Introduction

This file contains answers to the seven posed questions for the Software Developer C++/ Audio Test. The code answers were programmed in Visual Studio 2022 with C++20 Standard. For questions utilising code, there is a folder containing the .cpp source code and a compiled .exe application.

Each question is approached with care and attention, to highlight current skillsets.

Question One

This question creates a function to calculate the arithmetic mean of a set of given numbers. It is assumed the numbers given have been assigned into a vector with the float variable. Input from a user is not taken for this task. The created function is shown with include dependencies:

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

float mean(std::vector<float> nums) {
    int numEl = nums.size();
    float sumEls = std::accumulate(nums.begin(), nums.end(), decltype(nums)::value_type(0));
    float avgMean = sumEls / numEl;
    return avgMean;
}
```

The mean is calculated by adding each element of the vector together, before dividing by the number of elements. Usage of the std::vector<> container allows for the usage of functions such as the .size() function to count the number of elements. The float variable should be changed if requiring further precision.

Demonstrating this function is completed with various sized vectors, as well as an empty vector to show the null. This occurs within the main() function:

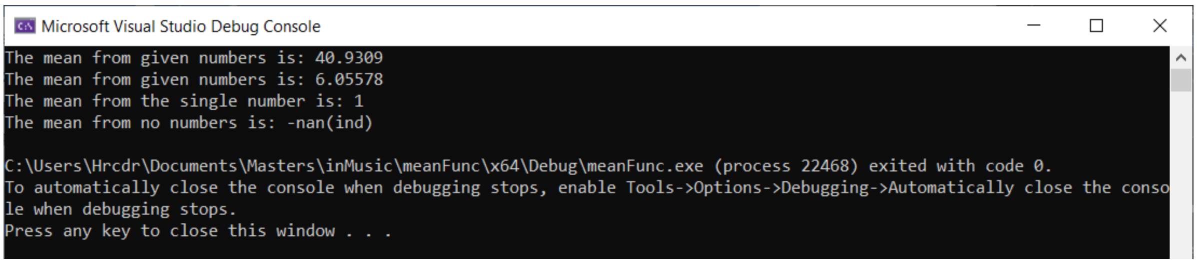
```cpp
int main()
{
    std::vector<float> numbers = {1, 6.0, 2, 3, 4, 5, 6.5, 19, 400, 3.14, 0.6};
// Float type to take integers without dropping the decimals
    std::vector<float> numbers2 = {6, 5.0, 4, 7.77777, 3.45689, 10.10};
    std::vector<float> numbers3 = { 1 };
    std::vector<float> numbers4 = {};
// Empty vector, checking for bugs

    float meanNum = mean(numbers);
    float meanNum2 = mean(numbers2);
    float meanNum3 = mean(numbers3);
    float meanNum4 = mean(numbers4);

    std::cout << "The mean from given numbers is: " << meanNum << std::endl;
    std::cout << "The mean from given numbers is: " << meanNum2 << std::endl;
    std::cout << "The mean from the single number is: " << meanNum3 << std::endl;
    std::cout << "The mean from no numbers is: " << meanNum4 << std::endl;
// -nan(ind) as no value

    return 0;
}
```

Upon compiling and running the program, the output is:



```
The mean from given numbers is: 40.9309
The mean from given numbers is: 6.05578
The mean from the single number is: 1
The mean from no numbers is: -nan(ind)

C:\Users\Hrcdr\Documents\Masters\inMusic\meanFunc\x64\Debug\meanFunc.exe (process 22468) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Showing functionality for various sizes and types of numbers. The full .cpp file is given within the folder, named Question1 with the compiled .exe file.

Question Two

To parse the method signature, the created "FuncParse()"function is called from within the test method functions. This function spaces each word, removes the 'test' beginning word and removes all capital letters except the first:

```
#define FunctionSend() FuncParse( __FUNCTION__ );

#include <iostream>
#include <string>
#include <regex>

void FuncParse(std::string functionName)
{
    std::regex e ("([a-z])([A-Z])");
    std::string spacedName = std::regex_replace(functionName, e,"$1 $2");  // spacing each word
    spacedName = spacedName.substr(spacedName.find_first_of(" \t") + 1);   // removing word:
'test'
    for (int i = 1; spacedName[i] != '\0'; i++)                            // changing upper
case characters to lowercase except for first letter
    {
        if (spacedName[i] >= 'A' && spacedName[i] <= 'Z')    //checking for uppercase characters
            spacedName[i] = spacedName[i] + 32;         //converting uppercase to lowercase
    }
    std::cout << spacedName << std::endl;
}
```

