Assignment 3 — Perl and Dynamic Scoping

Deadline: Apr 10, 2022 (Sunday) 23:59

1 Introduction

The purpose of this assignment is to offer most of you the first experience with Perl, a language supporting both dynamic and static scoping. Our main focus is on dynamic scoping.

The assignment consists of three parts. You need to implement a new game designed by the TAs, called "Tournament Duel", step by step. First, you are required to implement a simple version of this game in Perl, with the Python implementation provided to you for reference. Second, you need to implement a more complex version of this game in Perl with dynamic scoping. Third, you are required to re-implement the same game using Python with static scoping. The detailed Object-Oriented (OO) design for the Perl implementation is given. The OO design for the Python implementation is exactly the same. In the process, you will experience both programming flexibility and readability (good or bad) with dynamic scoping. Your implementation should be able to run on our VM environment with Perl v5.18.2 and Python 3.4.0. Besides, you are required to add "use warnings;" and "use strict;" at the start of your Perl programs. Good coding styles are expected.

IMPORTANT: All your codes will be graded on a VM that will be provided to you. You are welcome to write and test your codes in your own computing environments, but please test them on the given VM before your submission (run your Python code with the "python3" command instead of "python" on the VM).

NO PLAGIARISM! You are free to devise and implement the algorithms for the given tasks, but you must not "steal/borrow" codes from your classmates/friends without proper acknowledgement. If you use some code snippets from public sources or your classmates/friends, ensure you cite them in the comments of your code. Failure to comply will be considered as plagiarism. You are not allowed to ask/hire others to do the assignment for you either. All these are considered as serious cheating acts, which will be dealt with severely.

2 Task 1: Basic Tournament Duel

In this task, you need to implement the basic Tournament Duel game. The complete Python code and a Perl template are provided. You should strictly follow the specified OO design and game rules for this task. You have to follow the prototypes of the given classes exactly and you cannot change the provided skeleton code.

2.1 The Story

In an ancient tribe, the chief of the tribe wanted to select the most intelligent person as the next chief to rule the tribe. The chief of the tribe deployed a game called Tournament Duel to judge the candidates' intelligence. Tournament Duel is a strategy game played in rounds involving two players. Each player in Tournament Duel is a team leader with four fighters. In each round of the game, each team leader has to make an order list of the undefeated fighters of the team. The fighters are then paired according to the order positions in the two order lists. The pair of fighters

with the same order positions in the two teams' order lists will have to duel against each other in the round. If the HP of a fighter is reduced to 0 or below in a duel, the fighter is defeated and cannot be deployed in the remaining rounds. If all the fighters in one team are defeated, this team loses the game.

You, a promising and smart citizen in the tribe, made up your mind to participate in the qualification contest to be the next chief of the tribe. Meanwhile, other competitors are also striving for the chief title. So please be cool and quick; start your tournament duel now.

2.1.1 Task Description

You are required to realize the Tournament Duel game that involves two teams: Team 1 and Team 2. Each team consists of four fighters. We number Fighter 1 to Fighter 4 for Team 1, and Fighter 5 to Fighter 8 for Team 2 for simplicity. In particular, each fighter has four properties, which are HP, attack, defense, and speed respectively.

- *HP*: the health point of the fighter. If HP is less than or equal to 0. The fighter is defeated.

- *attack*: the attack strength of the fighter.

- *defense*: the defense ability of the fighter.

- *speed*: the speed of the fighter. The fighter with a higher speed attacks first in a duel.

HP, attack, defense, and speed are related to the duels. **The properties are all in integers.** The initial values of HP, attack, defense and speed are specified through user inputs that must satisfy the following constraint:

$$HP + 10 * (attack + defense + speed) \leq 500 \tag{1}$$

After the initialization, each team leader has to decide the dueling order of the undefeated fighters in each round. In each round, fighters are picked from the team's order list one by one and duel against the fighter with the same order from the opponent team. If one team has more undefeated fighters than the other team, the excess players will take a nap in the round.

