Important message on plagiarism

The single most important point for you to realize before the beginning of your studies at ShanghaiTech is the meaning of “plagiarism”:

Plagiarism is the practice of taking someone else's work or ideas and passing them off as one's own. It is the misrepresentation of the work of another as your own. It is academic theft; a serious infraction of a University honor code, and the latter is your responsibility to uphold. Instances of plagiarism or any other cheating will be reported to the university leadership, and will have serious consequences. Avoiding any form of plagiarism is in your own interest. If you plagiarize and it is unveiled at a later stage only, it will not only reflect badly on the university, but also on your image/career opportunities.

Plagiarism is academic misconduct, and we take it very serious at ShanghaiTech. In the past we have had lots of problems related to plagiarism especially with newly arriving students, so it is important to get this right upfront:

You may…

• … discuss with your peers about course material.
• … discuss generally about the programming language, some features, or abstract lines of code. As long as it is not directly related to any homework, but formulated in a general, abstract way, such discussion is acceptable.
• … share test cases with each other.
• … help each other with setting up the development environment etc.

You may not …

• … read, possess, copy or submit the solution code of anyone else (including people outside this course or university)!
• … receive direct help from someone else (i.e. a direct communication of some lines of code, no matter if it is visual, verbal, or written)!
• … give direct help to someone else. Helping one of your peers by letting him read your code or communicating even just part of the solution in written or in verbal form will have equal consequences.
• … gain access to another one’s account, no matter if with or without permission.
• … give your account access to another student. It is your responsibility to keep your account safe, always log out, and choose a safe password. Do not just share access to your computer with other students without prior lock--out and disabling of automatic login functionality. Do not just leave your computer on without a lock even if it is just for the sake of a 5--minute break.
• … work in teams. You may meet to discuss generally about the material, but any work on the homework is to be done individually and in privacy. Remember, you may not allow anyone to even just read your source code.

With the Internet, "paste", and "share" are easy operations. Don’t think that it is easy to hide and that we will not find you, we have just as easy to use, fully automatic and intelligent tools that will identify any potential cases of plagiarism. And do not think that being the original author will make any difference. Sharing an original solution with others is just as unethical as using someone else’s work.

CS100 Homework 5 (Spring, 2021)

In this homework, you are required to do simple practices with C++, to get familiar with <iostream> input and output, string operations, and OOP with inheritance and polymorphism. Have fun and enjoy coding. Practice, practice and practice.

Submission deadline:

2021-04-30 23:59:59

Late submission will open for 24 hours when the deadline is reached, with -50% point deduction.

Problem 1. Finding palindromic strings

A palindromic string is a string which reads the same backward as forward, such as *madam* or *racecar*. In this problem, you are required to find all palindromic strings in the input using C++. The input contains only one line, which contains many strings split by a comma (","). You need to read the input, split it by "," and find all palindromes among these strings. Then, you should print the palindromic strings to the screen. The order of the output strings should be the same as the order that they are in the input

For example, if the input string is:

```
abA,a1a,bbbb,xyz,foobar,madoka,QaQ,TAT
```

Your output should be:

```
a1a
bbbb
QaQ
TAT
```

Input description:

The input contains only one line, which is the strings split by ",". It's guaranteed that there is no empty input in all testcases, and the input contains at most 50 strings.

For each string, it's guaranteed that the length is: $1 \le length \le 50$, and each string will only contain letters and numbers (i.e., no whitespaces or "," will occur in these strings). Also, there is at least one string in the input.

Output description:

The output contains `n` lines, where `n` is the number of palindromic strings in the input. Each line contains one palindromic string. The output order should be the same as the input order. If there are no palindromic strings in the input sequence, do not output anything.

Hint: you can use "`#include <string>`" for string operations, and use `std::cin` and `std::cout` (`#include <iostream>` first) for input and output in C++.

