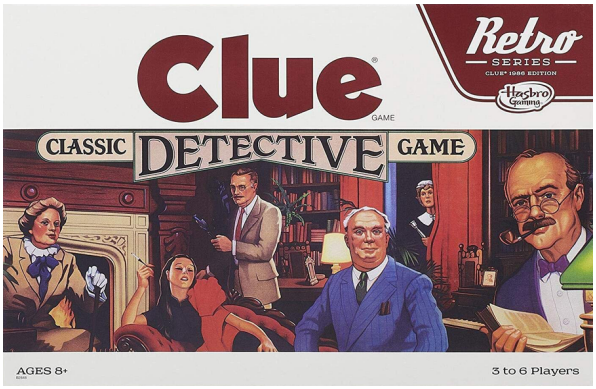

Clue

Project Profile

This project requires you to implement a text adventure game for Clue using the C programming language. A text adventure game is a form of interactive fiction in which players use text commands to control characters, influence the environment, and navigate a virtual world. The first text adventure game developed between 1975-1977 was called Colossal Cave Adventure. If you have never played a text adventure game you should give it a try to get the idea. The goal of this lab is to practice various aspects of the C programming language to help you prepare for implementing a text adventure of your own. Clue was published in 1943. In each turn, players can walk back and forth between the rooms in a mansion to solve the mystery of: who done it, with what, and where?

Game State

At the heart of a text adventure game is the state of the world that you interact with. This can include players, rooms, items or objects, and anything else that one could imagine simulating in a virtual environment. The virtual world consists of a set of “rooms” that are linked together to form locations that can be navigated through directions such as **north**, **south**, **east**, and **west**. A room has a description as well as objects that can be **taken** by a player, or more typically referred to as an avatar, controlled by the user of the game. An avatar can be a character that is controlled by a user or it can be a non-player character. You could also extend the notion of a player to one that is controlled by an AI. An avatar can also have an inventory (e.g., backpack) that can hold items collected as they navigate from room to room.

Game Play

A text adventure is played by placing a player’s avatar in a starting room or location and allowing the user to provide text input. To get an idea of how this works you can check out many example plays of the adventure game online. In general, there are several actions that can be played in a particular location including navigation such as *“look”* to provide a description of the location the player is currently in or *“go west”* to navigate to a new location (if you are able to go that direction). You can also pick up objects such as *“take key”* or get rid of an object with *“drop key”*. The objective of a text adventure game is up to the designer of that game. It could be to successfully navigate out of a maze of twisty passages or find a hidden item in a location and deliver it to a final location. The possibilities are endless, however, these games often involve a puzzle the player must solve and very often contain an easter egg to make it more interesting.

The goal of this project is to implement a functional text adventure game for Clue given the requirements below.

Requirements

Your primary objective is to use the C programming language to design and implement a functional text adventure game as described in the previous section. You are required to design and implement the appropriate data structures and corresponding algorithms that will enable a human player to navigate their avatar through a virtual world and solve a puzzle that you design. The general design of the game is up to you, however, you must satisfy the following requirements:

Game Requirements

1. Your game must have 9 rooms including the starting room. The game board is a 3 x 3 map.
2. Your game must randomly initialize the location of each room in the game board before the game starts.
3. Your game must have 5 characters other than the player's avatar and randomly initialize the location of each character in the game board before the game starts.
4. Your game must have at least 6 items. Before the game starts, Your game must randomly initialize the location of each item in the game board and each room can only contain at most one item.
5. Your game must randomly pick **a room**, **an item**, and **a character** as the **answer**.
6. Your game must allow each room to have a linkedList of items.
7. Your game must implement an avatar where the avatar has an inventory (a linkedList of items).
8. You must implement a table of commands:
    a. Your game must allow players to lookup the command in the table with the command "**help**".
    b. Your game must allow players to lookup the list of items, rooms, and characters with the command "**list**".
    c. Your game must allow an avatar to "see" the room they are in with the command "**look**", including the rooms in each direction, the characters in the room and the items in the room.
    d. Your game must allow an avatar to move through each room via room pointer using the command "go DIRECTION" where DIRECTION is one of these 4 directions: **north, south, east, west.**
    e. Your game must allow an avatar to pick up items with the command "**take ITEM**" where ITEM is the name of an item in a room.
    f. Your game must allow an avatar to drop an item with the command "**drop ITEM**" where ITEM is the name of an item in the avatar's inventory.
    g. Your game must allow an avatar to check items in the avatar's inventory with the command "**inventory**".
    h. Your game must allow an avatar to make a guess with the command "**clue CHARACTER**". (see 9)
9. After receiving the command "**clue**", your game must follow the three steps as below
    a. Move the **character** stated in the command to the same room of avatar.
    b. your game must tell the player the matching detail between the rumor and the answer based on the game status:
        i. **Room**: If the player avatar is in the room of the answer, show the message "**Room Match**".
        ii. **Character**: If the character of the answer is in the room, show the message "**Character Match**".
        iii. **Item**: if the item of the answer is in the room, show the message "**Item Match**". Notice that the item in the avatar's inventory can be treated as in the room. Also, it doesn't matter if other items are in the room/inventory.
    c. Check winning or losing state:
        i. Your game must have the winning state: if the player hit all 3 matches in a single **clue** command.
        ii. Your game must have the losing state: if the player cannot hit all 3 matches after she made her 10th clue command.