Example 1 *The undefeated fighters in Team 1 are Fighter 1 and Fighter 2, while the undefeated fighters in Team 2 are Fighter 5, Fighter 6, and Fighter 8. The proposed order of Team 1 is [2, 1] and the input order of Team 2 is [8, 5, 6]. Thus, Fighter 2 duels against Fighter 8 and Fighter 1 against Fighter 5. However, Fighter 6 takes a rest in this round.*

The duels are launched one by one according to the order in the order lists. Note that each duel allows only one strike from each fighter. The fighter with higher speed strikes first, followed by the other fighter striking back. The duel is then completed. If the two fighters have the same speed, the fighter from Team 1 strikes first. The first attacking fighter will cast damage to the latter attacking fighter. The damage is defined as the attack value of the attacker minus the defense value of the defender, but the damage is also an integer and should be no less than 1 as shown in (2).
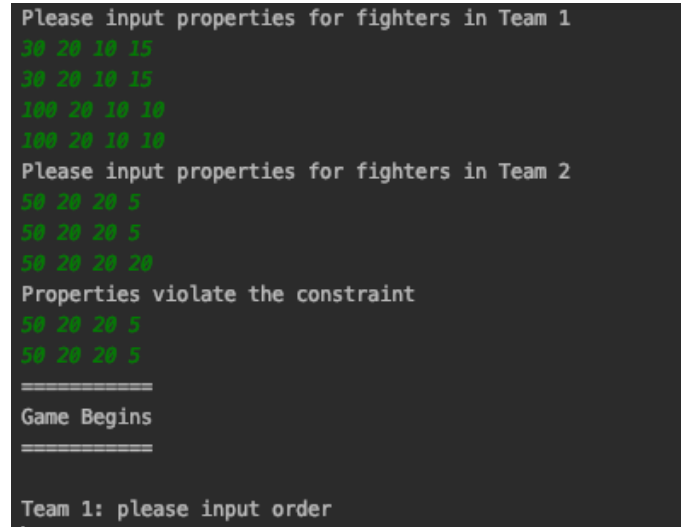
$$damage = \max(attack_{attacker} - defense_{defender}, 1) \tag{2}$$

Afterwards, the defending fighter gets a HP reduction by the value of the damage. If the HP of the defending fighter becomes non-positive, the defending fighter is defeated. If the defending fighter is not defeated, the roles of the two fighters are reversed. After a duel, the fighter casting more damage wins the round. If the damages are the same, this round is a tie.

After each round, we also monitor the states of all the fighters in each team. If all fighters in one team are defeated, this team loses the Tournament Duel game. We should announce that the winner is the other team.

2.2 Workflow

In this section, we specify the workflow of the basic Tournament Duel game. In the beginning, we initialize the tournament, teams, and fighters. The program asks the user to input the initial properties of each fighter satisfying the constraint in inequality 1. The program first output "Please input properties for fighters in Team 1". Then the user enters the properties of the first fighter by four integers and ends by pressing enter. We assume the input format and type are valid. The program only checks the semantic validity of the input that violate the constraint and recursively asks the user to enter properties of the remaining fighters. If the input properties violate the constraint, the program should output "Properties violate the constraint" and ask the user to input the properties again until the constraint is satisfied. After the user inputs four fighters' properties, the program outputs "Please input properties for fighters in Team 2" and follows the same process for Team 1. An example screenshot of the initialization is shown in Figure 1.

Figure 1: An example screenshot of the initialization.

After the initialization, the game starts with the program outputting "Game Begins" (as shown in Figure 2). Afterwards, the program asks the user to input the order of the first team, outputting "Team1: please input order". Then the user enters the fighter order of the first team by the fighters' numbers (NOs) and ends by pressing enter. We assume the input format and type are valid. The program only checks the semantic validity of the input that violates four simple constraints. The checking includes:

- The number of NOs is inconsistent with the number of undefeated fighters.

- There are duplicated fighters in the order.

- Fighter NO is not in the team.

- Fighter NO is defeated.

If the input order is invalid, the program outputs "Invalid input order". An example screenshot of the inputting orders is shown in Figure 2.

