Object Oriented Programming

Assignment 3 – SE All sections
Submission date : 6th April 2023

Submission Items:

1. Complete code files (that have been provided with the assignment). Comments with all core functions is mandatory and hold 10% of the assignment marks.

Assignment Information

• Coding language: C++

Instructions (before starting the assignment):

1. Assignments are to be done individually.

2. The code you write must be your own and you must understand each part of your code. You are encouraged to get help from the course instructors through google classroom and email.

3. Apply all validations for invalid inputs.

4. Plagiarism: Plagiarism of any kind (copying from others, copying from the internet, etc.) is not allowed. If found plagiarized, you will be awarded zero marks in the assignment. Repeating such an act can lead to strict disciplinary actions and failure in the course.

5. Please start early otherwise you will struggle with the assignment.

Submission Guidelines

a. Only submit .cpp file for each question Your submission.cpp file must contain your name, student-id, and assignment # on the top of the file in the comments. Example the first line of every questions should be //Maheen_Arshad_22i111. Missing this will result in 20% marks deduction in each question.

b. Move your all .cpp in one folder. The folder must contain only submission.cpp files (no binaries, no exe files etc.,). If we are unable to download your submission due to any reason you will be awarded zero mark.

c. Run and test your program on machine before submission. If there is a syntax error, zero marks will be awarded in that specific question.

d. Rename the folder as ROLL-NUM_SECTION (e.g. 21i-0001_A) and compress the folder as a **zip file**. (e.g. 21i-0001_A.zip). Only **zip file** will be acceptable.

e. Submit the .zip file on Google Classroom within the deadline. Late submission will be marked zero. No exceptions

f. Submission other than Google classroom (e.g. email etc.) will not be accepted.

g. The student is solely responsible to check the final zip files for issues like corrupt files, viruses in the file, mistakenly exe sent. If we cannot download the file from Google classroom due to any reason it will lead to zero marks in the assignment.

General Rules

• This is an individual assignment.
• All coding standards must be followed (meaningful variable name, code comments etc.)
• Apply all validations for invalid inputs

Assignment 3 is basically on Operator Overloading and Inheritance:

Q1. In this question you are going to build tower using blocks.

Class Block:

The class Block represents the "blocks" of a tower. There are different types of blocks, as shown in the following example:

ObliquedL ObliquedR Simple(red) Simple(white)

Every Block is characterized by a shape (i.e. ObliquedL, ObliquedR and simple) and a color
You must write the definition of the class Block by:

- **Adding a constructor** taking two parameters of type form and color; there is no default constructor for this class.
- Adding a public method with the prototype
  ostream & display (ostream& out) const； which prints the content of the object using the format:

  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

  If the color is not an empty string, the function should display the name of block shape along with color for example:
  (ObliquedL, Red)

  If the color is an empty string it should display only the shape.

- You should overload the **stream insertion operator <<** for class block, the implementation of this method should use the method display.

The class Build: The class Build is used to build “Towers” in our program. A “Tower ’‘is a collection of “different blocks” placed in a 3-dimensional space.
This class must have an attribute **`Content`** which is a 3-dimensional dynamic array of “different blocks”. The order to internally represent the 3 dimensions (from what direction you look at the tower) is shown here:

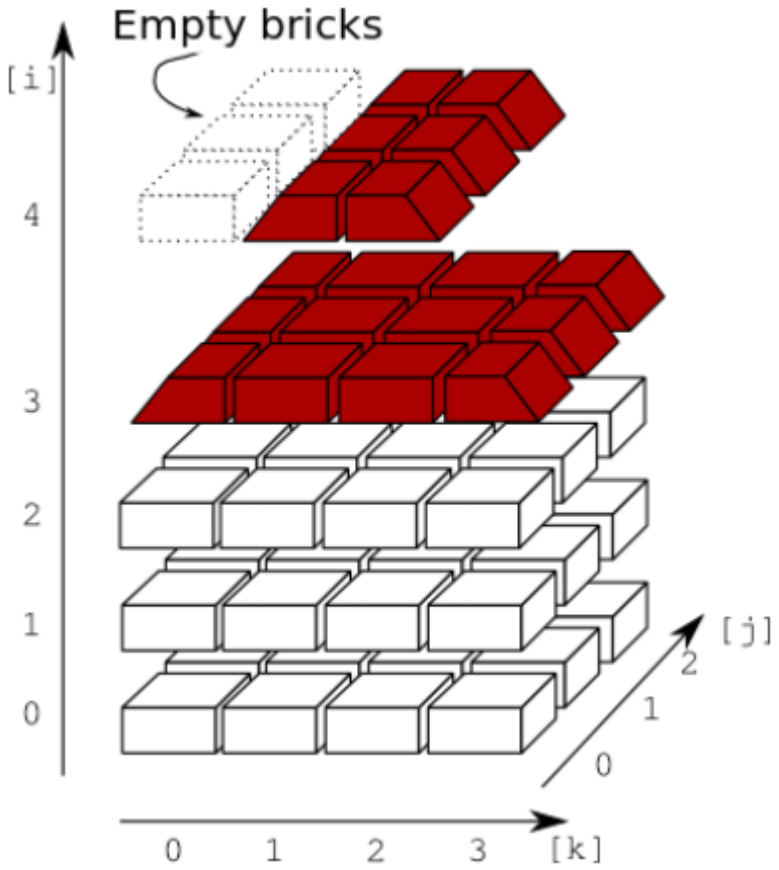


Figure 1

This means that the first index (i above) represents the height, the second (j) represents the depth and the third (k) represents the width so you must write the definition of class Build by:

- **Adding a constructor** which takes as parameter a Block and creating the content as an 1x1x1 array containing only this block;

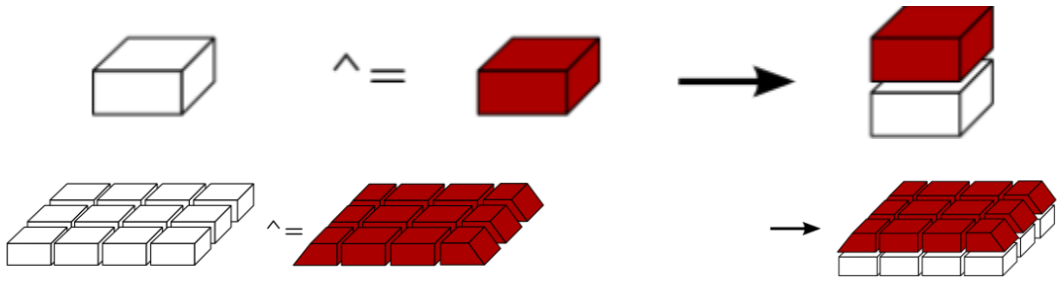
- Adding a public method with the prototype
  ostream& display(ostream& out) const ;
  displaying the tower's content layer by layer as shown further below in the output example (a message Layer number) must be displayed at the beginning of each layer of the tower Building). If tower is empty, nothing is displayed.

Operators for the class Build:

The following ten operators must be added to the class Build (Those operators are only used to describe the layers of the tower. It is not their purpose to check whether the corresponding result is physically possible or not.)

1.Overload the stream insertion operator for printing << which should use the method display;.
2. Overload operator **^=** and operator **^** which adds a layer of blocks on top (remember operator **^=** updates the existing object while operator **^** returns a new one. An example listed below explain the working of these operators.

- **a ^= b**; adds the layer/layers of block b on top of block a, as shown here:



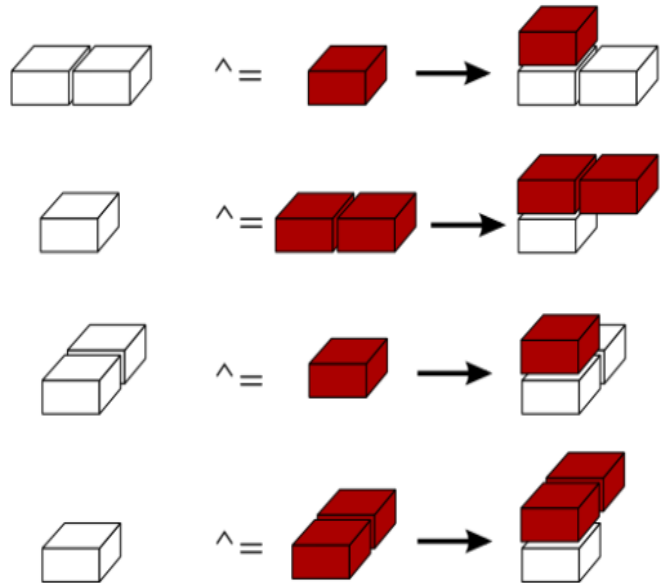


Figure 2

- **a ^ b;** creates a new Tower which is the result of block **b** layer placed on top of block a layer;

