COMP2123A Programming Technologies and Tools

Quiz 1.

Time: 110 minutes

- This quiz consists of 4 questions.
- **Submit your work to Moodle VPL**, for each question, we provide a backup option for you to submit your program if the VPL doesn't work for your program.
- **If you choose the backup option your marks in VPL will not be considered**.
- Please put your student ID card in front of the computer.

Question 1: Find the cheapest item

Please go to the directory `Q1`, there is a directory `quotations` and a shell script `find.sh`.

- The directory `quotations` contains files with `.qua` as suffix of their filenames.
  - Each `.qua` file represents one company's quotation of the products the company sell.
  - Each line in the `.qua` file shows the name and price of a product:

```
productName:productPrice
```

You can assume that there is no space in `productName`, and `productPrice` are all integer values.

- Your task is to update the shell script `find.sh`.

  If we call the script and pass the product name as **the first input argument** as follows:

```
./find.sh productName
```

the script **finds the lowest price of the product** with name equal to `productName` (exact match) among all companies' quotation files.

- You do not need to perform any input error checking.

Sample input

Suppose there are three quotation files `1.qua`, `2.qua`, and `3.qua` in the directory `quotations`.

coke:10 fanta:10 oolong:15 greentea:20	fanta:8 oolong:18 **coke:8** greentea:19 lemonade:13	fanta:20 oolong:14 lemonade:11 **coke:20** greentea:13 juice:30
1.qua	2.qua	3.qua

If we execute the following command with `coke` as the **first input argument**

```
./find.sh coke
```

Sample output

The following will be `echo` by the script, because the cheapest price among all quotations of `coke` is 8.

```
8
```

Question 2: Analysis the Gaussian distribution

Please go to the directory Q2. There are 4 files:

- `generator.cpp` **(provided)** – Source code of a C++ file that
  - Takes two integers `m` and `sd` as input (using `cin`).
  - Generates ONE random integer that follows the Gaussian distribution with mean equals to `m`, and standard deviation equals to `sd` (using `cout`).
- `input.txt` **(provided)** - Contains the input to the C++ program (i.e., `m` and `sd`) separated by a space.
  - E.g., the following `input.txt` has `m=10` and `sd=3`

    ```
    10 3
    ```
- `run.sh` – **(implemented by you)**. If we execute the following:

  ```
  ./run.sh
  ```

  The shell script will perform:
  - **Step 1.** Compile `generator.cpp` with the following command:

    ```
    g++ -std=c++11 -o generator generator.cpp
    ```
  - **Step 2.** Run the executable `generator` with the content in `input.txt` as input, **for 1000 times**.
    - The output numbers should be stored in `output.txt.`
    - Each line in `output.txt` contains one generated number.
  - **Step 3.** Manipulate the data in `output.txt` with some shell commands and PIPE into `plot.sh. plot.sh` generates a graph that shows the distribution of the 1000 generated numbers.

- `plot.sh` **(provided)** – A shell script that processes input with each line follows the following format:

  ```
  [numberOfOccurrence] [generated number]
  ```
  - `[numberOfOccurrence]` is the number of occurrences of the `[generated number]`.
  - For example, if `output.txt` consists of 20 numbers as in Figure 2a, then the input of `plot.sh` should be the number of occurrences of each distinct number in `output.txt`, **in ascending order of the distinct numbers**.

```
8
6
9
12
11
11
12
12
11
1
8
7
11
7
12
12
9
9
9
9
```

Figure 2a. `output.txt` (real one contains 1000 numbers)

```
1 1
1 6
2 7
2 8
5 9
4 11
5 12
```

Figure 2b. Input to `plot.sh`

	(e.g., 5 12 means 12 appears 5 times in `output.txt`)

Sample input

The content of `input.txt` as follows.

```
10 3
```

After executing

```
./run.sh
```

Sample output

A file `output.txt` that contains 1000 numbers generated by the generator is created.
The screen ouput will look like the following:

*Note: The graph may not be the same as yours due to randomness in `generator.cpp` , but it will look like a Gaussian distribution.

```
1
2     *
3     **
4     ***
5     *******
6     ***********
7     ***************
8     *********************
9     **********************
10    ****************************
11    ************************
12    ********************
13    *****************
14    ***********
15    *******
16    **
17    *
18
19
```

Question 3: Pokemon

Please go to directory Q3 and open Pokemon.h and Pokemon.cpp.

