EEEC10008(515169) S23: Homework 2

Due: 2023/5/16(Tues.) 23:59

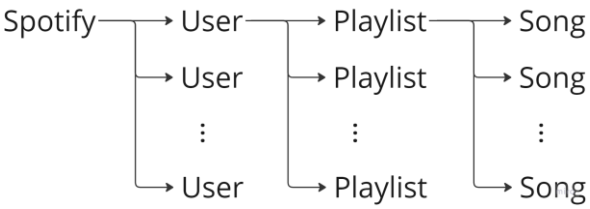
[Instruction]

- Please put your source code files of each problem into separate folder named StudentID_hw2_1 and StudentID_hw2_2, compress these two folders to **zip** files separately (Ex: 111511000_hw2_1.zip, 111511000_hw2_2.zip), and upload these two zip files to e3 before deadline.
- **If zipped file' s or source code file's name is wrong, your score of this homework is 30% off.**
- **Your source code files should be able to be compiled and executed on our server.**
- Your output should follow the example, otherwise you will not get full credit.
- If you have any question, please send an email to TA or leave a message in the line account.
- If necessary, you can get the files provided in each problem from `/home/share/hw2/`.

[Problem 1]: Digital Music Streaming Service

Spotify is a digital music, podcast, and video service that gives you access to millions of songs and other content from creators all over the world. In this problem, you should implement a simple version of it, including four classes (`class Spotify`, `class User`, `class Playlist`, and `class Song`).

- Hierarchy of four classes



- You will use keyboard to choose the command, the input should ignore the case (i.e. 'A' and 'a' belongs to same command).
- To pass the test, your program cannot contain memory leaks. You can use the following command to test for memory leaks.
  - **valgrind <your_executable_file> <arguments_if_needed>**
- `class Spotify`
  - User *curr_user: points to the current user, will be null at first
  - vector<User *> users: a vector of pointers used to store all users information.
  - vector<const Song *> songs: a vector of pointers used to store the songs provided in streaming service.
- `class User`
  - string name: The name of user.
  - string passwd: The password of user
  - Playlist *curr_list: points to the current playlist, will be null at first

- vector<Playlist *> all_list: a vector of pointers used to store all playlists of the user.
- const vector <const Song *> songs: a vector of pointers used to store the songs provided in streaming service.

- `class Playlist`
  - string name: The name of playlist.
  - int curr_song_index: The index used to record the current playing song, will point to the first song in list at first.
  - vector <const Song *> song_in_list: a vector of pointers used to store the songs in list.
  - const vector <const Song *> songs: a vector of pointers used to store the songs provided in streaming service.
- `class Song`
  - string name: The title of song.
  - string artist: The artist of song
  - string song_len: The length of song

File description

- You will be given `main.cpp`, `Spotify.h`, `User.h`, `Playlist.h`, `Song.h`, and `music.txt`.
- Files can get in `/home/share/hw2/hw2-1/`.
- Be careful, **`main.cpp`** cannot be modified, otherwise you will get 0 points.
- As for other header files, you can only add **Accessor & Mutator Member Functions**, **friend class relationship, and Boolean Function for sorting**.
- You must submit `Makefile`, `main.cpp`, `Spotify.h`, `User.h`, `Playlist.h` and `Song.h` these five files for this problem; however, if you have other program files, you may compress it into the same directory. In this directory, your code will be compiled and executed.

✓ Please **write every classes in separate files** and write a "`Makefile`" in your directory to compile your code. Your code can be compiled by typing "`make`", and the name of your program should be `hw2_1`.

✓ Here are the class template and the output.

✓ `main.cpp`

```
#include "Spotify.h"
using namespace std;

int main(int argc, char *argv[]) {
    string in_file = argv[1];
    Spotify s(in_file);
```

```
    s.controlManual();
    return 0;
}
```

✓ Spotify.h

```
#ifndef SPOTIFY_H
#define SPOTIFY_H
#include <fstream>
#include "User.h"
using namespace std;
// Add any Boolean Function for sorting you need
class Spotify {
  private:
    vector<User *> users;
    vector<const Song *> songs;
    User *curr_user;
    void scene1();
    void scene2();
    void scene3();
    void scene4();
    void addSong();     // add new song into songs
    void createUser();  // create new user
    void logIN();       // log in to specific user
    void logOUT();      // log out from current user
    void printSongList();  // list all songs in device
(sort by song length, if equal sort by song title)
    void printUserList();  // list all users in device
(sort by creation order)
  public:
    Spotify(string file);
    ~Spotify();
    void controlManual();  // controller of the whole
device
    // Add any Accessor & Mutator functions, or friend
```

```
    class declaration you need
};
#endif
```

✓ User.h

```
#ifndef USER_H
#define USER_H
#include "Playlist.h"
using namespace std;


// Add any Boolean Function for sorting you need
class User {
    private:
        string name;
        string passwd;
        Playlist *curr_list;
        vector<Playlist *> all_list;
        const vector<const Song *> songs;
        void show_list();     // show all playlists of the user
    (sort by list name)
        void choose_list();   // choose a playlist
        void add_list();     // add a playlist
    public:
        User(string name, string passwd, vector<const Song *>
    &songs);
        ~User();
        // Add any Accessor & Mutator functions, or friend
        class declaration you need
};
#endif
```

✓ Playlist.h

```
#ifndef PLAYLIST_H
#define PLAYLIST_H
```

```
#include "Song.h"
using namespace std;


// Add any Boolean Function for sorting you need
class Playlist {
  private:
   string name;
   int curr_song_index;
   vector<const Song *> song_in_list;
   const vector<const Song *> songs;
   void show_song();         // show song in playlist
(sort by song length, if equal sort by song title)
   void add_song();    // add song into playlist (song
must exists in streaming device)
   void remove_song();  // remove song from playlist
   void play_song();    // play current song
   void next_song();    // play next song (back to first
song if meet the end of playlist)
   void prev_song();    // play previous song (go to last
song if meet the start of playlist)
  public:
   Playlist(string name, const vector<const Song *>
&songs);
   ~Playlist();
   // Add any Accessor & Mutator functions, or
   friend class  declaration you need
};
#endif
```

✓ User.h

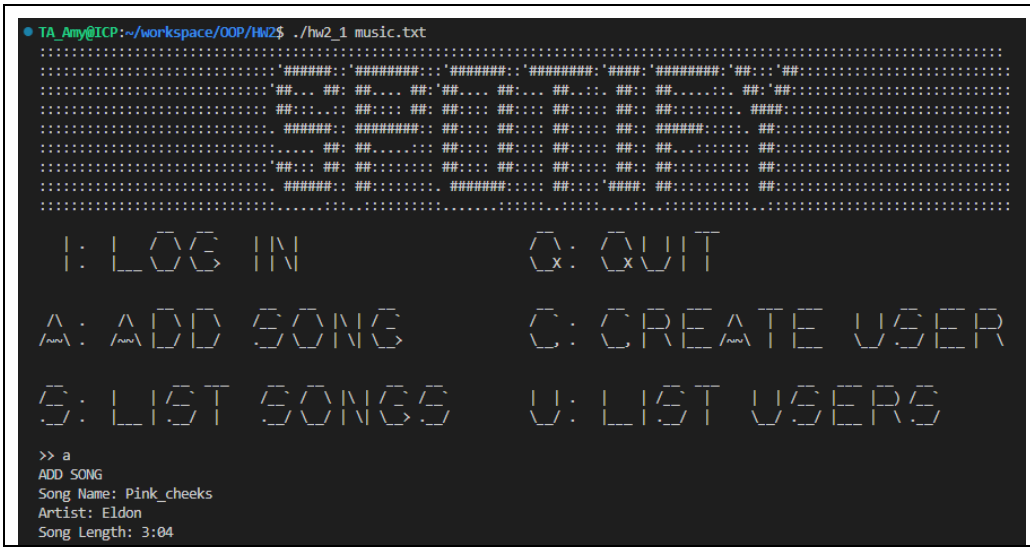
```
#ifndef SONG_H
#define SONG_H
using namespace std;
```

