VG101 Introductions to Computers & Programming
Final
2020 Summer

August 7, 2020

You are to abide by the University of Michigan-Shanghai Jiao Tong University Joint Institute (UM-SJTU JI) honor code. Please sign and upload your honor code file to signify that you have kept the honor code pledge.

We will use JOJ to grade your work. Please refer to JOJ for detailed input/ouptput format and naming requirements. For all problems in final: 1. compress file for each problem in a zip/tar file with no subdir. 2. Write comment section briefly at the beginning of each file describing how you solve this problem. Again: please refer to JOJ for detailed input/output format

1 Problem 1: Back to the beginning (2 sub-problems, 20Pts in total)

1.1 P1.1(10Pts)

Write a program that takes a series of numbers from the command-line argument and print 1 if the values they provide are strictly ascending or descending, and print

0 if otherwise. **The number of input numbers will be greater than 0 and less than 10.**

For example, input

```
ex1_1 1 2 3 4
```

Output should be

```
1
```

input

```
ex1_1 2 1 3 4
```

Output should be

```
0
```

input

```
ex1_1 0 0 1
```

Output should be

```
0
```

1.2 Problem 1.2 (10Pts)

Write a program that takes a string from users after runing the program, and print out the total number of numeric values, alphabetic values, and other special character type.

Input data will not include space character

Output format

```
num_numeric  num_alphabetic  num_others
```

For example, input

```
123ab
```

Output should be

```
3 2 0
```

Input

```
-1a3b#
```

Output should be

```
2 2 2
```

2 Problem 2 (20Pts)

Implement a class of Detector, which, given a series of numbers, prints the largest number that is adjacent to a zero.

```
class Detector
{
public:
/**
* REQUIRES: inputArray
* EFFECTS : print the largest number in the inputArray that is adjacent to a zero.
*/
void maxDetector(vector<int> inputArray);
};
```

Include the design and the implementation of your class in 'detector.h'. You can add private attributes and methods. But **you are not allowed to add or make any changes to the public interface: declarations of attributes and methods.** Your implementation should be able to be compiled with other files correctly using this class like

```
#include "detector.h"
int main()
{
Detector().maxDetector({1, 2, 3, 0, 4, 5, 0});
Detector().maxDetector({1, 5, 3, 0, 2, 7, 0, 8, 9, 1, 0});
Detector().maxDetector({−4,−6,−2,0,−9});
Detector().maxDetector({−1, 0, 0});
}
```

And the output for this sampled test should be

```
5
8
−2
0
```

3 Problem 3 (30Pts)

Implement a class of cashing machine, which supports making change with bills in the value of 100, 50, 20, 10, 5, 1.

The change function will return the change to the users from the highest available bills in the cashing machine.

```
class CashingMachine
{
public:
// Default Constructor
CashingMachine() {}

/**
* EFFECTS : print the status of the machine
*         including the amount of each bill, and the sum of money
*         refer to sample output for the specific format
*/
void printStatus() const;

/**
* REQUIRES: value: the nominal value of the bills to be added
*         can only be 100,50,20,10,5,1
* REQUIRES: amount(positive):  the number of bills to be added
* EFFECTS : add a number of bills to the cashing machine
*/
void add(const int value, const int amount);

/**
* REQUIRES: receive:  the amount of money that casher actually recevied
* REQUIRES: receivable, the money should be received
* EFFECTS : given the received money, make the change to the customers.
*EFFECTS: print "SUCCESS" if the change can be made; print "ERROR", otherwise.
*OTHERS: the elements in received  can only be 100,50,20,10,5,1
*/
void change(vector<int> receive, const int receivable);
};
```

Include the design and the implementation of your class in 'cashingmachine.h'. You can add private attributes and methods. But **you are not allowed to add or make any changes to the public interface: declarations of attributes and methods.** Your implementation should be able to be compiled with other files correctly using this class like

```
#include "cashingmachine.h"

int main()
{
CashingMachine CM;
CM.add(100, 1);
CM.add(20, 1);
CM.add(10, 1);
```

```
CM.printStatus();
\\ Casher received 100+20+1=121, he is supposed to receive 111 only.
\\Time to return some changes
CM.change({100, 1, 20}, 111);
CM.printStatus();
CM.change({100}, 50);
CM.printStatus();
return 0;
}
```

And the output for this sample test should be

```
100: 1
50 : 0
20 : 1
10 : 1
5  : 0
1  : 0
Sum: 130
SUCCESS
100: 2
50 : 0
20 : 2
10 : 0
5  : 0
1  : 1
Sum: 241
ERROR
100: 2
50 : 0
20 : 2
10 : 0
5  : 0
1  : 1
Sum: 241
```

4 Problem 4 (2 sub-problems, 30Pts in total)

In the northern ice lands, there lives a group of Hobbits; in their nearby forest, there lives a group of Elves; in the southern, there lives a group of humans. In this exercise, you are going to simulate the battle between groups.

