Question One

#include <iostream>

using namespace std;

void CalculateMean(int limit)

{

    double mean, sum = 0;

    double numbers[limit];

    cout << "Please enter numbers" << endl;

    for (int i = 0; i < limit; i++)

    {

        cin >> numbers[i];

        sum += numbers[i];

    }

    cout << "Sum of " << limit << "numbers is :" << sum << endl;

    mean = sum / limit;

    cout << "Mean of numbers is :" << mean << endl;

}

int main()

{

    int limit;

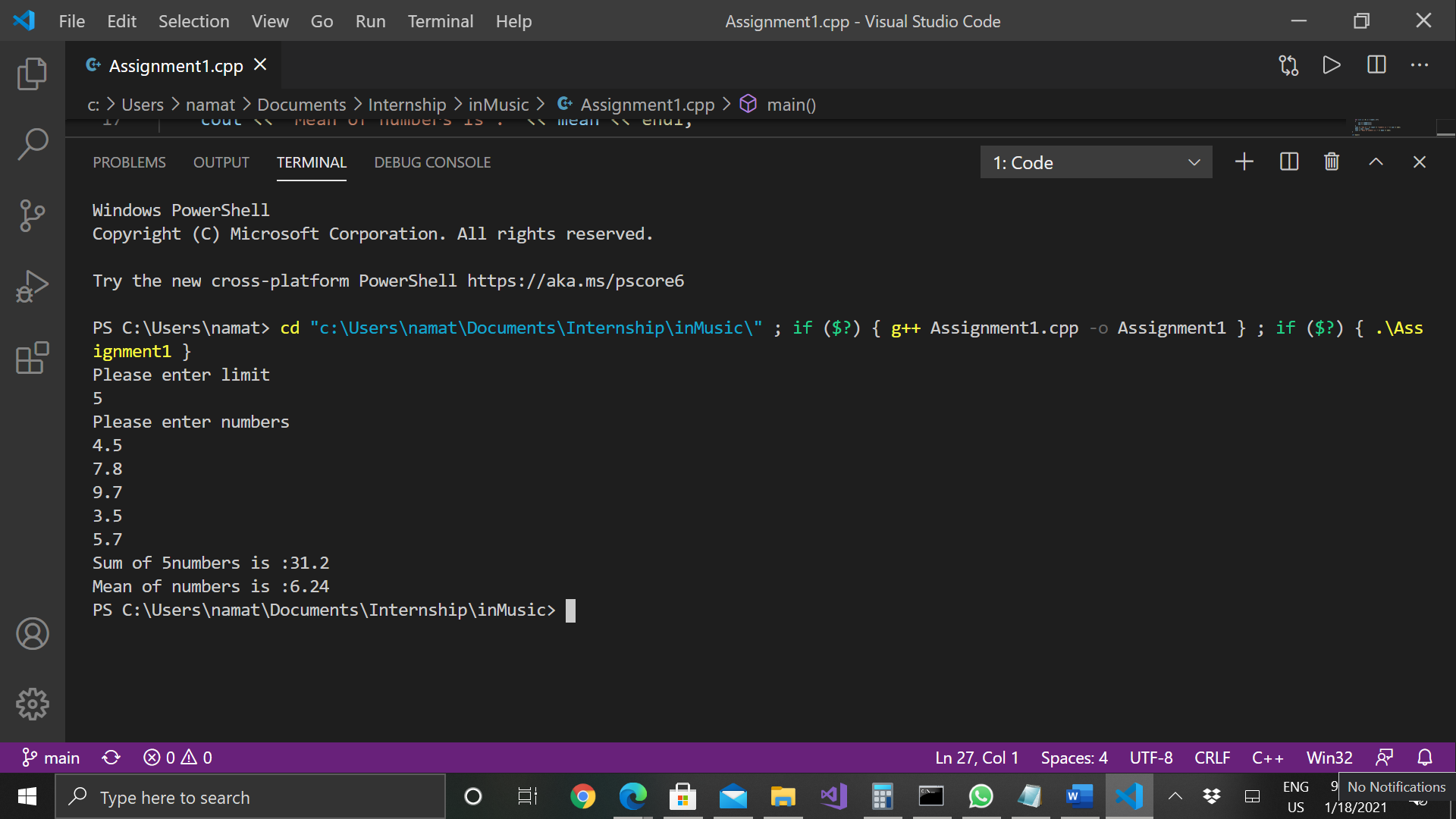
    cout << "Please enter limit " << endl;

    cin >> limit;