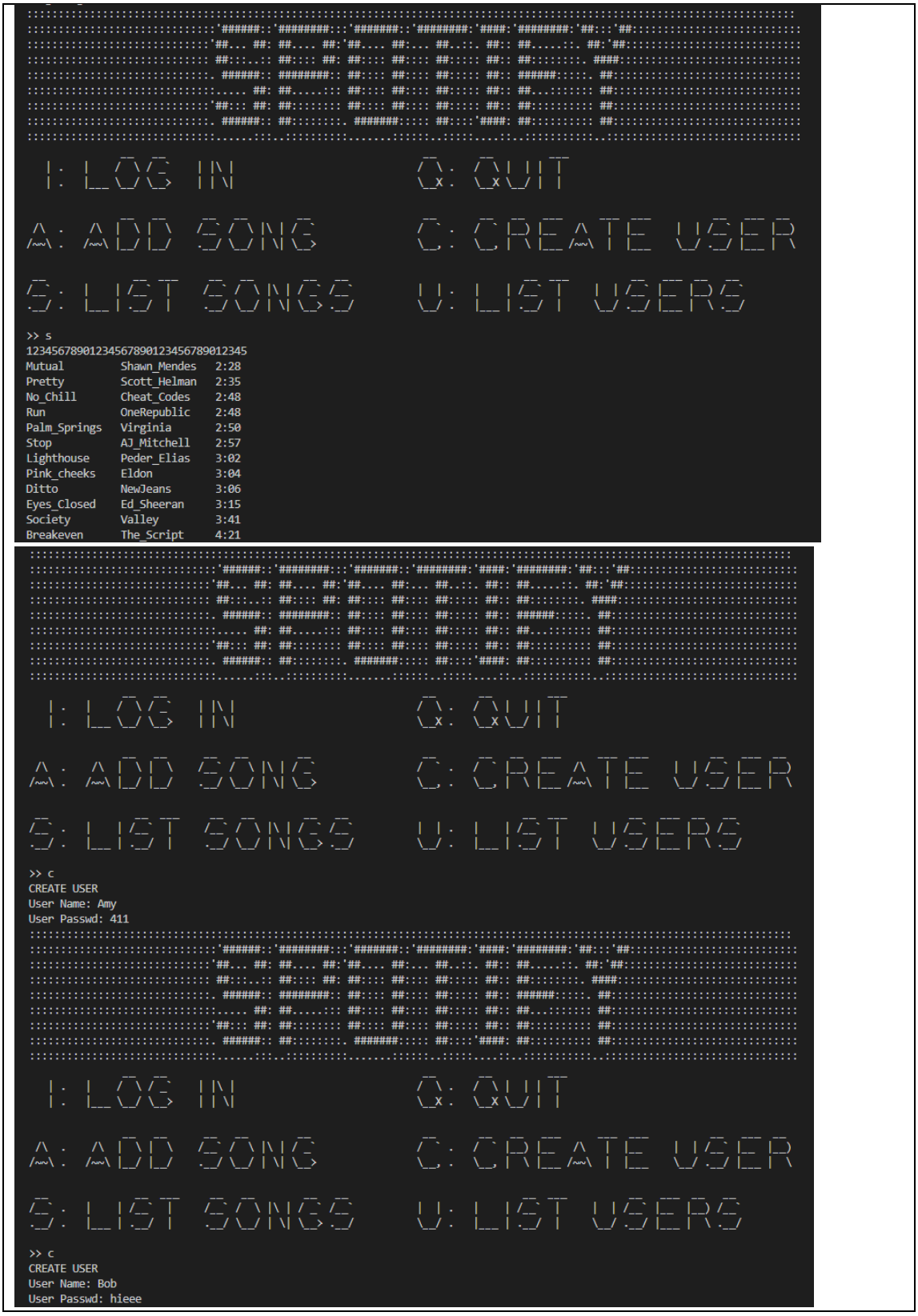
```
// Add any Boolean Function for sorting you need
class Song {
   private:
    string name;
    string artist;
    string song_len;
   public:
    Song(string name, string artist, string song_len);
    ~Song();
    // Add any Accessor & Mutator functions, or friend
    class declaration you need
};
#endif
```

