Submitted By Eric Miller 6/08/2014

The final Project

Intro:

I have had in mind for a while now to make a text based adventure game akin to 'Zork' for my final project. We had the same computer for a long time at my house, and text adventure games were one of the few things that it could run (if I remember correctly even the original monkey island was too much for it to handle). As such I have sort of a soft spot for them, and am interested in trying my hand at programming one.

It of course will not be nearly as long or interactive as 'Zork' (no grues), but nevertheless I want to create a basic interactive story of sorts. Now that we have both structs and classes in our toolbox, I finally feel that I can start taking a stab at this sort of program.

Of course, I plan on hitting every required component of the final project along the way. I hope that my program in the end will not only go above the minimum requirements, but be interesting to grade and play as well.

Understanding:

As outlined in the assignment, all of the below must appear somewhere in the program. I will keep updating this section of my report periodically to point out a line of code where requirements are met. Please see the *'Requirement Checklist'* PDF for code samples.

1. Demonstrates **simple output** (cout or printf is fine),
Example in Line 4074:

2. Demonstrates **simple input** (cin or getline are fine, better would be to use both :)!)

Example in Line 808:

3. Demonstrates **explicit type casting** (a couple ways to do this),

Example in line 1633:

4. Demonstrates **conditional** (should be easy, we use these all the time),
Example in Line 1746:

5. Demonstrates **logical** or **bitwise operator** (more specifically &, |, ^, &&, ||, !, ==),
Example in line 1646:

6. Demonstrates at least one **loop**,
Example in Line 654:

7. Demonstrates at least one **random number**,
Example in Line 766:

8. Demonstrates understanding of the three general **error categories** we talked about
Example of syntax error on line 223:

Example of Runtime Error on line 975:

Example of Logic Error on line 3855:

9. Demonstrates some form of **debugging** "tricks" that we have learned throughout the class
Example on line 972:

10. Demonstrates at least one **function** that you define
Example in Line 109:

11. Demonstrates general **functional decomposition** to reduce how large a single section of code is.
Example in Line 654:

12. Demonstrates how **scope** of variables works
Example in Line 68:

Example in Lines 4622 and 5593:

13. Demonstrates the different **passing mechanisms**
Example from line 103:

14. Demonstrates **function overloading**,
Example in Lines 109 and 215:

15. Demonstrates at least one **string** variable (std::string or c-style string),
Example in Line 47:

16. Demonstrates some form of **recursion**,
Example in Line 1660:

17. Demonstrates at least one **multi-dimensional array**,
Example in Line 33:

18. Demonstrates at least one **dynamically declared array**,
Example from Line 627:

19. Demonstrates at least one **command line argument**
Example from Line 617:

20. Demonstrates definition and use for at least one **struct**,
Example in Line 568 and 1246:

21. Demonstrates definition and use for at least one **class** and **object**,
Example in Lines 257 and 552

22. Attempts to demonstrate a **pointer to an array**
Example in Line 168:

23. Attempts to demonstrate a **pointer to a struct**
Example in line 1631:

24. Attempts to demonstrate a **pointer to an object**

Example from Line 638:

