Programming Project: Functions
CPSC 298-6 Programming in C++
cbonang@chapman.edu

Introduction

In this programming project, you'll update your Programming Project "Frequencies of Musical Notes with Loops and Arrays" to introduce functions to compute the frequency and wavelength of a musical note. You'll also introduce a function to return the name of the note given its octave number and halftone index value. Additionally, you'll create a header file to hold the definition of your functions as well as the definitions of a few constant values.

In addition to learning about functions, you'll also learn how to check their input values; in particular, you'll learn about function preconditions.

The framework for the project will be supplied in the form of two source files, a header file MusicalNoteApi_incomplete.h and an implementation file, MusicalNoteImpl_incomplete.cpp. As the names suggests, these are not complete, working programs.

The files contain comments similar to the following showing you the sections of code that you need to add.

```
// /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/ //
// TODO: Calculate the frequency in Hz based on the value of nu and k. Use the pow function
//       and supply it with a fractional exponent. Use the reference frequency constant,
//       k_dFrequencyRef, defined in the header file.
// /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/ //
```

The Background section below gives you some background information on the C Preprocessor and the role it plays in making sure your header files (the .h files) compile correctly. You can skip it if you're short of time.

Background

This project introduces you C++ header files and to a common usage of the C Preprocessor. The C Preprocessor plays an important role in ensuring your header files are not included multiple times (thus causing compilation errors).

Header File Preprocessor Sentinel

C and C++ header files all use C Preprocessor sentinels to avoid there being included multiple times and causing syntax errors (due to duplicate declarations). A sentinel ensures that your include file is included at most once. The idiom for a sentinel is shown below. Generally, the defined macro, in this case MUSICAL_NOTE_API_H mimics the name of the header file, in this case MusicalNoteApi.h.

```
#if !defined(MUSICAL_NOTE_API_H)
#define MUSICAL_NOTE_API_H  // Sentinel


#endif
```

The header file's code falls after the #define line and before the #endif line. The preprocessor says to only render the code if the macro MUSICAL_NOTE_API_H is not defined (The ! means not in the C Preprocessor language just as it does in C and C++.)

Once it is defined, the preprocessor ignores the code (doesn't insert it), so it can only ever be inserted once.

The included header file is literally inserted into what is called an intermediate file (or translation unit) and it is this file, with the header included at most once, that is presented to the compiler after preprocessing.

An alternative to the C Preprocessor syntax shown is to use the #pragma once directive. This does the same thing but is a more recent addition and may not be supported by the C++ compiler.

You just type
#pragma once
at the top of your header file and you're done.

The Assignment

In this assignment, you'll modify two incomplete files, MusicalNoteApi_incomplete.h and MusicalNoteImpl_incomplete.cpp by adding code. The files implement the familiar Musical Notes project, but this time using functions.

The sections below take you through the changes you need to make step by step. Alternatively, you can just scan the two files, search for the TODO comments and make the described changes.

Copy or Rename Files

Rename MusicalNoteApi_incomplete.h to MusicalNoteApi.h.
Rename MusicalNoteImpl_incomplete.cpp to MusicalNoteImpl.cpp.

Place both files in the same directory. (If you are using an alternate development environment, such as Microsoft Visual Studio on Windows, see the appendices.)

MusicalNoteApi.h Header File

First, observe that the contents of the header file is encapsulated in a preprocessor sentinel to prevent the header from being included multiple times. This isn't a concern on this project, but on larger projects headers can be included many times if not protected by a preprocessor sentinel, resulting in syntax errors.

```
#if !defined(MUSICAL_NOTE_API_H)
#define MUSICAL_NOTE_API_H  // Sentinel

<your header file code>

#endif
```

All the C++ header files you implement from now on (in this course and beyond) should have a preprocessor sentinel.

Header File Preamble

Remember to insert your name or email address in the file preamble in the header file:

```
/// @brief Musical Notes Application Programming Interface (API) Header File
/// CPSC-298-6 Programming in C++ Project Musical Note Frequencies with Functions
/// @author outstanding.student@chapman.edu
```

Header File Constants

The header file contains a number of constant values that you can use in your implementation file.

```
// ////////////////////////////////////////////////////////////////////////////
// CONSTANTS
// ////////////////////////////////////////////////////////////////////////////

/// Speed of sound in meters per second in air at standard temperature and pressure
const double k_dSpeedOfSoundMetersPerSecond = 345.0;

/// Speed of sound in centimeters per second in air at standard temperature and pressure
const double k_dSpeedOfSoundCmPerSecond = k_dSpeedOfSoundMetersPerSecond * 100;

/// Number of halftones in an octave.
const int k_nHalfTonesInOctave = 12;

/// Frequency of note C0; used as the reference frequency in computing the frequencies of other notes.
const double k_dFrequencyRef = 16.35;
```