Sample input 1:

```
X,y,1a3a
```

Sample output 1:

```
X
y
```

Sample input 2:

```
aba,AcA,F0ObArBaZ,aba
```

Sample output 2:

```
aba
AcA
aba
```

Sample input 3:

```
aA,Aa,AaAa,asdfghjkl
```

Sample output 3:

```

```

Problem 2. Structured Output

Assume you are one of TAs of *CS999*. One day, *Homework??* was released. After the deadline, you need to collect all `n` students' records from OJ and send the report to professor. There are `m` different records (like name, email, or scores) for each student.

> The original report you downloaded from OJ contains `n` lines, matching to `n` students.
>
> For each line, it contains `m` strings separated by a single whitespace (i.e., `" "`), representing someone's information (i.e., "Madoka Madoka@shanghaitech.edu.cn 99 100 81"), there are no whitespaces within these strings.

For an instance, `n = 3` and `m = 4` represents there are 3 students and 4 records per student, a sample score sheet is shown below.

```
Honoka 43253 65789 87912
Kotori 1.7 foo 44
Umi 20 aa 43
```

But the professor of CS999 is extremely strict with formats of reports, here are the requirements:

> The first line of the report should start with a "/", ends with a "\", interiors are filled with "-". (e.g., "`/----------\`")
>
> The last line of the report should start with a "\", ends with a "/", interiors are filled with "-". (e.g., "`\----------/`")

For middle lines, it can either be

(a)."line contains records":

- These lines only contain bar-separators ("|"), records (e.g., `100, GeZiWang, 1.233, OOP`), and whitespaces (" "). There is **at least one whitespace** between each "record" and "bar-separator", the record strings are **left-aligned**.
- e.g., "`| 100 | 200 | 300 |`"

(b)."separate-line":

- Starts and ends with a "|".
- These lines only contain bar-separators ("|") and minus signs ("-"). The "|" should be aligned with other lines.
- e.g. "`|-----|-----|-----|`"

Note:

- **(a) and (b) will occur alternately (i.e., abababa).**
- **All bar-separators should be aligned, all records are left-aligned.**
- **All lines you output should have the same length (same number of characters).**
- **The order of output should be the same as the input.**

For example, the following report is the structured report of report shown above.

```
/-------------------------------\
| Honoka | 43253 | 65789 | 87912 |
|--------|-------|-------|-------|
| Kotori | 1.7   | foo   | 44    |
|--------|-------|-------|-------|
| Umi    | 20    | aa    | 43    |
\-------------------------------/
```

So, you want to write a class called `ReportParser` to structure the report, and give the structured one. We also provide a template for you (as shown).

```
class ReportParser
{
public:
    // The constructor
    ReportParser(int numStudents, int numRecords);
    // The destructor
    ~ReportParser();
    // read & write functions
    void readReport();
    void writeStructuredReport();
    // Add your own functions and variables here
private:
    // Add your own functions and variables here
};
```

Some functions are provided for you:

• `ReportParser(int numStudents, int numRecords)`, The constructor of Report Parser, `numStudents` is the number of students, `numRecords` is the number of records.

• `~ReportParser()`, The destructor of ReportParser .

• `void readReport()`, Read the input report from the console (stdin)

• `void writeStructuredReport()`, Write the structured report to the console (stdout)

Input description:

The input contains `n+1` lines, the first line contains 2 numbers `n`, `m`, represents the total student number and the number of records per line. For the following `n` lines, each line contains `m` records separated by " ", representing the `m` records for this student.

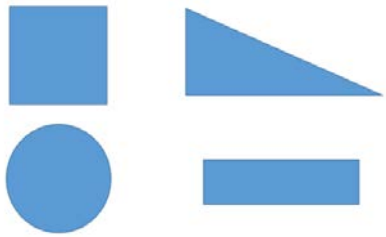
You need to implement the `ReportParser` class, read the input report and structured it into the required format.

Be careful about the requirements of structured report shown above, especially the bar-separators ("|") alignment, record alignment and the whitespaces.

Output description:

Your output is a structured report, as shown above.