    CalculateMean(limit);

    return 0;

}



Question Two

#include <iostream>

#include <string>

#include <cstring>

using namespace std;

void testCowsCanBeMilked();

void testSheepAreNotTheOnlyFruit();

void DisplayFunctionName(string FunName);

void testCowsCanBeMilked()

{

    cout << "Test Function name is : " << \_\_func\_\_ << endl;

    string FunName = \_\_func\_\_;

    DisplayFunctionName(FunName);

}

void testSheepAreNotTheOnlyFruit()

{

    cout << "Test Function name is : " << \_\_func\_\_ << endl;

    string FunName = \_\_func\_\_;

    DisplayFunctionName(FunName);

}

void DisplayFunctionName(string FunName)

{

    int len;

    string test = "test";

    if (FunName.find(test) != string::npos)

    {

        len = FunName.length() - 4;

        FunName = FunName.substr(4, len);

        cout << "Function Name  :" << FunName << endl;

    }

    char CharArray[FunName.length() + 1];

    strcpy(CharArray, FunName.c\_str());

    cout << "Display version is : " << CharArray[0];

    for (int i = 1; i < len; i++)

    {

        if (isupper(CharArray[i]))

        {

            cout << "  ";

        }

        cout << CharArray[i];

    }

}

int main()

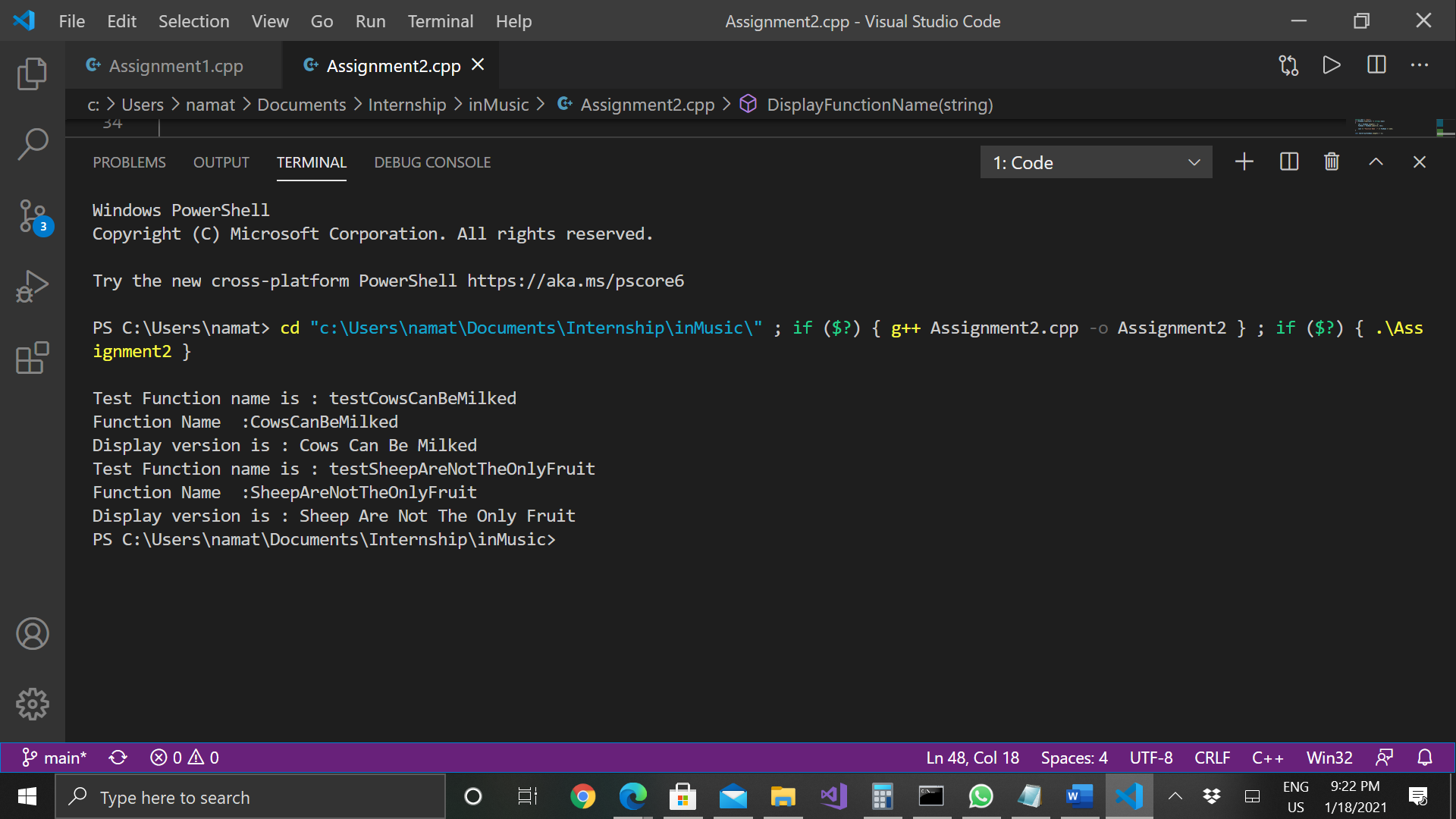
{

    testCowsCanBeMilked();

    testSheepAreNotTheOnlyFruit();

    return 0;

}



Question Three

class fubar : public widget // 1 Require header

{

void value\_parameter(widget ); // 2 Forward Declaration

void ref\_parameter(widget &); // 3 Forward Declaration

void ptr\_parameter(widget \*); // 4 Forward Declaration

virtual void value\_parameter(widget ); // 5 Require header

virtual void ref\_parameter(widget &); // 6 Require header

virtual void ptr\_parameter(widget \*); // 7 Require header

