Curtin University — Department of Computing

**Unix & C Programming** (COMP1000)

Semester 2, 2020

# Final Assessment

**Due:** Check the time limit on Blackboard when you start this assessment

**Weight:** 50% of the unit mark.

This is an online OPEN BOOK Assessment. You have 5 hours and 30 minutes to complete this. Finish all the tasks and submit the tarball before the time limit. Please ensure that you dedicate sufficient time to understand the tasks at hand before you begin coding.

## 1 Overall Program

A supplementary video has been included with this assessment specification. It is recommended you watch the video prior to start coding. The task will be to write a program that simulates an automatic turn-based role-playing game (RPG). A 'hero' and their 'enemies' will each be imported from files and the hero will fight until the battle is won or defeated. Each turn will have random elements that the program must manage (such as dealing more damage, missing the attacks, or receiving a "power-up").

### 1.1 Unassessed Items

Some items will not be assessed to assist with saving time. In particular:

- **Commenting your code**. Although commenting may help you with how you structure and attempt the task, you are not required to do so. However, other coding standards will still apply – your code must still be readable and must not contain multiple returns in a single function, as an example. C89 syntax must still be used.
- **Data type validation**. You may assume that the data type of the information from the input files is always correct. However, validating the data itself for correct values is still required. e.g. the ATTACK power of the hero should not be a negative number.

## 2 Academic Integrity

This is an assessable task, and as such there are strict rules. While the assessment is *open book*, this does not allow collusion. You must not ask for or accept help from anyone else on completing the tasks (whether or not they are themselves students). You must not show your work to another student enrolled in this unit who might gain unfair advantage from it. These actions are considered plagiarism or collusion.

Do NOT just copy some C source codes from internet resources. If you get caught, it will be considered plagiarism. Always reference your sources. Be aware that if significant portions of the assignment are not your own work (even if referenced), there may be penalties. If in doubt, ask!

Please see Curtin's Academic Integrity website for information on academic misconduct (which includes plagiarism and collusion).

The unit coordinator may require you to attend quick interview to answer questions about any piece of written work submitted in this unit. Your response(s) may be referred to as evidence in an Academic Misconduct inquiry. In addition, your assignment submission maybe analysed by systems to detect plagiarism and/or collusion.

# 3 The Tasks

## 3.1 File IO

The program will use two input files:

- **hero.txt** – This file will contain a single line in this format:

  HP, ATK, DEF, MED, Name
  e.g. 20,6,3,1,Arlen

  - HP (Hitpoints) represent the health of the Hero. The game is over if this reaches zero.
  - ATK (Attack Point) measures how much damage the Hero can deal to the enemy.
  - DEF (Defense Point) measures how much damage the Hero can absorb when receiving an attack from enemy.
  - MED (Medicine) is the medicine the hero can consume when his/her HP is 30% or less. However, if the HP reaches zero, this medicine cannot be used to revive the Hero.

- **enemies.txt** – This file will contain a list of any number of enemies that the hero must fight and will be in this format:

  HP, ATK, DEF, Monster_Name_1
  HP, ATK, DEF, Monster_Name_2
  ...

  - Every monster will have its own HP, ATK, and DEF. No monster will carry medicine.

## 3.2 Command Line Arguments

Your program should receive only 3 arguments: "hero.txt", "enemies.txt", and the amount of seconds to sleep for every screen transition/turn on the interface. You can use the

function atof() (this function is already included in stdlib.h) to convert the strings to float. Please check if the file can be opened before proceeding further.

## 3.3 Structs

Technically, you only need one struct here which contains four integers and one string for the Hero/enemy. The 4 integers are for the HP (Hit Points), ATK (Attack Point), DEF (Defense Point), and MED (Medicine). The enemy can just simply ignore the MED field. The string is for the name of the Hero/enemy. For simplicity, you can assume the name will only contain 50 characters maximum.

You should reuse the generic linked list you have written beforehand. You will need the linked list to store all the enemies information. Everytime the Hero start fighting the enemies, you remove it from the linked list one by one (and free it eventually). As the supplementary video mentioned, you store the enemies' information with the stack data structure manner. The first enemy in the text file, will be the last one to be fought.

## 3.4 The Battle

### 3.4.1 Format

When the battle starts, the Hero always get the first turn. Afterwards, the Hero and enemy will take turn to deal damage to each other's HP until one is defeated. The formula to calculate the dealt damage is:

$$\text{Damage} = (\text{Attacker's ATK}) - (\text{Defender's DEF})$$

### 3.4.2 Win/Lose Condition

The Hero wins the game by defeating all the enemies (basically, when the linked list is empty). The Hero loses the game when he/she is defeated (HP reaches zero) OR when his/her ATK is too weak to break through the enemy's DEF (i.e. the ATK is less than or equal to 50% of enemy's DEF). When the Hero loses, you can output the message stating end of the game and then proceed to free the remaining allocated memory before closing the program.

### 3.4.3 Battle Turn Randomness

For every attack, there are three possible outcomes:

- **Damage** (70% probability) – In this case, the damage is calculated as normal (ATK - DEF)
- **Critical Hit** (20% probability) – Here the ATK is doubled for that turn only, prior to subtracting DEF. However, you **must use bit shifting to achieve this effect**.
- **Miss** (10% probability) – The attacker will not deal any damage in this scenario. Please ensure that the DAMAGE does not become a negative number in this operation.