After the user inputs fighters' orders, the tournament engine could pair two fighters of the same order on each team to start a duel. The excess fighters have a rest in this round. The detailed dueling rules are given in Section 2.1.1. After each duel, we call the print_info() function to output the summaries of the duel and the properties of fighters. An example screenshot of the output in this round is shown in Figure 3.

3

Figure 2: An example screenshot of inputting orders.

Figure 3: An example screenshot of the output for a round.

At the end of each round, we should check if there is a winner of the Tournament Duel game. If all the fighters in one team are defeated, this team loses, and we announce the other team as the winner. If not, we will continue the next round of duels until we find the winning team. If Team NO is the winning team, the program should output "Team NO wins". An example screenshot of the winner output is shown in Figure 4.

Figure 4: An example screenshot of the output at the end of a game.

2.3 Perl Classes

Please follow the classes **Tournament**, **Team**, **Fighter** defined below in your Perl implementation. You cannot add new variables, methods or classes.

4

1. Class **Tournament**
   This class represents the game engine.

   - **Instance Variables**
     team1
       - This is a **Team** object for Team 1.

     team2
       - This is a **Team** object for Team 2.

     round_cnt
       - An integer to record the current round of the game. The initial value is 1.

   - **Instance Methods**
     new()
       - Initialize a **Tournament** object.

     play_game()
       - Start a new game and administer the whole process of the game. In each round, the user is asked for the fighters' order for both teams and then the duels are simulated accordingly. Finally, the winner of the game will be announced.

     set_teams(Team1, Team2)
       - Initialize the two **Team** object variables.

     play_one_round()
       - Simulate the duels in one round.

     check_winner()
       - Check the fighter states in each team and return the winning team NO if there is a winner. Otherwise, return 0.

     input_fighters(team_NO)
       - Input attributes for the 4 players in Team team_NO.

2. Class **Team**
   The class represents a team in the game. You have to implement it with the following components:

   - **Instance Variables**
     NO
       - The NO of the team. We name the two teams in the game with "Team 1" and "Team 2" respectively.

     fighter_list
       - An array recording all the fighters in the team. The order of each element is determined in the initialization.

     order
       - An array recording the fighters' order in each round. The order of each element is determined by user input.

     fight_cnt
       - An integer recording current fight number in one round, which is used to find the next fighter. The initial value is 0.

   - **Instance Methods**
     new(team_NO)
       - Initialize a **Team** object with the input team_NO.

     set_fighter_list(fighter_list)
       - Initialize all the fighters for the team.

get_fighter_list()

- Return the fighter list in the team.

set_order(order)

- Ask user for the fighters' order in each round for the team and also reset the fight_cnt.

get_next_fighter()

- Return the **Fighter** object of the next fight.

3. Class **Fighter**
The class represents a fighter in the game. You have to implement it with the following components:

- **Instance Variables**

  NO

  - The number of the fighter. We number the fighters in the game from "Fighter 1" to "Fighter 8" respectively.

  HP

  - The current health point of the fighter. If the HP of the fighter is not positive, the fighter is defeated. Note that HP is an integer.

  attack

  - The current attack value of the fighter. Note that attack is an integer.

  defense

  - The current defense value of the fighter. Note that defense is an integer.

  speed

  - The current speed value of the fighter. Note that speed is an integer.

  defeated

  - The state of the fighter is defeated or not.

- **Instance Methods**

  new(NO, HP, attack, defense, speed)

  - Initialize a **Fighter** object.

  get_properties()

  - Return an array including all the properties of the fighter, which has the format [NO, HP, attack, defense, speed, alive].

  reduce_HP(damage)

  - Reduce the HP of the fighter by the damage point.

  check_defeated()

  - Check the state of the fighter is defeated or not.

  print_info()

  - Output the state of the fighter. The code is provided. Do not change.

The complete Python code is provided, and the OO design is the same as Perl. Note that property decorator is used for the setter and getter methods in the Python code.

2.4   Grading Criteria

Your program should run by calling **perl main.pl** and will be tested this way. Do not change the file folder's directory structure. Otherwise, marks for the part will be lost. We will give one test case as an example. To obtain full marks in this part, the outputs of the program should be exactly the same as those given Python code. Moreover, proper error handling and good programming style are important. Poor programming style will receive mark deduction.