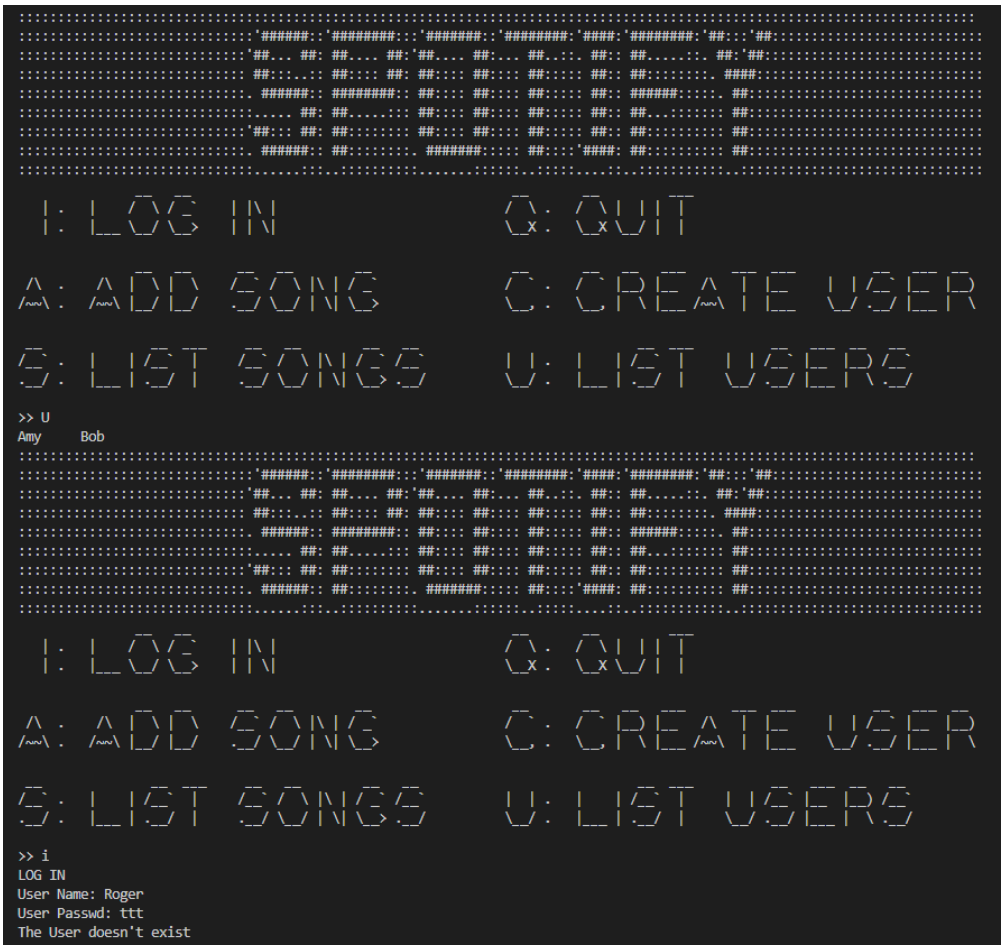
✓ `$ make`

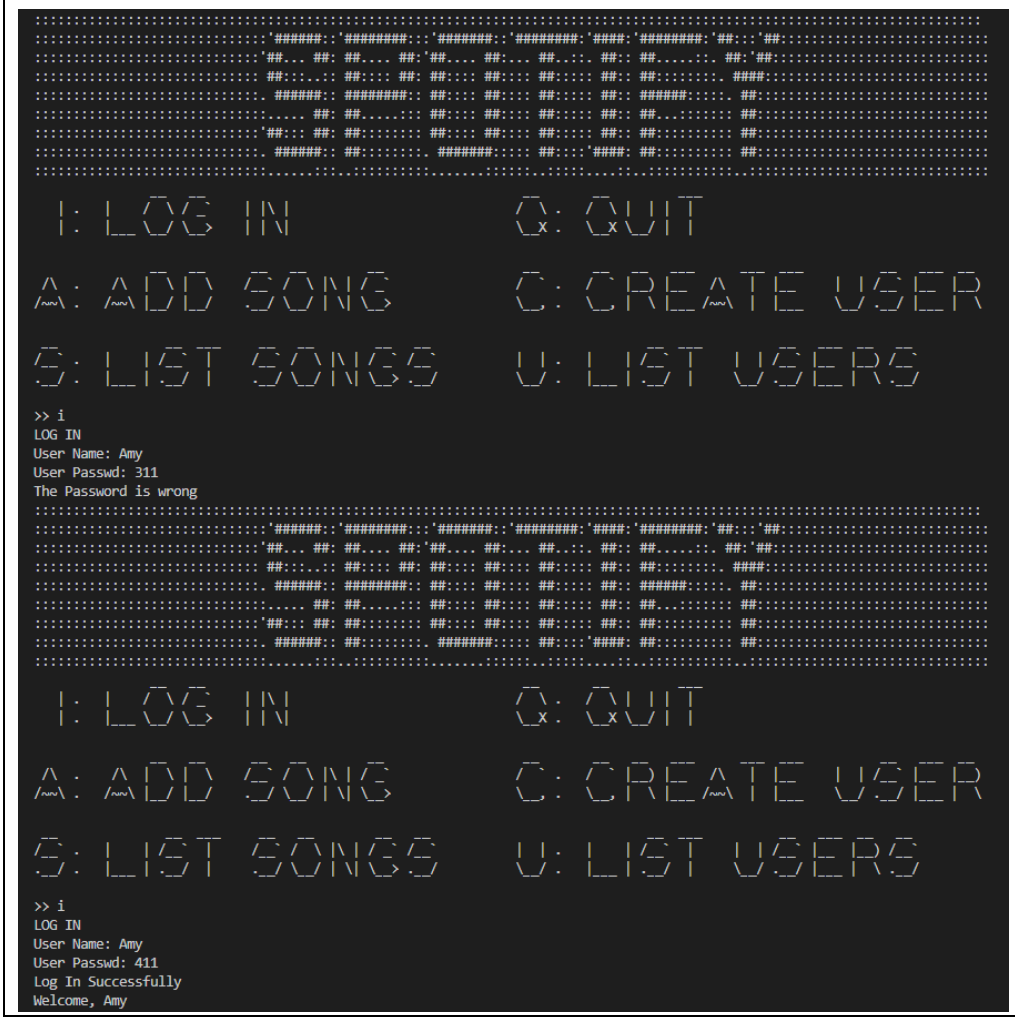
`$ ./hw2_1 [musicfile]`

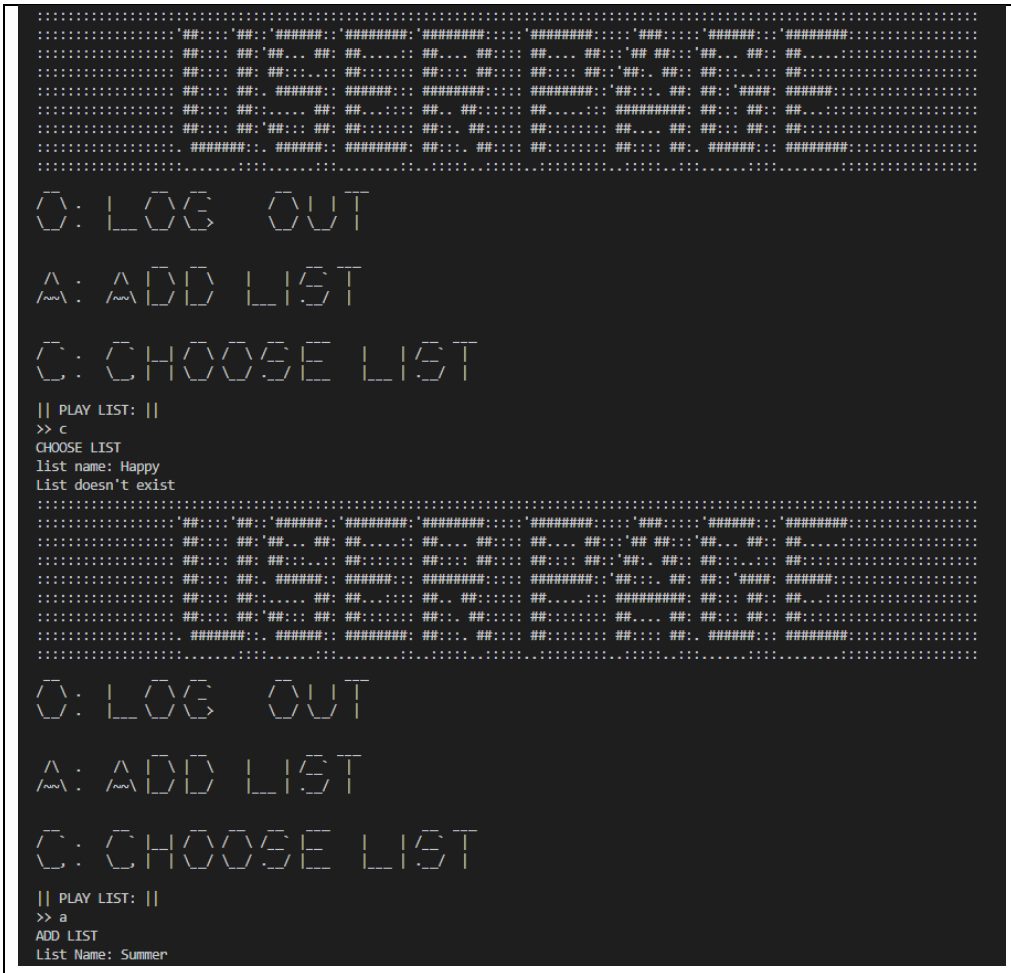
You can get clear version of output example at here: https://reurl.cc/MREo8p

