VE280 - Programming & Elem. Data Structures

Project 2

1. Hardworking Guy with a Touching Fish Heart...

Last year, you took an internship at SilverFOCS Inc., making web games in their game department. Life was busy but quite interesting there.

"Why not try a new company this year?"

Finding an internship was not easy for a full-time internationally-educated student in SJTU. Least working hours should not be too many. The company should not be too far. Anyway, you made it, with an internship in this Junior Involution Information & Technology Inc.

The first day of your work...

The time you arrived at the office, the boss was busy playing with his son, running here and there. Your mentor was shouting at some of your co-workers. He stopped to send you an internal link, "Choose one to work on. I will go back to you later.", then continued on his intense meeting.

You quickly skimmed through the task lists. Most of them were done, with three uncompleted tasks left.

- Write a simulation of an animal world
- Work on a project named Simple Twitter
- Write a chat bot for internal chat application

The simulation of animal world is for your boss 7-year-old naughty boy. You know, kids are energetic and curious about the world. Outside your office, this happy father and his son were still enjoying themselves in a warm, sunny morning.

You thought that Simple Twitter was something like Twitter, but it turns out not to be. The database of Simple Twitter is based on plain texts. You see a lot of files in a mess in the file tree. The project leader is your tough angry mentor over there. And your co-worker never backup the database. They just lost all their data this morning. "No wonder my mentor was so mad now..."

As for the chat bot...The chat application is based on C++, and it also uses plain texts as its database.

"Hard to make a choice..."

At the same time, your friend was playing happily with the repeater bot in Class of 2019 QQ Chat Group. That inspired you. You got some ideas and then started to work on the chat bot.

2. The Bot

The main task of the bot is to keep waiting for chat messages, receive one, parse it and return a response or do nothing (Fig. 1).

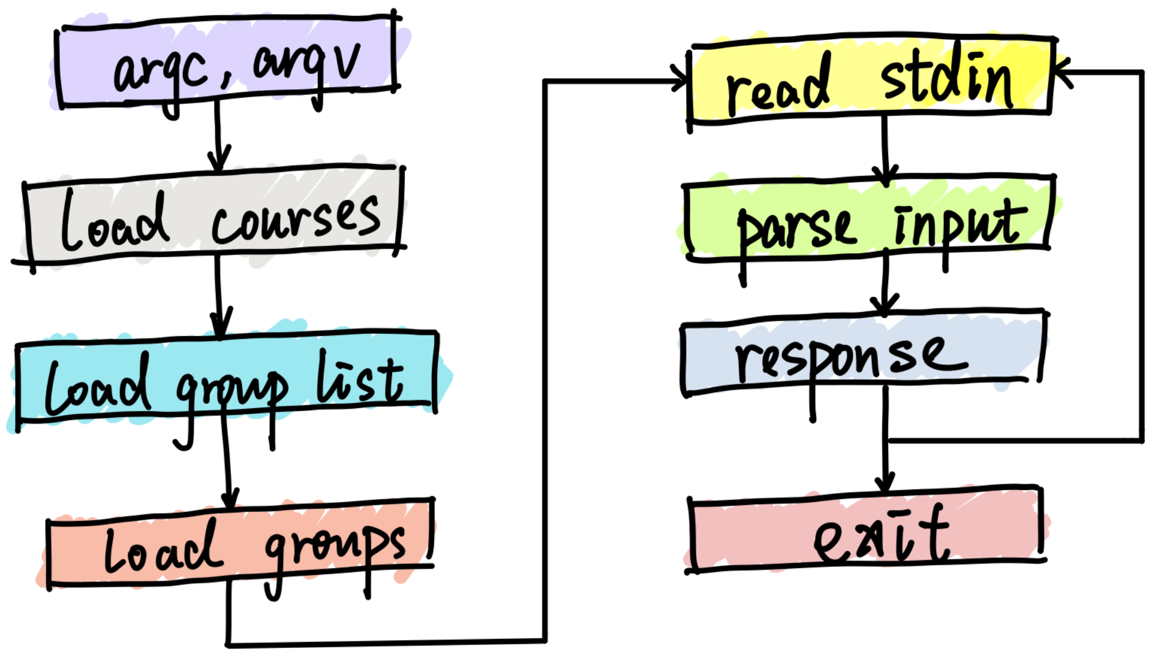

Figure 1: Flow Chart of the bot

The bot has to attend some chat groups. The bot will only accept message from these groups. These groups are stored in a group list. If a message is not from any of the groups, this message will be ignored. The format of the message is introduced in Section 4.3

Each group has its admins. When in one chat group, only the admins can stop the bot.

Here, to simplify the situation, we assume that once the bot is stopped in one group, the program will be stopped and exit. Actually, in real world, if the bot has been stopped in one group, we can still interact with it in other groups.

The working directory should be the root directory of the project. The root directory is the directory of `Makefile` or `CMakeLists.txt`. Otherwise it may have problems with finding data file.