Problem 3. Shapes



This problem consists of implementing an abstract class **`Shape`** from which the concrete child classes **`Square`**, **`Rectangle`**, **`Triangle`**, and **`Circle`** are derived. To simplify, the triangle here refers only to a right triangle (one with a 90° angle).

`Shape` should have procedures that permit the access to the area, the perimeter, and the number of corners onto the console. These procedures should be called `CalculateArea`, `CalculatePerimeter`, and `NumberCorners`. Since all child classes should have same procedures, but their calculation may differ, you should consider writing some of them as `virtual` functions.

In addition, these four types of shapes cannot be created similarly. Therefore, they should have different constructors. (All parameters are floating-point numbers)

> Square needs one parameter, its side length;
> Circle needs one parameter, its radius;
> Triangle needs two parameters, length of its two sides that are perpendicular;
> Rectangle needs two parameters, its height and width.

It's also required that, upon destruction of any **`Shape`** object, a line should be printed to the screen, in the form below: (**TYPE_OF_SHAPE should be the same as class name**)

```
A <TYPE_OF_SHAPE> has been destroyed.
```

In calculations of circles, you can either use the given constant `PI` or `M_PI` in `<math.h>`

In all, the interface of Shape and all child classes needs to be designed to be compliant with the main procedure, which is already given in advance. The main procedure first defines a mini-database of shapes, then fills it through user interaction.

Shapes are entered through the console through one of the following inputs:

```
Square <LENGTH_PARAMETER>
Circle <LENGTH_PARAMETER>
Triangle <LENGTH_PARAMETER1> <LENGTH_PARAMETER2>
Rectangle <LENGTH_PARAMETER1> <LENGTH_PARAMETER2>
```

Each time a shape has been entered, the main function creates the corresponding shape object and adds it to the list. Note that we create a pointer to some certain type of shape, but the pointer is implicitly converted into a Shape pointer!

Each time a shape has been entered, the user is prompted if he wants to add more shapes or not, by which the user is supposed to reply with either Y or N (anything other than Y will simply be interpreted as N).

The program finishes with looping through the list and printing the properties of each shape, and finally destroying each shape in the database.

Implement the classes Shape, Circle, Triangle, Square, and Rectangle to comply with the above requirements. You can modify anywhere in the template except the main function. A correct program runs like this: (red indicates input and black indicates your output)

```
Enter a type (Circle, Triangle, Square, or Rectangle) and one
or two size parameters, separated with blank spaces:
Circle 0.5
Do you want to add more shapes? (Enter Y or N)
Y
Enter a type (Circle, Triangle, Square, or Rectangle) and one
or two size parameters, separated with blank spaces:
Triangle 0.3 0.3
Do you want to add more shapes? (Enter Y or N)
N
Properties of shape 0:
Area: 0.7854
Perimeter: 3.1416
Corners: 0

Properties of shape 1:
Area: 0.045
Perimeter: 1.0243
Corners: 3

A Circle has been destroyed.
A Triangle has been destroyed.
```

Problem 4. Alarm Clock

In this problem, you are going to write C++ classes to implement an alarm clock that can add and trigger multiple alarms. Specifically, you will need to implement the `class` **`AlarmClock`** and several classes derived from `class` **`Alarm`**.

Class Implementations:

Your **`AlarmClock`** should keep track of its time. When created, its time will be at 00:00.

Your **`AlarmClock`** should store all the alarms, with a container of `Alarm*`, pointers to `Alarm`. It's up to you what container you choose. You can definitely use an array, but we strongly recommend an STL container (`vector`, `list`, etc.). STL containers are convenient to use, and it's also a good practice of what you learned in this class.

There are two member functions of `class` **`AlarmClock`** that you must implement:

> `void AlarmClock::AddAlarm(Alarm* alarm);`
> This function adds to the alarm clock a new alarm, which is a pointer to `Alarm`, created by the C++ keyword `new`.
> It is guaranteed that no two alarms set to the same time will be added.
>
> `void TimeElapse();`
> This function triggers any alarm at the alarm clock's current time, and then proceeds the alarm clock's time by one minute. (In other words, one minute elapses for this clock)
> If any alarm set at the current time is found, call `Alarm::Trigger()` on it. It's guaranteed that at most one alarm would be triggered at any certain moment.

