How to Compile Projects With Multiple Files

When working on programs and projects of some size it might be a good idea to separate different parts of your code in to several distinct source code files. Why you want to do this is not covered here in any length, but some advantages are:

- The same module can be used in several programs, but only needs to be compiled once.
- Shorter compile times when only one or a few of the files change betwen each compilation.
- You can keep pieces of code that does different things separate, increased readability.

Example Project

As a small example we will use the following files as our project:

- `main.c`: This is the program that we want to run in the end.
- `helper.c`: This is a source code file containing some functions we want to use.
- `helper.h`: This is a header file that show the compiler how the functions in `helper.c` is used.

Earlier in some of the weekly solutions you might have seen "header only" libraries used. This is when all the source code of the "helper" module is in the header file `helper.h`. Now we only have the function declarations in the header. In this example our header file look like this:

```
#ifndef HELPER_H
#define HELPER_H

int print_vector(int* v, int n);

#endif
```

The declaration contains only the function name, return type and parameter types. When this is included in our `main.c` file the compiler will know how to compile the program, and make it ready for linking with the compiled `helper.c`.

The code of `main.c`:

```
#include "helper.h"

int main(int argc, char const *argv[]) {
    int n = 10;
    int v[n];

    for (size_t i = 0; i < n; i++) {
```

```
        v[i] = i;
    }

    print_vector(v, n);

    return 0;
}
```

and `helper.c`:

```
#include <stdio.h>

int print_vector(int* v, int n){
    printf("[");
    for (size_t i = 0; i < n-1; i++) {
        printf("%d, ", v[i]);
    }
    printf("%d]\n", v[n-1]);

    return 0;
}
```

Compiling vs Linking

There are two steps to creating one program from several source code files. Compiling and linking. Compiling is the part where the source code is translated to machine code, and the linking is when several of these machine codes are "linked" together to form an entire program.

Take our `main.c` as an example. When the compiler compiles the source code into an object file the compiler simply inserts a call to the function `print_vector` at the correct spot. It assumes that the machine code that will do the actual work in that function will be linked later.

Using the Command Line

First we compile our two `.c` files in to object files:

```
gcc -c main.c helper.c
```

The output from this is `main.o` and `helper.o`. These two files are now ready to be linked together to form the final program.

```
gcc main.o helper.o -o main.exe
```

After this operation we have an executable `main.exe`. Running this executable we get the expected result.

```
$ ./main.exe
> [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

The compilation and the linking can be done in one step.

```
gcc main.c helper.c -o main.exe
```

Running this command will not generate object files, as they are discarded when the linking occurs. This is simpler, but both files are compiled every time even if no changes where made to one, or both.

Using a Makefile

When using a Makefile we can get the best of both worlds. We can compile and link all the source code files in to one program using one simple command, and we will not recompile parts that haven't changed.

To make this strategy work we need to define a couple of steps/recipes. We need to tell make how to make each object file. **Note:** do not copy paste from here, use the file bundled with this document. In Makefiles the difference between space and tab is very important, and it will not be correct if you copy/paste.

```
helper.o : helper.c
    gcc -c helper.c

main.o : main.c helper.h
    gcc -c main.c
```

These lines tell make how to, make, the object files which will be the dependencies of the executable later.

The rule for the executable is

```
main.exe : main.o helper.o helper.h
    gcc main.o helper.o -o main.exe
```

Now that we have told make how to create the program `main.exe` we only need one short line in the shell to compile and link.

```
make main.exe
```

This will create both object files and link them together to create `main.exe`. Repeated invocations of this line will only compile those source code files that changed since the last compilation and linking.

Extras

In the example Makefile there are a few more targets.

- `tar`: Used to make the bundle of files this document came with.

- `compiling_projects.pdf`: Target that creates the pdf version of this document.
- `clean`: Removes object and executable files.

Feel free to adapt the Makefile to suit your needs.