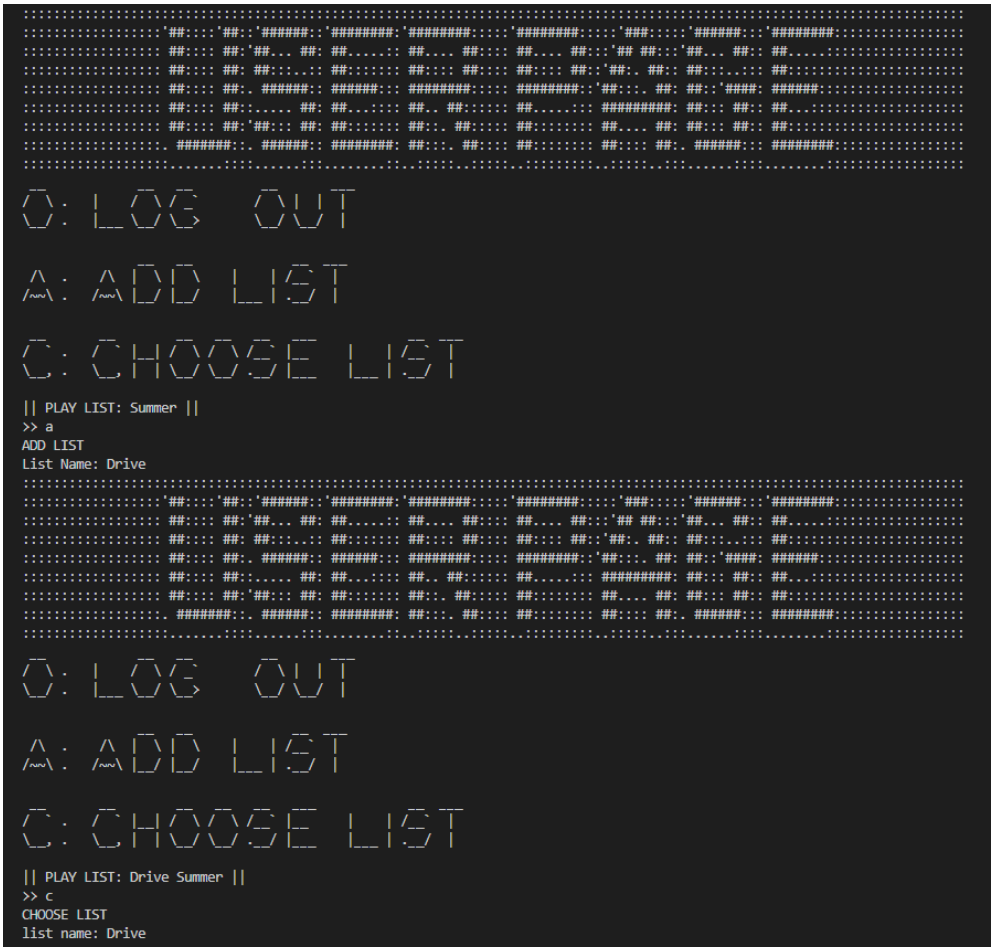


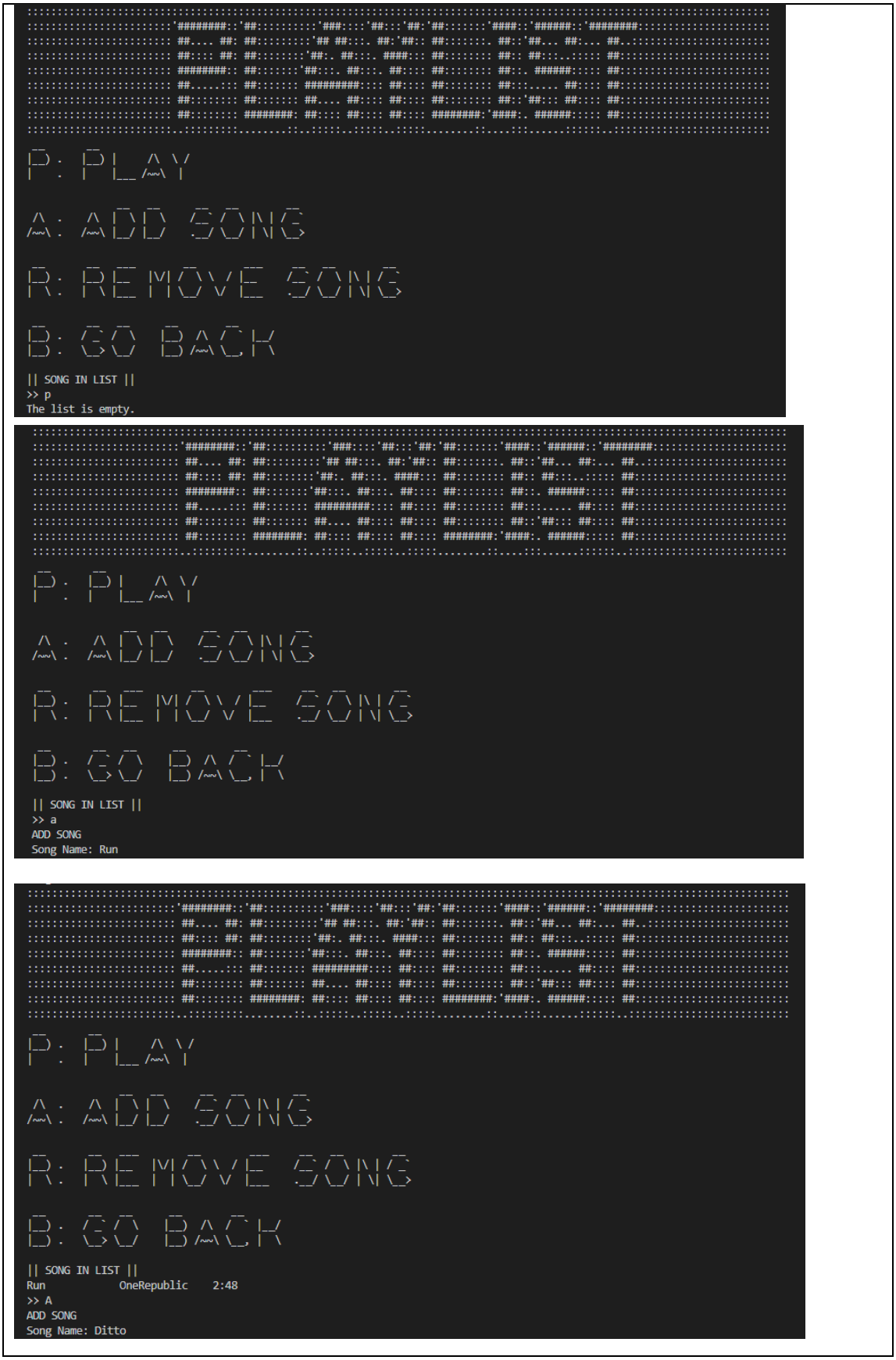

>> s
12345678901234567890123456789012345
Mutual Shawn_Mendes 2:28
Pretty Scott_Helman 2:35
No_Chill Cheat_Codes 2:48
Run OneRepublic 2:48
Palm_Springs Virginia 2:50
Stop AJ_Mitchell 2:57
Lighthouse Peder_Elias 3:02
Pink_cheeks Eldon 3:04
Ditto NewJeans 3:06
Eyes_Closed Ed_Sheeran 3:15
Society Valley 3:41
Breakeven The_Script 4:21
>> c
CREATE USER
User Name: Amy
User Passwd: 411
>> c
CREATE USER
User Name: Bob
User Passwd: hieee

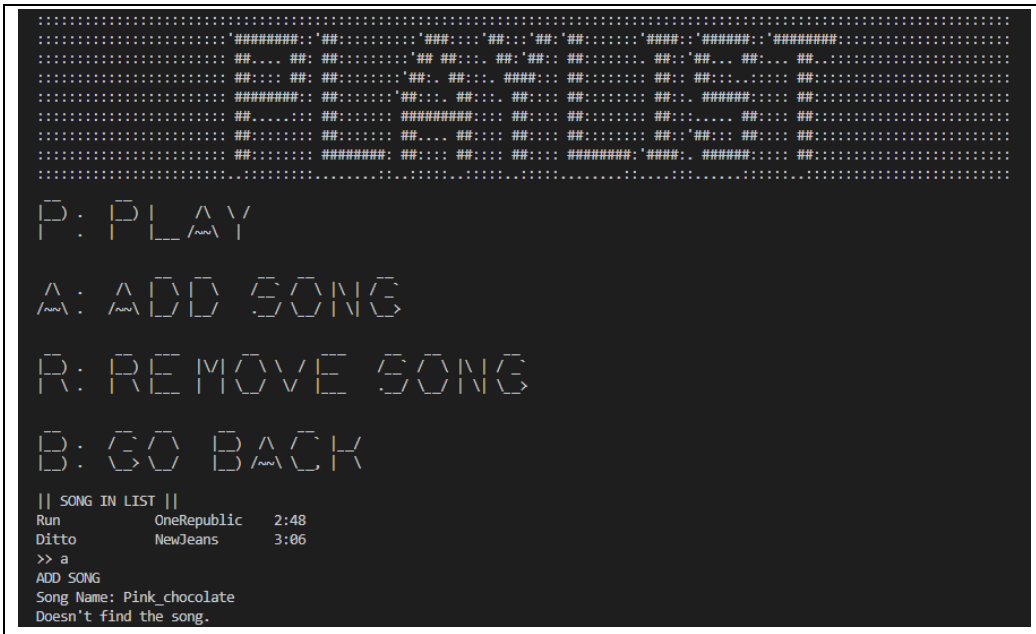

>> U
Amy Bob
>> i
LOG IN
User Name: Roger
User Passwd: ttt
The User doesn't exist

