Final Project, Part 1

Read the last page first!

You will complete this project in 2 parts. For the final assignment (#4) you will submit a design and the first increments of the program. The increments will be starting the program, opening a file and displaying it, prompting the user and opening each file in order and displaying it. **The rest will be completed for the final project.**

You will create a program that will enable the user to move through a maze. The maze will contain locked doors that require keys to be opened. Guarding the maze are roving killer wombats. To escape the maze, the player must navigate through the maze, wombats, and doors until they reach a ladder. Each maze level contains one ladder that takes the player to the next level of the maze until the final level is reached. On the final level there is an exit instead of a ladder.

An evil sorceress has captured the player. Her soulless minion didn't lock the door correctly so the player can try to escape. She uses killer wombats as the guards. Each level starts with one wombat. She knows the player has left the cell but can only summon one new wombat at a time. For every 30 steps the player takes, a new wombat will randomly appear on the current level.

There are strange marks in the dungeon (indicated by the letter 'T'). If the player moves on to one they are randomly transported within that level of the dungeon. If they land on another 'T' they are transported again! ☺ The random placement of the player is otherwise the same as for the wombats. They cannot be placed on physical objects, such as walls or doors. If the player is placed on a key he can pick it up.

Your code will implement this game as follows:

When your program starts it will prompt the user for the number of levels. The floor files will be named floor_# where # indicates which floor it holds. Your program will always start reading with the first floor.

The maze is defined by a set of text files, each of which contains a single level. Whenever the game starts or the player reaches a new level, that data must be read from the appropriate text file. The first line of each file will have two integer values which describe the number of rows and columns for that level. You must use this to create a dynamic array of that size and then read in the data for the level. Perform input validation on all characters in the file as you read them. Do not load the file and inform the user there is a problem.

The following characters or symbols describe objects in the levels:
‘#’- walls
‘D’- door
‘L’- Ladder
‘P’- Player
‘W’- Wombat (the starting location of the first wombat)
‘K’ Key
‘T’- Teleport square
‘E’- Starting location for each level
‘X’- Exit(one way out)
‘ ‘- empty space (blank character)

You must concatenate the file name with the new floor number. You will not use an array of filenames? HINT: This will require string streams.

You must display that level to the user. Then you will accept the following commands from the user:

‘w’ move up
‘a’ move left
‘s’ move down
‘d’ move right
‘u’ use ladder
‘q’ quit

Each input from the user must be validated before any action takes place. After each command, you will update the array, then print the current status of the game to the screen. Each command is one “step” which you will use to coordinate movement and summoning of the wombats.

You will update the position of the player and all wombats after every move entered by the player. The K (key) and A (apple) symbols will be removed from the maze after the player picks them up. The D (door) is removed once the player enters that space with a key. To move between floors, the player must get to the space with the ladder. On the final level the player must reach the X to exit the maze and make their escape.

The position of each Actor (Wombat or Player) will not be stored in the array. It will be stored in the Player or Wombat object. When you read in the initial floor array the ‘W’ should be changed to a blank character. When you display the floor after each move you will print the array, adding a ‘P’ or ‘W’ as appropriate for each Actor.

Movement- To move around a level the players must use the ‘w’, ‘a’. ‘s, and ‘d’ keys. Your move function should prevent them from moving through walls, or a door if they don’t have a key. If the player has a key then they can move through a door. Simply remove a key from the player and convert the D space to an empty space.

When the player gets to a ladder space they climb it with a 'U' command. The 'E' character designates where the player starts on the new level. Time (i.e. the number of steps) is reset on each new level. When the player moves from the starting space, the E must remain. If a wombat catches a player, their location must be changed to the E square. After each move, the updated maze must be displayed for the user.

When the player moves into a space that contains a key ('K') they pick up the key to add it to their inventory. When a player moves into a space with a teleport square ('T') they are randomly positioned. A player can hold no more than 3 keys. If they enter a space and cannot hold the key it remains in that space.

Wombats (10 points): The Wombats will be similar to the Player, except they move randomly. They cannot move through walls or doors. They will ignore teleport squares. If a wombat is in a square adjacent to the player, the player will be moved back to the starting space for that level. The wombats cannot move between levels. For every 30 steps the player takes, place a new wombat at a random valid location in the current level.

Classes. You will need the following classes:
Game class- dynamic array for current floor, array of pointers to actors, int total#_floors, int current_floor, function to read in floors from a file
Floor class- char [][] tiles, void print to screen, constructor
Actor class: ABSTRACT- int row, int column, char symbol, virtual move
Wombat class (inherits Actor)- move (automated),
Player class (inherits Actor)- move (prompt for wasd), keys(s), hold up to 3 keys
(You will need a function to send the player back to the start if a wombat moves adjacent to their position. **Where do you put that function?**)

You must **include a makefile and put all files for your assignment in a zip file**. Each class must have separate source and header files. If you do not do this assignment will NOT be graded.

Grading:

- programming style and documentation (10%)
- program design and development plan
  - your statement of the program requirements (10%)
  - your program design (20%)
  - your program development (10%)
  - your plan for incremental development (10%)
- read and create each maze level properly (20%)
- input validation for commands and reading the files (5%)
- reflections document to include the design description, test plan, test results, and comments about how you resolved problems while designing and implementing your program (15%)

You must include a makefile put **all** files for your assignment in a zip file. If you do not do this assignment will NOT be graded.
Each class must have separate source and header files. If you do not do this assignment will NOT be graded.
If your program segfaults at anytime, you will lose 20 points.

For assignment 4 you will create a detailed design and development plan. The only code to submit is the reading of the files, creating the maze, and processing user commands (most will do nothing at this time). You should use the Game and Floor classes to perform these activities.

Design your Program

As always you should sit down with pencil and paper and sketch out the design. Develop the necessary algorithms. Do a desk test, i.e. walk through your algorithms and code to look for logic errors.

Nope! No keyboard yet! Now design your incremental development. Your program should be decomposed into functional units. If not, go back to the previous paragraph. Look at the pieces and decide on the order you should use to implement and test each part. Reading the floor into your program and then moving an actor are probably good places to start. Then add a level to test using the ladder. Once you have the development you are ready!

Nope! No keyboard yet. Design and organize the directories for your project. You should have a working directory. Maybe you need a subdirectory to hold the your floor plans. Have you designed your floor plans? What is there to design? Maybe you can have a floor that allows you to test only one function or command at a time? Be careful with file names. The default first file always has the same name. If you overwrite one then you'll need to create it again. Always have one directory to save a copy of your code as you get each increment to compile. If you're paranoid (I know I am) maybe you have more than one stash and on different devices?

Now, you're ready to code!

The moral is: **Always plan ahead and make it as complete as you can!**