3   Task 2: Advanced Tournament Duel

A tribe elder thought that the basic Tournament Duel game was too easy and suggested to make it more complicated and harder. In this task, you are required to implement the advanced Tournament Duel game, which is an extension of the basic version. Dynamic scoping will be useful. A Python template and a Perl template are provided. You should strictly follow the specified OO design and game rules for this task. You have to follow the prototypes of the given classes exactly, and cannot add new member variables and methods.

3.1   Task Description

The advanced Tournament Duel game is an extension of the basic version with more fighter properties and several new dueling rules. The additional property of a fighter is coins and the new dueling rules can change fighters' properties. Each fighter possesses some coins, which can utilized to upgrade the properties *permanently*. The upgradable properties are attack, defense and speed. 50 coins can be used to upgrade one of the upgradable properties once, which increases the property by one point *permanently*. Initially, each fighter has 0 coins, and has to earn their coins after each round. There are two new variables relating to coins in **Fighter** objects:

- *coins*: The number of coins possessed by the fighter. Each fighter obtains some coins after each round and they can spend coins to upgrade their properties (*attack*, *defense*, and *speed*).

- *history_record*: An array recording the history of the recent three duels of the fighter. This is used to implement the new dueling rules.

There is one new package variable relating to coins:

- *coins_to_obtain*: the number of coins each fighter will obtain after one round. It is a package variable storing the default number of coins to be earned by a fighter after each round.

Besides, since fighters will get tired after each duel, their *attack*, *defense*, and *speed* will decrease by a certain value until the values become 1. We use three package variables *delta_attack*, *delta_defense* and *delta_speed* to maintain the changes of their properties correspondingly for each round. To be more specific, after each round, $attack \leftarrow \max(attack + delta\_attack, 1)$, $defense \leftarrow \max(defense + delta\_defense, 1)$, and $speed \leftarrow \max(speed + delta\_speed, 1)$. These three variables are -1 by default. However, under some situations, their values may vary *temporarily* and the related rules will be described in the following.

The team leaders also need to determine the strategy for fighters involved to upgrade properties before a duel starts in each round. If a fighter has 50 coins or more, the interface will ask the user to input the upgrade strategies repeatedly until the user has less than 50 coins or wants no more upgrades. The possible upgrade strategies are [A: attack, D: defense, S: speed, N: no], where A, D, or S represents upgrading the corresponding property, while N means no upgrade in this round. After the team leader chooses an upgrade strategy for a fighter, the fighter gets one point up on *attack*, *defense* or *speed* permanently and has 50 coins deducted. If the number of possessed coins is still no less than 50, the interface will continue to ask the user to upgrade until the leader inputs "N" or the number of possessed coins is less than 50. Then, the duel can begin. Each fighter obtains 20 coins by default after a duel in one round. Some fighters may obtain extra coins by satisfying advanced rules. The advanced rules which may also change *delta_attack*, *delta_defense*, *delta_speed* and *coins_to_obtain* for one round *temporarily* are listed as follows:

- **Fighters at rest** earn only half of the coins in that round. In addition, their *delta_attack*, *delta_defense* and *delta_speed* will be +1 for that round *temporarily* since they have built up their strength during the rest.

- A **consecutive winner** is a fighter winning three consecutive duels including the one in the latest round, and will take a more aggressive strategy for the next round. A consecutive winner is willing to sacrifice some defense in exchange for speed and attack, so that a consecutive winner's *delta_attack* and *delta_speed* will be +1, and *delta_defense* will be −2 for that round *temporarily*. Besides, a consecutive winner will obtain more coins in rewards and the *coins_to_obtain* will be increased by 10% in this round. Note that the *history_record* for the consecutive winner will be cleared after this round for the balance of the game in prevention of dominating players.

  To be more specific, assume that a fighter wins the $(i-2)^{th}$, the $(i-1)^{th}$, and the ith round. The fighter's *coins_to_obtain* will increase by 10% at the end of the ith round *temporarily*. Meanwhile, the fighter's *delta_attack* and *delta_speed* will be +1, and *delta_defense* will be −2 at the $(i+1)^{th}$ round *temporarily*.