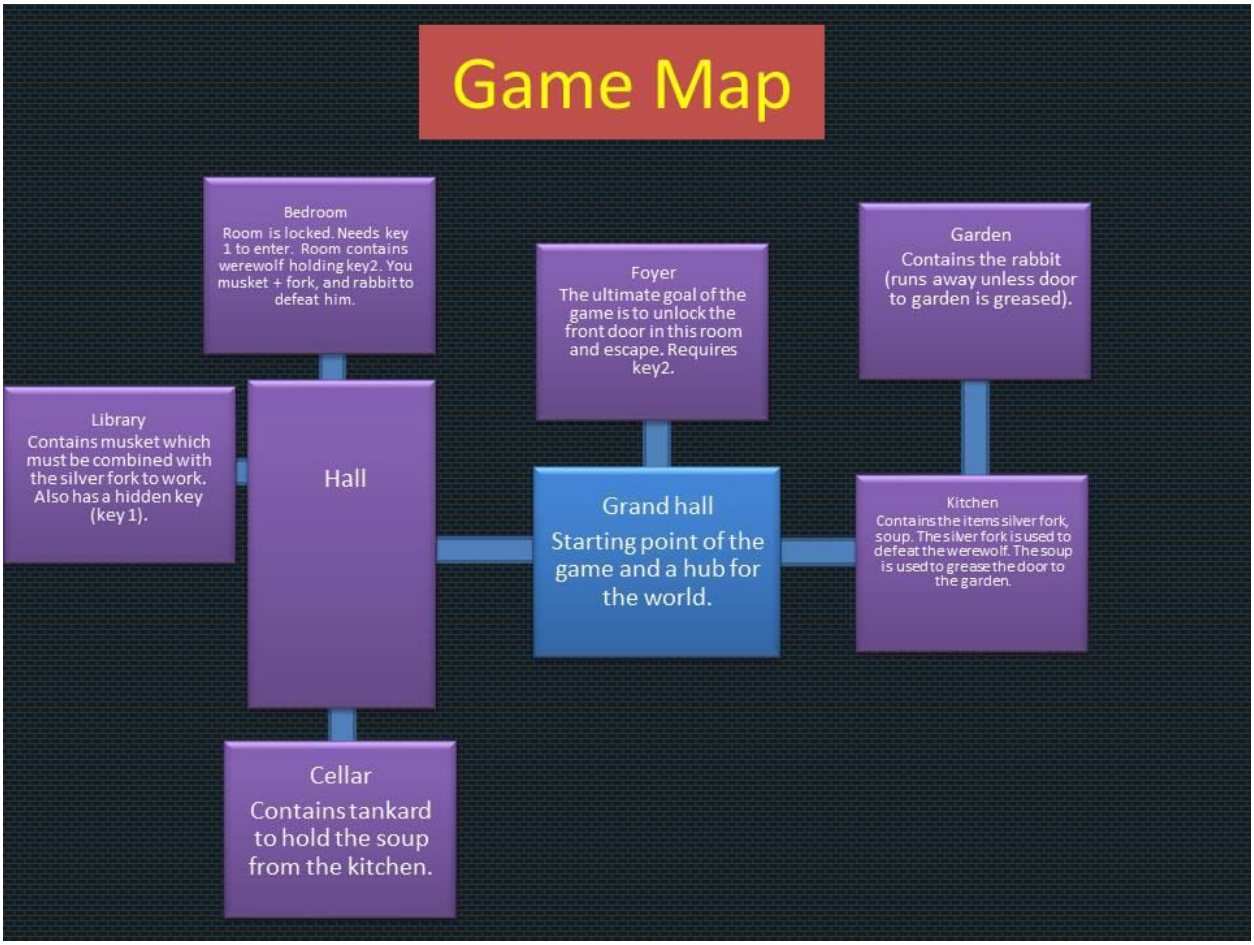
25. (extra credit) does **something awesome**

I sincerely hope that this program is enjoyable enough to qualify.

Program Outline / Design:

1. Instead of code, I am actually going to start off by creating a map and game outline. This map will contain the layout of the world, what connects to what, and what experience I want to program for each area. This will be used as a general guideline for how I build my program.
2. The way to beat the game will be to collect all the needed items and use them in the appropriate areas.
3. The general flow will be this:
    a. You need to exit from the front door, but it requires a key (key 2).

b. To get key 2, you need to defeat a werewolf behind a locked door (requires key 1). To defeat the werewolf you need the rabbit, musket, and silver fork.
c. Key 1 can be found in the library along with the musket.
d. The rabbit can be found in the garden sleeping, but the door to the garden is squeaky and scares it off. To silence the door you need soup.
e. The soup can be found in the kitchen along with the fork. However you can't pick up the soup without the tankard.
f. The tankard can be found in the cellar.
g. Once you've done all these things you will have defeated the werewolf and can leave through the front door to win.



