Please read all comments in main, that will help in understanding the game and what everything does. Also please see the pdf file titled DIFFERENT SCENARIOS FOR THE GAME in the zip file. This file contains the perfect route to beat the game, and gives some alternative routes to take that will display some of the game design and some of the functions, and the amount of points that the user have scored. Also please feel free to experiment with your own route. Also please see the file titled MAP FOR GAME file, that will give a layout of the structure of the rooms and their location for the game. I have included that file, to be kind to the grader as was suggested by prof Terry R., because I understand it can be time consuming to try and figure out the game.

This document will describe the major issues that I have encountered, and will give a description of how I set up the program.
The first thing that I did with this project was that I sat down with a paper in a pen trying to figure out what my game should be. I came with the idea that the game should be based on someone that woke up late trying to make it to work on time, and needs to perform certain tasks before going to work. In addition, I also designed the rooms that I will be needed, such as a bathroom, a hallway, a living room, a kitchen, a garage, a bedroom, and a car. After I did the designing as to the type of the room that I am looking for, my next goal was to come up with something to have the user move around in the bedroom which would be the starting point of the game. First thing I did, was designed a class called room. The class Room takes a pointer to all the different directions that the user can move to.
Example:

```
    Room *toEast;  // pointer to east
    Room *toWest; // pointer to west
    Room *toNorth; // pointer to north
    Room *toSouth;  // pointer to south.
```

Next, I then created, string variables, that would describe the name of the room, and to give a brief description. Setters and getters were created for the name and the description. The getter for description was declared to be a pure virtual function. Then I return the name, and the description for those functions. I also created setters for the directions, and the getters were described as pure virtual functions because each class would have it's own directions as to where the user can or cannot move. Then, I created a class called Boring Room that inherited from Room. The functions and variables for boring room were:

```
    BoringRoom();   // consctructor
    Room* getToEast();  // function to head east, returns the
room.
    Room* getToWest();  // function to head west, returns the
room.
    Room* getToNorth();  // function to head north, returns the
room.
    Room* getToSouth();  // function to head south, returns the
room.
    std::string getDesc();  // function that gives a description
of a room or an item
```

With those functions I went into main, and I created the Bedroom as a boring room. I called the setName function to let the user know that this is the bedroom. Then I also called the set Description function, to give a description of what the bedroom does.

```
BoringRoom *bedroom = new BoringRoom();
bedroom->setName("The Bedroom");
bedroom->setDesc("You are in your bedroom.)
```

With that same idea, I created another room called living room, and bathroom just for testing purposes. I followed the same format for the bedroom. I set the bedroom north of the Bathroom, and I set the bathroom east of the bedroom. I set the bedroom south of the living room, and the living room east of the bedroom. Then I tested out to see if I could move around going from the bedroom to the living room, or from the bedroom to the bathroom. I was able to go back and forth from all these rooms. Since I was able to move around, the next step involved figure out exactly how I want to set up the room, and how I wanted to connect them. In addition, one thing that I struggled a bit with was to figure out what type of classes that I would need. I looked up at some game design examples as how can I set up the classes, and I came up that I need to have the rooms based on what they will be doing. So the BoringRoom class describes a room that is boring, and not much action occur there. I came up with two other classes named LockedRoom, and ItemRoom. The locked room class is to handle a room that has a door that is locked and may require a key to open that room. The Item room class describes a room that may have an item that the user needs to get to perform a task. For example, if a door is locked, the user needs to find a key to unlock that door. The key would be feature in the item room class, and the locked door would be in the Locked Room class.

In addition, the LockedRom inherited from the Room class. The additional functions added specifically for the Locked room were:

```
// bool to check if the user can head in specific directions.
    bool eastLocked = false;
    bool westLocked = false;
    bool northLocked = false;
    bool southLocked = false;

    void lockEast();  // function to lock east
    void lockWest();  // function to lock west
    void lockSouth();  // function to lock south
    void lockNorth();  // function to lock north
    void unlockEast();  // function to unlock east
    void unlockWest();  // function to unlock west
    void unlockSouth();  // function to unlock south
    void unlockNorth();  // function to unlock north
```

The bools are to check on whether or not a room is east, west, south, or north is locked. The extra functions for the LockedRoom class are:

```
 // bool to check if the user can head in specific directions.
```

```
    bool eastLocked = false;
    bool westLocked = false;
    bool northLocked = false;
    bool southLocked = false;

    void lockEast();  // function to lock east
    void lockWest();  // function to lock west
    void lockSouth();  // function to lock south
    void lockNorth();  // function to lock north
    void unlockEast();  // function to unlock east
    void unlockWest();  // function to unlock west
    void unlockSouth();  // function to unlock south
    void unlockNorth();  // function to unlock north
```

In the constructor for Locked Room, the variables toEast, toWest, toNorth, and toSouth are set to NULL.   With the function for gettoEast() for the LockedRoom class. The bool was used to check if eastLocked was false, if it's locked, it cout East is Locked, and else it return NULL, and if toEast is not locked, it return toEast, else it returns NULL.
Pseudocode

```
If eastLocked{
Cout:: East is Locked
else
return NULL
 }
if toEast != NULL
 return toEast
else return NULL
```

The same format was followed for gettoNorth(), gettoWest(), and gettoSouth(); The showMoreOptions() function was used to show more options, and to display if one of the directions have been unlocked . So for example:

```
If(eastLocked)
Cout << "Unlock East << endl;
```

That same format was followed for West, North, and South.

