bottom of his/her deck.

     `dealCards`

     - Take a card from the top of his/her deck and deal it to the game.

     `numCards`

     - Return the number of player's cards.

3. **Class** `Game`

   This class represents the process of the cards game. You have to implement it with the following components:

   - **Instance Variable(s)**

     `deck`

     - This is a Deck object

     `players`

     - This variable is the reference to a one-dimensional array recording all players in the game. This array naturally encodes the order of all players.

     `cards`

- This variable is the reference to a one-dimensional array recording the cards stack during the game.

- **Instance Method(s)**

  `new`

  - Instantiate a variable deck with a Deck object and an array to record players.

  `set_players(players_name)`

  - "players_name" is a reference to a one-dimensional array recording the name of the players. The players variable will be instantiated by the players name.

  `getReturn`

  - Calculate how many will be returned to the player from the current cards stack. The operation follows the rule in description.

  `showCards`

  - Show the cards on the cards stack.

  `start_game`

  - Start a new game. First, deck will be shuffled and each participant will get a deck of cards evenly. And then players will deal cards and get cards return by turn. In each player's turn, he/she firstly deal a card and then get some or no return according to the rule. Finally, the winner will be shown. You are required to output very important information in this function and more details can be found in the output specifications.

### 2.3 Output Specification

You need to output some required information of the game. Please refer the output sample for more detail.

### 2.4 Grading Criteria

There is another package MannerDeckStudent.pm inherit the Deck.pm and implement the shuffle method, which use a certain algorithm to shuffle the decks. Besides, we also provide students with two input samples, "GoldenHookFishing_sample1.pl" and "GoldenHookFishing_sample2.pl", as well as their corresponding output, "output_sample1.txt" and "output_sample1.txt". Students can use them to test the correctness of their programs.

For scoring, we will modify the algorithm in "MannerDeckStudent.pm" and compare the output from students' program and standard output. **There are several test samples and only absolutely same outputs can get points.**

To be fair, the codes of Deck.pm and MannerDeckStudent.pm will be given to students. **You only need to implement the Player.pl and Game.pl.**

## 3 Task 2: GPU Management Server

This task is to simulate a gpu management server with Perl. You need to use dynamic scoping somewhere appropriate to simplify your implementation. With dynamic scoping, you can implicitly affect the behavior of functions with different context in your program . You should strictly follow our OO design in you implementation. For the OO design, you have to follow the prototypes of the given classes exactly, but can choose to add new member variables and functions.

### 3.1 Description

Artificial intelligence(AI) is quite hot recently and many AI tasks need the support of GPU's computing power. Thus, it's necessary to build a server to manage the GPUs. A server usually contains multiple GPUs, usually 1,2,4 or 8. Each GPU has two states, idel and busy. In each unit

time, the server may receives some takes with specific user and assigns them to idel GPUs. Each task has a unique PID and predefined execution time $T$. Normally, a task will finish if it has been executed for $T$ unit time. Besides, the server can kill any running/waiting task. Once a running task is completed or killed, the related GPU will be released and become idle. If no GPU is idle now, a new submitted task will be added into a waiting queue. When a GPU become idle, the server will first check whether the waiting queue is empty or not. If not, then the task come first will be assigned to this GPU. Your task is to implement such a GPU management system.

## 3.2 Perl Classes

Please follow the classes Task, GPU, Server defined below in your Perl implementation. You are free to add other variables, methods or classes. We would start the program by running: GPUManagementSystem.pl

1. **Class** `Task`

   A class representing a task submitted by user. You have to implement it with the following components:

   - **Instance Variable(s)**

     `pid`

     - This is the PID of the task.

     `name`

     - The name of user who submits this task.

     `time`

     - Total execution time of the task.

   - **Instance Method(s)**

     `new(name, time)`

     - Instantiate an object of Task with its name, time, and return the object. *The pid is counted from zero and plus one for each new task* (The first task has pid 0).

     `pid`

     - Return the PID.

     `name`

     - Return the name.

     `time`

     - Return the time.

2. **Class** `GPU`

   A class representing a GPU. You have to implement it with the following components:

   - **Instance Variable(s)**

     `time`

     - Current execution time of the task.

     `state`

     - The state of GPU, it has two value, 1 and 0. 1 means busy and 0 means idle.

     `task`

     - An object of Class Task, which means the task occupying it.

     `id`

     - The ID of this GPU in the server.

- **Instance Method(s)**

  `new(id)`

  - Instantiate an object of GPU with its ID and the initial state is set to idle.

  `assign_task(task)`

  - Assign a new task to this GPU. Its state will become busy and time is set to zero.

  `release`

  - Release the task and re-initialize the variable.

  `execute_one_time`

  - Execute the task one time.

  `id`

  - Return the GPU ID.

3. **Class** `Server`

   A class representing a server. You have to implement it with the following components:

   - **Instance Variable(s)**

     `gpus`

     - This variable is the reference to a one-dimensional array recording all GPUs in the server.

     `waitq`

     - This variable is the reference to a one-dimensional array recording all tasks in the waiting queue.

   - **Instance Method(s)**

     `new(gpu_num)`

     - Initialize the gpus array with gpu_num GPU Object. The array index represents the GPU index.