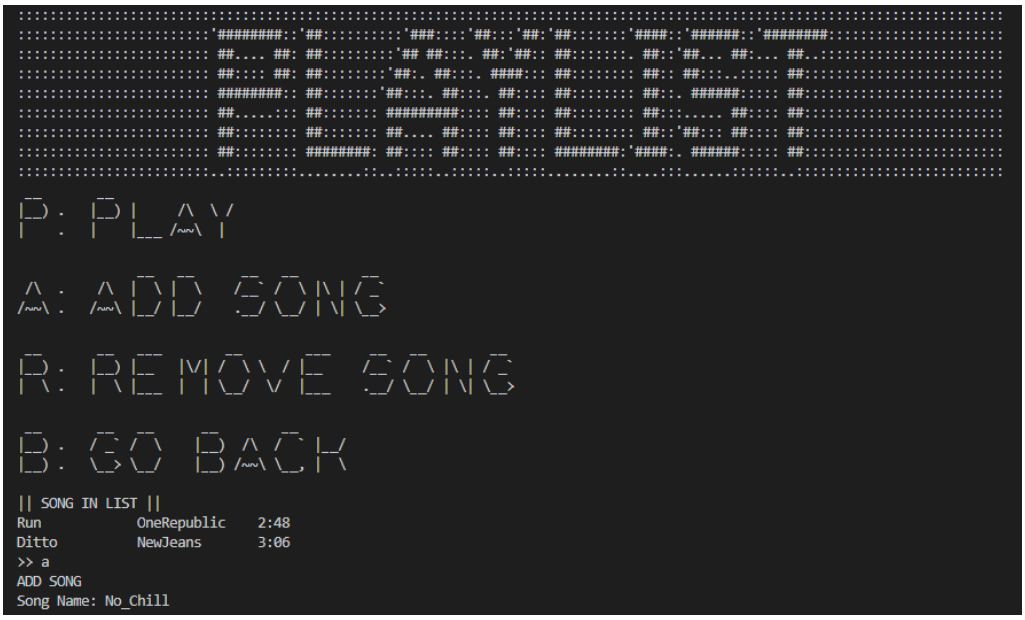

>> i
LOG IN
User Name: Amy
User Passwd: 311
The Password is wrong
>> i
LOG IN
User Name: Amy
User Passwd: 411
Log In Successfully
Welcome, Amy

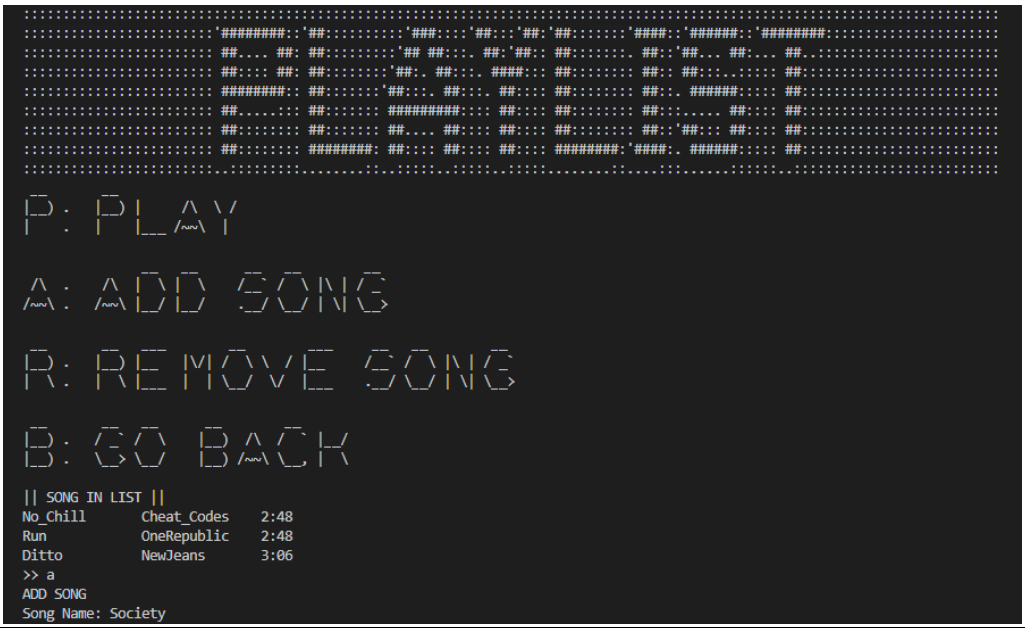

|| PLAY LIST: ||
>> c
CHOOSE LIST
list name: Happy
List doesn't exist
|| PLAY LIST: ||
>> a
ADD LIST
List Name: Summer

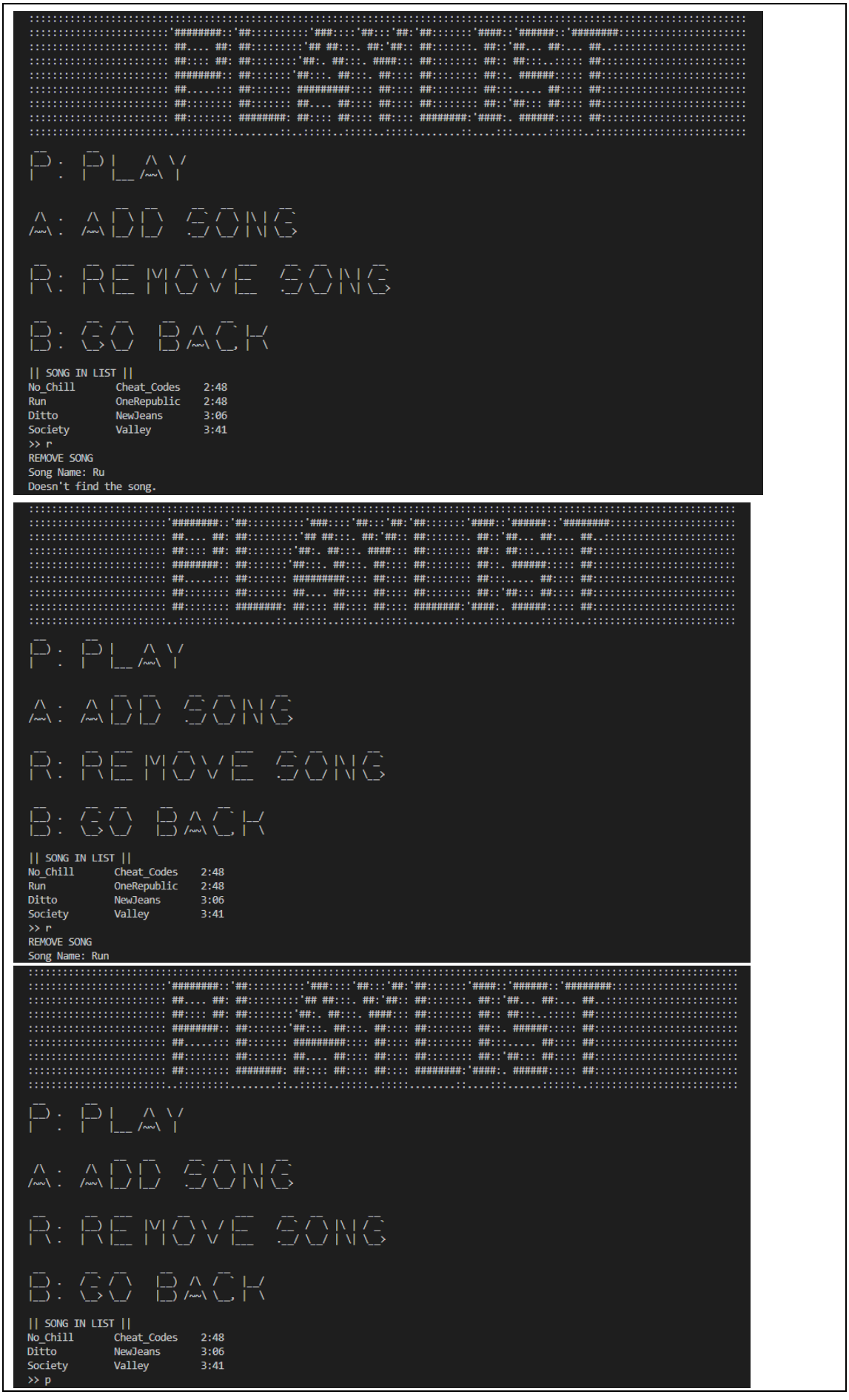

|| PLAY LIST: Summer ||
>> a
ADD LIST
List Name: Drive
|| PLAY LIST: Drive Summer ||
>> c
CHOOSE LIST
list name: Drive


|| SONG IN LIST ||
>> p
The list is empty.
|| SONG IN LIST ||
>> a
ADD SONG
Song Name: Run
|| SONG IN LIST ||
Run OneRepublic 2:48
>> A
ADD SONG
Song Name: Ditto

