Background

Young Tommy A. is trapped in a diabolical computer game. Forced to wander a macabre landscape studded with old broken–down Sesame Street letters (like 'Q' and 'H'), Tommy can escape only by saying the password. The password is not secret; it is clearly written on the wall of one of the rooms of the cave where Tommy finds himself. But the password will not release Tommy until each of its letters has been collected in the room in which Tommy makes the magical incantation. The good news is that ample letters are available somewhere in the cave to construct the password and thereby escape.

The bad news is that Tommy is not alone. An ancient insidious evil wanders the halls of this forlorn world. The blackhearted, execrable villain Doctor Chillburna Plotz has been banished to this tortured world to atone for unspeakable misdeeds. The evil Doctor Plotz has learned of the password, and means to use it to escape, leaving poor Tommy A. forever trapped in a nightmare beyond imagining.

Playing the Game

You are to implement a simple version of this Adventure game. To run your program the player will type: `proj3` *name–of–data–file* (see the Project 2 handout [Handout 21] for a description of how to use command–line arguments). Your program should first read in a file that contains a description of the rooms of the cave. Then it should repeatedly read and execute a command from the player, then obtain and execute a command from the nefarious Doctor Plotz.

When prompting for a command from the player, your program must first print information about the player's current environment. This information must include the following:

- The name of the room the player is in.
- The description of that room.
- The password, if the password appears on the wall of the room.
- The letters that lie on the floor of the room. The letters need appear in no particular order.
- A list of the rooms that are adjacent to this one.
- An indication of the presence of the dastardly Doctor Plotz, if the evil doctor is in the same room as the player.
- The letters currently held by the player.

Once this information has been imparted to the player, the program will read a command from `stdin`. The command may not be longer than `MAX_COMMAND_LEN` characters. This limitation implies limitations on the length of a room name and on the length of the password. Legal commands include:

- `move` *name-of-room:* if the named room is adjacent to the one the player is currently in, move the player to the new room.
- `take` *letter:* if the named letter is in the current room, then remove it from the room and add it to the inventory of letters the player is carrying. The player may carry no more than `MAX_PLAYER_INVENTORY_SIZE` letters at one time.
- `drop` *letter:* if the named letter is in the player's inventory, then remove it from the inventory and place it in the current room. There is no limit to the number of letters that a room may hold.
- `wait`: do nothing for one turn.
- `say` *word*: say the given word. If the word is the password, and all the letters of the password are on the floor of the room the player is currently in, the player escapes the maze and wins the game. If a given letter appears more than once in the password, then it must also appear more than once in the room for the game to be won.
- `quit`: quit the game. This command is reserved for miscreants and social hangers–on.
- `help`: Print a list of available commands.

If the player enters a blank line, a new prompt should be given, and she/he should not be charged for a move. In all other cases, an error message should be printed and a new prompt generated (again without charging the player for a move).

You may, if you so choose, report the actions of the dreaded Doctor Plotz to the player *while they are in the same room.* However, the noisome Doctor Plotz's actions must not be reported when they are in different rooms.

After the player has moved, the reprehensible Doctor Plotz makes a move (as described below), and the main play loop continues.

Example

Here is an example of a part of a game you might play with your system (commands entered by the player are shown in italics):

```
umbc9%  p3 p3.dat
You are in the red room
This is a medium-sized red room
Letters that are here: dfforsw
Adjacent rooms:
  green room
The despicable Dr. Plotz is here.
You are not holding any letters.
What do you want to do?  move green room
You are in the green room
This is a large green room
You see scrawled on the wall ``the password is swordfish''
Letters that are here: his
```

```
Adjacent rooms:
  yellow room
  red room
You are not holding any letters.
What do you want to do?  take s
You are in the green room
This is a large green room
You see scrawled on the wall ``the password is swordfish''
Letters that are here: hi
Adjacent rooms:
  yellow room
  red room
You are holding the letter s
What do you want to do?  move red room
You are in the red room
This is a medium-sized red room
Letters that are here: dfforsw
Adjacent rooms:
  green room
The vile Dr. Plotz is here.
You are holding the letter s
What do you want to do?  take f
You are in the red room
This is a medium-sized red room
Letters that are here: dforsw
Adjacent rooms:
  green room
The nefarious Dr. Plotz is here.
You are holding the letters f and s
What do you want to do?  take w
You're already carrying 2 letters; you can't hold any more.
You are in the red room
This is a medium-sized red room
Letters that are here: dforsw
Adjacent rooms:
  green room
The infamous Dr. Plotz is here.
You are holding the letters f and s
What do you want to do?  move green room
You are in the green room
This is a large green room
You see scrawled on the wall ``the password is swordfish''
Letters that are here: hi
Adjacent rooms:
  yellow room
  red room
You are holding the letters f and s
What do you want to do?  move yellow room
You are in the yellow room
This is a small yellow room
Letters that are here: dhiorssw
Adjacent rooms:
  green room
You are holding the letters f and s
```

```
What do you want to do? say swordfish
You say "swordfish"
You are in the yellow room
This is a small yellow room
Letters that are here: dhiorssw
Adjacent rooms:
  green room
You are holding the letters f and s
What do you want to do? drop f
You are in the yellow room
This is a small yellow room
Letters that are here: fdhiorssw
Adjacent rooms:
  green room
You are holding the letter s
What do you want to do? say swordfish
You say "swordfish"
Poof! You have escaped from the maze.
Congratulations, you are the winner.
```