     `submit_task(name, time)`

     - New a task with name and time. Then assign the task to a idle GPU(first check GPU0, then GPU1 and so on). If there is no empty GPU, it will be added in the waiting queue.

     `kill_task(name, pid)`

     - Kill the task with given pid and name. If there is no corresponding task, output the failure message.

     `deal_waitq`

     - Check the waiting queue and submit the task if any GPU is idle.

     `execute_one_time`

     - Excute all the GPU one unit time. *The task is finished once reaching the set execution time.* Once a task is finished, the related GPU will be released. And then the earlist task in waiting queue will be assigned to this GPU.

     `show`

     - Print the current GPU message.

     `task_info`

     - Output the task information.

     `task_attr`

     - Return the task attributes.

     `gpu_info`

     - Return the GPU information.

**You have to use dynamic scoping in this package. The implementation of last three methods are available and unmodifiable. You have to call them in your implementation.** You are allowed to add new method(s) to simplify your code with dynamic scoping.

### 3.3 Output Specification

You need to output some required information of the server. Please refer the output sample for more detail.

### 3.4 Grading Criteria

We provide students with two samples ("GPUManagementServer_sample1.pl", "GPUManagementServer_sample2.pl") and their standard output ("output_sample1.txt" and "output_sample2.txt"). Students can use them to test the correctness of their program.

For scoring, we will use other samples and compare the output from students' program and standard output. **There are several test samples and only absolutely same outputs can get points.** Besides, **marks will be deducted if you do not use dynamic scoping or not call specified functions.**

## 4 Report

Your report should simply answer the following questions within TWO A4 page:

1. Provide code and necessary elaborations for demonstrating how you use dynamic scoping in Task 2.
2. What're the advantages and disadvantages of using dynamic scoping, compared with lexical scoping?

## 5 Submission Guidelines

Please read the guidelines CAREFULLY. If you fail to meet the deadline because of submission problem on your side, marks will still be deducted. So please start your work early!

1. In the following, **SUPPOSE**

   your name is *Chan Tai Man*,
   your student ID is *1155234567*,
   your username is *tmchan*, and
   your email address is *tmchan@cse.cuhk.edu.hk*.

2. In your source files, insert the following header. REMEMBER to insert the header according to the comment rule of Perl.

```
/*
 * CSCI3180 Principles of Programming Languages
 *
 * --- Declaration ---
 *
 * I declare that the assignment here submitted is original except for source
 * material explicitly acknowledged.  I also acknowledge that I am aware of
 * University policy and regulations on honesty in academic work, and of the
 * disciplinary guidelines and procedures applicable to breaches of such policy
 * and regulations, as contained in the website
 * http://www.cuhk.edu.hk/policy/academichonesty/
 *
 * Assignment 3
```

```
 * Name : Chan Tai Man
 * Student ID : 1155234567
 * Email Addr : tmchan@cse.cuhk.edu.hk
 */
```

The sample file header is available at

`http://course.cse.cuhk.edu.hk/~csci3180/resource/header.txt`

3. Your submissions WILL BE accepted latest by 11:59 p.m. Apr. 8, but submissions made after the original deadline would be considered as late submissions and penalties will be imposed in the following manner:
   1. Late submission before 11:59 p.m. Apr. 9: Your marks will be deducted for 20%.
   2. Late submission before 11:59 p.m. Apr. 10 : Your marks will be deducted for 50%.

4. The report should be submitted to VeriGuide, which will generate a submission receipt. The report should be named "report.pdf". The VeriGuide receipt of report should be named "receipt.pdf". The report and receipt should be submitted together with codes in the same ZIP archive.

5. `Tar` your source files to `username.tar`
   First, `tar` all your source files for task 1 and task 2 into task1.tar and task2.tar respectively.

   ```
   tar cvf task1.tar Deck.pm Player.pm Game.pm MannerDeckStudent.pm <otherFiles>
   tar cvf task2.tar Server.pm Task.pm Gpu.pm <otherFiles>
   ```

   Then, `tar` task1.tar, task2.tar, report.pdf and receipt.pdf to `username.tar`.

   ```
   tar cvf tmchan.tar task1.tar task2.tar report.pdf receipt.pdf
   ```

6. `Gzip` the `tarred` file to `username.tar.gz` by

   ```
   gzip tmchan.tar
   ```

7. `Uuencode` the `gzipped` file and send it to the course account with the email title "HW3 *studentID yourName*" by

   ```
   uuencode tmchan.tar.gz tmchan.tar.gz \
   | mailx -s "HW3 1155234567 Chan Tai Man" csci3180@cse.cuhk.edu.hk
   ```

8. Please submit your assignment using your Unix accounts.

9. An acknowledgement email will be sent to you if your assignment is received. **DO NOT** delete or modify the acknowledgement email. You should contact your TAs for help if you do not receive the acknowledgement email within 5 minutes after your submission. **DO NOT** re-submit just because you do not receive the acknowledgement email.

10. You can check your submission status at

    `http://course.cse.cuhk.edu.hk/~csci3180/submit/hw3.html`.

11. You can re-submit your assignment, but we will only grade the latest submission.

12. Enjoy your work :>