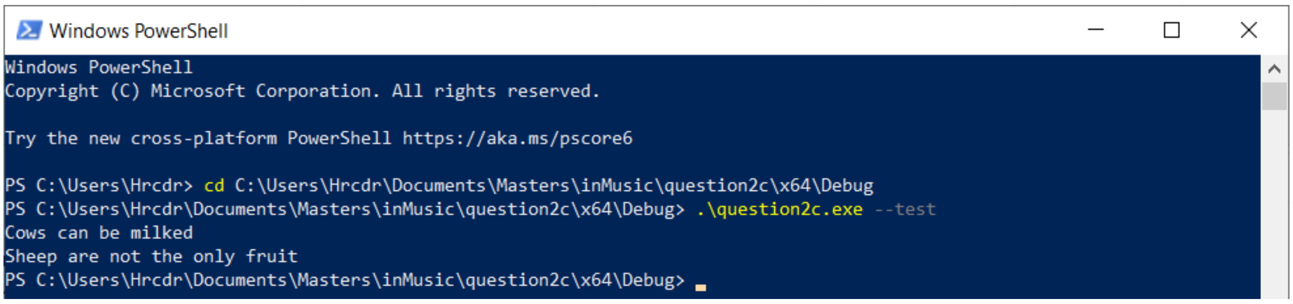
The demonstration of this function working is given as a test style method. The test functions are called when passed the '–test' argument in the command line:

```
void testCowsCanBeMilked() {
    //Test things Here.....
    //Function to be used when reporting the test running
    FunctionSend();
};

void testSheepAreNotTheOnlyFruit() {
    //More test things here.....
    //Function to be used when reporting the test running
    FunctionSend();
};

int main(int argc, const char* argv[])
{
    if (argc > 1 && std::string(argv[1]) == "--test") {
        testCowsCanBeMilked();
        testSheepAreNotTheOnlyFruit();
    }
    return 0;
}
```

This is shown:



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Hrcdr> cd C:\Users\Hrcdr\Documents\Masters\inMusic\question2c\x64\Debug
PS C:\Users\Hrcdr\Documents\Masters\inMusic\question2c\x64\Debug> .\question2c.exe --test
Cows can be milked
Sheep are not the only fruit
PS C:\Users\Hrcdr\Documents\Masters\inMusic\question2c\x64\Debug>
```

The full .cpp file is given within the folder, named Question2 with the compiled .exe file.

Question Three

The usage of #include or a forward dependency is tested via various declarations within this class. Within the numbered dependencies on the widget class, only 1 and 11 require the #include of the header file.

```
class fubar : public widget // 1                      // Widget object, uses #include
                                                         "widget.hpp", otherwise size of class
                                                         unknown
{
    void value_parameter(widget); // 2                 // Parameter, forward declare
    void ref_parameter(widget&); // 3                  // Widget reference, forward declare
    void ptr_parameter(widget*); // 4                  // Pointer, forward declare
                                                       // Note: cannot overload with virtual
                                                          functions with same name

    virtual void value_parameter(widget); // 5         // Parameter, forward declare
    virtual void ref_parameter(widget&); // 6          // run time polymorphism, parameter so
                                                          forward declare

    virtual void ptr_parameter(widget*); // 7          // run time polymorphism, parameter so
                                                          forward declare

    widget value_return(); // 8                        // Widget object, use #include
                                                          "widget.hpp"

    widget& ref_return(); // 9                         // Widget reference, forward declare
                                                          (compiler can assign a size)
    widget* ptr_return(); // 10                        // Widget pointer, forward declare

    widget instance_value_member; // 11                // #include "widget.hpp"
    widget& instance_ref_member; // 12                 // Forward declare
    widget* instance_ptr_member; // 13                 // Forward declare

    static widget static_value_member; // 14           // Does not need object, declaration in
                                                          class, use forward declare