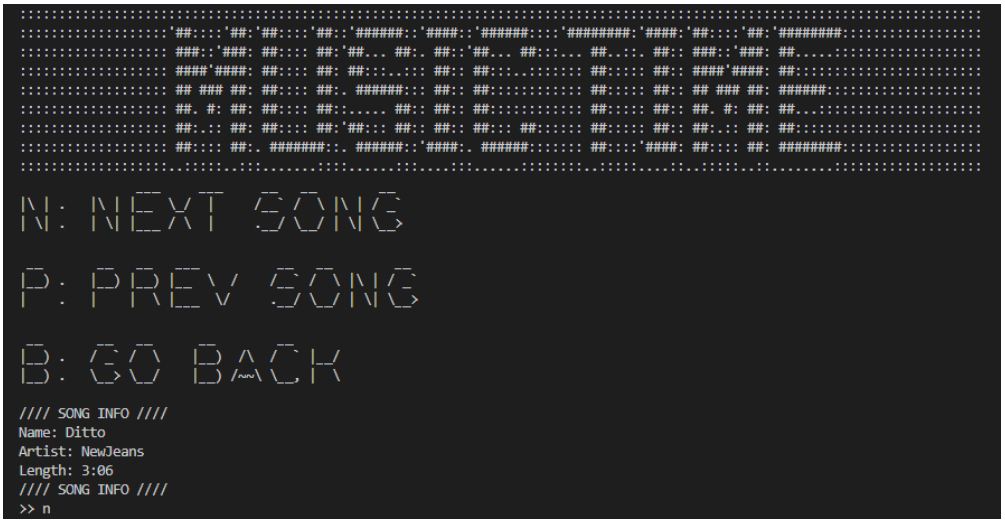

|| SONG IN LIST ||
Run OneRepublic 2:48
Ditto NewJeans 3:06
>> a
ADD SONG
Song Name: Pink_chocolate
Doesn't find the song.

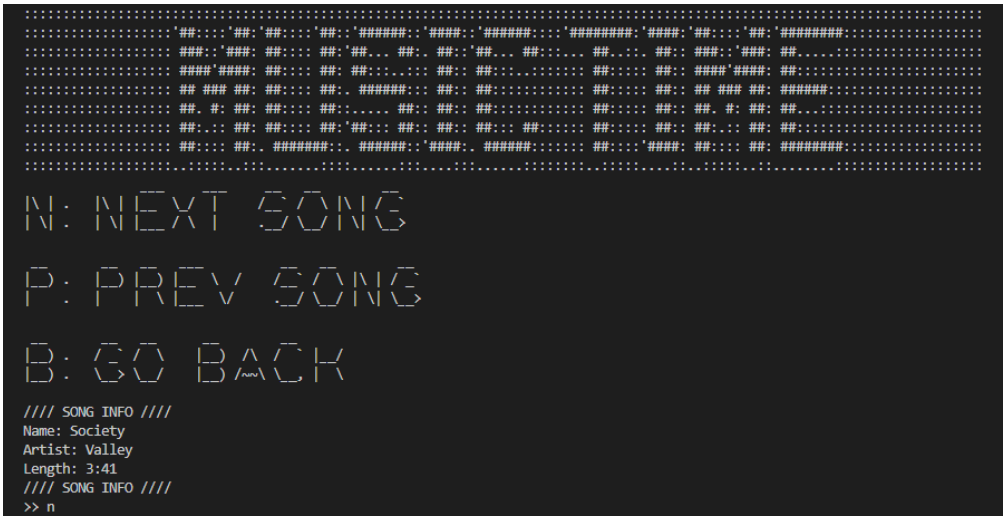

|| SONG IN LIST ||
Run OneRepublic 2:48
Ditto NewJeans 3:06
>> a
ADD SONG
Song Name: No_Chill


|| SONG IN LIST ||
No_Chill Cheat_Codes 2:48
Run OneRepublic 2:48
Ditto NewJeans 3:06
>> a
ADD SONG
Song Name: Society


|| SONG IN LIST ||
No_Chill Cheat_Codes 2:48
Run OneRepublic 2:48
Ditto NewJeans 3:06
Society Valley 3:41
>> r
REMOVE SONG
Song Name: Ru
Doesn't find the song.
|| SONG IN LIST ||
No_Chill Cheat_Codes 2:48
Run OneRepublic 2:48
Ditto NewJeans 3:06
Society Valley 3:41
>> r
REMOVE SONG
Song Name: Run
|| SONG IN LIST ||
No_Chill Cheat_Codes 2:48
Ditto NewJeans 3:06
Society Valley 3:41
>> p


//// SONG INFO ////
Name: No_Chill
Artist: Cheat_Codes
Length: 2:48
//// SONG INFO ////
>> n

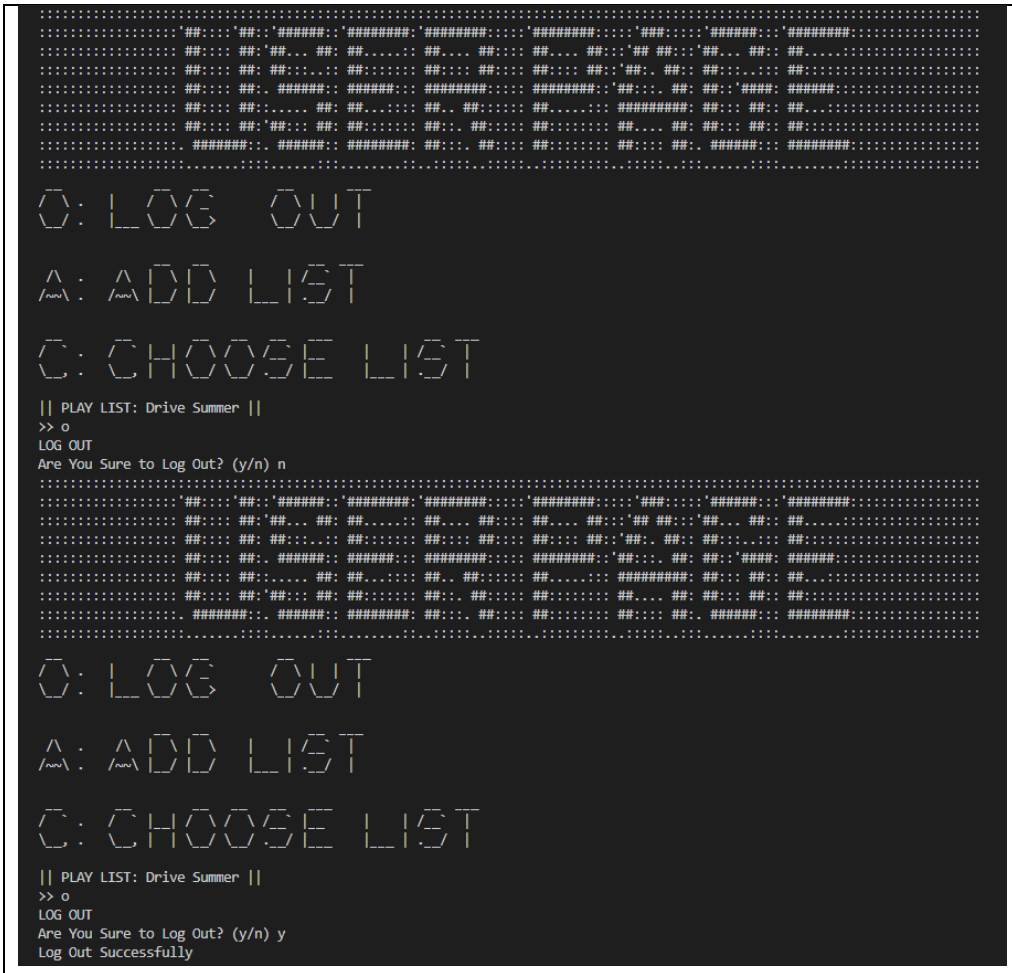

//// SONG INFO ////
Name: Ditto
Artist: NewJeans
Length: 3:06
//// SONG INFO ////
>> n

