C++

Muchang Bahng

Winter 2024

Contents

1 Translation

Translating C++ code to a binary consists of multiple steps:

1. Preprocessing the code.
2. Compiling each file independently.
3. Linking all the files.

Conventionally, all of these are called *compiling*, but it really isn't.

1.1 Preprocessing

When preprocessing, we do some boring stuff like removing comments. However, the main job is to take care of **preprocessing directives**, which are expressions with the `#` symbol. The most obvious is the `#include` directives, which **replaces the include directive with the contents of the included file**. That is, `#include` is really just a way to substitute code.

1. including with angle brackets, e.g. `#include <iostream>`, means that the compiler is looking for this file in the standard library files.
2. including with double quotes, e.g. `#include "tensor.h"`, means that the compiler is looking for this file locally in your project directory. It means you've written it.

Other directives is the `#define` directive.

1. You can define it to substitute text. It is conventionally in all upper-case.

```
#define NAME "Muchang"  // all instances of NAME will be replaced with "Muchang"
```

2. Or you can define it without substitution text, where further occurrences of `NAME` will be replaced by nothing.

```
#define NAME
```

The second isn't used for substitution, but rather for **conditional compilation**, which can be useful. You just wrap C++ statements around as such.

```
#ifdef NAME
...
#endif
```

```
#ifndef NAME
...
#endif
```

To see the output after preprocessing, use the `-E` flag.

```
g++ main.cpp -E
```

1.2 Compilation

We only compile files one at a time and independently. When the compiler compiles a file, it goes through each line sequentially. Therefore, we must ensure that all functions/variables/classes are *declared* first before they are called. *Forward declaration* makes this a lot easier.

If we

1.3 Linking

Remember, declaration is not the same thing as definition. When we do the linking, we go through all the source files in our project and match all the declarations with our definitions. The source files must all be written in the compile command.

```
g++ main.cpp add.cpp
g++ add.cpp main.cpp
```

This should not be order dependent. The source files can be

```
// main.cpp
int add(int x, int y); // declaration

int main() {
  int z = add(2, 3);
  return 0;
}
```

```
// add.cpp
// definition
int add(int x, int y) {
  return x + y;
}
.
.
```

2 Multiple Files