3. the operator **-=** and operator **-** which adds block/blocks behind

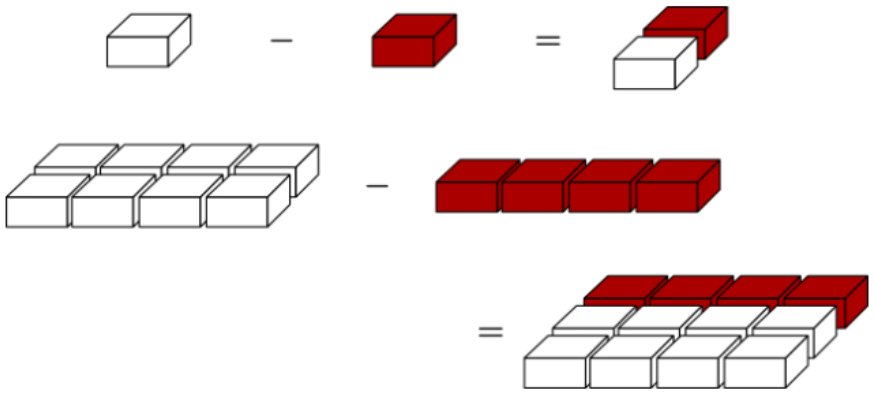


Figure 3

- To make it simple, **-=** and **-** operator does the following (this can be implemented as describe below):
- if the height of b (element added behind) is smaller than the height of a, no action is taken
- (a is not modified); if, by contrast, it is larger, then we only add the part of the same height as a and the rest of b is ignored.
- Here are a few examples:

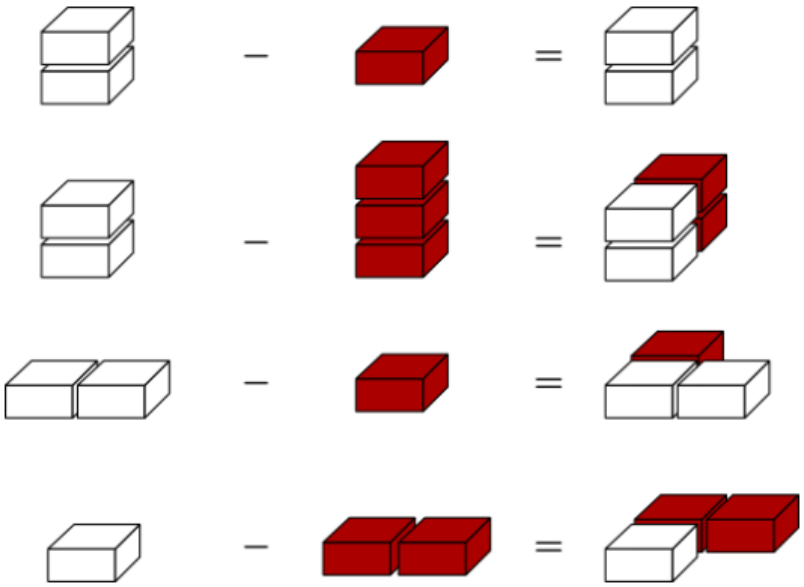


Figure 4

4. The operator**+=** and the operator**+** which adds a block to the right

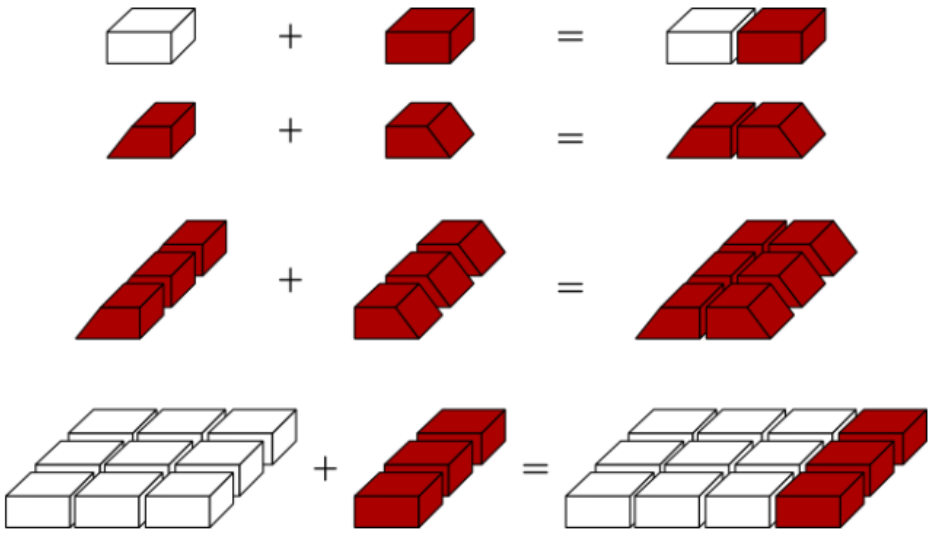


Figure 5

+= and + operator has the following functionality.

- if the height of b (the element added to the right) is smaller than the height of a, no action is taken (a is not modified); if, by contrast, it is higher, then we only add the part of the same height as a, the rest of b is ignored;
- if the depth of b is smaller than the depth of a, no action is taken (a is not modified); if, by contrast, it is larger, we only add the part of the same depth as a, the rest of b is ignored

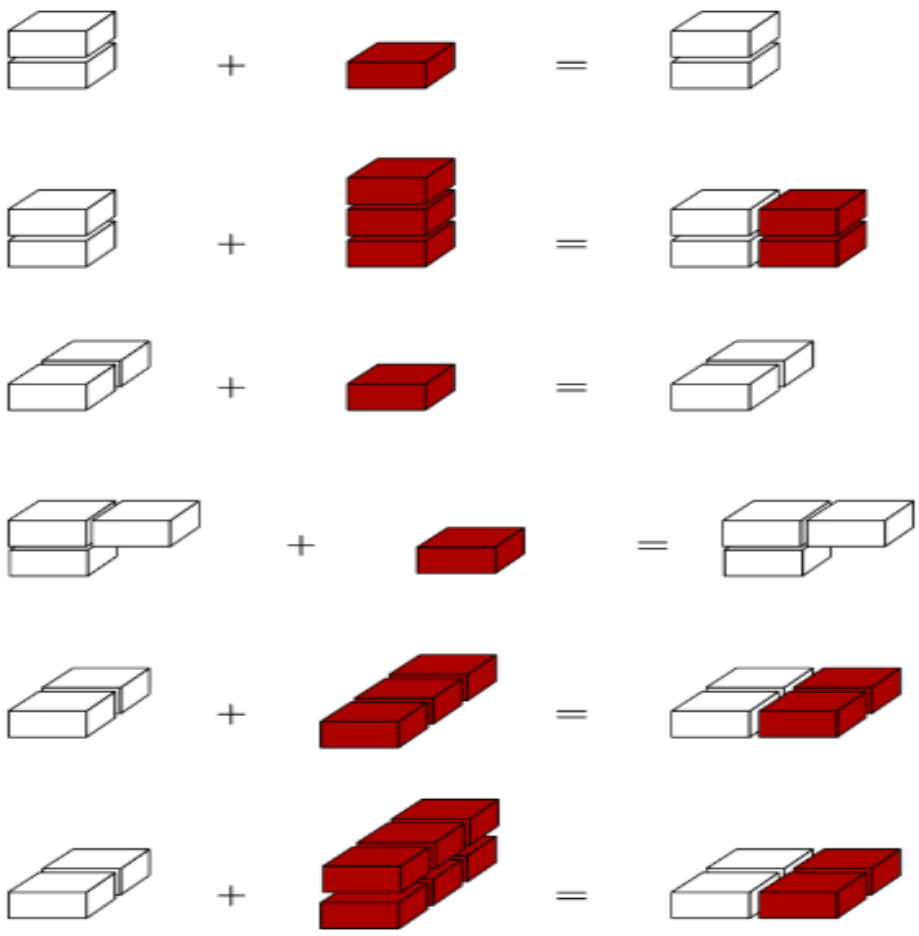


Figure 6

Here are a few more general examples:

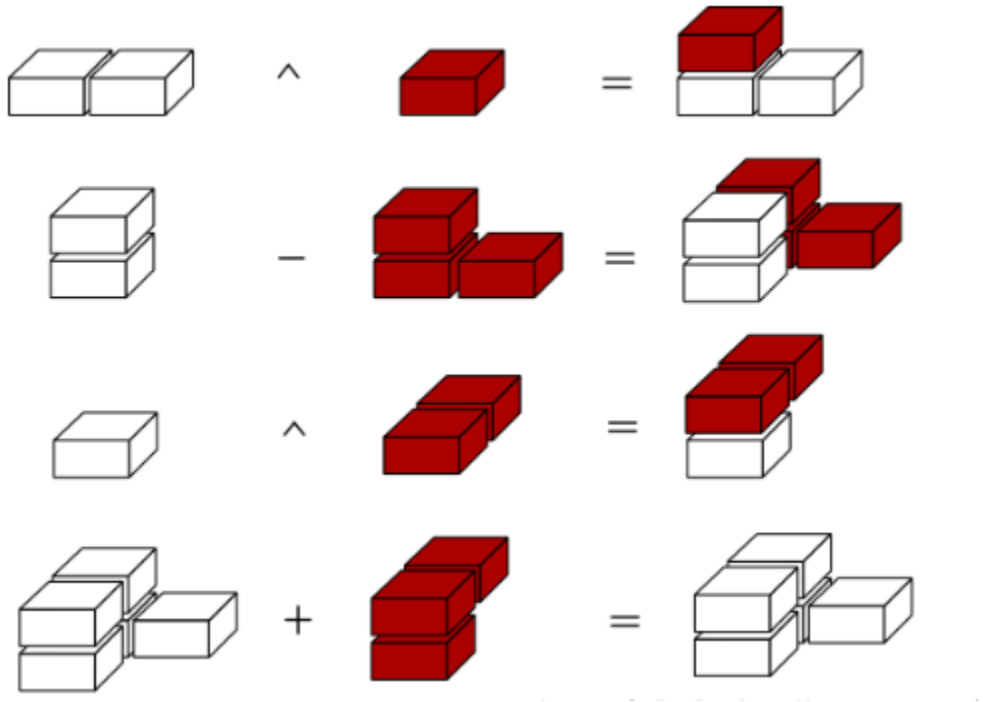


Figure 7

5. The following operators:

Construction operator*(unsigned int n, Construction const& a);
Construction operator/(unsigned int n, Construction const& a);
Construction operator%(unsigned int n, Construction const& a);

- n * a is the same as a + a + a, with + repeated n-1 times;
- n / a is the same as a ˆ a ˆ a, with ˆ repeated n-1 times;
- n % a is the same as a - a - a, with - repeated n-1 times;

Methodology

We suggest that you work carefully, step by step, testing your code after each step:
1. Start with the class Block (and test your implementation);
2. Write the basics of the class Build, including the constructor and the stream insertion operator ;
3. Start by overloading the operatorˆ= that simply adds a layer of blocks above the tower; test it right away with two simple “basic blocks”;
4. next, add and test the operator operator-= that adds blocks in the 2nd dimension;
5. then add the one that works in the 3rd dimension (that is operator+=);

6. continue with overloading their other counterparts operators (operatorˆ, operator- and operator+);
7. finally, add the “repeated operators” (operators operator/, operator% and operator*);
8. verify that the main () gives the expected results.