Function Preamble Comments

The header file provides function preamble comments that explain how the function works, what input parameters should be supplied and what the function returns. It also specifies restrictions on the function's input called preconditions, identified by the @pre notation.

```
/// Computes wavelength in centimeters of a musical note at room temperature and pressure given the note's frequency in Hertz.
///
/// @param dFrequency positive double precision value that is the frequency in Hertz (Hz) (non-negative, non-zero)
///
/// @pre dFrequencyHz > 0.0 (i.e. non-zero and non-negative)
/// @pre dFrequencyHz not a special floating point value such as positive or negative infinity, or a quiet or signaling NaN.
///
/// @return double precision value holding the computed wavelength in centimeters or -1.0 if an error occurred.
double computeWavelengthInCm(double dFrequencyHz);
```

The function preambles are written for you in this assignment, but you'll need to read them to complete several other steps. Also, you'll write your own function preambles from now on in the course.

Let’s Begin Coding

Function Prototype

You'll need to write the function prototype for the function "computeFrequency." (Incidentally, observe that all of the function names have the form verbObject. The verb indicates what the function does and the next word indicates what it acts on, the object (in the grammar sense). This naming convention is called camel case.)

```
// /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/ //
// TODO: code the prototype for function computeFrequency based on the method preamble comments above
// /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/ //
```

Notice the other prototypes in the file. For example, here's the preamble and prototype for getNoteName:

```
/// Returns the shorthand name of the musical note whose octave and half-tone are specified.
/// @param nu integer > 0 representing the octave number of the note.
/// @param k  integer from 0 to 11, inclusive, representing the half-step (or semitone) number within the octave.
///
/// @pre nu > 0
/// @pre k >= 0 and k < Number of Halftones (12)
///
/// @return a string object holding the shorthand name of the note corresponding to the specified
///         octave and half-tone, or "UNK" if nu and k are invalid.
const std::string getNoteName(int nu, int k);
```

Write a prototype for computeFrequency based on the preamble for compute frequency. The prototype will resemble that for getNoteName, but the return value will not be a string. (The preamble tells you what the function return type and arguments should be.)

```
/// Computes the frequency of a musical note given its octave and half-step (or semitone) number.
///
/// @param nu integer > 0 representing the octave number of the note.
/// @param k  integer from 0 to 11, inclusive, representing the half-step (or semitone) number within the octave.
///
/// @pre nu > 0
/// @pre k >= 0 and k < Number of Halftones (12)
///
/// @return double precision floating point value that is the frequency in Hertz (Hz), or
///         cycles per second, or -1.0 if an error occurred.

// /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/ //
// TODO: code the prototype for function computeFrequency based on the method preamble comments above
// /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/ //
```

Write the code for the **computeFrequency function prototype** right after **the TODO comment.**

That's it; you're done with the changes to the header file!

MusicalNoteImpl.cpp Implementation File

You'll now make modifications to the implementation file.

Implementation File Preamble

As you did for the header file, put your name or email in the @author entry in the file preamble comments for the Implementation file.

```
/// @brief Musical Notes Application Programming Interface (API) Implementation File
/// CPSC-298-6 Programming in C++ Project Musical Note Frequencies with Functions
/// @author outstanding.student@chapman.edu
```

Include MusicalNoteApi.h Header File

Next, include the MusicalNoteApi.h header file, remember to use double quotes ("MusicalNoteApi.h") rather than angle brackes <MusicalNoteApi.h>.

```
#include <iostream>
#include <string>
#include <cmath>
// /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/ //
// TODO: Include your MusicalNoteApi.h header file but use double quotes "header.h" rather
//       than angle brackets <header.h>
// /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/ //
```

Place your #include directive right after the TODO comments.

Complete Function computeFrequency

You'll make two additions to functions computeFrequency.

First, you'll write a conditional statement that implements the input checks on the function's arguments. These checks are called "preconditions." The preconditions ensure that the input values, nu and k, are correct. For instance, the user might have entered -1 for nu when nu must be >= 0.

