CMPS 12B

Introduction to Data Structures

Programming Assignment 4

Objective

The purpose of this assignment is to give you practice solving larger problems that involve multiple data structures, and with choosing the correct data structure based on the problem requirements.

Task

Your task is to write a program that lets the user play choose-your-own-adventure games. The games are described in text files with a specific format described in this assignment. Your program will read in an adventure files, parse it, and build up internal data structures representing all the rooms and options available in the adventure. The program then starts by showing the description of the first room and giving the player a list of options. The player then chooses a single letter option which leads to a new room. The player also has several other single letter commands for quitting, restarting the adventure from the beginning, and so on.

You may use Java or C for this assignment. You may use any data structures to represent the rooms and options of the adventure. There is no specific performance requirement, but you are required to defend your decisions and explain your design in a design document. If you are pair programming, this design document should also be written as a pair (both people present at all times working on the document).

The specifications for your program are presented in the format of a Unix man page.

NAME

cyoa - choose-your-own-adventure utility

SYNOPSIS

cyoa adventurefile

DESCRIPTION

The cyoa utility reads in lines from the given file. Once the entire file is finished, the utility starts a play session to play the adventure described in the file. Play starts in the first room described in the file. Players type one letter to make a choice or enter a command. Play continues until an error is encountered or the player enters the quit command ('q').

COMMANDS

a - l	Standard choices defined by the adventure.
r	Restart the adventure in the first room. Loses any existing saved undo state.
q	Quit the game.
y	Show information about the adventure. Prints one room per line, including tag of the room, then tags for destinations of all possible options from that room.
z	Undo the previous choice, go back to previous room (can be done multiple times).

EXIT STATUS

The following exit status codes are returned:

0	No errors were detected.
1	Invalid file, invalid contents while parsing file, or other internal adventure error.

Adventure File Format

The file format for adventure files is as follows. Any blank line is entirely ignored. Any non-blank line must start with a single character command, then a space, then may include any number of content characters followed by a newline. Content characters can be any non-newline characters.

The commands allowed are:

```
r     Add a new blank room with a tag given by the contents. The room has no
      options when created.
d     Add a line of description to the most recently added room. If no room has
      been added then this command generates an error.
o     Add a new option to the most recent room. The content is the text of the
      option that will be displayed to the player.
t     Update the destination tag of the most recently added option to the contents.
      The tag must match the tag of a room that appears somewhere in the input file
      (that room may occur later in the file than this command).
```

Example Adventure

Here is an example two room adventure. The player starts in room1 and can choose to move to room2 and back to room1.

```
r room1
d This is room one.
d You are awake.
o Go to sleep.
t room2
o Stay awake.
t room1

r room2
d You are sleeping.
o Stay asleep.
t room2
o Wake up.
t room1
```

Example Playthrough

Here is an example playthrough of the two room adventure. Bold text indicates player input.

```
$ java cyoa two.adventure

This is room one.

You are awake.

a  -  Go to sleep.
b  -  Stay awake.

a
[Go to sleep.]

You are sleeping.

a  -  Stay asleep.
b  -  Wake up.

a
[Stay asleep.]

You are sleeping.

a  -  Stay asleep.
b  -  Wake up.

z
[information]

room1 : room2 room1
room2 : room2 room1

You are sleeping.
```

```
a  -  Stay asleep.
b  -  Wake up.

b
[Wake up.]

This is room one.

You are awake.

a  -  Go to sleep.
b  -  Stay awake.

q
[quit]
```

Play Requirements

The adventure starts in the first room described in the text file. If the player tries to make a choice that is invalid an error message should be displayed but the program must not end; the player must be allowed to try a different choice. If the destination tag does not match any room tags, the program must show an error message and exit with failure status.

The information command must show information about all the room in alphabetically sorted order. Choice destination tags for each room must be shown in the order the options were added in the input file, matching the order the options are displayed to the player.

The undo command must flip back to the previous room visited by the player. If the player is at the start of the adventure, the undo command should display an error and let the player continue playing. Multiple undo commands must rewind the play history of the player, potentially all the way back to the initial state. The player is allowed to do many moves, undo some moves, make more moves, then undo any number of moves. For example, if the player moves 1 -> 2 -> 3, then does one undo back to room 2, then moves to room 4 (1 -> 2 -> 4), the player can then undo once back to room 2, and undo again back to room 1.

Two short example adventures can be found in the AFS directory for the assignment:

```
/afs/cats.ucsc.edu/users/r/nwhitehe/cmps12/asg4/
```

Design Document

You are required to prepare and submit a design document for your project. Your design document must be either a PDF or an ASCII text file. It should clearly explain how your project is organized, what data structures you are using, and why you choose those data structures. Your design document should describe the actual design of your submitted code. If your design changes over time, keep your design document updated to reflect the actual design. If you wish you may include a short section at the end describing the evolution of your design over time (this part is optional).

Note that your design document is designed to communicate with humans. It is more important to be clear and explain your ideas that to adhere to a specific format or be exhaustive in details. Your design document can include diagrams, tables, and code snippets. When describing which data structures you chose you should discuss the alternatives you considered and the trade-offs you made.

Demo Adventure

In addition to the provided test adventures, you are required to create an original adventure following the provided file format. Your adventure must be a coherent narrative without syntax errors. It must have at least 10 rooms and include rooms that indicate when the player has won (the description of the room should include a message such as, "You have won!"). It must be possible to beat the adventure. The subject matter must be appropriate to project to the class during lecture. Your adventure must be named "demo.adventure".

What to Submit

All files you turn in for every assignment and lab should begin with a comment block that includes your name, CruzID, class, date, filename, short description of the file's role in the assignment, and any special instructions related to the file. Also create a file called README. The README file should have the normal comment block, then list all the files being submitted (including itself) along with any special notes to the graders. If you are pair programming then both names must be included in every comment block, and both names and CruzIDs must be included in the README.

Your README should include the choices necessary to beat your demo adventure in one line (e.g. "To win choose: aabcbaabcbdaab").

Submit the following files:

```
      README
      design.pdf OR design.txt
      Makefile
      [your project files]
      demo.adventure
```

To submit, use the submit command.

```
      submit cmps12b-nojw.f14 asg4 files
```

Your makefile should have a target "test" that builds and runs your program with your demo adventure.