Execution examples

The example of output below corresponds to the tower shown in figure 1.

```
Layer 4 :
(obliqueL, red) (obliqueR, red)
(obliqueL, red) (obliqueR, red)
(obliqueL, red) (obliqueR, red)
Layer 3 :
(obliqueL, red) ( simple , red) ( simple , red) (obliqueR, red)
(obliqueL, red) ( simple , red) ( simple , red) (obliqueR, red)
(obliqueL, red) ( simple , red) ( simple , red) (obliqueR, red)
Layer 2 :
( simple , white) ( simple , white) ( simple , white) ( simple ,white)
( simple , white) ( simple , white) ( simple , white) ( simple ,white)
( simple , white) ( simple , white) ( simple , white) ( simple ,white)
Layer 1 :
( simple , white) ( simple , white) ( simple , white) ( simple ,white)
( simple , white) ( simple , white) ( simple , white) ( simple ,white)
( simple , white) ( simple , white) ( simple , white) ( simple ,white)
Layer 0 :
( simple , white) ( simple , white) ( simple , white) ( simple ,white)
( simple , white) ( simple , white) ( simple , white) ( simple ,white)
( simple , white) ( simple , white) ( simple , white) ( simple ,white)
```

Question 2:

In this question, you are going to have dragons and ichneumons fight each other.

The provided main program simulates a Fight between a dragon and a ichneumon. 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).

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 (flame range 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).

The class Ichneumon: A Ichneumon is a Creature. It has a specific characteristic the length of its neck (neck length, an integer) and the dose of poison it can inject in an attack (poison dose, 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 Inject Poison(Creature&) which does not return anything and simulates what happens when the ichneumon poisons another Creature:

1. if the ichneumon and the creature are alive and the creature is in range of the head of the ichneumon, then the ichneumon inflicts damage to the creature; the creature weakens by the number of attack points of the ichneumon plus its dose of poison;

2. if at the end of the fight the creature is no longer alive, the ichneumon increases in level by one unit;

The creature is “in range of the head of the ichneumon” if the distance the creature and the ichneumon is smaller or equal to the length of the neck of the ichneumon.

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

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

Execution examples:

The example of output below corresponds to the provided creaturemain.cpp program.

```
Dragon red, level: 2, health_status: 10, force: 3, Attacking Points:
6, position: 0 is preparing for fight with :

Ichneumon evil, level: 2, health_status: 10, force: 1, Attacking
Points: 2, position: 42

1st Fight :

    the creature-s are not within range, so can not Attacke.
```

After the Fight :

Dragon red, level: 2, health_status: 10, force: 3, Attacking Points: 6, position: 0

Ichneumon evil, level: 2, health_status: 10, force: 1, Attacking Points: 2, position: 42

Dragon has flown close to Ichneumon :

Dragon red, level: 2, health_status: 10, force: 3, Attacking Points: 6, position: 41

Ichneumon moves :

Ichneumon evil, level: 2, health_status: 10, force: 1, Attacking Points: 2, position: 43

2nd Fight :

+ Ichneumon inflicts a 3-point attack on dragon

[level (2) * force (1) + poison (1) = 3] ;

+ Dragon inflicts a 6-point attack on Ichneumon

[level (2) * force (3) = 6] ;

+ during his attack, dragon loses two additional points

[corresponding to the distance between dragon and ichneumon : 43 - 41 = 2].

After the Fight :

Dragon red, level: 2, health_status: 5, force: 3, Attacking Points: 6, position: 41

Ichneumon evil, level: 2, health_status: 2, force: 1, Attacking Points: 2, position: 43

Dragon moves by one step

Dragon red, level: 2, health_status: 5, force: 3, Attacking Points: 6, position: 42

3rd Fight :

+ Ichneumon inflicts a 3-point attack on dragon

[level (2) * force (1) + poison (1) = 3] ;

+ Dragon inflicts a 6-point attack on Ichneumon

[level (2) * force (3) = 6] ;

+ during his attack, dragon lost 1 additional life point.

[corresponding to the distance between dragon and ichneumon : 43 - 42 = 1] ;

+ Ichneumon is defeated and the dragon rises to level 3

Ichneumon evil is no more!After the Fight :

Dragon red, level: 3, health_status: 1, force: 3, Attacking Points: 9, position: 42

Ichneumon evil, level: 2, health_status: 0, force: 1, Attacking Points: 0, position: 43

4th Fight :

    when one creatures is defeated, nothing happpens

After the Fight :

Dragon red, level: 3, health_status: 1, force: 3, Attacking Points: 9, position: 42

Ichneumon evil, level: 2, health_status: 0, force: 1, Attacking Points: 0, position: 43