The precondition check will have the form
if ((? >=0) && (? < ?) && (?? > = 0)) where the question marks correspond to nu, k and the constant `k_nHalfTonesInOctave`.

```
double computeFrequency(int nu, int k)
{
    double f = -1.0; // computed frequency in Hertz (Hz)

    // Preconditions:
    // k > 0
    // k < Number of Halftones (12)
    // n > 0

    // /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/ //
    // TODO: Write a conditional expression that checks the preconditions of the function as
    //       specified in the function's preamble comments in the header file (and also listed
    //       above). The expression consists of three relational expressions (involving the >=
    //       and < operators) that ANDed togeher (using the && operator).
    //       Observant students will note these preconditions are identical to those of
    //       another function in the Music Notes API.
    //      if ((? >= 0) && (? < ?) && (?? >= 0))
    // /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/ //
    if ( /* ((? >= 0) && (? < ?) && (?? >= 0)) */ )
    {
        // /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/ //
        // TODO: Calculate the frequency in Hz based on the value of nu and k. Use the pow function
        //       and supply it with a fractional exponent. Use the reference frequency constant,
        //       k_dFrequencyRef, defined in the header file.
        // /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/ //
        f = /* ??? */
    }
    else
    {
```

The second change, also shown above, is to calculate the frequency in Hertz (Hz) based on the value of nu and k. You've already done this calculation in earlier assignments and just need to insert it here at the point f = /* ???? */

Note that the function returns f, the frequency. (What does it return in the event of an error?)

Complete Function computeWavelengthInCm

Next, you'll complete function computeWavelengthInCm by implementing the computation for the wavelength in centimeters. The precondition checks have already been implemented, you just need to insert your calculation of the wavelength from your previous assignments.

```
double computeWavelengthInCm(double dFrequencyHz)
{
    double dWavelengthCm = -1.0; // Computed wavelength in centimeters returned by computeWavelengthInCm

    // Preconditions:
    // dFrequencyHz > 0.0
    // dFrequencyHz must be finite (i.e. not positive or negative infinity, or a signaling or quiet Not-a-Number).
    //       Infinite frequency makes no sense (not even Jimi Hendrix can do that) and is treated as
    //       an error.

    // The expressions below check the negation of the preconditions and report an error.
    // Report an error if dFrequencyHz < 0.0
    // Report an error if dFrequencyHz is not finite (a NaN or positive or negative infinity)
    // Report an error if dFrequencyHz is exactly 0 (including either positive or negative zero).
    //      A frequency of zero produces no wave at all and thus has no wavelength at all; zero
    //      frequency makes no sense for this
    //      function and is treated as an error.)
    if ((dFrequencyHz < 0.0) || (!isfinite(dFrequencyHz)) || (FP_ZERO == std::fpclassify(dFrequencyHz)))
    {
        dWavelengthCm = -1.0; // Indicate Error
        std::cerr << "ERROR: computeWavelengthInCm preconditions not satisfied; dFrequencyHz invalid: " << dFrequencyHz << std::endl;
    }
    else {
        // Input satisfies all preconditions; perform computation

        // /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/ //
        // TODO: Calculate the wavelength in centimeters and assign the value to the double precision
        //       variable dWavelengthCm. Use one of the constants defined in the header file.
        // /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/ //
        dWavelengthCm = /* ????? */
```

Complete Function getNoteName

Next, you'll complete function getNoteName.

getNoteName returns the name of a note as a string given the values of nu and k for the note. The function allows you to build the string for a note name in an arbitrary octave, not just 0 though 8.

For instance, the string "C#0" can be formed as follows by combining "C#" with a "stringized" form of the integer 0. "Stringize" means to convert the integer 0 to its representation as a string,"0". "C#0" is jut the concatenation of "C#" and "0".

```
const std::string getNoteName(int nu, int k)
{

    const static std::string a_strNotePrefix[] = {
            "C", "C#", "D","D#", "E", "F", "F#", "G", "G#", "A", "A#", "B",
    };

    std::string strNote = "UNK";

    // Preconditions:
    // k > 0
    // k < Number of Halftones (12)
    // n > 0
    if ((k >= 0) && (k < k_nHalfTonesInOctave) && (nu >= 0))
    {
        // /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/ //
        // TODO: Express strNote in terms of an element of a_strNotePrefix and the stringized form of
        //       the integer nu.
        //
        //       Use std::to_string to covert integer nu to a string.
        //
        //       Concatenate the prefix with the string representation
        //       of nu to form the note name. For example, "C#0" = "C#" + "0"
        //
        //       strNote = a_strNotePrefix[?] + ?
        // /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/ //
```

You're almost done, just one more function to go.

Complete Function Main

Finally, you'll complete the main function by calling your three new functions.

First, you'll call them in the nested for loop used to compute the frequency and wavelengths of the notes from octave 0 through octave 10.

