UEE1303(1009) F22: Homework 1

Due: 2022/6/20(Mon.) 23:55

[Instruction]

- Please put your source code files of each problem into separate folder named StudentID_hw1_1 and StudentID_hw1_2, compress these two folders to **zip** files separately (Ex: 110511000_hw1_1.zip, 110511000_hw1_2.zip), and upload these two zip files to e3 before deadline.
- **If zipped file' s or source code file's name is wrong, your score of this homework is 30% off.**
- **Your source code files should be able to be compiled and executed on our server.**
- Your output should follow the example, otherwise you will not get full credit.
- If you have any question, please send an email to TA or leave a message in the line account.
- You can get files shown in each problem under `/home/share/hw1/`.

[Problem 1]: Ordering System

Please complete the definition of the class `Subway` to support the correct execution of the main program (written in `hw1-1.cpp`).

You should write the `Subway.cpp` and add proper codes in `Subway.h`. You only need to put these three files in the folder StudentID_hw1_1 and zip it to StudentID_hw1_1.zip.

Do not modify `hw1-1.cpp`.

You can only add codes in `Subway.h` but do not modify the existed codes.

Execute: ./hw1-1

Define a class called Subway that has member variables to track the Bread of subway (either Honey Oat, Parmesan Oregano, or Wheat) along with the combo (either Combo A, Combo B, or Combo C) and the number of extras (e.g. avocado, bacon, ……). You can use constants to represent the bread and combo. Create a void function, outputDescription( ), that outputs a textual description of the subway object. Also include a function, computePrice( ), that computes the cost of the subway and returns it as a double according to the rules:

Subway with any type of bread: $15 + $2 per extras
Combo A: $3, Combo B: $5, Combo C: $7

About Input:

1. The user must enter Bread, Combo and Extras in order.
2. Among them, extras must allow the user to repeatedly input different ingredients, and input 0 is regarded as the end of the selection of ingredients. If the user enters duplicate ingredients, the program must send an error message and ask again.
3. At any time, the program should shut down when the user enters -1.

4. When entering, only the numbers in the menu can be entered. Regardless of any illegal input (including all possibilities), the program must output an error message and re-query.

`Subway.h` is shown as below:

```
// Subway.h
#ifndef _SUBWAY_H_
#define _SUBWAY_H_
#include<string>
#include<iostream>
using namespace std;

class Subway{
   private:
      string bread;
      string extra;
      string combo;
      void checkInputNumber(int &n);

   public:
      Subway();
      int getInputBread();
      int getInputCombo();
      void getInputExtra(int inputExtra[]);
      void setBread(int inputBread);
      void setExtra(int inputExtra[]);
      void setCombo(int inputCombo);
      void outputDescription();
      double computePrice(int numberofExtra, int inputCombo);
};

#endif
```

`hw1-1.cpp` is shown as below:

```
//hw1-1.cpp
#include <iostream>
#include "Subway.h"
using namespace std;

int main(){
  while(true){
      int inputBread, inputCombo, inputExtra[10] = {0};
      Subway yourSubway;
      inputBread = yourSubway.getInputBread();
      inputCombo = yourSubway.getInputCombo();
      yourSubway.getInputExtra(inputExtra);
      yourSubway.setBread(inputBread);
      yourSubway.setCombo(inputCombo);
      yourSubway.setExtra(inputExtra);
      yourSubway.outputDescription();

      int numberofExtra = 0, i = 0;
      while (inputExtra[i] != 0){
         numberofExtra++;
         i++;
      }
      double price = yourSubway.computePrice(numberofExtra,
inputCombo);
      cout << endl << "Your subway's price: $" << price << endl
           << "Thank you for your coming!\n" << endl;
   }

   return 0;

}
```

