My first order of business is to increase readability of the code. Specifically, I shall be standardizing the identifiers of certain types of variables, such as “whichDirection” always referring to a compass direction (in place of the more ambiguous “whichExit”), and I shall be replacing the “char\*” data types with the “string” data type included in “std::string”.

Having done this, I changed the “RotatingRoom” display method to display the text from its axis Room if “areOtherExitsVisible” is not set to TRUE. I also changed the “operator<<” method for “Room” so that, if “isVisible” returns FALSE, it displays text that indicates the room is not visible rather than its normal text.

In order to facilitate a more natural implementation of the “rotateRoom()” method of “RotatingRoom”, I have changed the values of the compass direction constants. However, this introduced errors in the “directionToIndex()” methods of “Room” and its subclasses. These errors were ultimately rectified.

Renamed “rotateRoom()” to “rotate()”. Attempted to implement “rotate()” by using the “linkRooms()” method. This ended up being problematic in that it introduced unexpected errors and would delete the “RoomExit” objects. Instead decided to implement an “unlinkRooms” method for “Room” to preserve the “RoomExit” object. Also implemented a “rotateDirection()” global method that rotates a compass direction a certain number of degrees, relative to the defined direction global constants. Used these two methods to successfully implement “rotate()”.

Decided to improve memory usage by defining destructors for the various classes. Encountered some problems, especially concerning the “RotatingRoom” class, possibly unique to how C++ implements subclasses. Have decided to leave this be for the moment, as the destructors now seem to work mostly as intended, apart from “RotatingRoom”.

Was able to solve this problem. One part of the error arose because when C++ deletes an object of a subclass, it also calls the destructor of each of its superclasses. So, if memory is deallocated in a subclass, it is deallocated again for each superclass. To fix this, only the Room destructor now deallocates memory. The other part of the error arose due to something a bit similar. When a CorridorRoom is created, an array of length 2 is dynamically allocated. However, when a CorridorRoom is deleted, the destructor of Room would traverse this array up to its default getMaxExits() value, which is 4. To avert this, the CorriorRoom constructor now dynamically allocates an array with a length of at least 4, initializing to NULL any entries that are not specified in an argument.

Improved the functionality of RotatingRoom so that other rooms now interact correctly with it. Created an Object class. Created a Container subclass of Object that contains an array of Objects. Made Room to be a subclass of Container. Created static variables and functions to keep track of Rooms that are created. Each created Room now has a unique ID and is stored in a static array within the Room class (except for the Room contained in a RotatingRoom).

Changed the structure of the Makefile and the directories of the compile paths so that similar code is in the same directory. Created an Item class which will be the superclass of all item types. Specific item instances will have their own different class.

Made each Room object store its “maxExits” as a member variable instead of being returned by a virtual method. This fixed some errors that would occur when a Room object is deconstructed, namely that the method being called would always return the “maxExits” of the base Room class. Items and RoomExits are now stored in a static array, similar to Rooms.

Created a Character class so that characters can be represented. Created a parse() function that parses input and can perform actions based on the input. Changed main.cc to prompt the user for input, then parse() the input, then repeat indefinitely.

Creted a separate commands.h file to handle the specific action for each command, such as “GO” and “LOOK”. Improved the flow of the parse() decisions. Added a display method and operator<<() overload for RoomExit. Improved the display method for RotatingRoom. Made the identifiers of the arguments for the activate() methods more intuitive. Fixed some errors with the RotatingRoom that were not obvious until now, namely that all RoomExits are now always linked to the RotatingRoom instead of its m\_axisRoom. Improved the functionality of parse() and executeEXIT() so that executeEXIT() can return to main(), where allocated data can be properly deallocated.

Added a HELP command that displays help information about any command.

Touched up some of the code by making some things more readable and adding some more error conditions and display calls. Also added a virtual isAccessible() method to Room so that Rooms interact better with RotatingRooms.

Made isExitBlocked() behave differently when the RoomExit is shared with a RotatingRoom. Created a Switch class. A Switch can store two sets of parameters for activate() methods, and calling activate() on the Switch can use one set of these stored parameters to activate() the Object indicated by the “target” parameter by using the remaining three paramaters, and the set called depends on whether the Switch is “on” or “off,” and whether the Switch “can be toggled.” Furthermore, calling activate() on a Switch can set either set of four parameters.

Expanded the capabilities of a Container so that Objects can be added and accessed. Added a USE command, as well as its HELP information. Made Rooms display the Objects they contain. Added a “title” string field to Objects, and a “position” int field. Added a subDescription string field to Objects, which, for now, indicates whether an Object is on the FLOOR, WALL, or CEILING. Although not yet implemented fully, this can be used, along with a position, to disambiguate the Object of a USE command if there are multiple Objects with the same “title” via the Container method getObject(). Objects located in a RotatingRoom have their position rotated when the rotate() method is validly called, depending on if areOtherExitsVisible() and the value of the Object’s m\_subDescription. An Object cannot be added to a Container if it shares a case-insensitive m\_title, m\_position, or case-insensitive m\_subDescription with an Object already in the Container.

Made PLAYER a constant static field of Character so that various source code files can make reference to PLAYER without compiler conflicts via Character::PLAYER. Made commandUSE accept up to 3 arguments. Made getObject() accommodate 3 arguments. If there are multiple Objects, or no Objects, that match the criteria set by the arguments, then NULL is returned. Otherwise, the matching Object\* is returned.

When Switches activate RoomExits, an appropriate message is now displayed. Made Container.addObject able to be called by Container.activate(). The isCurrentRoom() and isAdjacentRoom() methods are now virtual methods of the Object class and now take a Room\* argument.

Fixed a problem where m\_isOn was not changing its state when a Switch that canToggle() successfully activated its SWITCH\_PRESS action. Fixed Container.addObject() so that an Object will not be added to the Container if the Container already has an Object with the same “title”, “position”, and “subDescription”. Also made Container.addObject() only set the m\_subDescription of the Object being added if it is successfully added to the Container.

Made each Switch able to hold up to 10 on\_actions and 10 off\_actions. Changed Switch.setOnAction() to Switch.addOnAction(), and Switch.setOffAction() to Switch.addOffAction(). Added Switch.deleteOnAction() and Switch.deleteOffAction().

Fixed a bug with a dynamic array field of CorridorRoom not being initialized. Created a sample playable dungeon with an attainable win condition.