4.1 Problem 4.1 (10Pts)

Print out the corresponding settings described in the input

4.2 Problem 4.2 (20Pts)

Print out which group will win after the duel

4.3 Code requirement

1. First write a class called Creatures, which includes all possible features and methods.
2. For Elves and hobbits, write classes inherited from the creatures.
3. Simulate the whole game.

4.4 Format

Input format Group size, life value, and damage per attack (DPA) are all positive integers.

number of Elves Humans Hobbits (seperated by space) maximum life of Elves Humans Hobbits (seperated by space) DPA of Elves Humans Hobbits (seperated by space) Group1 Group2(choosing two from the opponents)

Output format for Problem 4.1

num takes the corresponding value in the input

Group size(Elves, Humans, Hobbits): num, num, num Life value(Elves, Humans, Hobbits): num, num, num DPA(Elves, Humans, Hobbits): num, num, num When Group1 fights with Group2

Output format for Problem 4.2: $group\_name \in \{Elves, Humans, Hobbits\}$

group_name wins!

4.5 Details

General logic:
In this game, you will first set up the attributes (life value, damage per attack, group size) for each group via standard input. You will then nominate two groups to fight. The program will determine which group will win in this battle.

Basics:
all members in these groups are creatures, so they would have some common attributes, including life and their damage per attack(DPA).

Victory mechanism
In this game, if a member's life is cut down to 0, he will die. If all members of a group die, this group loses.

Damage mechanism:
If the life value of human is 8, and his DPA is 2; the life value of a elf is 4 and its DPA is 3. Then, if the elf attacks the human successfully, the life of the human will be cut down to 5 (linear subtraction, 8-3).

Duel mechanism
Members from each group start attacking following the attack sequence mentioned later. Every time, one creature v.s one creature. If one creature's life is cut down to 0 or less than 0, he dies. Its place is replaced by another creature from the same group. For example, there are elf 1, elf 2 v.s human 1, human 2. First human 1 vs elf 1, and if elf dies, then elf 2 goes on challenge human 1. If human 1 dies then human 2 goes on to challenge elf 2.

Group feature
1. For Hobbits, if its life is less than 3, it can double its attack every round.
For example, if a Hobbits' remaining life is 2 and his original damage per attack is 3, then his damage per attack will double to 6 during his remaining life time.
2. For Elves, they can escape from one attack for every three round. For example, if it has been attacked for two times, when the third time someone wants to attack it, it will escape from this attack and no life value will be reduced.
3. For humans, no extra group features:(

Attack sequence
1. Initial attack priority is Elves $>$ Humans $>$ Hobbits
2. After the initial attack, members from each group take turns to attack.

4.6 Sample run

For example, for sample input

```
1 2 2
10 3 4
2 3 2
Hobbits Humans
```

The corresponding output for Problem 4.1 is

```
Group size(Elves, Humans, Hobbits): 1, 2, 2
Life value(Elves, Humans, Hobbits): 10, 3, 4
DPA(Elves, Humans, Hobbits): 2, 3, 2
When Hobbits fights with Humans
```

The corresponding output for Problem 4.2 is

```
Hobbits wins!
```

Explanation: In the above battle, it is a battle between a group of two humans and a group of two Hobbits. The details of the duel is: (As the attack sequence says Humans > Hobbits, Humans attack first)

1. Human 1 cause three damage to Hobbit 1, Hobbit 1 life 4-3=1
2. Hobbit 1 DPA becomes 2x2=4 (since its life is smaller than 1), after its attack, Human 1 life '3-2x2=-1', Humans 1 is out.
3. Human 2 takes place. Human 2 causes three damage to Hobbit 1, Hobbit 1 life 1-3=-2, then Hobbit 1 dies.
4. Hobbit 2 with DPA 2, after its attack, Human 2 life 3-2=1.
5. Human 2 with DPA 3, Hobbit 2 life 4-3=1.
6. Hobbit 2 causes attack 2x2=4, Human 2 life 1-2x2=-3, Human 2 dies.