- A **consecutive loser** is a fighter losing three consecutive duels including the one in the latest round. A consecutive loser will take a more conservative strategy for the next round and will put nearly all the focus on defense and speed for survival. Therefore, a consecutive loser's *delta_defense* and *delta_speed* will be +2, and *delta_attack* will be −2 in that round *temporarily*. Besides, a consecutive loser will obtain more coins in rewards and the *coins_to_obtain* will be increased by 10% in this round *temporarily* for compensation. Note that the *history_record* for the consecutive loser will be cleared after this round too.

- A **fighter defeating another fighter** will have morality boosted and thus the fighter's *delta_attack* will be increased by 1 for the next round *temporarily*. Beside, the coins to obtain becomes doubled for the current round *temporarily*.

3.2 Workflow

In this section, we specify the workflow of the advanced Tournament Duel game. The workflow is essentially the same as the basic version, except that the user will obtain coins and have their properties updated.

The tournament starts with asking two teams to input their fighters' orders. After that, the tournament checks whether each fighter has enough coins to upgrade their properties permanently before each duel. If a fighter has enough coins, the program outputs "Do you want to upgrade properties for Fighter NO? A for attack. D for defense. S for speed. N for no" to ask the user to input the upgrade strategy for the fighter, and upgrades the fighter's properties accordingly. We assume inputs are valid, so that you do not need to check the inputs.

The tournament also checks whether the fighter satisfies the four advanced rules in Section 3.1. If the fighter satisfies some advanced rules, the tournament changes *delta_attack*, *delta_defense*, *delta_speed* and *coins_to_obtain* for one round temporarily. If a fighter is a consecutive winner and defeats another fighter in the current round, the advanced rules should change *delta_attack*, *delta_defense*, *delta_speed* and *coins_to_obtain* for one round temporarily as: $delta\_attack = +2$, $delta\_defense = -2$, $delta\_speed = +1$, $coins\_to\_obtain = \text{int}(2 * \text{int}(1.1 * \text{coins\_to\_obtain}))$. An example screenshot of permanent upgrading and temporary updating is shown in Figure 5.

Example 2 *Take Fighter 5 in Figure 5 as an example. In round 3, the properties of Fighter 5 is: $attack = 18, defense = 18, speed = 3$. Since Fighter 5 is a consecutive winner, i.e. Fighter 5 wins the first round, second round and the third round, the properties of Fighter 5 should be changed into: $attack = 19, defense = 16, speed = 4$. Besides, Fighter 5 defeats Fighter 1 in round 3, the properties of Fighter 5 should be changed into: $attack = 20, defense = 16, speed = 4$. Lastly, in*

Figure 5: An example of screenshot of permanent upgrading and temporary updating.

the beginning of round 4, Fighter 5 uses 50 coins to upgrade defense by one point. Therefore, the properties of Fighter 5 should be changed into: $attack = 20, defense = 17, speed = 4$.

After a duel is finished, fighters obtain their coins in this round. We should check the fighter again whether the fighter satisfies the advanced rules in Section 3.1. If the fighter satisfies some advanced rules, the *coins_to_obtain* for the fighter has a temporary change. Otherwise, the fighter obtains 20 coins by default. Remember to give coins to fighters at rest in the round too.

After each duel, you should output the duel results and call print_info() to generate a summary string containing fighters' states after the duel, which is the same as the basic version. At the end of each round, we should check if there is a winner of the Tournament Duel game. If all fighters in one team are defeated, this team loses, and we announce the other team as winner. If not, we will continue the next round of duels until we find the winning team. If we find the winning team, the program should output "Team NO wins", where NO is the number of the winning team. This part is the same as the basic version.

3.3 Perl Classes

Please follow the classes **Tournament**, **Team**, **Fighter** defined below in your Perl implementation. You cannot add other variables, methods or classes. We would start the program by running: perl

9

main.pl.

