CSC 407 Programming Assignment #3: OOPL
Due Date: Friday, March 30

In this assignment, select an OOPL that you have no experience with. You may select any OOP language you like other than C++, Common Lisp, JavaScript, and Java. One of C#, F#, Smalltalk (or Squeak) or Ruby is recommended. Other possible languages are VB, Python (if you have not taken INF 120), Objective-C, Ada 95, Delphi, Eiffel, Oberon, or Modula-3.

Implement a small adventure game that consists of at least 3 classes and then some subclasses of those classes. The classes are a Creature (or Fighter) class, a Weapon class and an Armor class. Also, optionally, you could create classes for Treasure and Rooms. The Creature class should describe one creature of the game. The main features of the Creature are that it has hit points, a weapon, some type of armor and a method to fight. A fight method might be something like the following.

```
fight(Creature c) {
        while(isAlive() and c.isAlive()) {
                for(i=0;i<weapon.numAttacks();i++)
                        generate a random number (say between 1 and 20) and if it is
                        greater than c's armor value, you strike c for weapon.getDamage()
                        amount which is a random amount between 1 and the max that your
                        weapon can produce
                if(c.isAlive()) c.fight(me);
        }
}
```

The main character (the user) will be a Creature. There will also be other Creatures that the main character may have to fight.

When the user enters a Room, if there is a Creature, the user may fight the Creature there. The fight will consist of the user and the other Creature taking turns trying to attack each other. To hit the other, generate a random number (say between 1 and 20) and if this number is greater than the other's armor value, it is a hit. Then generate a random number to determine the damage. Subtract this amount from the other's hit points. This continues until the one of the Creatures has no more hit points in which case it dies. The amount of damage possible and the number of "swings per turn" is based on the weapon type. For instance, a sword might be capable of 2 swings per turn and inflict between 1 and 8 points of damage. A dagger might be capable of 4 swings per turn and inflict between 1 and 5 points of damage. Weapons and Armor may have magical properties so that, as long as the Creature or the Room does not emit a magic-dampening field, they have more damage or swings per turn/more protection. For instance, an enchanted sword can be swung an additional time per turn and inflict 2 extra bonus points of damage per hit, or be able to hit a Creature more often. But if the Room dampens magic, then the enchanted sword is the same as an ordinary sword.

Creatures could potentially have more than one Weapon in which case, prior to the fight, a decision has to be made about which Weapon to use. For instance, if the user has a magical dagger and a

non-magical sword and the user enters a magic dampening Room, the user may select the sword, but if magic is allowed, the user may select the dagger. There are also magical weapons that are of no use in a magic dampening Room or against a magic dampening Creature like fireballs. The user can obtain such weapons via Treasure picked up or from a dead Creature. There may be magical Scrolls, Wands and other types of objects.

Create two subclasses of Creature: Human (or Hero) to represent the user and Monster or Fighter that will represent Creatures the user will fight). There can be many subclasses of Monster/Fighter including Magic-Dampening or Magical in which case they have their own form of magical attack or defense (for instance, one might be able to teleport and thus miss a majority of your attacks). There may also be scrolls that generate a magical dampening field so that the user has an easier time against a magical Creature.

Not all Creatures are to be fought. Perhaps provide an aggression value to every Creature. The higher the value, the more likely the Creature will fight the user. The user, upon entering a Room and is not attacked may decide to attack the Creature there or try to avoid the Creature. However, if the user attempts to take anything in the Room (say Treasure), the Creature may attack at that point. The Human/Hero subclass should be able to carry multiple items. The Monster type of class may only be able to carry say one or two items (a Weapon and Armor, unless the Creature has natural weapons (claws) and Armor (tough skin)). This will be your choice. But limit how much the user can carry so that the user may have to decide to drop an item in order to pick up another item.

Populate your game with at least 8 Rooms including an entry Room which is empty and an exit Room which will denote the end of the game. Rooms do not have to be classes but should be stored in an array where the Room number is its index. A Room will contain exits which is an array of int values that are the indices of the adjacent Rooms. For instance, if you have 8 Rooms, all placed adjacently, then Room 0 would have as an exit Room 1, and Room 1 would have as exits Room 0 and Room 2, etc. A different pattern of Rooms would make your game more interesting. For instance, Room 0 may have only one exit, Room 1, but Room 1 might be adjacent to Room 0, Room 2, Room 3 and Room 7. Hardcode in your program the Rooms' adjacencies.

Each Room will have any combination of the following:

- A Creature to fight
- Treasure (money)
- Exits
- Magic dampening
- Discarded items like Weapons, Armor, Scrolls (or other similar items)

You may populate your Rooms by hand (that is, hardcode what goes where) or have this all generated randomly, or some combination.

Upon entering a Room, the action is as follows:

- Display information about the Room (what the user will see)
- Determine if the Creature will fight the user or not (based on a random value and the Creature's aggressive value)
    - If it will fight, the user may either retreat or fight

    - If no fight and the user tries to take something from the Room, this will increase the Creature's aggression but it still may not fight
- If a fight takes place and the user wins, the user can then take anything it fights in the Room as long as the user has room to carry it
- Interact with the user via a simple menu-driven input such as "(1) Fight or (2) Retreat" or "Now that the Creature is dead, do you want to (1) Pick up item ____, (2) Pick up item ____, (3) leave via exit ___ or (4) retreat?" The number of choices will vary based on the number of items to pick up and the number of adjacent Rooms. If the user tries to pick up an item and has no spare room, provide a menu of choices for the user to select an item to drop.

You can populate Rooms with additional items such as food and healing potions. Upon eating food or taking a potion, the user's hit points can be increased (but no more than the starting value). Upon eating food or using a healing potion, it must be removed from the user's list of items. A user should be able to use any item while in a room with a dead Creature, or a Creature that is not attacking the user. However, if a user eats food to restore hit points, after each turn, test the Creature to see if its aggression will cause it to attack. In addition, if desired, you could have a couple of wandering Creatures that move from Room to Room so that a Creature may enter a Room where the user currently is. Any wandering Creatures should have high aggression so that they will most likely attack.

At a minimum, implement the classes of Creature (with subclasses as noted above), Weapons and Armor. You can make Rooms and Treasure classes or primitive types (e.g., Treasure may simply be an int value such as 10 gold pieces, 5 gold pieces, 2 gold pieces). Don't be too overly elaborate in this program in that the main thing to understand is how to implement classes, instance data, methods, use inheritance and polymorphism in the language you select. With that said, try to have as much fun with the project as possible.

Submit: all of your (commented) code, some output generated by the program showing the user interaction and results (do not show all of the output from a run, just a good sample of about 1 page in length) and a report that describes: 1. How well the language matched what you tried to do, 2. What challenges you had in learning the language, 3. What challenges you had in implementing the adventure game using this language and if another language might have been a better choice, 4. How the language you chose differs from Java, and 5. What you would do to enhance your game given more time.