widget value\_return(); // 8 Forward Declaration

widget & ref\_return(); // 9 Forward Declaration

widget \* ptr\_return(); // 10 Forward Declaration

widget instance\_value\_member; // 11 Require header

widget & instance\_ref\_member; // 12 Require header

widget \* instance\_ptr\_member; // 13 Forward declaration

static widget static\_value\_member; // 14 Forward Declaration

static widget & static\_ref\_member; // 15 Forward Declaration

static widget \* static\_ptr\_member; // 16 Forward Declaration

};

Header File

Header file in C++ is a file that can contain function definitions, data type definition etc and which is written only once and can be used in different files. When we include these files with the preprocessor directive “#include”, compiler understand that the file has to processed before compilation. We can create reusable code to a file and create a user defined file to another file for more readability and also to use its functionality.

Say we have a arithmetic function in arithmetic.h

double FindSumOfTwoNumbers(double a, double b){

    return a+b;

}

Now we can use FindSumOfTwoNumbers() function in all the files where we included the arithmetic.h

Syntax - #include <arithmetic.h> or #include “arithmetic.h”

Forward Declaration

In C++, the compiler compiles top to bottom. If we write user defined function A after the main method or another method B using method A, the compiler will be so confused while compiling the code main or method B as it doesn’t know what exactly is method B. In order to solve the problem we an just declare method A on top or before its usage. So while compiling the Compiler will understand the existence of Method A.

