Concordia University
Department of Computer Science
and Software Engineering

Advanced program design with C++
COMP 345 --- Fall 2016

Individual assignment #1

Deadline:	Friday, October 21st, 2016
Evaluation:	5% of final mark
Late submission:	not accepted
Teams:	this is an individual assignment

Problem statement

This is an individual assignment. It is divided into three distinct parts. Each individual student is expected to select one of these parts as his/her assignment. Each part is about the development of a part of the topic presented as the team project. Even though it is about the development of a part of your team project, each assignment has to be developed independently of the others and is not to be presented as an integrated part of the team project, or include the implementation of one another's aspects. Each member of your team is free to choose to do any part, and is expected to follow a different design approach than the other team members that have selected the same assignment topic. Note that the following descriptions describe the baseline of the assignment, and are related to the project description (see the course web page for a full description of the team project).

Part 1: Character

Implement a group of C++ classes that allow the generation of player characters following the d20 game rules. The baseline of this assignment is to create characters belonging to the *fighter* class. The character constructor must automatically generate the following, based on the level (provided at creation time) and class of the character: (1) ability scores (generated randomly) and ability modifiers, (2) hit points (based on constitution modifier and level), (3) armor class (based on dexterity modifier), (4) attack bonus (based level and strength/dexterity modifiers), (5) damage bonus (based on strength modifier). The character must be able to wear one item of each of the following kinds: armor/shield/weapon/boots/ring/helmet.

Part 2: Map

Implement a group of C++ classes that allow the generation of custom maps. This must allow for the creation of custom maps of variable size to be determined prior to the creation of the map. As stated in the project description, it is advised (for simplicity) that you design the map as a grid, where each grid cell is either (1) an empty cell where a character can move or (2) a wall where a character cannot move, or (3) an occupied cell containing a character, opponent, chest, a door, or whatever else your game rules might include. The class should be designed to allow the creation of a blank map given breadth and width, and provide member functions to set any cell to anything it might eventually contain as stated above. Your class should have a method to verify the validity of a map, which verifies that there is at least one clear path between the mandatory begin and end cell.

Part 3: Items and item container

Implement a group of C++ classes that implement an item container. Items can be of type helmet, ring, weapon, shield, armor, and boots. Items may be enchanted with a +1 to +5 enchantment bonus that upon wearing will eventually influence one of the character's statistics (Strength, Dexterity, Constitution, Intelligence, Wisdom or Charisma), armor class, or attack and damage bonus. Different types of items should provide only with a certain possibility of enhancement types as per the table below. Each item must have a member function to return the enhancement it bears (enhancement type and bonus) in order to apply it to the character that wears it. The item container may contain any number of items and must provide member functions to get a specific item from it, or put a new item into it. Eventually the following item containers will be needed in the game: character backpack, character worn items, and treasure chest.

Item	May increase either
Helmet	Intelligence, Wisdom, Armor class
Armor	Armor class
Shield	Armor class
Ring	Armor class, Strength, Constitution, Wisdom, Charisma
Belt	Constitution, Strength
Boots	Armor class, Dexterity
Weapon	Attack bonus, Damage bonus

Part 4: Dice

Implement a C++ class that provides support for rolling dice in the game. The dice are of the following kinds (d4, d6, d8, d10, d12, d20, d100). The Dice class must allow to roll any number of dice of one kind at once using the following regular expression: $x\mathbf{d}y[+z]$. Where x is the number of dice of the dy kind, optionally added with z after all the dice results have been added. The Dice class must parse a string of this form, roll the individual dice, make the addition and return an integer value.

Assignment submission requirements and procedure

You are expected to submit a group of C++ files implementing a solution to one of the problems stated above (Part 1, 2, 3 or 4). Your code must include a *driver* that allows the marker to compile and execute your code on a standard lab computer. The driver should simply create a character, map, item container, or dice object and somehow demonstrate that the character, map, item container or dice was created following the above-mentioned specifications, as well as following the applicable d20 game rules.

For each part, you are given a CppUnit test suite. The code that you deliver must be passing the provided test cases for the part you chose. You must also deliver at least two other relevant test cases and include them in the test suite. The provided test cases are documented using Doxygen. The test cases you provide must also be documented likewise. The header file(s) of your implementation must include in Doxygen documentation a statement of: 1) the game rules involved in their respective implementation, 3) a brief description of your design, 3) the libraries used in the code, including a rationale for the selection of these libraries. The submitted code must include a driver (i.e. a main function) that demonstrates possible uses of the implemented code. The focus of this course being the coding aspect of software development, you are discouraged to submit external documentation.

You have to submit your assignment before midnight on the due date using the ENCS Electronic Assignment Submission system under the category “programming assignment 1”. Late assignments are not accepted. The file submitted must be a .zip file containing all your code. You are allowed to use any C++ programming environment as long as they are usable in the labs. No matter what programming environment you are using, you are responsible to give proper compilation and usage instructions to the marker in a README file to be included in the zip file.

Evaluation Criteria

Analysis:	
Completeness/orrectness of statement of game rules involved (Doxygen) :	5 pts (indicator 4.1)
Design:	
Compliance of solution with stated problem and game rules:	15 pts (indicator 4.4)
Simplicity and appropriateness of the solution:	7 pts (indicator 4.3)
Clarity/completeness of Doxygen documentation:	3 pts (indicator 7.3)
Use of tools:	
Rationale for selection of tools/libraries used:	2 pts (indicator 5.2)
Proper use of language/tools/libraries:	3 pts (indicator 5.1)
Implementation:	
Code readability: naming conventions, clarity, use of comments:	2 pts (indicator 7.3)
Coding style: `.h` and .cpp files:	2 pts (indicator 5.1)
Relevance of test cases provided:	4 pts (indicator 4.4)
Relevance of driver and completeness of presented results:	7 pts (indicator 4.4)
Total	**50 pts (indicator 6.4)**