Each time the bot starts, it will

- load the course data file
- load the group list from the file, then according to the file, load details for each group on the list
- initialize a random seed,

All of them are specified by program arguments as

```
./bot [course data file] [group list file] [random seed]
```

3. Bot Manual

- **NAME**

  bot -- a chat bot

- **SYNOPSIS FOR A COMMAND**

  #**command** [keyword]

- **DESCRIPTION**

  - The bot keeps waiting for a message. When a message arrives, it attempts to parse it and return a response or just do nothing.
  - If the message can be parsed into a known command, it will execute the command. The commands are as follows:
    - course: find all the courses that contain the keyword
    - instructor: find all the instructors that contain the keyword
    - help: show help message
    - time: show the time when the message was sent
    - stop: (For group admins only) stop the bot
  - All commands start with `#`. A message is regarded as a command only if it starts with `#` and can be parsed into one of the known commands above. A command is made up of a prefix (#), a command name (shown below), and keywords (optional). Like

```
#course ve482
#instructor Korosensei
#help
#time
#stop
```

  - If the message cannot be parsed into one of the known commands shown above,
    - Sometimes it will repeat the message back. The probability is defined by `REPEAT_ROLL` (unit: %).

      Sample

      Input:

      ```
      haha
      ```

      Output:

      ```
      haha
      ```

- Sometimes it will admire the message by adding a prefix to the original message. For the prefix, the bot will flip a coin. If the value of the coin is **1** (or head), it will use **Respect**, otherwise it will use **I really admire**. The probability of the overall admire action is defined by `ADMIRE_ROLL` (unit: %).

  Sample

  Input:

  ```
  Poor at C++
  ```

  Output:

  ```
  Respect Poor at C++
  ```

  OR

  ```
  I really admire Poor at C++
  ```

  Please be aware of the space after the prefix
- Otherwise it will do nothing.

4. I/O Standard

4.1 Data file for courses/instructor

A sample course data file is provided with the starter files and it looks like

```
6
VE482,Intro to Operating System,Manuel Charlemagne
VE477,Intro to Algorithm,Manuel Charlemagne
VR080,Intro to Involution,Shi Li
VE280,Programming & Elem. Data Structure,Paul Weng
VR418,Intro to coffee,Small Arms Big Heart
VE999,Intro to noble Elm,Yue
```

First line is the number of the courses `n`.

Assume that $1 \leq n \leq 2147483647$. 2147483647 is the maximum value of `int`. This is actually not a limit. It means that you should use `int` to store the number.

Each line below represents one course in the following format

```
[Course code],[Course name],[Instructor]
```

Note that a comma, with the symbol `","`, is the separator. There will be no commas in the name/code/instructor, which also applies to all the other file/input.

4.2 Group list file and group detail file

A sample group list file is provided with starter files and it looks like

```
cases/groups
2
-6
-1
```

The first line is the path of the directory that contains all the file with group information.

The path is the relative path from the root directory of your project. The root directory is the directory of all your c++ source files.

Make sure that the bot is started at the root directory of the project. Otherwise it may have problems with finding data file.

The second line is the number of the groups that the bot has attended `n`.

Assume that $1 \leq n \leq 2147483647$. 2147483647 is the maximum value of `int`. This is actually not a limit. It means that you should use `int` to store the number.

Starting from the third line, each line represents a group. The group name is a string. (QQ chooses to use negative numbers to represent the id of its chat group and that is the sample here)

For the sample file, the detail file for group `-6` is `${PROJECT_ROOT_DIR}/cases/groups/-6`

Group detail files are provided with starter files and they look like

```
2
60381
Dio
```

The first line is the number of admins of this group `n`.

Assume that $1 \leq n \leq 2147483647$. 2147483647 is the maximum value of `int`. This is actually not a limit. It means that you should use `int` to store the number.

Starting from the second line, each line stands for one admin.

4.3 Message input

All the messages come from the standard input in the format given below

```
[time],[group],[user],[content]
```

- time: the time when the message was sent
- group: the group where the message is from
- user: the sender of the message
- content: content of the message

Input samples are provided in the starter file named `query`.

If a message is not from any of the groups in the group list, this message will be ignored, which means that the bot will do nothing.

Note that a comma, with the symbol `","`, is the separator. There will be no commas in time/group/user/content.

4.4 Commands

No matter what response it will be, add a new line at the end.

- **course**

Finds all the courses that contain the keyword. It should be case sensitive.

```
13:00:31,-1,Den-O,#course 4
```

```
Course Code: VE482
Course Name: Intro to Operating System
Instructor: Manuel Charlemagne
Course Code: VE477
Course Name: Intro to Algorithm
Instructor: Manuel Charlemagne
Course Code: VR418
Course Name: Intro to coffee
Instructor: Mr. Teapot

```

ERROR: Missing keyword:

```
13:00:31,-1,Natsu Dragneel,#course
```

```
Oh, input the search keyword first...

```

ERROR: Not found:

```
13:00:31,-1,wowaka,#course typescript
```

```
I don't know this course

```

- **instructor**

Find all the instructors that contain the keyword. It should be case sensitive.

```
13:00:32,-1,hachi,#instructor e
```

```
Instructor: Manuel Charlemagne
Courses: VE482 VE477
Instructor: Paul Weng
Courses: VE280
Instructor: Mr. Teapot
Courses: VR418
Instructor: Yue
Courses: VE999

```

ERROR: Missing keyword:

```
13:00:32,-1,DECO*27,#instructor
```

```
Oh, input the search keyword first...