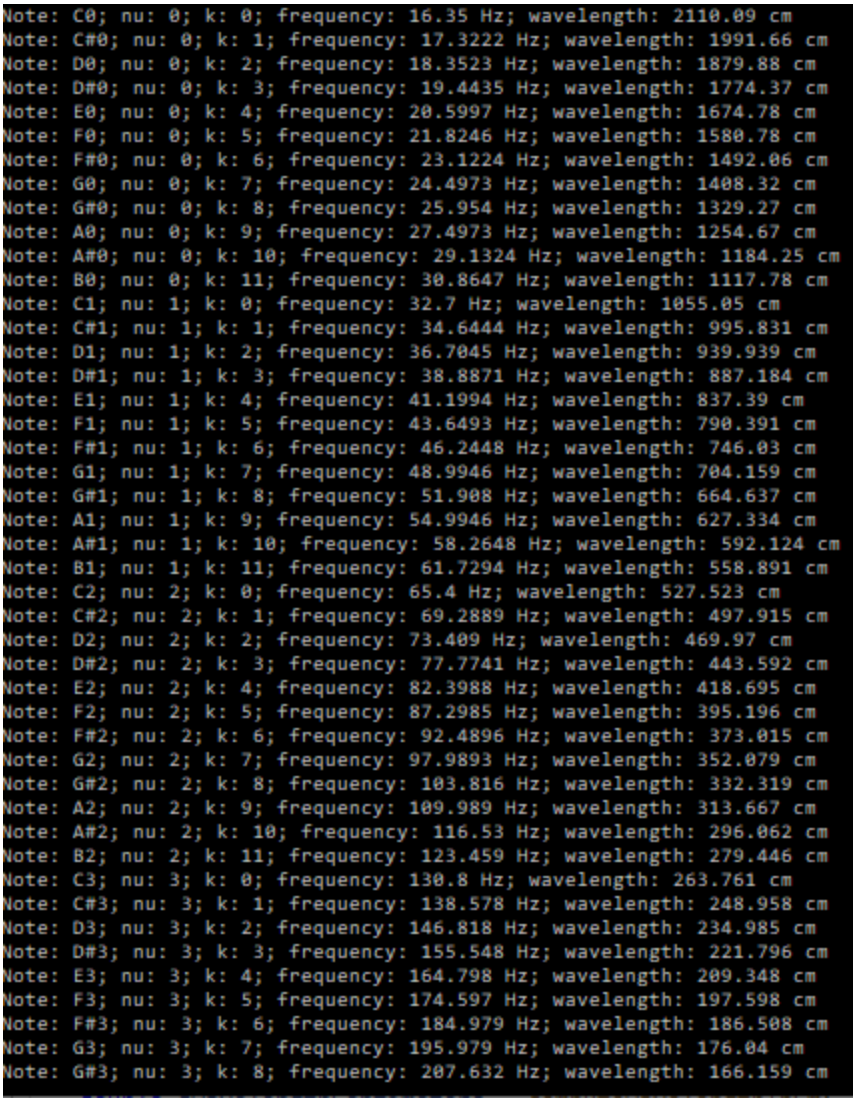
```
int main()
{
    int nOctaves = 11; // Number of octaves over which to compute frequencies.

    // Compute for Octaves 0 through 10 (previously you computed only zero through 8)
    for (int nu = 0; nu < nOctaves; nu++)
    {
        for (int k = 0; k < k_nHalfTonesInOctave; k++)
        {
            // /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/ //
            // TODO: Call your function to compute the frequency in Hertz of the note corresponding to
            //       octave nu and half-tone k and assign it to a variable named f of the appropriate type.
            // /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/ //

            // /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/ //
            // TODO: Call your function to compute the wavelength of the note given its frequency in Hertz
            //       and assign it to a variable named dWavelengthCentimeters of the appropriate type.
            // /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/ //

            // /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/ //
            // TODO: Call function getNoteName to obtain the name of the note as a string variable and
            //       assign it to a variable named strNote of the appropriate type.
            // /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/ //
```

When you execute the program, this portion will produce output similar to the following:



The last note whose frequency and wavelength are computed is "B10":

```
Note: D#7; nu: 7; k: 3; frequency: 2488.77 Hz; wavelength: 13.8623 cm
Note: E7; nu: 7; k: 4; frequency: 2636.76 Hz; wavelength: 13.0842 cm
Note: F7; nu: 7; k: 5; frequency: 2793.55 Hz; wavelength: 12.3499 cm
Note: F#7; nu: 7; k: 6; frequency: 2959.67 Hz; wavelength: 11.6567 cm
Note: G7; nu: 7; k: 7; frequency: 3135.66 Hz; wavelength: 11.0025 cm
Note: G#7; nu: 7; k: 8; frequency: 3322.11 Hz; wavelength: 10.385 cm
Note: A7; nu: 7; k: 9; frequency: 3519.66 Hz; wavelength: 9.80209 cm
Note: A#7; nu: 7; k: 10; frequency: 3728.95 Hz; wavelength: 9.25194 cm
Note: B7; nu: 7; k: 11; frequency: 3950.68 Hz; wavelength: 8.73267 cm
Note: C8; nu: 8; k: 0; frequency: 4185.6 Hz; wavelength: 8.24255 cm
Note: C#8; nu: 8; k: 1; frequency: 4434.49 Hz; wavelength: 7.77993 cm
Note: D8; nu: 8; k: 2; frequency: 4698.18 Hz; wavelength: 7.34327 cm
Note: D#8; nu: 8; k: 3; frequency: 4977.55 Hz; wavelength: 6.93113 cm
Note: E8; nu: 8; k: 4; frequency: 5273.53 Hz; wavelength: 6.54211 cm
Note: F8; nu: 8; k: 5; frequency: 5587.11 Hz; wavelength: 6.17493 cm
Note: F#8; nu: 8; k: 6; frequency: 5919.33 Hz; wavelength: 5.82836 cm
Note: G8; nu: 8; k: 7; frequency: 6271.31 Hz; wavelength: 5.50124 cm
Note: G#8; nu: 8; k: 8; frequency: 6644.23 Hz; wavelength: 5.19248 cm
Note: A8; nu: 8; k: 9; frequency: 7039.31 Hz; wavelength: 4.90105 cm
Note: A#8; nu: 8; k: 10; frequency: 7457.89 Hz; wavelength: 4.62597 cm
Note: B8; nu: 8; k: 11; frequency: 7901.36 Hz; wavelength: 4.36634 cm
Note: C9; nu: 9; k: 0; frequency: 8371.2 Hz; wavelength: 4.12127 cm
Note: C#9; nu: 9; k: 1; frequency: 8868.98 Hz; wavelength: 3.88996 cm
Note: D9; nu: 9; k: 2; frequency: 9396.35 Hz; wavelength: 3.67164 cm
Note: D#9; nu: 9; k: 3; frequency: 9955.09 Hz; wavelength: 3.46556 cm
Note: E9; nu: 9; k: 4; frequency: 10547.1 Hz; wavelength: 3.27106 cm
Note: F9; nu: 9; k: 5; frequency: 11174.2 Hz; wavelength: 3.08747 cm
Note: F#9; nu: 9; k: 6; frequency: 11838.7 Hz; wavelength: 2.91418 cm
Note: G9; nu: 9; k: 7; frequency: 12542.6 Hz; wavelength: 2.75062 cm
Note: G#9; nu: 9; k: 8; frequency: 13288.5 Hz; wavelength: 2.59624 cm
Note: A9; nu: 9; k: 9; frequency: 14078.6 Hz; wavelength: 2.45052 cm
Note: A#9; nu: 9; k: 10; frequency: 14915.8 Hz; wavelength: 2.31299 cm
Note: B9; nu: 9; k: 11; frequency: 15802.7 Hz; wavelength: 2.18317 cm
Note: C10; nu: 10; k: 0; frequency: 16742.4 Hz; wavelength: 2.06064 cm
Note: C#10; nu: 10; k: 1; frequency: 17738 Hz; wavelength: 1.94498 cm
Note: D10; nu: 10; k: 2; frequency: 18792.7 Hz; wavelength: 1.83582 cm
Note: D#10; nu: 10; k: 3; frequency: 19910.2 Hz; wavelength: 1.73278 cm
Note: E10; nu: 10; k: 4; frequency: 21094.1 Hz; wavelength: 1.63553 cm
Note: F10; nu: 10; k: 5; frequency: 22348.4 Hz; wavelength: 1.54373 cm
Note: F#10; nu: 10; k: 6; frequency: 23677.3 Hz; wavelength: 1.45709 cm
Note: G10; nu: 10; k: 7; frequency: 25085.3 Hz; wavelength: 1.37531 cm
Note: G#10; nu: 10; k: 8; frequency: 26576.9 Hz; wavelength: 1.29812 cm
Note: A10; nu: 10; k: 9; frequency: 28157.2 Hz; wavelength: 1.22526 cm
Note: A#10; nu: 10; k: 10; frequency: 29831.6 Hz; wavelength: 1.15649 cm
Note: B10; nu: 10; k: 11; frequency: 31605.4 Hz; wavelength: 1.09158 cm
```