The Data File

The data file contains an exact description of a cave at the start of play. A sample data file is available in `~mayfield/p3.dat`. The program `~mayfield/make-p3-datafile` will generate other sample datafiles for your experimentation and enjoyment.

A data file contains a description of each room in the cave, of all the letters in those rooms, of the password to be used, and of the room in which the players will start. The data file consists of a series of `room`, `letters`, `password` and `start` descriptions *in any order.*

Each `room` description consists of five parts:

1. The word `room` on a line by itself.
2. The name of the room, again on a line by itself.
3. A one–line description of the room. The description will be no longer than `MAX_DESCRIPTION_LEN` characters.
4. The names of all adjacent rooms, each on its own line. No room will have more than `MAX_ADJACENT_ROOMS` adjacent rooms. Note that if room A is adjacent to room B, it does not necessarily follow that room B is adjacent to room A.
5. A blank line to indicate the end of the `room` description.

A `letters` specification consists of four parts:

1. The word `letters` on a line by itself.

2. A sequence of letters, on a line by itself.
3. The name of the room that holds these letters at the start of the game. A letter may appear more than once in a room. If more than one `letters` command appears for a given room, then all of the mentioned letters are included in the room.
4. A blank line to indicate the end of the `letters` description.

A `password` specification consists of four parts:

1. The word `password` on a line by itself.
2. The word that will serve as the password for both players, on a line by itself.
3. The name of the room on whose wall the password is to appear.
4. A blank line to indicate the end of the `password` description.

Each `start` specification consists of three parts:

1. The word `start` on a line by itself.
2. The name of the room that will serve as the starting room for both players, on a line by itself.
3. A blank line to indicate the end of the `start` description.

There will be exactly one `start` description and one `password` description per data file. You may assume that the data file is well–formed.

Your Data Structures

A *graph* is a data structure that stores data in much the same way as a tree. However, in a graph, any node may be connected to any other node. The nodes of a graph are usually called *vertices*, and the connections between nodes are usually called *edges*. A *directed graph* is a graph in which the edges are one–way (*i.e.*, an edge that goes from vertex A to vertex B in a directed graph can take you from A to B, but not back from B to A).

The cave that your program must read from the data file is effectively a directed graph. While there are many ways to represent a graph in a computer program, your program *must* use a pointer–based representation. Your program must read in the data file specified on the command–line and build a linked structure to represent the rooms and the letters in those rooms. Each room must be represented by a pointer to a `struct` that contains (among other things) a collection of adjacent rooms (*not* room names). You may use a linked list if you like, or an array (the constant `MAX_ADJACENT_ROOMS` expresses the maximum number of neighbors that any room may have). It is important though that the name of each room appear only once in your data structures.

Because the room descriptions in the data file may appear in any order, you will encounter situations in which you need a reference to a room that has not yet been defined. To handle such cases, you might want to write a function called `intern_room`. `Intern_room` takes a room name as its argument, and returns a room pointer for the room of that name. If such a room has not yet been created, it is created by `intern_room` (but it is not filled in). If the room has already been created by an earlier call to `intern_room`, then the existing room pointer is returned. Every time you encounter a reference to a particular room, you will call `intern_room`; in this way, you can be certain that every reference to a given room will generate a pointer to the same structure.

You should strive to create strong abstraction barriers throughout your program, and to exploit commonalities. You will want to create one or more new ADTs, including one for the rooms of the cave (which should end up looking very much like a graph ADT). You might find other ADTs useful. For example, you might want an ADT for collections of letters, or one to represent players (the stark contrast between good and evil notwithstanding, the player and the diabolical Doctor Plotz are very much alike from the programmatic point of view).