4. For the main character, I will have them be a class object something like this:
   Class player
   {
   Public

Function change name
Function change health
*String *food*
Private
String name
Int health

}

5. If the default arguments are equal to 2, then the second argument will be used for activating a cheat code that give the player full items:

```
If (argc  ==  2)

        Fill_inventory ();
```

6. Not sure at this point how it will filter into the game, but you may have noticed that food is an attribute of the player. Food will mostly be there to meet the requirements of a pointer as well as a 2-d array plus a random number. It will also demonstrate a global as opposed to local variable.
7. A global string 2d array called what_I_ate will be initialized to contain a 2x2 array holding food names. A separate function will randomly choose something from this array via pointer to be set as the value of what the player ate. This meets requirements 22, 17, 12, and 7.
8. The items will all be 'stored' in a separate struct object called <inventory>.

```
struct inventory

{

        String items []; //array holds item names//

}
```

9. The werewolf will simply be a struct manipulated by pointers (meeting requirements 20 and 23) :

```
Struct werewolf

{

        Int health = 1;

        Int attack = 1;

}
```

10. I am thinking that each room will be its own class containing items and descriptions similar to the following:

```
class room
{
   public:

   void look ();
 //Gives description of room

   void goto_room ();
//enters the room and sets up interactions

   void set_objects (string obj[]);
   //enters interactive objects into room_objects
   //Parameter one is the list of objects

   void use_objects (string action, string item);
   //Gives results of object interaction
   //Parameter one is the user action
   //Parameter two is the item being interacted with

   private:

   string room_objects[6]; //interactive items in this room//

   int arr_size = 6;
//size of object array

};
```

11. User input will be a list of predefined actions, and four directions (N<S<E<W)
12. To make sure that there are no logic errors due to input being misread by the program, all text input will be set to lowercase via function (this is a form of a debugging trick…meets requirements 8 and 9).
13. The rest of the requirements I think I shall be able to achieve over the course of my programming.
14. The first actual programing I will do is to try and make a bare bones adventure with no real interaction besides going to rooms and looking at them (see below for testing).

Testing A-1:

Conditions	Expected Output	Actual Output
I made a global string array containing a list of all valid 'actions' by the player. Whenever the player inputs something, it is compared to this list in order to see if it is a recognized action through a function. I am going to see now if input checking works. TEST 1: Invalid TEST 4-5: Valid TEST 6: Invalid	"Yay!" if the word is valid. "Nay!" if the word is invalid. TEST 1: NAY TEST 4-5: YAY TEST 6: NAY	Tests 1-6: As expected.

Testing A-2:

Conditions	Expected Output	Actual Output
Depending on the room, entering a direction will change a global variable for the room number. This transports the user from room to room. I only have two rooms to start, but I want to see if I can travel between them and use the 'LOOK' command in each area.	From the grand hall if I go north I end up in the foyer and vice versa. Each room will let me look at its unique surroundings	As expected I can travel freely between the rooms and examine them. It actually is kind of exciting to see it work out on screen!

Testing A-3:

Conditions	Expected Output	Actual Output
Each room will have its own list of objects you can 'interact' with. I want to have the user input an action and (if the action is not "look" or a direction) have the user choose an object to interact with. I want the program to say "ACTION" to show that an object exists in that room (this is for debugging only and will be replaced by dialogue in the final game).	If an object isn't located in the room, a message will tell the user that their input wasn't valid. On the other hand if that object does exist, the message "ACTION!" will be output to the screen.	Surprisingly as expected. I mostly copied the code pattern for valid actions and transferred it to find valid objects (including error filtering by making any input capital letters).

Testing A-4:

Conditions	Expected Output	Actual Output
At this point I have created all of the rooms for the world map (connected as shown on the game map). I will now try and travel to each room and make sure that You can look at each area as well as only travel into connecting rooms.	All rooms are accessible, but you can only travel to rooms that are connected to each other. You can enter the 'LOOK' command to get a description of each room.	As expected. Granted at this point there is no interactivity with items in the room, but at first I just wanted to get the traveling portion down.

15. Now that I have the game world foundation built, I am going to move on to item interactivity.
16. I am thinking that this will require a new function for the room classes that will process user input for the item that the user wants to interact with, then output a proper text or action in response.
17. Initially I will ignore items that you can pick up for your inventory.
18. Each room will have its own string array of "objects" that can be interacted with. Should both the player action be valid as well as the object input (it must be on the room object list), then each room will activate a function that will print out a response based on action/object combination.