For the itemthatLetsmeDo() function, the whole idea is the function should allow something to be done. So for example, in that particular case if the user select one of the option that is presented, it allows the user to do something. If option 5 is selected, it allows the user to do 5. Code Example below:

```
std::string LockedRoom::itemThatLetsMeDo(int option)
{
    if(option == 5)
        return letsMeDo5;
    else if(option == 6)
        return letsMeDo6;
    else if(option == 7)
        return letsMeDo7;
    else if(option == 8)
```

```
        return letsMeDo8;
    return "";
}
```
The same format is also followed for itemthatPreventsMeDo() function.
```
std::string LockedRoom::itemThatPreventsMeDo(int option)
{
    if(option == 5)
        return preventsMeDo5;
    else if(option == 6)
        return preventsMeDo6;
    else if(option == 7)
        return preventsMeDo7;
    else if(option == 8)
        return preventsMeDo8;
    return "";
}
```

The real challenged for the LockedRoom class, was with the makeItSo() function. That function was used to allow the user to unlock a room that is locked once a key is obtained. So with that, it was trial and error to try to see how it would be. At first when I first wrote the code, I tried to have the user to unlock something, and it wasn't working. The main idea is that we assume that the user will unlock something by setting a bool variable called userFailed = false; then if the room is east, we call in on the letsmeDo variable to allow it to be done, and that is for all directions. And If the user failed, it returned true. With that idea, I needed a way to know for sure that the door has been unlocked. The next thing I decided to do was to work on the items that the user would be carrying. I decided to use the player as the backpack meaning that the player will have the items to carry. I used an array called body to do that. The user carry stuff that needs to be done and also items.
```
for(int i=0;i<10;i++)
    {
        body[i] = "";
    }
    body[0] = "Unshaven"; //doing shave removes item
    body[1] = "Unshowered"; //doing shower removes item
    body[2] = "Wearing Pajamas"; //acquiring suit removes item
    body[3] = "Barefoot"; //acquiring shoes removes item
    body[4] = "Holding a Nail"; //hall2 removes item on unlock
    body[6] = "Missing your Car Keys"; //closet get keys removes
    //need: suit
    //need: shoes
    //need: handcuff key
    //need: razor
    //need: carkeys
```
The user start by carrying unshaven as an item, that will later get to be replaced when the player finds his razor to shave. The player also carries an item called Missing your car Keys, that will be replaced with holding your car keys later. This was the basic idea. Next I decided to have the bathroom door locked and have the player wake up with a nail in his hand that can be used to unlock the door of the bathroom. Going back to the

makeitSo() function I decided to try again, to see if I could get it to work. I got back to the idea of wanting to try to find a way to know when something happen and to be able to make a change in the array for the item description that the user is carrying. So for example I needed a way that when the user picks up his keys, it would update to say that the key have been picked up. I used the idea of parsing a string. To parse the strings to be able to switch things in the array, I used commands like, get, lose, goodbye, say, trade. Once that function was tested and working, I went back to finish the makeitSo() function. I went back to my original idea, and switch the codes around, and then I realized, I wasn't callingeastLocked to lock east. I was trying to unlock something that was already unlock. So I had this to unlock something east. The same format was followed for the other directions also.

```
if((option == 5)
    {
        usedItem = letsMeDo5;    // east is being unlocked
        unlockEast();
    }
```

As soon as I realized that I was trying to unlock something that wasn't locked, I switched the code around and I had this:

```
if((option == 5) && eastLocked)
    {
        usedItem = letsMeDo5;    // east is being unlocked
        unlockEast();
    }
```

That same format was followed for all the other directions. Then after, I called on the showingBasicDescription bool, and used to switch the description of the room, if the room that was locked has been unlocked. With that format, I was able to make to make the makeItSo() function work.

After the makeitSo function became fully functional, I realized that I needed to start working on the ItemRoom class which also inherited from Room to have the player pick up an item that may be needed to perform a task, such as getting a razor to shave or maybe getting a key to unlock a door. While I started to work on that class, I came up with the idea of a function called getExtendedDesc that will give more description about what is happening after a task has been completed, and also to give hints and different directions to take. The ItemRoom class follows a similar format with the other classes. The good thing with this program is that, once a class is working, it became easier to have other classes. For gettoNorth () in that class, it simply return toNorth, if the user is able to head that way, and the same format was followed for all the other directions. The getDesc() function returns both the descripton and the extended description. The extendeddescripton() function only returns the extended description. The showMoreOption() function display the action being display at the moment for the class. Item that letsmeDo() returns what is to be done, and itemThatPreventsMeDo() returns what is not to be done.

With the classes being set, then I went to main, and set up all the rooms and their pointes and give their description. This was actually the easiest thing for the program, and I took my time to enjoy it. Then after all the rooms were linked, I tested out the program and trying to use different routes. One major issue that I encountered is that the player was able to shave even without a razor. I just couldn't understand why? After going back and forth thinking that it was the ItemthatPreventsMedDo that wasn't working. I tried to set the sink to prevent MeDo(shave) to see if that would work, but to no avail. I tried that:

```
//sink->setPreventsMeDo(5, "Shave");
```

After a lot of thinking and debugging, I realized that I was missing this :

```
sink->setLetsMeDo(5, "Holding a Chilly Razor");
```

I need to useLetsMedo, to have the user used a razor before he is able to shave. Once that was used, it was working properly. Then the next thing was to work on a way to keep up the time for the game so it doesn't go indefinitely. I used an integet variable called minutes and set = to 11. Then I incremented minutes causing one minute each time the user moved perform an action such as unlocking a door, going from one room to another, or picking up car keys etc. The logic, is that the user wakes up at 8:11, and needs to make it to work at 0900. After the time was implemented, then the last thing I did was to create a function called tallyScore that keeps track of the score of the game based on different scenarios that can happen and also based on what tasks were accomplished by the user.