Execution result is shown as below:

```
$ ./hw1-1
```

Output Sample:

```
Choose bread’s type:
1. Honey Oat
2. Parmesan Oregano
3. Wheat
4

Invalid input. Please input again:
2

Choose Combo:
1. Combo A
2. Combo B
3. Combo C
```

```
10

Invalid input. Please input again:
1

Choose extras:
1. Avocado
2. Bacon
3. Egg Patty
4. Extra Cheese
5. Double Meat
6. Egg Mayo
3

Choose extras:
1. Avocado
2. Bacon
3. Egg Patty
4. Extra Cheese
5. Double Meat
6. Egg Mayo
5

Choose extras:
1. Avocado
2. Bacon
3. Egg Patty
4. Extra Cheese
5. Double Meat
6. Egg Mayo
0

Your order:
Bread: Parmesan Oregano
Combo: Combo A
Extras: Egg Patty & Double Meat

Your subway’s price: $22
Thank you for your coming!

Choose bread’s type:
1. Honey Oat
2. Parmesan Oregano
3. Wheat
OOP-HW1!!

Invalid input. Please input again:
3

Choose Combo:
1. Combo A
2. Combo B
3. Combo C
2
```

```

Choose extras:
1. Avocado
2. Bacon
3. Egg Patty
4. Extra Cheese
5. Double Meat
6. Egg Mayo
0

Your order:
Bread: Wheat
Combo: Combo B
Extras: None

Your subway’s price: $20
Thank you for your coming!

Choose bread’s type:
1. Honey Oat
2. Parmesan Oregano
3. Wheat
-1
```

[Problem 2]: Battle Cards

Please complete the definition of class `Char` and `Weapon` in `Object.h` to support the correct execution of the main program (written in `hw1-2.cpp`).

You should complete the `Object.cpp` and add proper codes in `Object.h`. You only need to put these three files in the folder StudentID_hw1_2 and zip it to StudentID_hw1_2.zip.

Do not modify `hw1-2.cpp`.

You can only add codes in `Object.h` but do not modify the existed codes.

- ✓ Input: Character file, Weapon file.
- ✓ Output: The result of each problems.

Execute: $ ./hw1-2 character.in weapon.in

`Object.h` is shown as below:

```
//Object.h
class Weapon
{
  private:
    string name;
    int ATK;
    int MAG;
    int DEF;
    int MDEF;
    int USE;
  public:
    Weapon();
    Weapon(string name, int ATK, int MAG, int DEF, int MDEF, int
USE);
    Weapon(const Weapon&);
    //Input single line of string to initialize weapon object
    Weapon(string);
    //Reset the value of weapon, set all value to 0
    void reset();
    //Decrease the usage of weapon
    void operator--(int);
    friend ostream& operator<<(ostream&, const Weapon&);
};

class Char
{
  private:
    string name;
    int HP;     //Health
    int ATK;    //Attack
    int MAG;    //Magic
    int DEF;    //Defense
    int MDEF;   //Magical Defense
    int POW;    //Amount of power to use magic attack
    int POW_cnt;//Power storage
    Weapon weapon;
  public:
    Char();
    Char(string name, int HP, int ATK, int MAG, int DEF, int MDEF,
int POW);
    Char(const Char&);
    //Input single line of string to initialize Char object
    Char(string);
    //Equip weapon to character and return a new object
    Char operator+(const Weapon&);
    //Increase the POW_cnt by 1
    void operator++(int);
    //Battle
    void operator-(Char&);
    friend ostream& operator<<(ostream&, const Char&);
};
```

`hw1-2.cpp` is shown as below:

```
//hw1-2.cpp
int main(int argc, char** argv)
{
  string line;
  vector<Char> chars;
  vector<Weapon> weapons;
  Char* c;
  Weapon* w;

  ifstream fin(argv[1]);
  while(getline(fin, line))
  {
    //Constructor with input string (single row)
    c = new Char(line);
    chars.push_back((*c));
  }
  fin.close();
  fin.open(argv[2]);
  while(getline(fin, line))
  {
    //Constructor with input string (single row)
    w = new Weapon(line);
    weapons.push_back((*w));
  }
  fin.close();

  /*
  Q1 (20%)
  1. Complete constructors
  2. Complete operator overloading of <<
  */

  cout << "[Problem1]" << endl;

  for(int i=0; i<chars.size(); i++)
    cout << chars[i] << endl;
  for(int i=0; i<weapons.size(); i++)
    cout << weapons[i] << endl;

  /*
  Q2 (20%)
  1. Equip the character with weapon with operator +
  */
  cout << "[Problem2]" << endl;

  Char t1 = chars[0] + weapons[0];
  cout << chars[0] << endl;
  cout << weapons[0] << endl;;
  cout << t1 << endl;

  Char t2 = chars[1] + weapons[1];
  cout << chars[1] << endl;
  cout << weapons[1] << endl;;
  cout << t2 << endl;

  /*
  Q3 (30%)
```

```
  1. Charge power of character with operator ++
  2. Decrease the usage of weapon with operator --
  */

  cout << "[Problem3]" << endl;
  t1++;
  cout << t1 << endl;
  Weapon w1(weapons[0]);
  w1--;
  cout << w1 << endl;

  /*
  Q4 (30%)
  1. Battle operation with operator -
  */
  cout << "[Problem4]" << endl;
  t1-t2;
  cout << t1 << endl;
  cout << t2 << endl;

  return 0;
}
```

