As part of a development team at FlexTrade you will regularly review your colleagues work prior to submission

Imagine an intern in your team has written the attached code “VWAPer.cpp” and submitted it for code review

> a) Review the code as you would if it were it a real review in your day as a developer; add comments and suggestions as /*code comments*/; leave any general comments or feedback at the end of the file (this example is massively contrived and badly written - don’t expect code this bad in your day to day life)
>
> b) Write your own implementation of this functionality, you can make any assumptions outside of the specification given in the file
>
> c) How would you check this code works as intended?

there is no time or word limit for this task

a)
1. code style issues:

`typedef basic_string<char> string;` #this redefines a standard library type, it would be better/clearer to use std::string

`if (!strcmp("version", argv[1]))` #the question in the comments didn’t say the first argument had to be the version number

`while (fgets(line, 256, file){` #braces aren’t needed for this while statement if there’s just one call to sscanf

`using namespace std;` #bad coding practice

`int sum = 0;` #this is an unused variable

`FILE* file = fopen(argv[2], "r");` #alot of c style code is used, such as fopen, which doesn’t have exception handling. If argv[0] points to a file that doesn’t exist there will be a segfault. This will also happen if no input parameters are given because it doesn’t check the array length before dereferencing.

`map<std::string, CUpperLower>::iterator itr` #auto could be used for the iterator type

2. this function returns a reference to a local variable which will be destroyed after the function returns so it’s value will be undefined and the code that reads that value will not produce the correct result

```
int& getSum()
{
   int sum = nCurLwr + nCurUpr;
   return sum;
}
```

3. `CUpperLower() : nCurLwr(0), nCurUpr(0)`

nCurLwr is initialized to 0 in the constructor so the comparison with nLow in the add function will always fail and it will never be updated.

4. use const when possible: the getSum function doesn't change any of the object's variables so it can be made const

5.

```
   float     Highs[1000];
   float     Lows[1000];
```

floats are used here but doubles would be better if there were files with higher precision values or if more computations were going to be done on those values

6.

```
sscanf(line, "%s %d %d %f %f",
       Stocks[i], &Intervals[i],
       &Volumes[i], &Highs[i], &Lows[i++]);
```

The order the arguments are evaluated in before they are passed to sscanf is compiler dependent so the post increment might happen before the other arguments are evaluated, meaning some of the arrays will start at 0 and others at 1

7.

```
    int         nCurLwr;
    int         nCurUpr;
```

These variables are ints but the values in the Highs/Lows arrays are floats and so are the values in the example output in the comments, the parameters to the add function are also ints so accuracy will be lost in the narrowing conversion

8.

```
for (int s = 0; s <= i; ++s)
```

The for loop that starts on line 105 will loop over all the items that were added by sscanf (i+1 times) without considering duplicates so the total amount for each stock will be multiplied by how many duplicate values of that stock there were

9.

```
cout << Stocks[s] << "," << Intervals[s] << "," <<
    TotalVolumes[Stocks[s]] / Volumes[s] * 100 << endl;
```

Volume[s] could be 0 and this doesn't check for division by 0 so there will be an arithmetic exception if this happens (if the &Volumes[i] parameter passed to sscanf was evaluated before &Lows[i++] when i=0 and if the getSum function was fixed to not return a local reference).

10. in the code above endl is used instead of just '\n' which has lower performance because it also flushes the stream

11. most of the code is in the main function, the code for calculating the total volume and printing the high/low values could be moved to member functions or custom operators inside the class

12.

```
    while (itr != HighLows.end())
    {
        cout << (*itr).first << "," << (*itr).second.nCurLwr << "," <<
             (*itr).second.nCurUpr << endl;
        ++itr;
    }
```

This could be a range for loop, e.g:

```
for (const auto &item: HighLows) {
    std::cout << item.first << "," << item.second << '\n';
}
```

13. the main function returns 1 which other programs will interpret as an error code

b) the code for this is in the same folder as this document

c) I've added some skeleton code that uses gtest and some comments to explain how it could be done, I've tested the code on some sample inputs and checked the values of the variables by adding breakpoints in gdb