The Vile Doctor Chillburna Plotz

The player of this game is competing against the devilish Doctor Chillburna Plotz to be the first to assemble the letters of the password in one place. Thus, your program must alternate between carrying out the commands of the player, and carrying out those of the sinful Doctor Plotz. You need not write code to dream up actions for the abhorrent Doctor Plotz to take; such a routine has already been written for you. It is called `get_doctorPlotz_command`, and its prototype is as follows:

```
command_type
get_doctorPlotz_command(char *string_result,
                        char *current_room_name,
                        char *current_room_description,
                        char *password,
                        char *room_letters,
                        char *adjacent_room_names[],
                        int num_adjacent_rooms,
                        char *holding_letters,
                        int num_players_in_room);
```

The return type of `get_doctorPlotz_command` is `command_type`, which is defined in `doctorPlotz.h`. This return value will indicate what action the iniquitous Doctor Plotz wishes to carry out next. The arguments to `get_doctorPlotz_command` are as follows:

1. `string_result`: this is used to return a string from `get_doctorPlotz_command`. It must be an array containing at least `MAX_COMMAND_LEN` characters.
2. `current_room_name`: a string containing the name of the room that the heinous Doctor Plotz is currently in.
3. `current_room_description`: a string containing a description of the room that the infamous Doctor Plotz is currently in.
4. `password`: the password, if the password is on the wall of the current room; `NULL` otherwise.
5. `room_letters`: a string containing the letters to be found in the room.

6. adjacent_room_names: an array of char *s that holds the names of all of the rooms adjacent to the current room.
7. num_adjacent_rooms: the number of names in adjacent_room_names.
8. holding_letters: a string containing the letters currently held by the abominable Doctor Plotz. There may be no more than MAX_PLAYER_INVENTORY_SIZE of these.
9. num_players_in_room: a positive integer that get_doctorPlotz_command can use to determine whether there are other players in the same room.

You may assume that the values returned by get_doctorPlotz_command are syntactically well–formed, but you may not assume that they are legal commands. For example, get_doctorPlotz_command might try to pick up a letter that is not present in the room, or to move to a non–adjacent room.

Compiling Your Program

You will need to use at least two items from lib202 to complete your project. First, you must #include the file doctorPlotz.h at the top of your .c files to gain access to the type command_type, to the prototype for the function get_doctorPlotz_command, and to various constants. Second, you will need to link the executable for get_doctorPlotz_command into your program. To use these items, you will need to modify your Makefile (as you did when you wanted to use lib202 with previous assignments). Lines that compile .c files will need to use the -I flag:

```
cc -g -c -I ~mayfield/lib202 myfile.c
```

Lines that build executables will need to use the -L and -l flags:

```
cc -o proj3 file1.o file2.o ... -L ~mayfield/lib202 -l202
```

Submission

Once again, submission of your program will be electronic, using the program called 'submit.' To submit your files, use proj3 as the project name. Thus, to submit the files proj3.h, proj3.c and Makefile, you should enter:

```
submit cs2021 proj3 proj3.h proj3.c Makefile
```

You must submit all files necessary to successfully compile your program, along with a Makefile. The executable must be named proj3, but you may name your other files as you wish. You must submit a Makefile that correctly compiles your source files on the gl machines upon issue of the command make. The executable resulting from the compilation must be named proj3, and it must execute on the gl machines.

You may submit your program more than once, but only the last version you submit will be graded. Submissions made by midnight on the due date (Tuesday, 7 May) will receive the on–time bonus; submission of homework will be disabled at midnight the following Thursday night. Don't forget to include a comment at the top of your program stating your sources of help (as described in the Prospectus).

Extra Credit

For extra credit, you may do your own implementation of the function `get_doctorPlotz_command`. Your function must match the prototype given in the file `proj3.h` found in `~mayfield/lib202`. Each time it is called, your function must return the next command that the despicable Doctor Plotz would like to execute. If the command takes an argument (*i.e.*, it is a `say`, `move`, `take` or `drop` command), `get_doctorPlotz_command` must also place the characters of the argument in the first parameter handed to it. Don't forget to terminate this string with the ASCII null character (`'\0'`).

A simple version of `get_doctorPlotz_command` is easy to write; here is perhaps the simplest version:

```
command_type
get_doctorPlotz_command(char *string_result,
                        char *current_room_name,
                        char *current_room_description,
                        char *password,
                        char *room_letters,
                        char *adjacent_room_names[],
                        int num_adjacent_rooms,
                        char *holding_letters,
                        int num_players_in_room)
{
  *string_result = '\0';  /* Just to be safe */
  return(WAIT);
}
```

A version of `get_doctorPlotz_command` that gives a person a run for their money is significantly more difficult to write. Such a program would need to do (among other things) the following:

- Explore the cave and build a map of the room connections.
- Remember where it has placed and/or seen various letters.
- Determine the fastest way to get from one room to another.
- Develop a strategy for interfering with the opponent's progress while making progress itself.
- Handle gracefully any interference from the opponent.
- Ideally, build a model of the actions of the opponent, and alter its behavior accordingly.

A wide range of techniques covering the breadth of computer science might be used to achieve these goals. For example, graph search, computer learning, user modeling, planning, and reasoning under uncertainty are subfields of computer science each of which might contribute to a solution.

Of course, your solution need not take advantage of these techniques. You have enough weapons in your arsenal right now to develop a credible version of `get_doctorPlotz_command`. Use your imagination.