

SEARCH

RESOURCES

CONCEPTS

1. Intro

2. Header Files

3. Using Headers with Multiple Files

4. Bjarne on Build Systems

5. CMake and Make

6. References

7. Pointers

8. Pointers Continued

9. Bjarne on pointers

10. References vs Pointers

11. Bjarne on References

12. Maps

13. Classes and Object-Oriented Pro...

14. Classes and OOP Continued

15. This Pointer

16. How Long Does it Take to Learn ...

17. Outro

Single File Code

In the previous concept, you saw some example code that wouldn't compile because the functions were out of order:

```
#include <iostream>
#include <vector>
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) {
    for (auto& i: v) {
        i++;
    }
}

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

In the last exercise of the notebook, you were to separate that code into a header .h file and a .cpp file. But what if you wanted to use completely separate files for each of the functions? For example, you might want to do this if the functions were going to belong to different classes or libraries.

Multi-file Code

In the next few cells these functions have been separated into several different files. The structure of the included files is as follows:



vect_add_one.h and vect_add_one.cpp

```
In [ ]: #ifndef VECT_ADD_ONE_H
#define VECT_ADD_ONE_H

#include <vector>
using std::vector;

// AddOneToEach method declaration.
void AddOneToEach(vector<int> &v);

#endif
```

```
In [ ]: #include "vect_add_one.h"

void AddOneToEach(vector<int> &v)
{
    for (auto& i: v) {
        i++;
    }
}
```

increment_and_sum.h and increment_and_sum.cpp

```
In [ ]: #ifndef INCREMENT_AND_SUM_H
#define INCREMENT_AND_SUM_H

#include <vector>
using std::vector;

// IncrementAndComputeVectorSum method declaration.
int IncrementAndComputeVectorSum(vector<int> v);

#endif
```

```
In [ ]: #include "vect_add_one.h"

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

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

main.cpp

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

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

Compile & Execute See Explanation

Loading terminal (id_yvcrmxo), please wait...

If you look carefully at the files above, you will see several things:

- vect_add_one.h is included in increment_and_sum.cpp .

This is because AddOneToEach is used in IncrementAndComputeVectorSum . Including the vect_add_one.h header means that the AddOneToEach function declaration is pasted into increment_and_sum.cpp , so no compiler error will occur when the AddOneToEach function is used.
- Only the header file needs to be included in another file.

As long as the header file is included, the corresponding function declarations will be included. When the compiler finds an undefined function, it has already seen the function's declaration. This means the compiler can continue on without error until it finds the definition of the function, regardless of where that definition is.
- Some libraries, like <vector> are included in multiple files.

Each file is compiled alone and must have all the declarations and libraries necessary to compile, so the necessary libraries must be included. This is another reason why [include guards \(https://github.com/isocpp/CppCoreGuidelines/blob/master/CppCoreGuidelines.md#Rs-guards\)](https://github.com/isocpp/CppCoreGuidelines/blob/master/CppCoreGuidelines.md#Rs-guards) are important - if multiple headers were included in main , each with the same #include <vector> statement, you wouldn't want the vector header pasted multiple times into the code.
- The g++ compile command from the "Run Code" button is:

g++ -std=c++17 ./code/main.cpp ./code/increment_and_sum.cpp ./code/vect_add_one.cpp && ./a.out

When compiling, each of the relevant .cpp files must be included in the compile command. The -std=c++17 specifies that we are using the C++ 17 standard (which happens automatically in the terminal).

Your Turn!

In the course project code, all the files will be set up for you, so you will not need to manually create the header files and write #include statments yourself. Nonetheless, it is still helpful to see how this works.

Now that you have seen the code split into multiple files, try compiling and running the code yourself from the command line above, instead of relying on the "Compile & Execute" button.

Remember that you will need to include all the files in your compile command:

```
g++ -std=c++17 ./code/main.cpp ./code/increment_and_sum.cpp ./code/vect_add_one.cpp
```

followed by

```
./a.out
```