You may use random number functions to simulate these scenarios. Ensure that every game will yield a different result. (Hint: Use srand() function)

### 3.4.4 Battle Reward Randomness

After every battle, the Hero will receive one of these three items:

- **Better Sword** (30% probability) – This will increase the Hero's ATK by 50% permanently. Make sure the final number is still an integer by rounding the number OR typecasting it into integer.
- **Better Shield/Armor** (30% probability) – Similar idea to "better sword", but the DEF is increased by 50% permanently instead.
- **1 Medicine** (40% probability) – The Hero gained one Medicine for him/her to consume later. There is no upgrade on the ATK or DEF in this case.

## 3.5 Gameplay

Once again, you can watch the supplementary video to see how the gameplay looks like. Here is the written explanation of the program interface. Keep in mind the transition between one screen/event to the next one always requires the program to clear the screen ( system("clear") ) and use newSleep with the length depending on the provided command line arguments.

**Welcome Message**
When the program starts with valid input files, it should clear the screen and show a simple welcoming message. You can choose the sentence. Here is one example:

```
Welcome to auto RPG!
```

**Hero/Enemy Status**
The program will then list the status of the Hero and the enemies. Keep in mind that the list of the enemies has been reversed. Make sure you store the enemy information on the linked list with stack data structure (first IN, last OUT). Here is an example:

```
Hero:
Arlen           HP:20   ATK:6   DEF:3   MED:1

Enemies:
Slime           HP:5    ATK:4   DEF:3
Overlord        HP:20   ATK:10  DEF:4
```

**Battle Screen**
After the status screen, then the first battle will start. The Hero always strikes first, then they will take turn to attack each other until one is defeated. Here is how the interface

looks like:

```
Arlen
HP:20
ATK:6
DEF:3
MED:1

Arlen attacks Slime, dealing 3 damage!

        Slime
        HP:2
        ATK:4
        DEF:3
```

```
Arlen
HP:19
ATK:6
DEF:3
MED:1

        Slime attacks Arlen, dealing 1 damage!

        Slime
        HP:2
        ATK:4
        DEF:3
```

```
Arlen
HP:19
ATK:6
DEF:3
MED:1

Arlen attacks Slime, Critical Hit!! dealing 9 damage!

        Slime
        HP:0
        ATK:4
        DEF:3

Slime is defeated!
```

As you can see, the Hero's current status will be on the top left, while the enemy is at the bottom (you can use the tab '\t' to create the indent). The HP is updated in real time

as the battle goes on. Once the enemy is defeated, it should be removed from the linked list, then print the sentence "enemy_name is defeated!".

If the Hero's HP is 30% or less remaining (but not zero), he/she can use the medicine (if any) to recover full HP, and still can attack at the same turn:

```
Arlen
HP:20
ATK:9
DEF:3
MED:0

Arlen is badly injured, eating 1 medicine. HP is restored!
Arlen attacks Overlord, Critical Hit! dealing 14 damage

        Overlord
        HP:6
        ATK:10
        DEF:4
```

**Item Drop**

After every battle, the Hero will receive an item. Please print what item the hero received and the new status. Here is an example:

```
Arlen got better sword! ATK is now 9!
```

**Win/Lose**

Lastly, print the result of the overall battle. If the Hero defeats all the enemies, print the winning message. If the Hero loses the battle, print the losing message:

```
Arlen defeats all enemies! The world is saved.
cg@ubuntu:~/Desktop$
```

OR

```
Arlen loses the battle! The world is doomed.
cg@ubuntu:~/Desktop$
```

## 3.6 Shell Scripting

Please write a short Shell script called "auto.sh" to detect whether the executable file exists. If not, run the makefile. Afterwards, run the executable with the "hero.txt" and "enemies.txt" as the arguments, along with "5.0" seconds argument.

### 3.7 Makefile

As usual, please write a proper makefile with all the variables and rules (including clean rules). There is NO conditional compilation on this final assessment. If this makefile is missing or broken, we will assume the program cannot be compiled.

## 4 Marking Distribution

- Struct for Hero/enemy (5 marks)
- Command line arguments validation (5 marks)
- Correct gameplay interface (with proper screen clear, newSleep and real-time update) (15 marks)
- Read files correctly, validating the content, and store the content in a struct. Utilizing the linked list correctly to store/remove the information of the enemies correctly. (20 marks)
- Implementation of battle and item drop with correct probability. (15 marks)
- Usage of bit shifting for the Critical hit. (5 marks)
- Correct update on the status after receiving item. (5 marks)
- Makefile (5 marks)
- Shell Scripting (5 marks)
- Memory Management (Memory Leaks Check) (10 marks)
- Coding Standard (10 marks)

# 5 Final Check & Submission

After you complete your assignment, please make sure it can be compiled and run on our Linux lab environment. If you do your assignment on other environments (e.g on Windows operating system), then it is your responsibility to ensure that it works on the lab environment. In the case of incompatibility, it will be considered "not working" and some penalties will apply. You have to submit a tarball (see practical worksheet for the tarball creation instruction) containing:

- **Your essential assessment files** — Submit all .c, .h, and .sh files along with your makefile. Please do not submit any executable or object (.o) files.

The name of the tarball should be in the format of:

$$\langle student - ID \rangle\_\langle full - name \rangle\_Final.tar.gz$$

Example: 12345678_Antoni-Liang_Final.tar.gz

Once the tarball is ready, please submit it in **TWO places**:

- The first place is the "e-test" webpage where you download the assessment specification PDF file.
- The second place is the "backup submission link".

Make sure these 2 submission are the SAME final tarball you want to submit. You can double check your submission on the backup submission link.

Please make sure your submission is complete and not corrupted. You can re-download the submission and check if you can compile and run the program again. Corrupted or empty submissions will not receive marks. You may submit multiple times, but only your latest submission will be marked (assuming not over the time limit). If you need to submit it multiple times, make sure the latest submission is still a complete submission (i.e not just one file to complement previous submission).

# End of Final Assessment