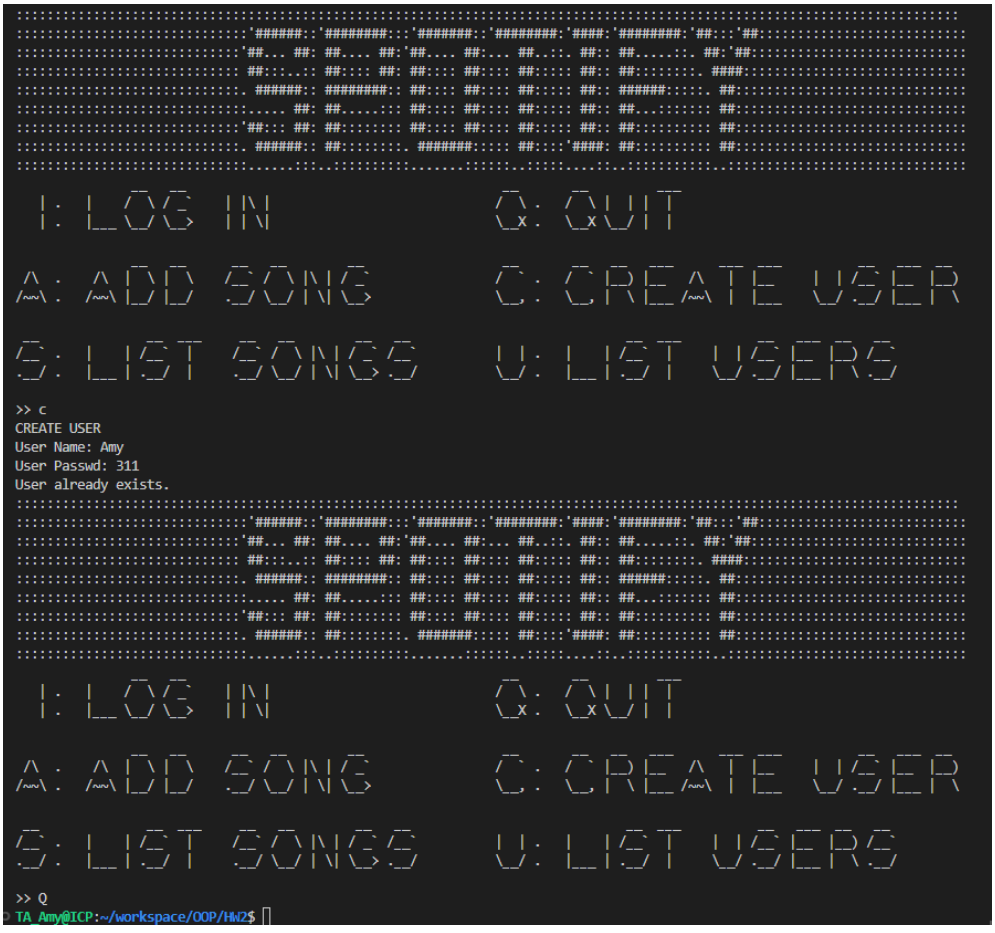

//// SONG INFO ////
Name: Society
Artist: Valley
Length: 3:41
//// SONG INFO ////
>> n


//// SONG INFO ////
Name: No_Chill
Artist: Cheat_Codes
Length: 2:48
//// SONG INFO ////
>> p


//// SONG INFO ////
Name: Society
Artist: Valley
Length: 3:41
//// SONG INFO ////
>> b


|| SONG IN LIST ||
No_Chill Cheat_Codes 2:48
Ditto NewJeans 3:06
Society Valley 3:41
>> b


|| PLAY LIST: Drive Summer ||
>> o
LOG OUT
Are You Sure to Log Out? (y/n) n
|| PLAY LIST: Drive Summer ||
>> o
LOG OUT
Are You Sure to Log Out? (y/n) y
Log Out Successfully


>> c
CREATE USER
User Name: Amy
User Passwd: 311
User already exists.
>> Q
TA_Amy@ICP:~/workspace/OOP/HW2$

- ✓ Reference
  - How to sort a vector:
    https://www.geeksforgeeks.org/sorting-a-vector-in-c/
  - How To Create And Use Makefile In C++
    https://www.softwaretestinghelp.com/cpp-makefile-tutorial/

[Problem 2]: Duel game (simple)

In the recent course, we learned the concept of OOP inheritance, which helps us to have better extensibility when declaring objects. In this problem, there will be a base class called Role, which includes the basic HP volume, attack power, etc. of a character. And there will be various child classes based on different types of roles, which will be described in more detail below.

- `class Role`
  - int hp: The HP of the character, When the blood volume is 0 or less than 0, the character loses.
  - int attack: The attack of the character, represents the damage dealt by each attack.
  - int defense: The character's defense power, if the character is in a defensive state, it can resist part of the damage when it is damaged.
  - int speed: The movement speed of characters, fast characters can move first.
  - bool isDenfense: It is used to record whether the character is in a defensive state, and the state is released after each attack is resisted.

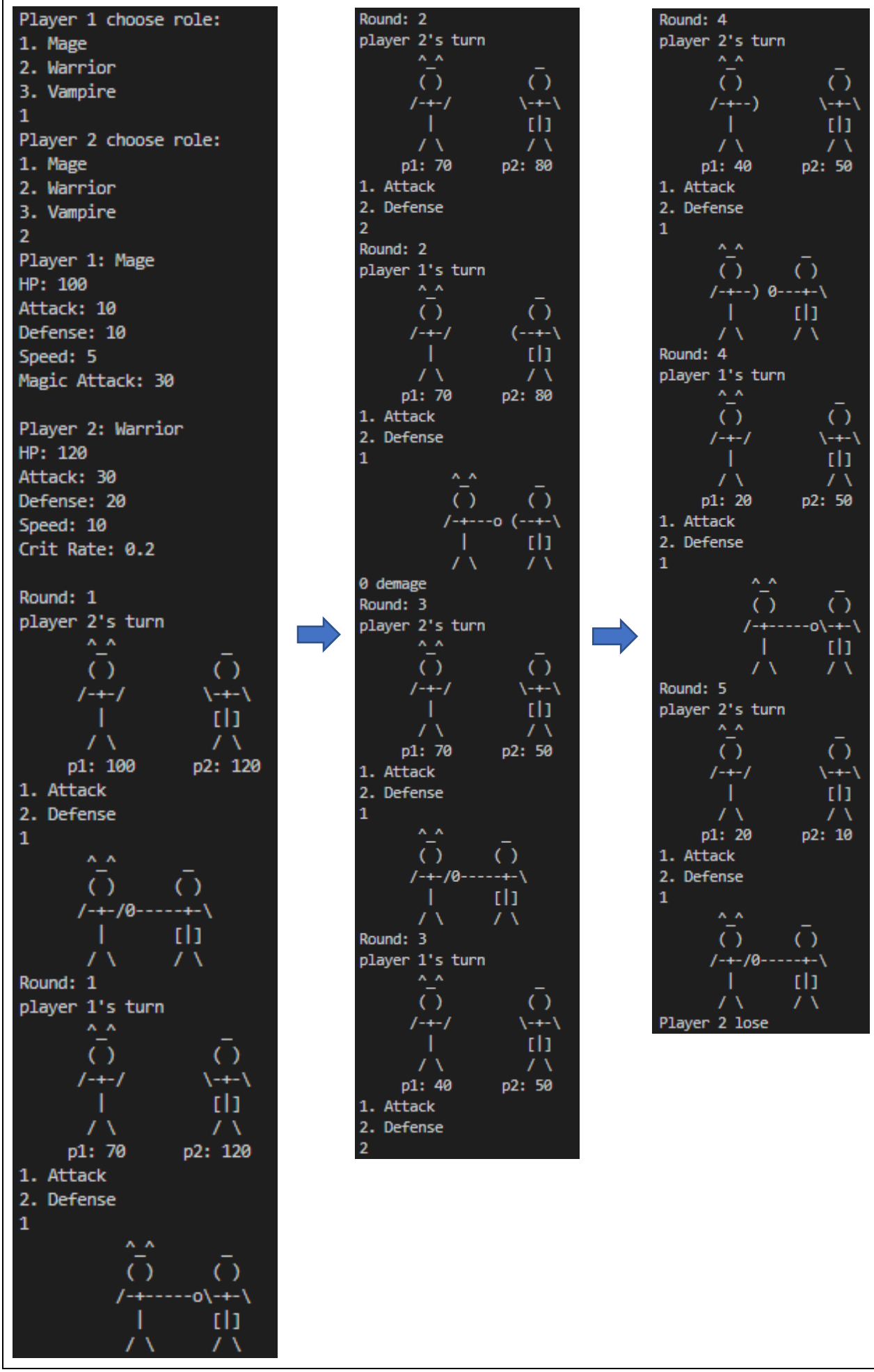
- `class Mage: public Role`
  - int magicAttack: The character's special ability can add magic damage when attacking, and this damage will not be resisted by the defense.
- `class Warrior: public Role`
  - float critRate: A decimal number between 0 and 1. When a warrior attacks, there is a chance to trigger a critical strike. This parameter is used to store the probability of a critical strike. The random method is srand(time(NULL)). And the case of a critical strike is rand()%100 < 100 * critRate. When a critical strike is triggered, the attack power will be doubled.
- `class Vampire: public Role`
  - float lifeSteal: A decimal number between 0 and 1. In the turn of the vampire's action, the vampire can suck the opponent's blood to itself according to the value of the lifeSteal * its own max HP, and the maximum fills up to its own full HP volume.