```

ERROR: Not found:

```
13:00:32,-1,Harumaki Gohan,#instructor null
```

```
I don't know this instructor

```

- **time**

```
13:31:40,-1,Orangestar,#time
```

```
13:31:40

```

- **stop**

```
13:31:40,-1,admin X,#stop
```

```
Good night. I am going to sleep

```

ERROR: Unauthorized

```
13:31:40,-1,bad student B,#stop
```

```
You are not qualified to stop me

```

- **help**

```
13:31:40,-1,40mP,#help
```

```
Cheat Sheet for Repeater Bot:
Notice: Commands start with #
   course [keyword]:     find all the courses that contain the keyword
   instructor [keyword]:    find all the instructors that contain the keyword
   help:    show help message
   time:    show the time when the message was sent
   stop:    (For bot admins only) stop the bot

```

5. Errors and Exceptions

- Errors may happen with the command line arguments. In this case, print the error and exit the program
  - If any of the arguments is missing
  - The random seed is not a number
  - The random seed exceeds the range of integer
  - The course data file cannot be open
  - The group list file cannot be open

```
./bot [course data file] [group list file] [random seed]
```

- Assume that all the database/group/group-list files are correctly generated described in Section 4. That means that all files will be in the right format and each group on group list will have its configuration file.
- Assume that all the message input can be correctly parsed into

  ```
  [time],[group],[user],[content]
  ```

- When encountering an error that is mentioned in Section 4.4, the bot will reply an error message then wait for the next message. The bot will not stop.

6. Limitations and Notes

1. For your convenience, most of the error/prompt messages are written in `constants.h`. You may not define any global variables yourself.
2. Randomness will bring much trouble to grading. So we offer two files `rand.cpp` and `rand.h`. Use
   - `int flipCoin()` to decide which admire prefix to use

   - `RespChoice randomResponse()` to decide to repeat or to admire or do nothing
3. Never trust user input. The message content may come in various ways that you would never dream of. If handled well, life will be easier. We make some choices below.
   - `#stop` is a command. `     #stop` and `#stop1` is only a plain message. `#stop jiit` will be the same result as `#stop` .
   - `#course ve482` is a command but `#courseve482` nor `#courseve482 ve482` is just a plain message.
   - `#course` or `#course  ` should result in an error.
   - `#course         ve482` will be the same as `#course ve482`.
   - In `#instructor Namikaze Minato` , the keyword is `Namikaze Minato`.
4. In writing your code, you may use the following standard header files: `<iostream>`,`<fstream>`, `<sstream>`, `<iomanip>`, `<string>`, `<cstdlib>` and `<cassert>`.
5. Pass large structures by reference rather than value. Where appropriate, pass constant references / pointers-to-constant. Do not pass lots of little arguments when you can pass an appropriate, larger structure instead.
6. All required output should be sent to the standard output stream; none to the standard error stream.
7. You should strive not to duplicate identical or nearly-identical code, and instead collect such code into a single function that can be called from various places. Each function should do a single job, though the definition of "job" is obviously open to interpretation. Most students write too few functions that are too large.

7. Source Code Files and Compiling

There are three code files located in the `starters.zip` from our resources:

`rand.h`: The header file which defines the random methods

`rand.cpp`: The cpp file which implements the random methods

`constants.h`: The header file which defines all the constants

You should copy this file into your working directory. **DO NOT modify it!**

You need to write three other source code files.

`bot.h`: contains the declarations for all the functions you write

`bot.cpp`: contains all the implementations of the functions written in `bot.h`

`main.cpp`: should only contain the main function.

After you have written these files, you can type the following command in the terminal to compile the program:

```
g++ -Wall -o bot bot.cpp rand.cpp main.cpp -std=c++11
```

This will generate a program called `bot` in your working directory. In order to ensure that the online judge compiles your program successfully, you should name you source code files exactly like how they are specified above.

8. Submission

Due at `23:59, June 25th`.

Zip your `bot.cpp`, `bot.h`, `main.cpp` in a tar/zip file and submit it via JOJ.

9. Grading

Your program will be graded along three criteria:

- Functional Correctness
- Implementation Constraints
- General Style

Functional Correctness is determined by running a variety of test cases against your program, checking against our reference solution. We will grade Implementation Constraints to see if you have met all of the implementation requirements and restrictions. General Style refers to the ease with which TAs can read and understand your program, and the cleanliness and elegance of your code. For example, significant code duplication will lead to General Style deductions.

10. Endings

Special Thanks to BoyanZh and his QQGroupRepeater.

If you have not experienced his repeater yet, you can reach for the chat group for 2019 class at 730448868 via QQ (Content is in Chinese)

For his bot, things may be a little bit more complicated but also easy to understand though...

I DON'T LIKE WRITING DOCUMENTATIONS....

But it is very important. Hard..