- Pokemon.h – Consists of code to define a class Pokemon:

```
#ifndef POKEMON_H
#define POKEMON_H
#include <string>
#include <iostream>
using namespace std;
class Pokemon{

   private:
       string name;
       int level;

   public:
       /* More codes here */
};

/* More codes here */

#endif
```

Update Pokemon.h and Pokemon.cpp so that

- The class Pokemon supports a **constructor** that accepts a string as the initial name of the Pokemon, and set the initial level of the Pokemon to 1.

client1.cpp	Expected output

#include "Pokemon.h"
#include "iostream"
using namespace std;
int main(){
  Pokemon p1("Pikachu");
  return 0;
}
	No output but a Pokemon called Pikachu is created with level 1.

- The class Pokemon supports the **insertion operator (<<)** to output the information as follows.

client2.cpp	Expected output

#include "Pokemon.h"
#include "iostream"
using namespace std;
int main(){
  Pokemon p1("Pikachu");
  cout << p1 ;
  Pokemon p2("Charmander");
  cout << p1 << p2;
	Pikachu(Level 1) **[endl]**
	Pikachu(Level 1) **[endl]**
	Charmander(Level 1) **[endl]**

return 0;
}	

- The class `Pokemon` supports the extraction operator (>>) to update the Pokemon’s name and level.

`client3.cpp`	User input	Expected output

#include "Pokemon.h"
#include "iostream"
using namespace std;
int main(){
  Pokemon p1("Pikachu");
  cout << p1 ;
  cin >> p1;
  cout << p1;
  return 0;
}
	PIKACHU 10 **[enter]**	Pikachu(Level 1) **[endl]**
		PIKACHU(Level 10) **[endl]**

- The class `Pokemon` supports the post-increment operator (++) to increase the Pokemon’s level by one and **return the Pokemon with level before the increment**.

`client4.cpp`	Expected output

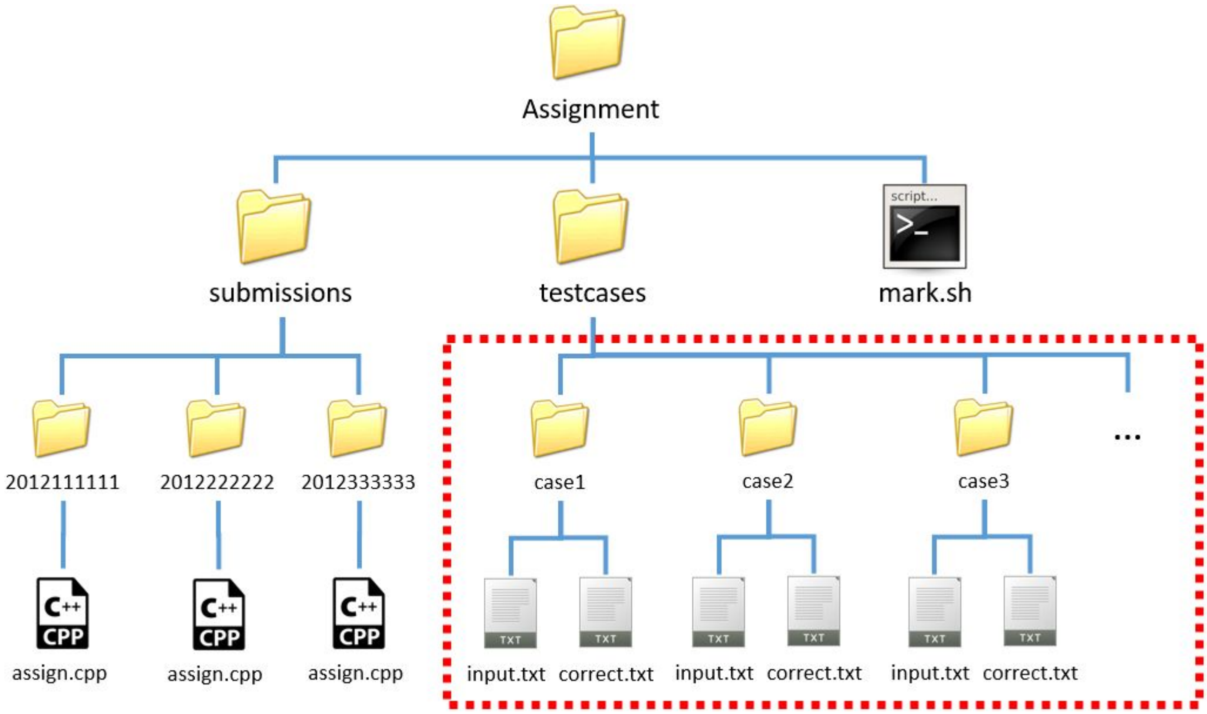
#include "Pokemon.h"
#include "iostream"
using namespace std;
int main(){
  Pokemon p1("Pikachu");
  cout << p1++;
  cout << p1;
  return 0;
}
	Pikachu(Level 1) **[endl]**
	Pikachu(Level 2) **[endl]**

Question 4: Enhancing mark.sh

Please go to directory Q4 and **open Assignment/mark.sh**

- In checkpoint 2.4, we have implemented mark.sh that supports grading with 1 test case only.
- In this question you will update mark.sh to support multiple test cases.