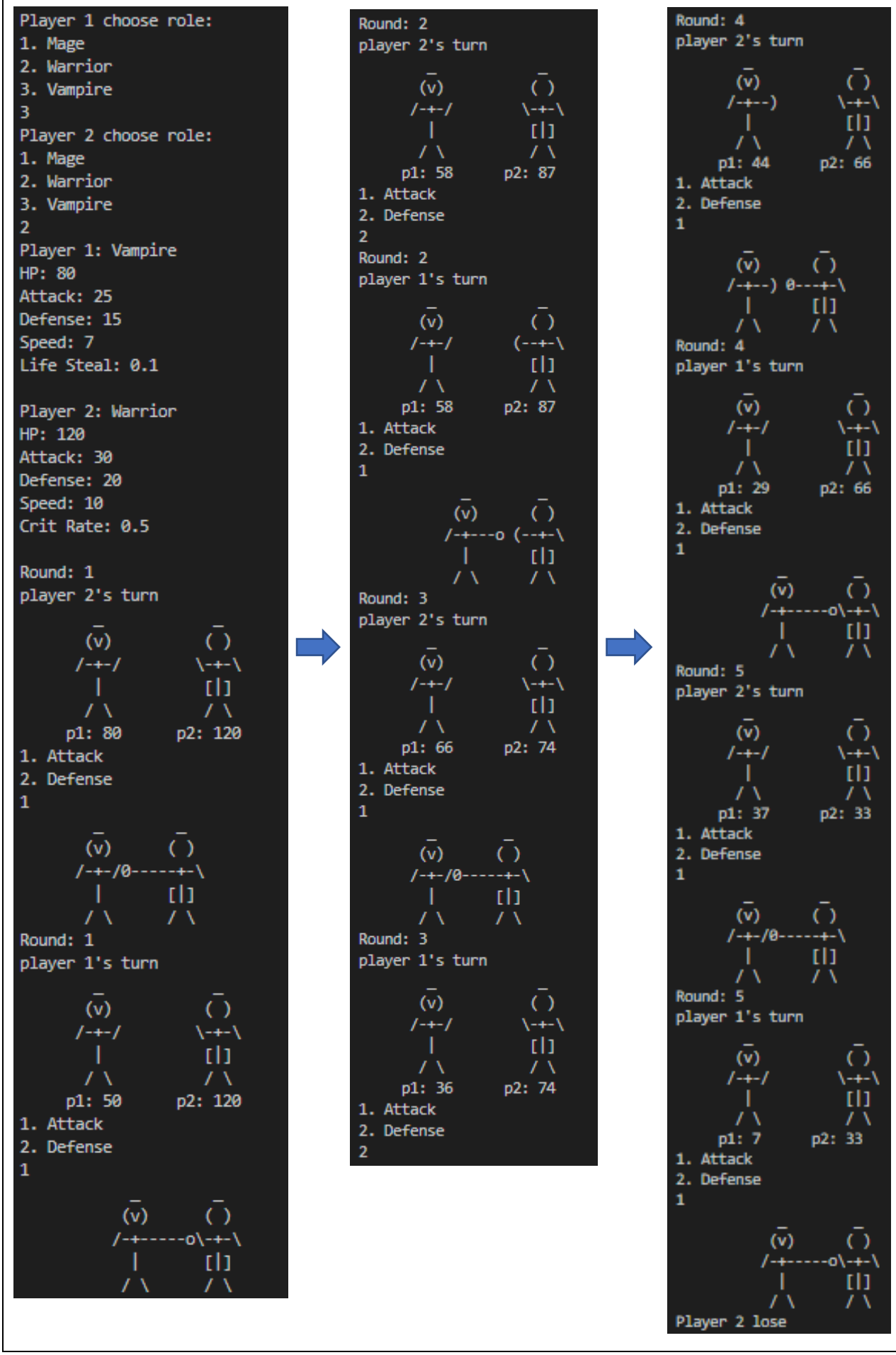
- `Main function`
  - Step 1: Two players choose their respective roles, note that they can choose the same character.
  - Step 2: Player 1, Player 2 set parameters according to the characteristics of their roles.
  - The battle starts until one player's HP returns to 0.
  - In each round (one action for each side is a round), the player with the fastest speed will make moves first.
  - It will display different appearances on the terminal according to the player's role and actions.
  - The character's kaomoji will be placed in the folder

Execution result is shown as below:

```
$ ./hw2-2
```

Output Sample:


Player 1 choose role:
1. Mage
2. Warrior
3. Vampire
1
Player 2 choose role:
1. Mage
2. Warrior
3. Vampire
2
Player 1: Mage
HP: 100
Attack: 10
Defense: 10
Speed: 5
Magic Attack: 30
Player 2: Warrior
HP: 120
Attack: 30
Defense: 20
Speed: 10
Crit Rate: 0.2
Round: 1
player 2's turn
p1: 100
p2: 120
1. Attack
2. Defense
1
Round: 1
player 1's turn
p1: 70
p2: 120
1. Attack
2. Defense
1
Round: 2
player 2's turn
p1: 70
p2: 80
1. Attack
2. Defense
2
Round: 2
player 1's turn
p1: 70
p2: 80
1. Attack
2. Defense
1
0 demage
Round: 3
player 2's turn
p1: 70
p2: 50
1. Attack
2. Defense
1
Round: 3
player 1's turn
p1: 40
p2: 50
1. Attack
2. Defense
2
Round: 4
player 2's turn
p1: 40
p2: 50
1. Attack
2. Defense
1
Round: 4
player 1's turn
p1: 20
p2: 50
1. Attack
2. Defense
1
Round: 5
player 2's turn
p1: 20
p2: 10
1. Attack
2. Defense
1
Player 2 lose


Player 1 choose role:
1. Mage
2. Warrior
3. Vampire
3
Player 2 choose role:
1. Mage
2. Warrior
3. Vampire
2
Player 1: Vampire
HP: 80
Attack: 25
Defense: 15
Speed: 7
Life Steal: 0.1
Player 2: Warrior
HP: 120
Attack: 30
Defense: 20
Speed: 10
Crit Rate: 0.5
Round: 1
player 2's turn
p1: 80
p2: 120
1. Attack
2. Defense
1
Round: 1
player 1's turn
p1: 50
p2: 120
1. Attack
2. Defense
1
Round: 2
player 2's turn
p1: 58
p2: 87
1. Attack
2. Defense
2
Round: 2
player 1's turn
p1: 58
p2: 87
1. Attack
2. Defense
1
Round: 3
player 2's turn
p1: 66
p2: 74
1. Attack
2. Defense
1
Round: 3
player 1's turn
p1: 36
p2: 74
1. Attack
2. Defense
2
Round: 4
player 2's turn
p1: 44
p2: 66
1. Attack
2. Defense
1
Round: 4
player 1's turn
p1: 29
p2: 66
1. Attack
2. Defense
1
Round: 5
player 2's turn
p1: 37
p2: 33
1. Attack
2. Defense
1
Round: 5
player 1's turn
p1: 7
p2: 33
1. Attack
2. Defense
1
Player 2 lose