For other functions, like constructors, destructor, or any auxiliary function you add, you are free on how you implement them.

There are two kinds of alarms for this problem. One is called a repeatable alarm, which will trigger every day at the same time; the other is called a snooze-able alarm, which does not repeat, but has a "snooze" button. If you choose to snooze, the alarm will go off again 10 minutes later. You can also snooze again, or for as many times as you want. Repeatable alarms do not have the snooze feature. Therefore, these two different types of alarms must inherit from an abstract base class **`Alarm`**.

An **`Alarm`** should store information about when it will go off. Also, an **`Alarm`** has a name that can be stored as a `string`.

In the `class` **`Alarm`**, we have provided a pure virtual function, `void Trigger`. This means every child class of **`Alarm`** should implement it. You should also finish the `class` **`Alarm`** with member variables to store the common traits of two alarm types, and functions like constructors and common member functions.

When a `RepeatableAlarm` triggers, it should print a line in the form below:

`"Alarm <NAME> has triggered at <TIME>."` (with a new line)

`<TIME>` is a 24-hour time. For example:

```
Alarm CS100 has triggered at 10:00.
```

Remember, repeatable alarms will trigger every day at their set time.

When a `SnoozeableAlarm` triggers, it should print a line in the form below:

`"Alarm <NAME> has triggered at <TIME>. Snooze? (Y/N)"` (with a new line)

`<TIME>` is a 24-hour time. For example:

```
Alarm MyWeekend has triggered at 08:00. Snooze? (Y/N)
```

It then waits for user input. If the input is a `"Y"`, this snooze-able alarm will be able to go off 10 minutes later (08:10 in this case.), and will also be snooze-able then. If the input is a `"N"`, this alarm will be turned off, and will never trigger, even at the same time of the next day. It's guaranteed that an alarm will never be snoozed into the next day.

Main Function & Input:

The main function simulates your alarm clock for 3 days. At the beginning of each day, you can add alarms by input. Then `TimeElapse()` is called on your clock for 24*60 times, simulating every minute in the day. This process repeats for 3 times.

You are not allowed to change the main function, but you can write the function `void Input(AlarmClock& clock)` to handle the input in your own way, and to create `new` instances of Alarms according to how your own constructors are written.

Input description:

At the beginning of each day, new alarms will be added. The input format is:

The first line is a number `N`, indicating the number of alarms to be added.

Each of the next `N` lines indicate an alarm, in the following format:

```
<TYPE> HH:MM <NAME>
```

`<TYPE>` is a single character, either `'R'` or `'S'`, indicating a [R]epeatable alarm or a [S]nooze-able alarm.

After these alarms are added, the main function simulates `TimeElapse()`. If any snooze-able alarms go off during this process, there will be an additional line of input, either `"Y"` or `"N"`.

Output description:

Apart from output produced in the given main function, for each triggered alarm, there will be a line of output.

On next page is a sample result of a correct program: (red indicates input and black indicates your output)

Hints on formatting time:

If you used int to represent hour and minutes, you will find the functions `std::setw()` and `std::setfill()` in the header `<iomanip>` helpful.

```
Do you want to add any alarms?
2
R 10:00 CS100
S 08:00 MyWeekend
Alarm MyWeekend has triggered at 08:00. Snooze? (Y/N)
Y
Alarm MyWeekend has triggered at 08:10. Snooze? (Y/N)
Y
Alarm MyWeekend has triggered at 08:20. Snooze? (Y/N)
N
Alarm CS100 has triggered at 10:00.
A day has elapsed.
Do you want to add any alarms?
1
R 15:00 afternoon
Alarm CS100 has triggered at 10:00.
Alarm afternoon has triggered at 15:00.
A day has elapsed.
Do you want to add any alarms?
0
Alarm CS100 has triggered at 10:00.
Alarm afternoon has triggered at 15:00.
A day has elapsed.
```