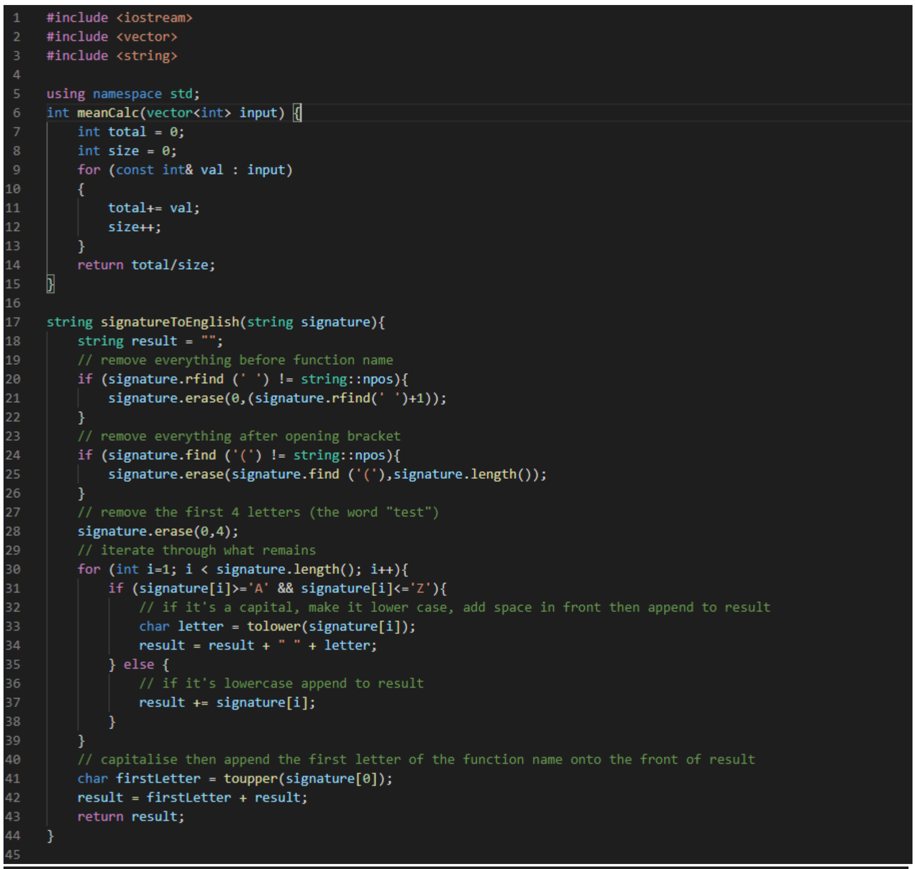
Questions 1&2:

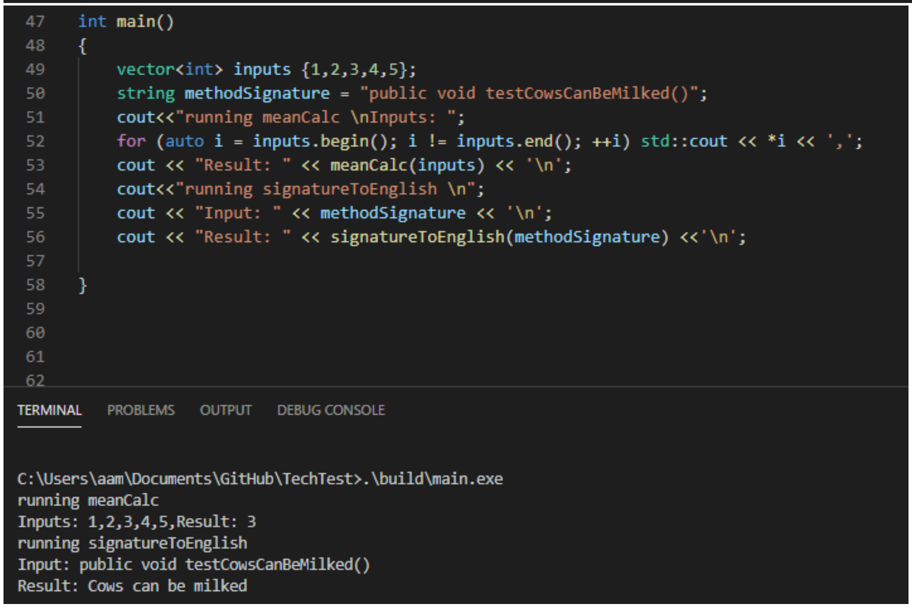


```cpp
#include <iostream>
#include <vector>
#include <string>

using namespace std;
int meanCalc(vector<int> input) {
    int total = 0;
    int size = 0;
    for (const int& val : input)
    {
        total+= val;
        size++;
    }
    return total/size;
}

string signatureToEnglish(string signature){
    string result = "";
    // remove everything before function name
    if (signature.rfind (' ') != string::npos){
        signature.erase(0,(signature.rfind(' ')+1));
    }
    // remove everything after opening bracket
    if (signature.find ('(') != string::npos){
        signature.erase(signature.find ('('),signature.length());
    }
    // remove the first 4 letters (the word "test")
    signature.erase(0,4);
    // iterate through what remains
    for (int i=1; i < signature.length(); i++){
        if (signature[i]>='A' && signature[i]<='Z'){
            // if it's a capital, make it lower case, add space in front then append to result
            char letter = tolower(signature[i]);
            result = result + " " + letter;
        } else {
            // if it's lowercase append to result
            result += signature[i];
        }
    }
    // capitalise then append the first letter of the function name onto the front of result
    char firstLetter = toupper(signature[0]);
    result = firstLetter + result;
    return result;
}

```



```cpp
int main()
{
    vector<int> inputs {1,2,3,4,5};
    string methodSignature = "public void testCowsCanBeMilked()";
    cout<<"running meanCalc \nInputs: ";
    for (auto i = inputs.begin(); i != inputs.end(); ++i) std::cout << *i << ',';
    cout << "Result: " << meanCalc(inputs) << '\n';
    cout<<"running signatureToEnglish \n";
    cout << "Input: " << methodSignature << '\n';
    cout << "Result: " << signatureToEnglish(methodSignature) <<'\n';

}
```

```
C:\Users\aam\Documents\GitHub\TechTest>.\build\main.exe
running meanCalc
Inputs: 1,2,3,4,5,Result: 3
running signatureToEnglish
Input: public void testCowsCanBeMilked()
Result: Cows can be milked
```

all code can be found built inside the "build" folder, the uncompiled code can be read in main.cpp.

Question 3:

11 and 1 will require a hash include as opposed to a forward declaration.

1 is straightforward you cannot use inheritance without #including the class. 11 is a declaration of a non-static member, therefore it needs to be of a complete type or the compiler will throw an error.

Number 14 is interesting as the declaration of a static data member in its class definition is not a definition and can be of an incomplete type.

It's also worth mentioning that you will need a complete definition of the widget class before you can declare an instance of the object, as a result many of these member functions cannot be called and member data values assigned without it.

Question 4:

1. Your exception doesn't appear to be being caught anywhere and as a result will not be displayed, you should implement some try-catch blocks
2. The FileStarError class appears to be expecting a constant char pointer as an input but you are instead passing in a string, I would recommend having it expect a string pointer
3. Your member functions of the FileStar class routinely use class member data before it's defined, I believe this will cause an error in compilation and as a result you should move your data member declarations before your function member declarations within the class definition. (move const char *filename; and FILE *f; to top of class definition)
4. It's not expressly mentioned but unless the FileStarError class is included in the same file as the FileStar class then you will need to add a #include statement for it

Question 5:

Memory management – ensuring that critical path code does not suffer page faults and is still able to store necessary information to reduce repetition when designing the software should improve the speed with which the program runs these pre-calculations.

soft timing based systems – the pre-calculations should be timed and after passing a deadline (or believing it won't be able to complete the computation before the deadline) degrade the complexity of the task it is trying to complete to maintain responsiveness

multithreading - some tasks like these pre-calculations could be completed or at least started on a separate thread allowing for a reduction in time to apply a change based on a control.