#include <iostream>

using namespace std;

int main()

{

    findsum(3, 6);

}

void findsum(int x, int y)

{

    cout << x + y;

}

In above program the compiler starts compiling from top to bottom and compiler will not understand about findsum while compiling main method. So we have to forward declare that function

#include <iostream>

void findsum(int x, int y);

using namespace std;

int main()

{

    findsum(3, 6);

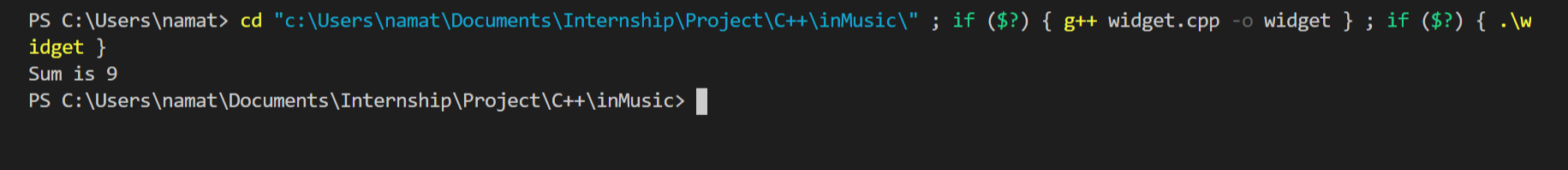
}

void findsum(int x, int y)

{

    cout << "Sum is " << x + y;

}



But while using forward declaration, the compiler would know the existence it doesn’t know the memory it might consume and any other details. So, it is kind of incomplete because of that we cannot declare a member, a base class.

But we can do like create reference or pointer member, create method which pass incomplete type etc.

Question Four

1. Need to include #include <cstudio.h>
2. Need to include #include <string.h>
3. Modify destructor to hand the “ destructors default to noexcept”

Question Five

1. Review and optimize the functions that handle lots of arrays and data, could change functions to load data to cache and manipulate it there before storing the final results to stack.
2. When we have nested array, the memory usage is more and cause performance issues. So could consider to change the array dimensions instead of having nested arrays.
3. Could consider data smoothening.
4. Use multiple threads and by dedicating main thread to continue performing while others could run in background 😊
5. Could use loop tiling by loading large data to cache and perform all the operations before paging another set of new data. But it is completely depending on the application and requirements.
6. Consider simple logical changes such
7. Consider doing any constant calculation out side the loop

Ii. Instead of for(int i=0;i<array.length()-1;i++) try to use obtain length out side the loop by Int len = array.length()-1;

1. Careful selection of storage class and scope of variables
2. Could use preincrement (++i) instead od post increment(i++) as post increment contains copying the iterator.

Question Six

Inter-Sample peaks are creating during the digital to analog conversion process. All the music which are created digitally has to be converted to analogue signals in order to listen. For this conversion, a reconstruction filter is used to round off the stepped digital audio signal to provide a smooth listening experience.

These may cause a slight change in the level of audio due to use of filters. These changes in the level could cause issue for the signals that are close to the 0dBFS and may result in clipping. Usually, a headroom in a high-end digital to analogue converter can compensate this but still cause issue for the audio played in cheap speakers.

As per the theorem states for every sampled digital signal, there is only one correct way of reconstructing a band-limited analog signal into a digital one such that the analog signal passes through each digital sample. Ditigal-to-analog converts try to approximate this correct analog waveform as closely as possible.

BS.1770, the international standards document used as the base for regional loudness specifications, gives a suggested algorithm to detect the true peak level of a digital signal. This algorithm is a relatively simple one: first, unsampled the signal to four times its original sampling rate, and then take the digital peak of the new, unsampled signal. A True-Peak-meter is a digital peak meter along with data pre - processing. Audio signals is up-sampled (usually by factor of 4) to take inter-sample peaks into consideration. Although, the DPM make use of identical scale, true-peak meter use the unit dBTP which is identical to the dBFS except on which it may be larger than zero to indicate peaks.