CS 103 Computer Programming

Assignment Number 5*

April 7, 2017

Deadline: Friday April 14, 2017 before 22h00

Attention

- Make sure that you read and understand each and every instruction. If you have any questions or comments you are encouraged to discuss your problems with your colleagues (and instructors) on Piazza.
- Plagiarism is strongly forbidden and will be very strongly punished. If we find that you have copied from someone else or someone else has copied from you (with or without your knowledge) both of you will be punished. You will be awarded straight zero in this assignment or all assignments.

1 Exercise 1 — Clash of the titans

In this exercise, you are going to have dragons and hydras fight each other.

Hydra

Dragon

The provided main program simulates a Fight between a dragon and a hydra. The hierarchy of classes that model the creatures of this game are missing and you are asked to provide them.

The class Creature A creature is characterized by:

- its name (a constant string);
- its level (an integer);
- its number of health points (health_status; an integer);
- its force (an integer);
- its position (position_, also an integer; for simplicity, our game takes place in 1D; Can be 2D as well an object of Point class).

*Errors and Omissions expected.

These attributes must be accessible to classes deriving from Creature.
The methods defined for this class are:

- a constructor allowing the initialization of the name, level, health points, force and position of the creature using the values passed as parameters, in this order; the constructor accepts zero as default value for the position;
- a method bool alive() returning true if the creature is alive (number of health points greater than zero) or false otherwise;
- a method AttackPoints returning the number of attack points that can be inflicted by the creature to others; the value is computed as the level multiplied by the force if the creature is alive, or zero otherwise;
- a method Move(int), which does not return anything and adds the integer passed as parameter to the position of the creature;
- a method GoodBye() which does not return anything and displays the message (English: <name> is no more!): using strictly this format. `<name>` is the name of the creature;
- a method Weak, which does not return anything and substracts the number of points passed as parameter from the number of health points of the creature, if it is alive; if the creature dies, its number of health points is set to zero and the method GoodBye is called;
- a method Display(), which does not return anything and displays informations about the creature using strictly the following format:

  ```
  <name>, level: <level>, health_status: <points>, force: <force>, \
  Attacking Points: <attack>, position: <position>
  ```

  `<name>` is the name of the creature, `<level>` is its level, `<points>` is its number of health points, `<force>` is its force, `<attack>` is its number of attack points and `<position>` is its position.

The class Dragon A Dragon is a Creature. It has as specific characteristic the range of its flame (flamerange an integer). Its specific methods are:

- a constructor which initializes its name, level, number of health points, the force, the range of the flame and the position of the dragon using the values passed as parameters, in this order; the constructor accepts zero as default value for the position;
- a method Fly(int pos) which does not return anything and allows the dragon to move to the given position pos;
- a method BlowFlame(Creature&) which does not return anything and simulates what happens when the dragon blows its flame towards another Creature:

  1. if the dragon and the creature are both alive and if the creature is in range of its flame, the dragon inflicts its attack points as damage to the creature; the creature weakens by the number of attack points; The dragon also weakens; it loses “of ’ health points, with “of ’ being the distance between the dragon and the creature (the further the dragon has to blow, the more it weakens);
  2. if after this epic fight the dragon is still alive and the creature dies, the dragon increases in level by one unit;

  The creature is in the range of the flame of the dragon if the distance between them is smaller or equal to the range of the flame (you should use the function distance we provide).

The class Hydra A Hydra is a Creature. It has a specific characteristics the length of its neck (necklength, an integer) and the dose of poison it can inject in an attack (poisondose, an integer). Its specific methods are:

- a constructor which initializes its name, level, number of health points, force, the length of its neck, the poison dose and the position using the values passed as parameters, in this order; the constructor accepts zero as default value for the position;
- a method InjectPoison(Creature&) which does not return anything and simulates what happens when the hydra poisons another Creature:

    1. if the hydra and the creature are alive and the creature is in range of the head of the hydra, then the hydra inflicts damage to the creature; the creature weakens by the number of attack points of the hydra plus its dose of poison;
    2. if at the end of the fight the creature is no longer alive, the hydra increases in level by one unit;

    The creature is "in range of the head of the hydra" if the distance the creature and the hydra is smaller or equal to the length of the neck of the hydra.