Testing B-1:

Conditions	Expected Output	Actual Output
Every room (except for the hallway) has at least a few objects that can be interacted with when a valid action is input. The valid actions are as follows: ***"USE", "LOOK", "EAT", "TASTE", "ATTACK","HIT","OPEN", "TAKE", "PICKUP","KILL", "BREAK", "INSULT", "HARASS", "EXAMINE","TALK","SPEAK", "YELL", "PUNCH", "PICK", "READ"*** I will try combining these actions with room objects to test for appropriate output.	Desired output will result from combining appropriate action and object that the user wishes to interact with.	As expected. This part of the program has taken me just as much time as all the parts before it due to the sheer number of possible interactions that needed to be programmed. Fortunately I think I have mostly gotten them all down, and now only the collectable items need to be tackled.

19. While programing the interactions, I already started making if/else switches for when collectable items are added. The conditions for these switches are based on array values for the inventory struct int array “backpack”. Each index corresponds to a different item, with a value of ‘1’ meaning that the item is in the player’s possession.

```
struct inventory
{
   int backpack [8] {0,0,0,0,0,0,0};

  // index [0]  is key 2
   //index [1] is key 1
  // item [2] is the silver fork
  // item [3] is the lantern
   //item [4] is the tankard
  // item [5] is the rabbit
 //  item [6] is the soup
   //item [7] is the musket

 int arr_size = 7;
   //size of the key item array//
};
```

20. I also added a lantern item which I initially didn’t have in my game plans. This is required to enter rooms that have no other light source and can be found in the kitchen.
21. Just like room objects, I want inventory objects to be able to be used in certain situations. To do this I will create a global string array that contains the names of the collectable items and (via function) check to see if these are called by the player. In rooms where both a collectable item and a room item share the same name, exceptions will be added so that the room object (as opposed to the collectable object) will be interacted with.

```
Stiring key_items[] = {FORK, MUSKET, SOUP….};

Void validate_key_item (string user_input, string key_items[ ], int arr_size);

Void use_key_item (string validated_input, string key_items[ ]);
```

Testing B-2:

Conditions	Expected Output	Actual Output
This time I will : Go through the rooms that have collectable items. Pick up the items. Make sure that the game recognizes the items are gone. Combine items when I can (Fork + Musket, Soup + Gate) Use collectable items instead of room objects. To save time, for some rooms where I combine items I will pre-insert the items into my inventory to I don't have to go searching for them every time.	I will be able to pick up items. Once items are picked up they are no longer in the room, but carried on the player (so to speak). In certain situations, collectable items can be used instead of room object (such as when combining things).	Error when I set a variable as '==1' instead of '=1', but that was easily fixed. After fixing: as expected. I am able to pick up items only once, and combine items when appropriate. So far so good!

22. Now that all of the items seem working, it is time to program the werewolf encounter.
23. Previously I had planned to make the werewolf as a struct like the following

```
Struct werewolf
{
        Int health = 1;

        Int attack = 1;

}
```

24. Now however I think I would prefer to create a class for the werewolf in a manner similar to the room classes. This will allow member functions that can be called to start the encounter.
25. Unlike a room however, the encounter will not end until the player or the werewolf dies. Certain actions (or failure to take actions) will change the health points for the werewolf and player.
26. I will have werewolf class functions that both create output for the room object (in this case, just the werewolf) as well as unique output for the collectable items when used on the werewolf.
27. The action order to defeat the werewolf will be USE RABBIT -> USE MUSKET. Any other actions will result in the player taking damage.
28. A game over function displaying text will be called should the player health be equal to '0'

Testing C-1:

Conditions	Expected Output	Actual Output
This time I will : First I will try and defeat the wolf using the proper input commands and having all the items. Next, I will try and fight the werewolf using invalid as well as all the valid action commands. I will set the player at infinite health for this portion of the testing. Next I will go and fight the werewolf while missing some of the required equipment and purposefully loose.	The werewolf will be defeated should the player use the rabbit, then the combined fork and musket. The player will lose if their health reaches zero.	Error as I was trying to access the private player class member for health from inside the separate werewolf class. To fix this, I amended the player member function that controls health to show the game over message when it reaches zero (instead of the message coming from within werewolf member function). Else: As expected.