Lastly, you'll call the functions again outside of the nested for loop. You'll only call each once and will pass erroneous input in all cases.

```
std::cout << std::endl << std::endl;
std::cout << "---------------------------------------------- CHECK ERROR HANDLING ------------------------------------------------------------" << std::endl;

// /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/ //
// TODO: Call your function to compute the frequency in Hertz of the note corresponding to
//       octave nu and half-tone k and assign it to a variable named f of the appropriate type.
//       Pass -1 as the value for argument nu and -1 for the value for arugment k.
// /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/ //

// /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/ //
// TODO: Call your function to compute the wavelength of the note given its frequency in Hertz
//       and assign it to a variable named dWavelengthCentimeters of the appropriate type.
//       Pass -16.35 as the value for the argument dFrequencyHz.
// /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/ //

// /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/ //
// TODO: Call function getNoteName to obtain the name of the note as a string variable and
//       assign it to a variable named strNote of the appropriate type.
//       Pass -1 as the value for argument nu and -1 for the value for arugment k.
// /\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/ //
```

Assign the return values of the functions to appropriately named variables so that the values can be printed out:

```
std::cout << "Note: " << strNote << "; nu: " << nu << "; k: " << k << "; frequency: " << f << " Hz; wavelength: " << dWavelengthCentimeters << " cm" << std::endl;
```

When you execute the program, this portion will produce output similar to the following:

```
---------------------------------------------- CHECK ERROR HANDLING ----------------------------------------------------------------
ERROR: computeFrequency preconditions not satisfied; dFrequencyHz nu (-1) or k (-1) invalid
ERROR: computeWavelengthInCm preconditions not satisfied; dFrequencyHz invalid: -16.35
Note: UNK; nu: -1; k: -1; frequency: -1 Hz; wavelength: -1 cm
```

Notice that the errors are reported and that values are returned (such as "UNK" and -1) indicating errors occurred in the function invocations.

You've now completed all of the changes to both the header file and the implementation file. It's time to compile, build and run.

Building the Program

When building using the Chapman FSE Linux image running in a Docker container, place both the MusicalNoteApi.h and MusicalNoteImpl.cpp files in the same directory. This allows the compiler to find the header file (it's in the current working directory).

```
$ g++ -o music MusicalNoteImpl.cpp
```

If they weren't in the same directory, you can use the -I option to tell the compiler where to find them. The example below shows the -I option with an argument of ./, which simply tells the compiler the header files are in the current directory. However, the argument could just as easily be any other directory.

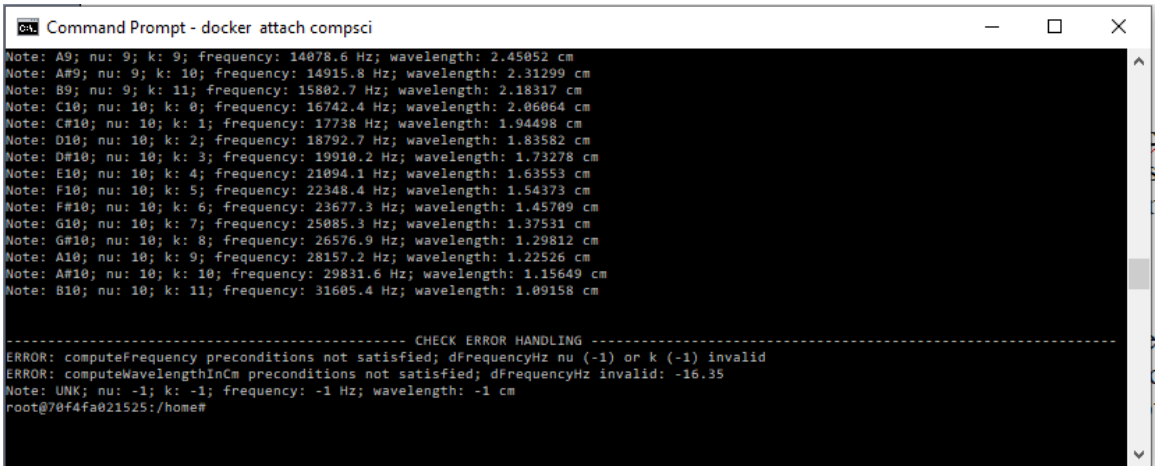
```
$ g++ -I ./ -o music MusicalNoteImpl.cpp
```

If you had multiple header files scattered in different directories throughout a large project, you could specify multiple -I options so that the compiler could find them all.