The function Fight (fight) takes as parameters a dragon and a hydra. It allows:

- the hydra to poison the dragon;
- and the dragon to blow on the hydra.

1.0.1 Execution examples

The example of output below corresponds to the provided program.

```
Dragon red, level: 2, health_status: 10, force: 3, points of attack: 6, positi
is preparing for fight with :
Hydra evil, level: 2, health_status: 10, force: 1, points of attack: 2, positi

1st Fight :
    the creatures are not within range, so can not Attacke.
After the Fight :
Dragon red, level: 2, health_status: 10, force: 3, points of attack: 6, positi
Hydra evil, level: 2, health_status: 10, force: 1, points of attack: 2, positi

Dragon has flown close to Hydra :
Dragon red, level: 2, health_status: 10, force: 3, points of attack: 6, positi

Hydra moves :
Hydra evil, level: 2, health_status: 10, force: 1, points of attack: 2, positi

2nd Fight :
  + Hydra inflicts a 3-point attack on dragon
      [ level (2) * force (1) + poison (1) = 3 ] ;
  + Dragon inflicts a 6-point attack on Hydra
      [ level (2) * force (3) = 6 ] ;
  + during his attack, dragon loses two additional points
       [ corresponding to the distance between dragon and hydra : 43 - 41 = 2
After the Fight :
Dragon red, level: 2, health_status: 5, force: 3, points of attack: 6, positio
Hydra evil, level: 2, health_status: 4, force: 1, points of attack: 2, positio
```

```
Dragon moves by one step :
Dragon red, level: 2, health_status: 5, force: 3, points of attack: 6, positio

3rd Fight :
  + Hydra inflicts a 3-point attack on dragon
      [ level (2) * force (1) + poison (1) = 3 ] ;
  + Dragon inflicts a 6-point attack on Hydra
      [ level (2) * force (3) = 6 ] ;
  + during his attack, dragon lost 1 additional life point.
       [ corresponding to the distance between dragon and hydra : 43 - 42 = 1
  + Hydra is defeated and the dragon rises to level 3
Hydra evil is no more!
After the Fight :
Dragon red, level: 3, health_status: 1, force: 3, points of attack: 9, positio
Hydra evil, level: 2, health_status: 0, force: 1, points of attack: 0, positio

4th Fight:
   when one creatures is defeated, nothing happpens.
After the Fight :
Dragon red, level: 3, health_status: 1, force: 3, points
of attack: 9, position: 42
Hydra evil, level: 2, health_status: 0, force: 1, points of attack: 0, positio
```

2 Exercise 2 — Travel agency

A travel agent wishes that you help him to handle travel offers of his traveling agency.

2.1 Description

Download the source code available at the Slate (voyages.cpp) and complete it according to the instructions below.

1. save the downloaded file as voyages.cpp;
2. write your code so that it perform according to given main function:
3. save and test your program to be sure that it works properly; try for instance the values used in the example given below;

The travel options Our travel agent sells traveling kits composed of different options. These will be modeled using a class `TravellingOptions` (means `TravelOption`).

An `TravellingOptions` is characterized by:

- its name (a string);
- its default price, *flat fee* (a `double`).

The `TravellingOptions` class includes:

- a constructor initializing the attributes using parameters in an order compatible with the given `main`;
- a method `name()` returning the name of the option;

- a method `double price()` returning the flat fee for the option;
- a method `void display(ostream&)` displaying the option according to the following format:

  ```
  <name> -> <price> PKR
  ```

  where `<name>` is the name of the option and `<price>` is its price.

You will also implement an overloading of the operator << able to display by the means of its `display` method.

The traveling options can be specialized using different sub-classes. You will first have to model the following two of them: the transport means (the `Transport` class) and the kind of accommodation (the `Hotel` class).