`character.in` is shown as below:

Each row specifies a character.

<Name, HP, Attack, Magic, Defense, Magical Defense, Power to use magic>

```
//character.in
Aries 800 26 44 11 9 5
Taurus 1000 29 47 1 8 5
Gemini 700 30 40 13 9 4
Cancer 1000 13 47 14 10 2
Leo 700 20 36 12 5 4
```

`weapon.in` is shown as below:

Each row specifies a weapon.

<Name, Attack, Magic, Defense, Magical Defense, Usage Limit>

```
//weapon.in
WeaponA 5 11 4 5 5
WeaponB 1 8 3 8 4
WeaponC 3 10 4 8 3
```

```
WeaponD 2 12 4 8 1
```

Rule of battle:

1. For each round, two characters will attack each other with physical attack or magical attack, and the character who calls the operator – will attack first.
   For example, there are two characters c1 and c2.
   For physical attack, the damage that c1 causes on c2 will be **the physical attack of c1 (ATK)** + **additional physical attack of weapon that c1 equipped (ATK of weapon)** – **the physical defense of c2 (DEF)** – **additional physical defense of weapon that c2 equipped (DEF of weapon)** and vice versa.
   For magical attack, the damage that c1 causes on c2 will be **the magical attack of c1 (MAG)** + **additional magical attack of weapon that c1 equipped (MAG of weapon)** – **the magical defense of c2 (MDEF)** – **additional magical defense of weapon that c2 equipped (MDEF of weapon)** and vice versa.
2. Only when the magic power storage (POW_cnt) equals to the power requirement (POW) will the character use magical attack to attach the other. After using magic to attack, the power storage will be reset to 0.
3. If the power storage is not enough, the character will use physical attack. After each physical attack, the power storage will increase by 1.
4. There is a usage limit for each weapon. The usage limit of the weapon will decrease by 1 after each **physical** attack. When the usage limit decreases to 0, all the attack and defense values will be reset to 0.
5. The battle will keep going until the HP of one character equals to 0.
6. If a character's HP turns to zero, it cannot attack the other character.

`Example` is shown as below:

```
//Result
[Problem1]
Aries 800 26 44 11 9 5 0
Taurus 1000 29 47 1 8 5 0
Gemini 700 30 40 13 9 4 0
Cancer 1000 13 47 14 10 2 0
Leo 700 20 36 12 5 4 0
WeaponA 5 11 4 5 5
WeaponB 1 8 3 8 4
WeaponC 3 10 4 8 3
WeaponD 2 12 4 8 1
[Problem2]
Aries 800 26 44 11 9 5 0
WeaponA 5 11 4 5 5
Aries 800 26 44 11 9 5 0
```

```
WeaponA 5 11 4 5 5
Taurus 1000 29 47 1 8 5 0
WeaponB 1 8 3 8 4
Taurus 1000 29 47 1 8 5 0
WeaponB 1 8 3 8 4
[Problem3]
Aries 800 26 44 11 9 5 1
WeaponA 5 11 4 5 5
WeaponA 5 11 4 5 4
[Problem4]
Aries 48 26 44 11 9 5 2
Taurus 0 29 47 1 8 5 0
```

Requirements:

1. The output format of a Char object:
   Name, HP, ATK, MAG, DEF, MDEF, POW, POW_cnt
   (separated with whitespace)
2. The output format of a Weapon object:
   Name, ATK, MAG, DEF, MDEF, USE
   (separated with whitespace)
3. If a character equipped with a weapon, print out the information of the weapon at the next line if the usage limit of the weapon is greater than zero.