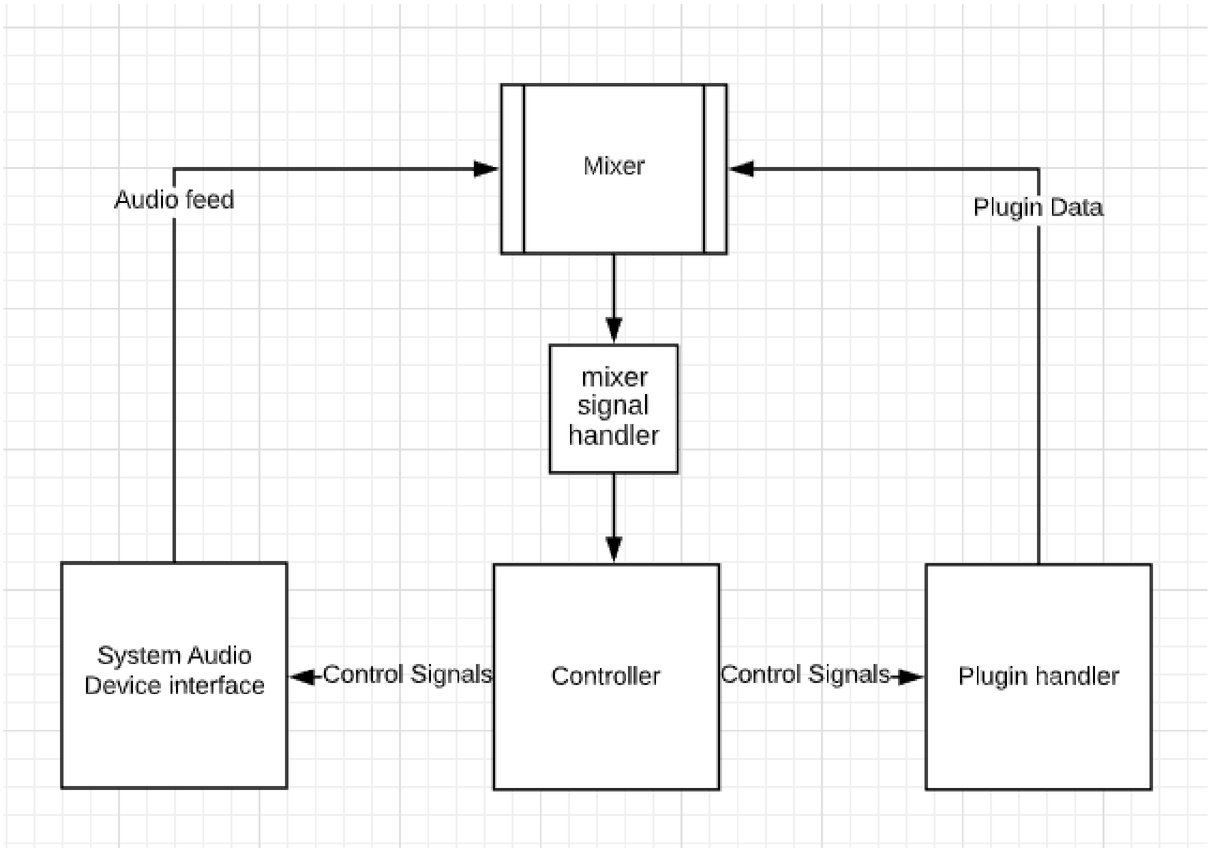
Question 6:

Performing digital to audio signal conversion then finding the peak of that signal

Question 7:

I'm going to freely admit here to not knowing the intricacies of interfacing with system audio drivers, a mixer (I assume an external one?) and whatever that mixer's interface is for managing plugins. I however gave it my best go and the result is a super simplified diagram below.
Essentially in my diagram I delegated interfacing with whatever plugin handling protocols the mixer requires to a "plugin handler" class, I also used similar reasoning and implemented a system audio device interface class for passing audio to the mixer. Both of which are controlled by the controller which takes input signals from the mixer (which are first parsed by a "mixer signal handler" class to simplify the inputs and outputs of the controller class. The goal being modularity to allow for easy

optimisation and testing of individual classes.


Mixer
Audio feed
Plugin Data
mixer signal handler
System Audio Device interface
Control Signals
Controller
Control Signals
Plugin handler