The Hotel class A `Hotel` is a type of `TravellingOptions` characterized by a number of nights (an integer) and the price per night (a `double`).

The price of a `Hotel` is simply the number of nights multiplied by the price per night, to which the flat fee of the option is added. The computation of that price shall not contain any code duplication at all.

The `Hotel` class shall of course have a constructor conforming to the provided `main` (arguments order: the name, the flat fee, the number of nights and the price per night).

Remember to inherit virtually "class B: virtual public A" to avoid the diamond problem in the case of multiple inheritance (see `https://isocpp.org/wiki/faq/multiple-inheritance#mi-diamond` and `https://isocpp.org/wiki/faq/multiple-inheritance#virtual-inherita`

The Transport class A `Transport` is a `TravellingOptions` characterized by a boolean indicating whether the trip is long or not.

The price of a transport is the constant `LONG_TARIF` (`1500.0`) if the trip is long and `BASIC_TARIF` (`200.0`) if not, to which we add the flat fee of the option. These two constants shall be declared as static members of the class.

The `Transport` class shall of course have a constructor conforming to the provided `main` with arguments given in following order: the name of the option, the flat fee and a boolean with value `true` for long trips and `false` otherwise. By default, a `Transport` has a short trip.

Combined offers The travel agent also proposes combined offers including both transport and hotel. You are asked to implement a class `CombinedOffers` for modeling such offers. A `CombinedOffers` is a `Transport` as well as a `Hotel`, but should behave as a *single* `TravellingOptions`.

The class `CombinedOffers` will contain:

- a constructor conforming to the provided `main`; the arguments will be given according to the following order: the name of the option, the flat fee, the number of nights, the price per night and a boolean with value `true` for long trips and `false` otherwise (trips are short by default);
- a specific redefinition of the method `price`. The price of an `CombinedOffers` is the 75% of the sum of the hotel's price (as computed for a `Hotel`) and the transport's price (as computed for a `Transport`).

Travel kit The travel agent sells kits consisting of multiple options.

You are asked to code a class `TravelKit` as a "heterogeneous collection" of `TravellingOptions` (an `array` of "`TravellingOptions*`").

The class `TravelKit` will also be characterized by the locations of departure and destination (two `strings`).

The class `TravelKit` will have:

- a constructor compatible with the provided `main` (the first argument being the departure and the second the destination);
- a method `double price()` that will calculate the price of the kit as the sum of the prices of all the options;
- a method `AddOption`, compatible with the provided `main` and allowing to add an `TravellingOption` to the collection of options of the kit (the options will be added at the end of the collection);
- a method `Cancel()` that resets the collection to an empty set.

Your design must reflect the fact that a travel kit is an (i.e. can be displayed). The display format for a travel kit must be as follows:

```
Travel from <depart> to <destination>: You have not reserved yet!
```

if the set of travel options is empty, and otherwise:

```
Travel From <depart> to <destination>, With the Options:
- <name option1> -> <price option1> PKR
- ....
- <name optionN> -> <price optionN> PKR
Total Price: <price of kit> PKR
```

where `<depart>` is the departure location of the kit, `destination`, its destination and `<price of kit>` its price. The display will be followed by a newline.

2.2 Execution example

```
Travel from Zurich to Paris, With the options :
   - Trip in train ->  250 PKR
   - Hotel 3* : Les amandiers  ->  540 PKR
Total Price : 790 PKR

Travel from Zurich to New York, With the options :
   - Trip by air ->  1550 PKR
   - Hotel 4* : Ambassador Plazza   ->  600 PKR
Total Price : 2150 PKR

Travel from Zurich to New York, With the options :
   - Hotel 4* : Ambassador Plazza and  trajet by air ->  1650 PKR
Total Price : 1650 PKR

Travel from Zurich to Paris:  You have not reserved yet! !

Travel from Zurich to New York:  You have not reserved yet! !
```

3 Exercise 3 — A Banking System

Please see the document “assignment4-part-II.pdf”.