C++ Project -- A Zork Game

For the first assignment we will be looking back to one of the first interactive fiction computer games called Zork, a text-based adventure game. You will design a simpler variant of this game which will be able to read as input an XML file with complete information of a specific adventure and create the set of objects to engage the player in that environment. <\p>

example game

- http://iplayif.com/?story=http://www.ifarchive.org/if-archive/games/zcode/zdungeon.z5

You should write the program from scratch. The only code you can download from other sources is an XML parser. This specification is given "as is" If anything is incomplete, inconsistent, or incorrect, please explain your interpretation or improvement of the specification in the README.txt file you will turn in with the rest of your project. One of the purposes of this assignment is to provide an opportunity to learn how to handle an imperfect specification. If you are completely baffled talk to the TA about the project.

Game Commands

- **n, s, e, w** – movement commands to put the player in a different room. If the indicated direction leads to a new room, the description of the new room is be printed to the screen. Otherwise print “Can’t go that way.”
- **i** – short for “inventory”, lists all items in the player’s inventory with each item separated by commas, if the player has more than one item. If there are no items in the player's inventory print "Inventory: empty"
- **take** (item) – changes item ownership from room or container to inventory. If successful print “Item (item) added to inventory”. Hint: “take” can be written as a shortcut for the “put” command)
- **open** (container) – prints contents of container in format “(container) contains (item), (item), …” and makes those items available to pick up. If empty, you should output "(container) is empty."
- **open exit** – if the room is of type exit prints “Game Over” and gracefully ends the program.
- **read (item)** – prints whatever is written on an object that the player has, if something is written on the object and prints “Nothing written.” if nothing is written on the object. If the object is not in the players inventory indicate that by printing an appropriate message.
- **drop** (item) – changes item ownership from inventory to present room and prints “(item) dropped.”

  If the object is not in the players inventory indicate that by printing an appropriate message.
- **put** (item) in (container) – adds the item to the containers inventory and and prints “Item (item) added to (container).” If the object is not in the players inventory indicate that by printing an appropriate message.
- **turn on** (item) – activates an item if it is in the player’s inventory printing “You activate the (item).” and executing commands in the “turnon” element. If the object is not in the players inventory indicate that by printing an appropriate message.
- **>attack** (creature) with (item) – prints “You assault the (creature) with the (item).” and executes “attack” elements if item matches creature’s “vulnerability” and existing conditions are met. If the object is not in the players inventory indicate that by printing an appropriate message.

Behind the Scenes Commands

- **Add** (object) to (room/container) – creates instance of object with a specific owner (does not work on the player's inventory).

- **Delete** (object) – removes object references from game, but the item can still be brought back into the game using the 'Add' command. If a room is removed other rooms will have references to the removed room as a 'border' that was removed, but there is no means for adding a room back in.

- **Update** (object) to (status) – creates new status for object that can be checked by triggers
- **Game Over** – ends the game with a declaration of "Victory!"

Order of Operations

When the user enters a command:

1. Check if triggers override the command
2. Execute command if not overridden
3. Check if the effects of command activate a trigger
4. If the command activates a trigger, perform the indicated actions and check if new the object status resulting from the trigger activate additional triggers until no changes are detected.

Objects

In the descriptions below, if an element is followed by [ ] there may be multiple objects for that element.

- **Room** – may contain the following elements: name, status, type, description, border[ ], container[ ], item[ ], creature[ ], trigger[ ]. Type is assumed to be 'regular' unless specified otherwise.
- **Item** – may contain name, status, description, writing, status, turn on, and trigger[ ]. If an item has a "turn on" element and the "turn on" command is issued by the user, action elements in 'turn on' are to be executed if any of the given conditions are met.
- **Container** – may contain name, status, description, accept[ ], item[ ], trigger[ ]. If an 'accept' element is present, only specific items may be put into the container. The container need not be opened to insert these specific items, and, in fact, cannot be opened until one of those items is inserted. This is used, e.g., to require a key to be placed into a lock before opening a door.
- **Creature** – may contain name, status, description, vulnerability[ ], attack, trigger[ ]. If the "attack" command is issued by a user that match the creature's vulnerability, action elements in 'attack' are to executed if any of the given conditions are met.

A Special "Object"

Triggers – contains one or more conditions, including special conditions of type *command*, all of which need to be satisfied for the corresponding actions to take place; of type, *one or more things to print*, and of type *one or more actions*. The type will be a 'single' (only used once) unless specified to be 'permanent' (unlimited use). The order of execution is to output any 'print' action(s) first followed by action commands, if any, in the order given in the XML description file.

- **Conditions** – owner, status
- **Commands** A user command, recognized as the entire string (i.e. <command >take sword</command>). trigger will pass the 'command' portion of it's condition if there is no command element or if any one of the command element's contents are matched.
- **Owner** – will have *object*, *has*, and *owner* elements.
- **Status** – will have object and status elements.

Context is important. Only triggers in the present context of the game should be checked. This includes the inventory, the present room, and any items, containers, creatures, and items in containers in the present room. The actions those triggers perform, however, can act on any object in the game.>

XML Formatting

- **Root element**: <map >>
- Each type of object will have elements associated with their given descriptions above with the addition of a ‘name’ element.
- Triggers will have the additional complexity of containing a condition and action elements, with the condition having three additional elements to create an ‘if’ statement of the form “if (object) is/isn’t in (owner)” with the is/isn’t being determined by the <has> element.

Grading (total 100 points)

- 24 pts (3 test cases) – rooms - movement between entrance, regular room(s), and the exit
- 24 pts, (4 test cases) items – take/drop, read, turn on, and add/remove
- 16 pts, (4 test cases) containers – take, put in, open, restrictions (accepts element)
- 8 pts (2 test cases) creatures – attack and add/remove
- 28 pts (5 test cases) triggers – permanence and activation with each other object type

Notes

- Commands are case sensitive.
- Triggers always take precedence over default actions, and are only relevant in their given contexts (i.e., a trigger associated with a room should not be tested by your programm if player is not in that room). When testing for triggers, only test triggers of objects in your present context (present room, objects in present room, objects in containers in present room, inventory), however, the trigger can affect any object in the map.
- The Initial room will always be named “Entrance”
- If an error message is not specified, the general error message is "Error".

Tips

- A good structure might have classes corresponding to each object type, with all objects inheriting from a common class which can be searched for triggers.o
- Their should be a clear division between creation of the game and its objects and play.
- For creation, you will find there are several open source XML parsers for C++. One is RAPIDXML.
- Check Blackboard for updates and don't be afraid to ask the TA questions.

A Sample Run-through

- Here is a sample XML File: sample.txt.xml.
- The output correspnding to that input is here: runThroughResults.txt.
- NOTE: in the runThroughResults.txt file the ">" character is to depict input and should NOT be included in your project! It is for ease of reading only!!!>
- A sample pack for further testing: samplepack.zip.

Submission

Please submit your program through Blackboard. You should submit a .zip or .tar file which should expand into a directory with the name .. The directory should contain a Makefile that allows an executable to be biult on one of the standard lab machines.