The directory Assignment is organized as follows.

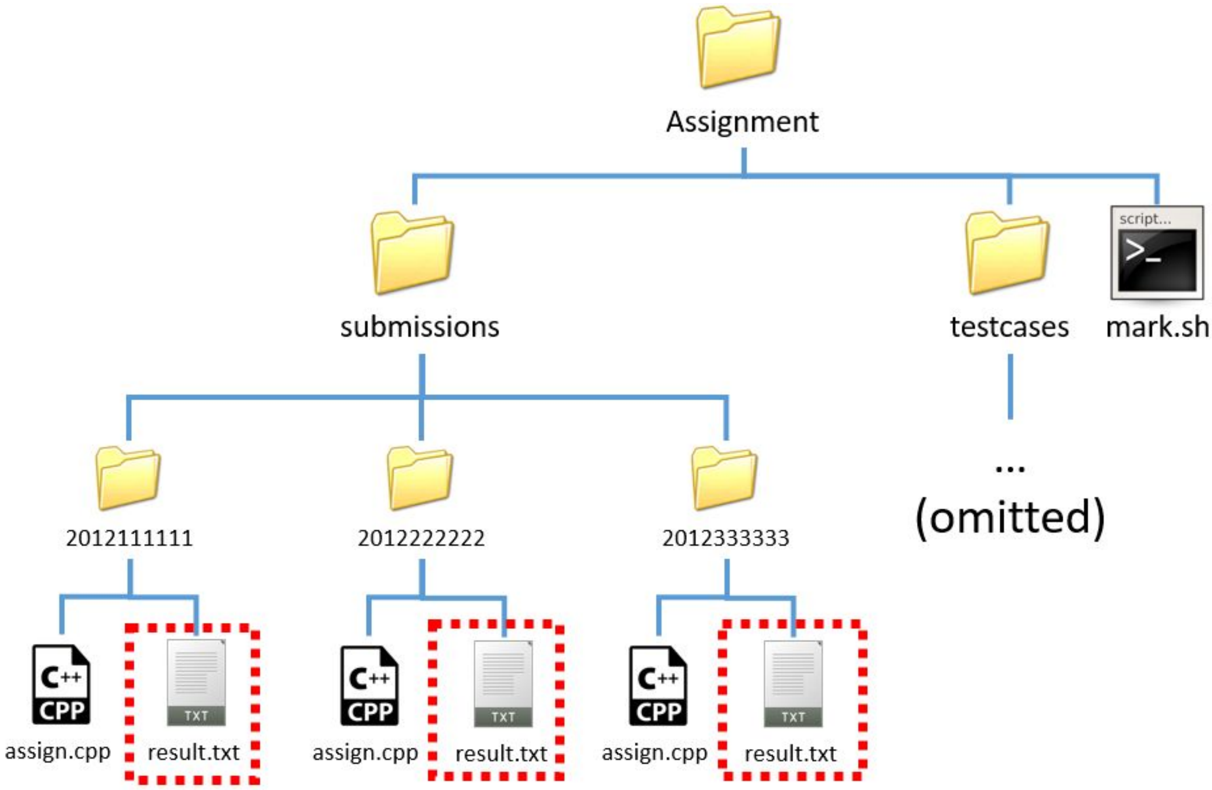


- The test case directories are located in Assignment/testcases/[CASENAME] .
- In each test case directory, there are two files input.txt and correct.txt.

Update mark.sh so that it generates a report file result.txt inside each student's directory.

	Format	Example
When assign.cpp failed to compile.	The result of [UID]: Cannot be compiled.	The result of 2012111111: Cannot be compiled.
When assign.cpp can compile, the executable is then check against each test case.	The result of [UID]: [CASENAME]:Correct! [CASENAME]:Wrong answer.	The result of 2012222222: case1:Correct! case2:Correct! case3:Wrong answer. case4:Wrong answer. case5:Wrong answer.

After running the script `./mark.sh` the directories should look like this:



In the given sample, the `result.txt` of the three students should be as follows:

UID	result.txt

2012111111	`The result of 2012111111:`
	`Cannot be compiled.`

2012222222	`The result of 2012222222:`
	`case1:Correct!`
	`case2:Correct!`
	`case3:Wrong answer.`
	`case4:Wrong answer.`
	`case5:Wrong answer.`

2012333333	`The result of 2012333333:`
	`case1:Correct!`
	`case2:Correct!`
	`case3:Correct!`
	`case4:Correct!`
	`case5:Correct!`

- END OF PAPER -