10. You should organize your code into multiple C files. In particular, you are to have the following files:
    - **`rooms.c`**, **`rooms.h`** - the source and header file implementing data structures and functions for rooms.
    - **`items.c`**, **`items.h`** - the source and header file implementing data structures and functions for items. You must implement add_item and drop_item function for adding/removing an item to/from a linkedList.
    - **`characters.c, characters.h`** - the source and header file implementing data structures and functions for characters. You must implement move_character function for moving a character to the assigned room.
    - **`adventure.c`** - the source file with functions to read user input and interpret commands as well as the main function.
    - **`Makefile`** - the **`Makefile`** to use with the **`make`** command to build your code. **Please compile with gcc -std=c99** in your makefile.

Code Requirements

1. You must use C structs in your code to represent various game objects.
2. You must use pointers in your code.
3. You must use dynamic allocation in your code using **`malloc`**/**`calloc`**.
4. You must deallocate memory in your code using **`free`**.

Video Demonstration

You must provide a link to a 2-minute video demonstration of your game being played. Your video should be short and get to the point, showing a game being played to completion. This means that you should be able to demonstrate the starting state of the game, navigation, looking, taking, and using an item. **We only accept a link of video, i.e, youtube video. Uploading a video on Gradescope will result in 0 in video category**.

Grading and Rubric

You will be graded according to the following rubric. The scoring of the rubric below falls under various categories and headings. Categories may also be multiplied - that is, some categories count more than others (e.g, x2, x3).

1. Project Requirements (40 points)
    a. See the requirement section.
    b. Includes a Makefile that compiles your submitted code.
2. Design and Implementation (30 points)
    a. Functions are declared and used properly.
    b. Data structures and data types are declared and used properly.
    c. Variable and function naming is clear and helps with understanding the purpose of the program.
    d. Global variables are minimized, declared, and used properly.
    e. Control flow (e.g., if statements, looping constructs, function calls) are used properly.
    f. Algorithms are clearly constructed and are efficient. For example, there is no extra looping, unreachable code, confusing or misleading constructions, and missing or incomplete cases.
3. Coding Style (5 points)
    a. Code is written in a consistent style. For example, curly brace placement is the same across all if/then/else and looping constructors.
    b. Proper and consistent indenting is adhered to across the entire implementation.
    c. Proper spacing is used making the code understandable and readable.
4. Comments (5 points)
    a. Comments in the code are used to document algorithms.
    b. Variables are documented such that they aid the reader in understanding your code.

    c. Functions are documented to indicate the purpose of the parameters and return values.
5. Video (14 points)
    a. The video shows the code being compiled from the command line.
    b. The video shows the code being executed and satisfying the output requirements.
    c. We can reproduce the game shown in your video by running your code
6. README.txt (5 points)
    a. The README.txt file is well written.
    b. The README.txt file provides an overview of your implementation.
    c. The README.txt file explains how your code satisfies each of the requirements.
7. GradeScope Submission (1 point)
    a. You automatically get a point for submitting to gradescope.

Submission

You must submit the following to Gradescope by the assigned due date:

- **`adventure.c`**, **`rooms.c`**, **`rooms.h`**, **`items.c`**, **`items.h`**, **`characters.c`**, **`characters.h`** - this is your implementation of the text adventure game.
- **`Makefile`** - the build file to use with the **`make`** program.
- **`README.txt`** - this is a text file (made in Vim, Emacs, or Nano - not Word, TextEdit, etc.) containing an overview/description of your submission highlighting the important parts of your implementation. You should also explain where in your implementation your code satisfies each of the requirements of the project or any requirements that you did not satisfy. The goal should make it easy and obvious for the person grading your submission to find the important rubric items. This text file should also clearly include a URL to your video for us to review.

You must use the Edlab environment to write your code. Make sure you submit this project to the correct assignment on Gradescope!

On Gradescope Errors

Students often post on Piazza about “Gradescope errors” which turn out to be errors with their code unrelated to Gradescope. In general, please test your code in Edlab before uploading to Gradescope - Gradescope is a grading tool, not a debugging tool. We recommend the following flags when compiling to test your code: **-std=c99 -Wall**. The latter flag will provide helpful warnings.