29. My next step of action now that the werewolf is working, is to add an ending (simply a text output).
30. I will also the add command line arguments now (mentioned in step 5!) that will give the player all the items from the start of the game if their 2nd argument is “CHEAT”.

Testing C-2:

Conditions	Expected Output	Actual Output
This time I will : Run through the full game, collecting all the items and defeating the werewolf along the way to re-check for errors. I will then start the game with command line arguments to save time on collecting items.	The game will go as planned, with item collection, room object interaction, and game over/ victory conditions. If “CHEAT” is used as a command line argument, all items are available from the start.	All as expected, though I came across a lot of typing errors that needed fixing.

31. Lastly if I have time, I want to polish the program. This will be adding features like the following:
    a. Fixing grammar, spelling, and spacing
    b. Because typing out 'EXAMINE' to look at objects instead of rooms is a pain to type, I want players to be able use the 'LOOK' command as well for objects. Until now typing "LOOK" would skip the object entering process and return a description of the room.
    c. I want to have extra werewolves running around once the player gets the final key. Unlike the first encounter however, these are not required to complete (nor can you beat them)
32. Making the "LOOK" command universal for both rooms and objects is kind of difficult at this stage in programing, and will require some extra code that might get a bit messy. I think I will add "ROOM" as an object players can interact with, but solely with the "LOOK" command.

Testing D-1:

Conditions	Expected Output	Actual Output
This time I will : Run through the game using both LOOK and EXAMINE for rooms as well as room objects. I will try and interact with the extra werewolves after the first has been defeated.	This time "LOOK" will work for both rooms and items. Extra werewolves will appear on the map at the end of the game.	Mostly as expected, with a few missing semicolons, parentheses, etc that needed to be sorted out.

Testing D-2:

Conditions	Expected Output	Actual Output
One last run-through of the game to clean up any missed errors.	Hopefully, just the game with no lingering bugs or issues.	A lot of text editing, and a situation where the "LOOK" command didn't work in the library. All fixed.

Testing D-3:

Conditions	Expected Output	Actual Output
Testing on Putty Server	The game will play.	LOTS of Errors- all stemming from my initializing variables inside of classes. This is allowed on my complier (CodeBlocks) but it didn't work in Putty. To fix it I had to make constructors for each class object…

Reflection:

Wow, so that was quite the experience. Although they were the last thing we learned in this class, I ended up relying very heavily on class objects- so much so that the main function really only serves as a hub point. I think this project really showed me the value of 'Object Based Programming', and I am excited to move forward with this method from here.

The program itself followed the plan that I had made for it - for something this big it was really important to have an overarching game flow in mind (which started with the world map). That said, if I had more time I would have liked to have added an inventory display, maybe 1-2 more rooms, and the option to type "L" for "LOOK".

One challenge that I wasn't expecting was text formatting. It is actually kind of difficult to make sure that it looks right on the screen and that words aren't cut off if the sentence runs on for too long. Even now I am not completely happy with the test layout on screen, but at this point I've done what I can. One tool that really helped me was the application on this website: (http://tomeko.net/online_tools/cpp_text_escape.php?lang=en)

While I fit them in, for the purposes of this game dynamic arrays and pointers weren't really required, and bitwise operators definitely weren't necessary. However at least for bitwise operators I've never really used them before, so it was a good chance to play around with them and review some binary.

Besides the value of Object Based Programming, this project also really showed me how much work goes into even the simplest of programs. This game is so tiny, yet took over a week to make. I realize that part of this is due to inexperience and me working alone, but it still made me respect even more the professional programmers out there.

Lastly, I want to thank you for taking the time to grade this project as well as all the projects prior to it. I hope that it was at least a little bit interesting to go through, and I always appreciate your feedback!