Running the Program

To run the program, type ./music at the command prompt.

```
root@70f4fa021525:/home# ./music
```

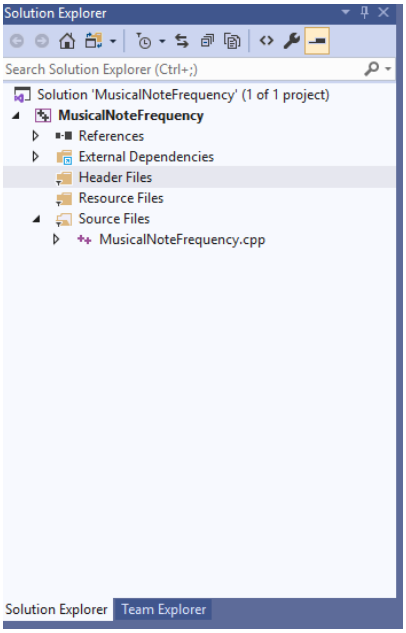


That's it; you're done!

Appendix A: Adding Header Files in the Microsoft Visual Studio Integrated Development Environment

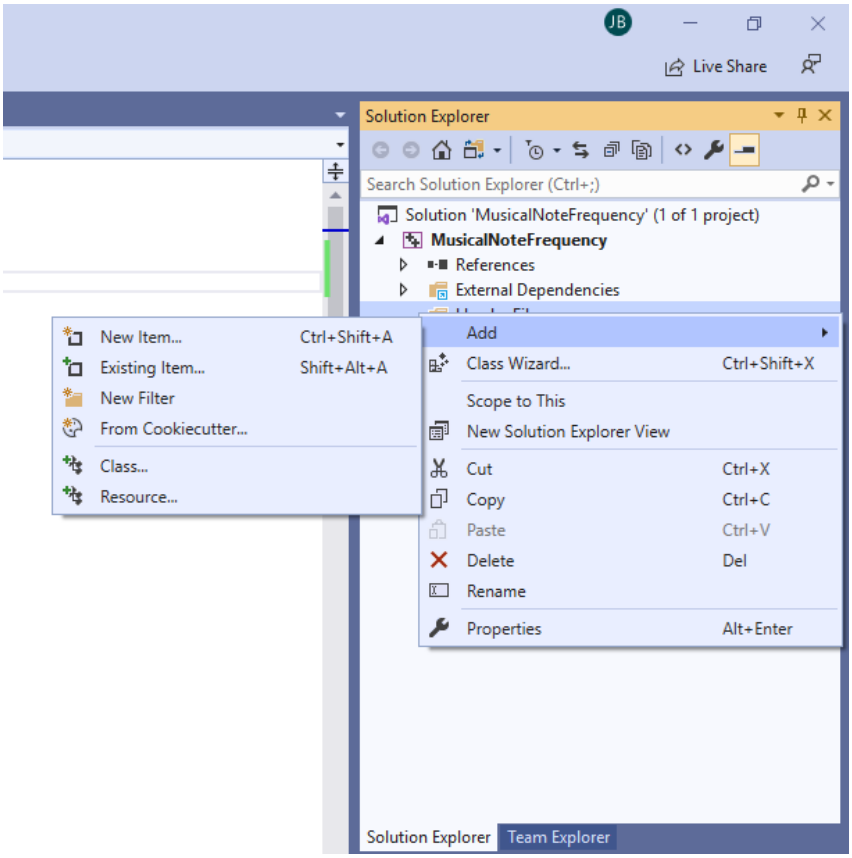
You may be using the Microsoft Visual Studio Integrated Development Environment (IDE) in lieu of the standard Chapman FSE Linux Image running in a Docker container. If so, you'll need to add the MusicalNotesApi.h header file to your project.

In the Solution Explorer window on the right hand side (typically) of the Visual Studio IDE find the Header Files folder icon.

