Header Files

May 5, 2020

0.1 Function Order in a Single File

In the following code example, the functions are out of order, and the code will not compile. Try to fix this by rearranging the functions to be in the correct order.

In [ ]:
```
#include <iostream>
using std::cout;

void InnerFunction(int i)
{
    cout << "The value of the integer is: " << i << "\n";
}

void OuterFunction(int i)
{
    InnerFunction(i);
}

int main()
{
    int a = 5;
    OuterFunction(a);
}
```

Compile & Execute

Show Solution
Loading terminal (id_4wc3hya), please wait...

In the mini-project for the first half of the course, the instructions were very careful to indicate where each function should be placed, so you didn't run into the problem of functions being out of order.

0.2 Using a Header

One other way to solve the code problem above (without rearranging the functions) would have been to declare each function at the top of the file. A function declaration is much like the first line of a function definition - it contains the return type, function name, and input variable types. The details of the function definition are not needed for the declaration though.

To avoid a single file from becomming cluttered with declarations and definitions for every function, it is customary to declare the functions in another file, called the header file. In C++, the header file will have filetype `.h`, and the contents of the header file must be included at the top of the `.cpp` file. See the following example for a refactoring of the code above into a header and a cpp file.

In [ ]:
```
// The header file with just the function declarations.
// When you click the "Run Code" button, this file will
// be saved as header_example.h.
#ifndef HEADER_EXAMPLE_H
#define HEADER_EXAMPLE_H

void OuterFunction(int);
void InnerFunction(int);

#endif
```

In [ ]:
```
// The contents of header_example.h are included in
// the corresponding .cpp file using quotes:
#include "header_example.h"

#include <iostream>
using std::cout;

void OuterFunction(int i)
{
    InnerFunction(i);
}

void InnerFunction(int i)
{
    cout << "The value of the integer is: " << i << "\n";
}

int main()
{
    int a = 5;
    OuterFunction(a);
}
```

Compile & Execute

Explain
Loading terminal (id_017mcyj), please wait...

Notice that the code from the first example was fixed without having to rearrange the functions! In the code above, you might also have noticed several other things: - The function declarations in the header file don't need variable names, just variable types. You can put names in the declaration, however, and doing this often makes the code easier to read. - The `#include` statement

for the header used quotes " " around the file name, and not angle brackets <>. We have stored the header in the same directory as the `.cpp` file, and the quotes tell the preprocessor to look for the file in the same directory as the current file - not in the usual set of directories where libraries are typically stored. - Finally, there is a preprocessor directive: `cpp #ifndef HEADER_EXAMPLE_H #define HEADER_EXAMPLE_H` at the top of the header, along with an `#endif` at the end. This is called an "include guard". Since the header will be included into another file, and `#include` just pastes contents into a file, the include guard prevents the same file from being pasted multiple times into another file. This might happen if multiple files include the same header, and then are all included into the same `main.cpp`, for example. The `ifndef` checks if `HEADER_EXAMPLE_H` has not been defined in the file already. If it has not been defined yet, then it is defined with `#define HEADER_EXAMPLE_H`, and the rest of the header is used. If `HEADER_EXAMPLE_H` has already been defined, then the preprocessor does not enter the `ifndef` block. **Note:** There are other ways to do this. Another common way is to use an `#pragma once`preprocessor directive, but we won't cover that in detail here. See this Wikipedia article for examples.

0.3 Practice

In the following two cells, there is a blank header file and a short program that won't compile due to the functions being out of order. The code should take a vector of `int`s, add 1 to each of the vector entries, and then print the sum over the vector entries.

Without rearranging the functions in the main .cpp file, add some function declarations to the header file to fix this problem. Don't forget to include the "header_practice.h" file in your `.cpp` file!

In [ ]:
```
// This file will be saved as "header_practice.h"
#ifndef HEADER_PRACTICE_H
#define HEADER_PRACTICE_H

#include <vector>
using std::vector;

int IncrementAndComputeVectorSum(vector<int> v);
void AddOneToEach(vector<int> &v);

#endif
```

In [ ]:
```
#include <iostream>
#include <vector>
#include "header_practice.h"
using std::vector;
using std::cout;


int IncrementAndComputeVectorSum(vector<int> v)
{
    int total = 0;
    AddOneToEach(v);
```

```
    for (auto i: v) {
        total += i;
    }
    return total;
}

void AddOneToEach(vector<int> &v)
{
    // Note that the function passes a reference to v
    // and the for loop below uses references to
    // each item in v. This means the actual
    // ints that v holds will be incremented.
    for (auto& i: v) {
        i++;
    }
}

int main()
{
    vector<int> v{1, 2, 3, 4};
    int total = IncrementAndComputeVectorSum(v);
    cout << "The total is: " << total << "\n";
}
```

Compile & Execute

Show Solution
Loading terminal (id_qk25dsd), please wait...