Lab 6

This is a C++ lab. You need to use g++ to compile your source code.

This lab mainly help you understand how to define and implement a class as well as how to take advantage of std::string to substitute C-style string.

This lab is easier than lab4 and lab5 ~

pre-lab reading: string and vector

Comparing with C, C++ is more convenient and efficient when you deal with many problems.

We know that in C, an array or a C-style string cannot be treated as a whole entity. For example, you cannot return an array in a function, or assign an array to another array.

```
int a[10]={0,1,2};
int b[10];
b=a;//wrong in C
char str1[]="820";
char str2[4]=str1;//wrong in C
```

Also, the size of the array is fixed. You can't resize them.

However, in C++, things becomes different.

As long as you include the header file `<string>`, you can treat a `string` type variable as a whole entity. This type of string is far more useful than C-style string!

```
#include<string>
#include<iostream>
using namespace std;
int main()
{
    string s1, s2;
    s1="test1";//OK
    s2=s1+s1;//OK, s2 now is "test1test1"
    s1.push_back('1');//s1 now is "test11"
    cout<<"s1="<<s1<<", s2="<<s2<<endl;//output:"s1=test11, s2=test1test1"
    char ch=s1[2];//ch == 's' now. The index is still avaliable.
    string s3="test3";
    s3[1]='E';//'e' changed to 'E'
    cout<<s3.size()<<endl;//output is 5. No need to consider '\0'
    return 0;
}
```

You can also pass a "string" type variable to a function as a parameter, and allow return value of a function to be a "string" type.

You can see that the length of a "string" variable is not fixed. It can be appended with "+" operator or "push_back(char)" function. You can also explicitly resize it with `resize` function.

```
s3.resize(30);
cout<<s3.size();//30
```

You can also use "==" to judge whether two strings are equal.

While "string" is an upgraded version of C-style string, "vector" is an upgraded version of array.

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    // Create a vector containing integers
    vector<int> v = { 7, 5, 16, 8 };

    // Add two more integers to vector
    v.push_back(25);
    v.push_back(13);

    // Print out the vector
    cout << "v = { ";
    for (int i=0;i<(int)v.size();i++) {
        cout << v[i] << ", ";
    }
    cout << "}; \n";
}
```

output:

```
v = { 7, 5, 16, 8, 25, 13, };
```

You see, the syntax `vector<int> v;` declared a variable v which is an "int" vector. You can treat it as array, but it's more powerful then C-style array. You can change the size, copy the whole vector, or return a vector in a function.

You're recommended to refer to internet resources, like cppreference.com or cplusplus.com for details.

https://en.cppreference.com/w/cpp/container/vector

https://en.cppreference.com/w/cpp/string/basic_string

lab task

Design a class to implement a stock pricing system, which can complete these tasks:

You'll be given a data stream with regard to the stock price. Each datum is consists of two elements: a time stamp and a price. They're both integers. Unluckily, the data stream is not sorted by the time. Also, some data is wrong. If you find two or more records with the same time stamp, the latest record is regarded as correct.

The data stream will be provided with the syntax below:

```
time1 price1
time2 price2
...
```

for example

```
1 20
4 15
2 13
4 18
```

Then the record "4 15" is no longer valid. The price at time stamp=4 is updated to 18. The price at time stamp=1, 2 are 20 and 13 respectively.

Now, as the chief programmer at New York Stock Exchange, you're asked to design a class to implement a stock pricing system, which have these functions:

- **Update/generate** the price of the stock at a time stamp. If the time stamp does not exist before, you need to generate the price. Otherwise, you need to update the price.
- Find the **Latest** stock price in the data stream **until now**. Latest stock price means the price corresponding to the largest time stamp.
- Find the **highest** stock price in the data stream until now.
- Find the **lowest** stock price in the data stream until now.
- Print out all the records which is sorted by time.

A header file is provided for you. You need to implement the four functions at stock.cpp file and write your main function. You're allowed to add other data or functions to help you in the header file.

input commands

update

The syntax is `update <nums>`, for example, `update 3`.

Then you need to input <nums> lines of records. Each record consists of two integers. The first integer is the time stamp, and the second integer is the price.

This command will update/generate the price of the stock at a time stamp.

show all, show highest, show lowest, show latest

show all: print out all the records

show highest: print out the highest price until now

show lowest: print out the lowest price until now

show latest: print out the latest price until now

The final effect of your program should be like this:

stdin:

```
update 3
1 20
4 15
2 13
show all
show latest
update 1
4 18
show highest
show lowest
show all
exit
```

stdout:

```
1 20
2 13
4 15
15
20
13
1 20
2 13
4 18
```

restrictions and clarifications

We make some restrictions and clarifications to help you

1. The commands, including update, show all, show highest, show lowest and show latest, will be called less than 30 times.
2. In command `update <nums>`, the nums will be less than 10.
3. The range for time stamp is [0, 1000]. The time stamp and price are all non-negative integers.
4. Feel free to `#include` more standard libraries in the header file as long as you know how to use them. However, different from lab5, you **must** use the header file and implement the five functions.