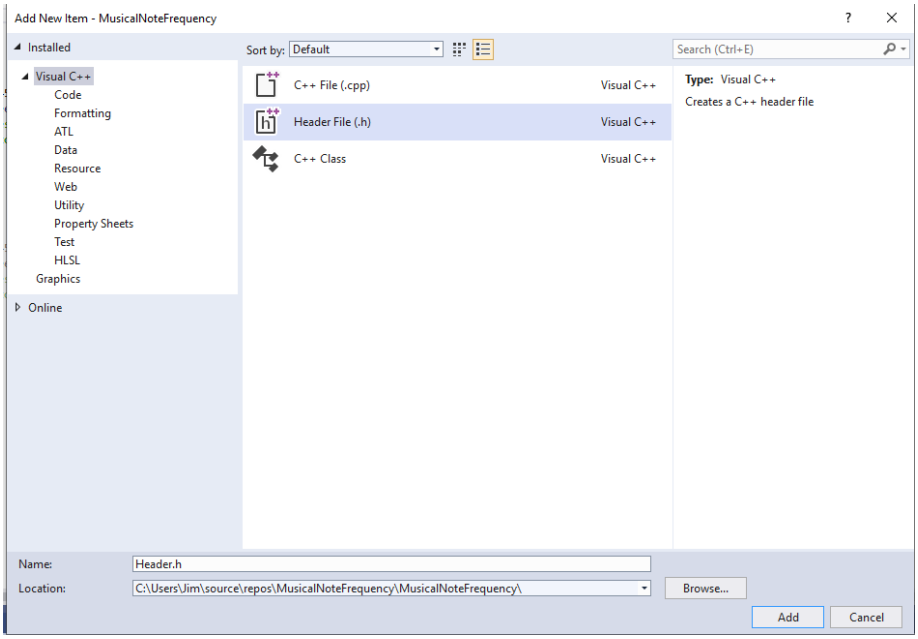


Right Click on the Header files Folder.

In the pop-up menu that appears, select Add, then select New Item from the Add sub-menu as shown below.

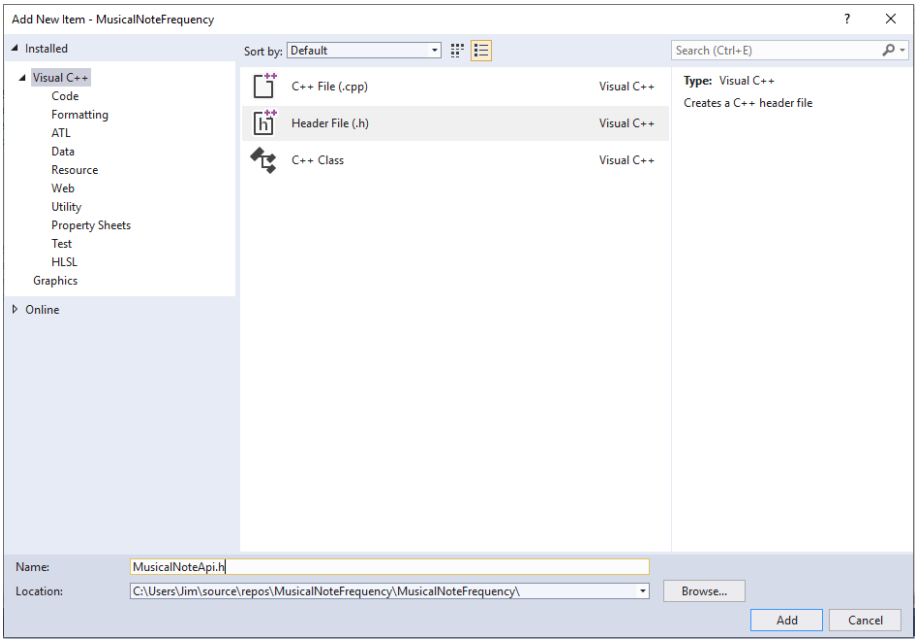


The Add New Item dialog window appears.



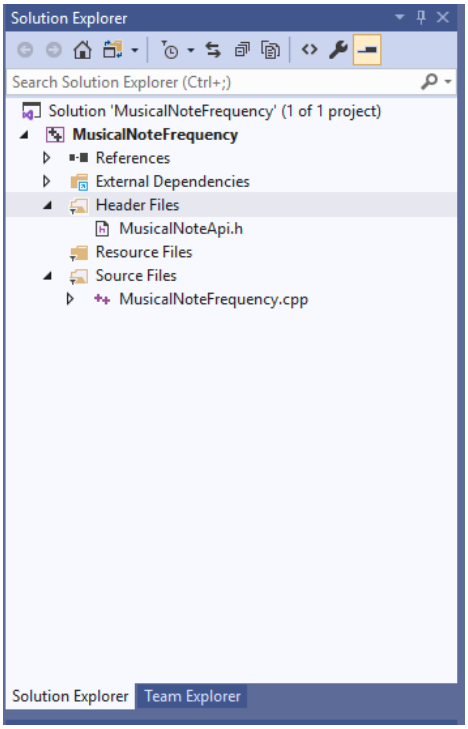
Click on Header File (.h) as shown above.

Next, enter a name for your header file in the Name textbox at the bottom of the dialog window as shown in the figure beow.



Now, press the Add button.

The new header file, MusicalNoteApi.h appears under the Header files folder.



Double click to open the MusicalNoteApi.h header file in the editor.