1. Class **AdvancedTournament**

   The class represents an advanced tournament in the advanced version. This class inherits the variables and methods in the class **Tournament**. You have to implement it with the following components:

   - **Instance Variables**

     defeat_record
       - An array recording the defeated fighter NOs in the last round.

   - **Instance Methods**

     new()
       - Initialize a **AdvancedTournament** object.

     input_fighters(team_NO)
       - Input attributes for the 4 advanced fighters in team_NO.

     play_one_round()
       - Execute all the duels in one round.

     update_fighter_properties_and_award_coins()
       - Update fighter's properties and award coins to fighters. Check whether the fighter satisfies the four advanced rules and temporarily change *delta_attack*, *delta_defense*, *delta_speed* and *coins_to_obtain*. Please follow the advanced rules in Section 3.1. The amount of awarded coins follow the rules we mention in Section 3.1 too.

2. Class **AdvancedFighter**

   The class represents an advanced fighter in the advanced version. This class inherits the variables and methods in the class **Fighter**. You have to implement it with the following components:

   - **Instance Variables**

     coins
       - The coins possessed by the fighter.

     history_record
       - An array storing the latest three duel results. The initial value is an empty array.

   - **Instance Methods**

     new(NO, HP, attack, defense, speed)
       - Initialize an **AdvancedFighter** object.

     record_fight(fight_result)
       - Record the latest fight result into the history_record.

     obtain_coins()
       - Fighter obtains coins according to rules.

     buy_prop_upgrade()
       - Check whether the possessed coins are enough to upgrade the properties of the fighter recursively. If the coins are enough, asks the user to input the strategy repeatedly until the fighter has less than 50 coins or refuse to upgrade. The strategies are [A: attack, D: defense, S: speed, N: no]. Upgrade the properties of the fighter based on the upgrade strategy for the fighter and change the possessed coins of the fighter, and repeat until the user wants no more upgrade or there are less than 50 coins.

     update_properties()
       - Update the fighter's *attack*, *defense* and *speed* properties using *delta_attack*, *delta_defense*, and *delta_speed*.

10

3.4 Grading Criteria

Your program should run by calling **perl main.pl** and we will test your basic or advanced version by changing the main file. The variable coins_to_obtain is a default value for all the fighters and we will change the value of this variable to test your code under different settings. Do not change the locations of the file. Before submission. Otherwise, marks for the part will be lost. We will give one test case as an example. To obtain full marks in this part, the outputs of the program should be exactly the same as the given. Moreover, proper error handling and good programming style are important. Poor programming style will receive mark deduction.

3.5 Implementation in Python

In this task, you are asked to implement the Advanced Tournament Dual using Python with exactly the same class design. The skeleton code is provided. Do not change! Also, the input/output specifications and grading criteria are the same as those in the previous task. We will run **python3 main.py** during evaluation. Note that Python does not support dynamic scoping.

4 Report

You should give a simple report to answer the following questions within two A4 pages:

1. Provide example code and necessary elaborations for demonstrating the advantages of Dynamic Scoping in using Perl to implement the Advanced Tournament Dual game as compared to the corresponding codes in Python.

2. Discuss the keyword local in Perl (e.g. its origin, its role in Perl, and real practical applications of it) and give your own opinions.

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.

The late submission policy is as follows:

- 1 day late: -20 marks

- 2 days late: -40 marks

- 3 days late: -100 marks

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 and Python.

```
/*
 * 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. The report should be submitted to VeriGuide, which will generate a submission receipt. The report and receipt should be submitted together with your Perl and Python codes in the same ZIP archive.

4. For Task 1, the Perl source should have the filename "Tournament.pl", "Team.pl", and "Fighter.pl". The Python source should have the filename "Tournament.py", "Team.py", and "Fighter.py". For Task 2, the Perl source should have the filename "AdvancedTournament.pl" and "AdvancedFighter.pl". The Python source should have the filename "AdvancedTournament.py" and "AdvancedFighter.py". The VeriGuide receipt of report should have the filename "receipt.pdf". All file naming should be followed strictly and without the quotes.

5. `Tar` your source files to `username.tar` by

> `tar cvf tmchan.tar task1 task2 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 :>