    static widget& static_ref_member; // 15            // Forward declare
    static widget* static_ptr_member; // 16            // Forward declare
};
```

For the cases that require the #include, the reasoning behind it is needing to know the size of the object, whereas in other cases such as number 9, the compiler can allocate a size. Fubar may depend on widget by using its methods or fields.

Question Four

The given code has features that may need to be reviewed and tested for the wanted functionality. A colleague presenting this code would invoke the main comments:

- The constructor for the FileStar class contains definitions of functions as well as passing attributions. Consider rewriting to solely pass attributes.
- The POSIX name ‘strdup’ is deprecated, consider ‘_strdup’.
- Single line variable declaration does not require {} brackets.
- There are variable definitions within the function parameters. Not needed to be placed there.
- Within the ‘void read()’ function of the ‘FileStar’ class, consider passing the ‘buf’ by reference to give the address of the ‘buf’ stored.

The outcome of this review would be to simplify some areas of the code and check the functionality is what is required. Updates to variable naming practice may need to occur for similarity across all the code.

Question Five

For real time audio, the latency of operations occurring within the audio call back buffer needs to be limited, with the worst-case scenario of call back being 10 milliseconds. Without operations, latency can immediately be affected by sample rate and buffer size. If also accepting audio input, this will also be affected in the same way.

General concepts that come to mind when thinking about writing code for operations taking 100s of milliseconds is to pre process the changes. Once the pre-processing has occurred, the real time portion of code would need to not allocate/ deallocate memory or take locks on the audio thread. Algorithms used would also need to be limited in complexity (>O(1)).

Question Six

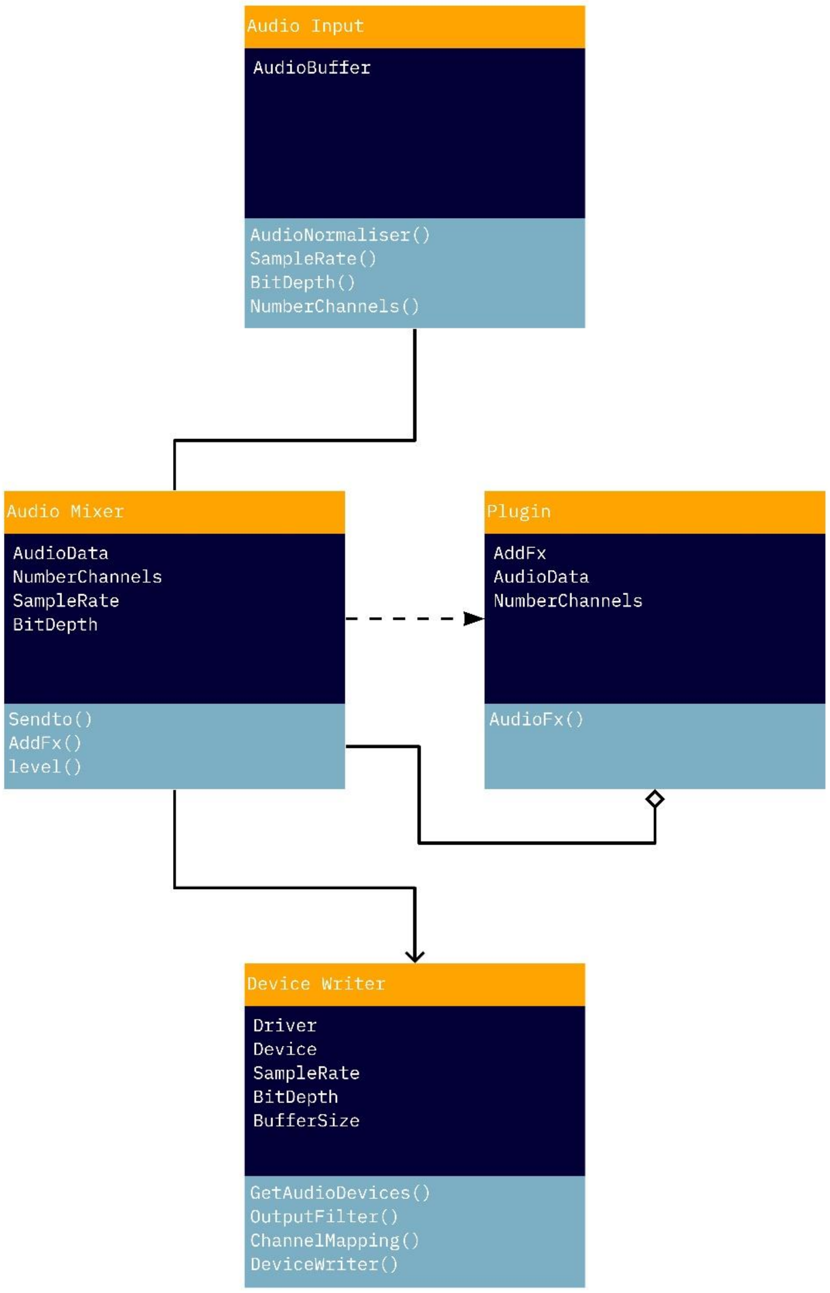
To measure the true peak level of a digital signal, the signal would first need to be oversampled. This allows for the interpolation of data between the samples taken. Depending on the initial sample rate, oversampling by four times for high accuracy is usually observed, however higher sampling rates require proportionally less oversampling. This would then pass through a FIR lowpass filter with a scalar integer and stopband attenuation.

Upon completion of interpolation, this filtered signal (S) would be rectified and converted to the decibel true peak (dBTP) scale:

$$c = 20 \times log_{10}(|S|)$$

Therefore, the true peak would be determined to be the maximum decibels of the converted digital signal.

Question Seven



A program that utilises the installed audio device of a system, a mixer components and external plugins requires prior knowledge of the inputted audio, obtained before moving into the mixer component. This data can be shared with the device send for digital to analogue conversion and any plugin that would feedback to the mixer. This attempt to solve the given question comes at a level of minimal execution due to estimations of UML type diagrams.