# 1 Linking in C++

Each translation unit (.ccp file) has no relation to other translation units. Because of this, if we want to split our program into multiple files, we need a way to link them together into one executable file.

Even if we don't use multiple files, we need to link functions to each other so that we know the entry point (where to start the instructions).

## 1.1 When does linking occur

Linking occurs whenever a function may be called. Errors regarding function linking are different from compilation errors, which occur when there is no function declaration available.

In the following example, the project will build successfully although you might think it would raise a linking error.

```
\\ Log.cpp
void Logr(const char* message) {
    return message;
}

\\ Main.cpp
void Log(const char* message);

static int multiply(int a, int b) {
    Log("message");
    return a * b;
}

int main() {
    return 0;
}
```

This does not raise an error because `multiply` is a static function, which means that it can only be accessed from inside `main.cpp`. Since it is not called inside of the main function, the compiler ignores it. If `multiply` is not marked as a static function, there will be a linking error, since it could be called from other cpp files.

## 1.2 Ambiguous symbols

Another common linking error can occur when multiple functions are defined with the same signature. This may seem like an uncommon error, however, look at this example.

```
\\ log.h
#pragma once
#include <iostream>

void log(const char* message) {
    std::cout << message << std::endl;
}

\\ initLog.cpp
#include "log.h"

void initLog() {
    log("Initialized");
}

\\ main.cpp
#include "log.h"

int main() {
    log("message");
    return 0;
}
```

In this example, the contents of `log.h` are copied into both `initLog.cpp` and `main.cpp`. This means that there are two identical functions, which causes a linking error. This is why you should **never** put function definitions inside of a header file!

This error can be avoided by identifying `log` as a static function, so that the function in `initLog.cpp` has a different signature from the function in `main.cpp`, by marking `log` as inline, or by moving the definition into a .cpp file and leaving only the declaration in the header.

## 1.3 Errors

Since compilation and linking are separate processes, compilation and linking will give us separate errors. It is very important to know the difference when debugging your program. The error code of compilation errors will start with a C i. e. *C0239*.