C in Practice

Kaiyuan Xu

February 21, 2022

Make it work, but also make it good!

Make C program link correctly

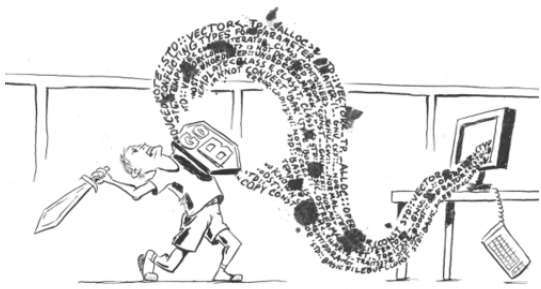
How to write header files

Debugging and Compiling

Q. & A.

Compile and run your code while writing it!

Useful tools: Google Test (test framework), Valgrind (detects memory bugs), Gcov (coverage and profiling).



Pitfall: Ignoring warnings.

Always use `-Wall -Wextra` flags.

Get even more advice from a linter such as Clang-Tidy.

Fallacy: The code works (compiles) will always work (compile).

C is not designed carefully, so many things could go wrong if you do not know what you are doing.

- problematic header files
- linkage errors
- runtime errors
- memory leak
- ...

Make it work, but also make it good!

Make C program link correctly

How to write header files

Debugging and Compiling

Q. & A.

Declaration Vs. Definition.

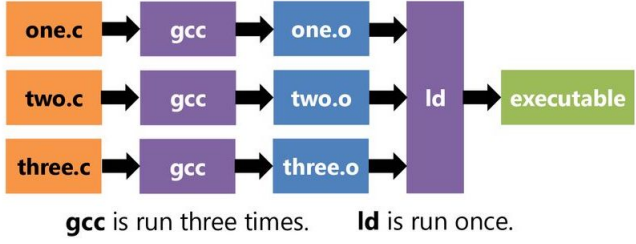
You can declare a function/structure (exactly the same) as many times as you want, but you can only define a function/structure once.

```
1 int fn(void); /* declaration */
2
3 int fn(void) { return 0; } /* definition */
4
5 struct a; /* declaration */
6
7 struct a {
8     int first;
9 }; /* definition */
```

Translation Unit

A **translation unit** is the ultimate input to the compiler from which an object the file is generated.

The compiler can only see part of the program, which means some errors can only be discovered at link time.



Linkage Error: Why it happens?

```
/usr/bin/ld: b.o: in function `fn_a':
b.c:(.text+0x0): multiple definition of `fn_a';
a.o:a.c:(.text+0x0): first defined here
/usr/bin/ld: a.o: in function `main':
a.c:(.text+0x52): undefined reference to `var_c'
collect2: error: ld returned 1 exit status
```

Where is My Symbol?

```c
int fn_a(void) {
    return 0;
}
static int fn_b(void) {
    return 0;
}
extern int fn_c(void);
int fn_d(void);
int var_a = 0;
static int var_b = 0;
extern int var_c;
int var_d;
```

```
Section Headers:
  [Nr] Name              Type             Flags
  [ 1] .text             PROGBITS          AX
  [ 4] .bss              NOBITS            WA

Symbol table '.symtab':
Size Type    Bind   Vis       Ndx Name
  15 FUNC    GLOBAL DEFAULT     1 fn_a
  15 FUNC    LOCAL  DEFAULT     1 fn_b
   0 NOTYPE  GLOBAL DEFAULT   UND fn_c
   0 NOTYPE  GLOBAL DEFAULT   UND fn_d
   4 OBJECT  GLOBAL DEFAULT     4 var_a
   4 OBJECT  LOCAL  DEFAULT     4 var_b
   0 NOTYPE  GLOBAL DEFAULT   UND var_c
   4 OBJECT  GLOBAL DEFAULT   COM var_d

Relocation section '.rela.text':
  Offset             Type         Sym. Name + Addend
00000000004c  R_X86_64_PLT32     fn_c - 4
00000000005c  R_X86_64_PLT32     fn_d - 4
000000000052  R_X86_64_PC32      var_c - 4
000000000062  R_X86_64_PC32      var_d - 4
```

Global Variables are Evil!

Linker could merge any `COMMON` global variable against any other global variable (even with a different type) with the same name. Using `-fno-common` flag (default in GCC 10) can avoid generating `COMMON` global variables.

Using `extern` variables in header files is better but global variables are still bad.

Never use global variables! Define a static variable and access it through some functions instead. This could help prevent concurrent bugs.

Make it work, but also make it good!

Make C program link correctly

How to write header files

Debugging and Compiling

Q. & A.

`#include`: Nothing magical!

Preprocessor simply replaces `#include` directive with the file specified.

Header files are used to avoid writing things multiple times in different translation units.

Mistake: Not using `#include` guard.

This will not necessarily cause errors, but is considered as bad practice.

```
1  #ifndef PROJECT_PATH_NAME_H
2  #define PROJECT_PATH_NAME_H
3
4
5  /*****
6   *
7   * The code here will never be 'included' twice.
8   * The #include guard should be used in all header files.
9   *
10  *****/
11
12
13 #endif /* PROJECT_PATH_NAME_H */
```

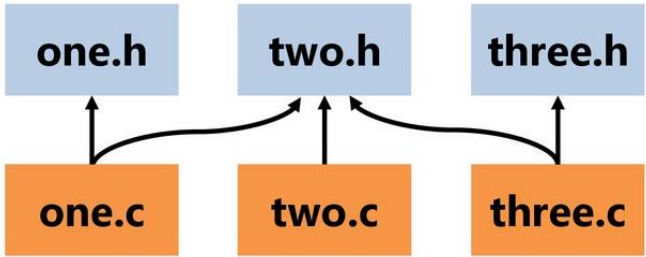
Mistake: Recursive include.

With include guard, one of the recursive includes will have of effect.
This could lead to some mysterious ‘undefined reference’ errors.

Reorder your header file and break the recursive dependency.
Declare the function you need instead of including it if necessary.

Mistake: Not making the header file compilable by itself.

A header file should include any dependency file, not relying on the file it includes.



What should go into an header file?

Header files are used to provide information needed in different translation units. Things can be put in header files:

- macro definitions
- structure (enum, typedef) definitions (declarations)
- function declarations
- static inline functions which are small and simple (since C99)

Mistake: Place static variable in header file.

Compiler will generate the same static variable multiple times in each translation unit!

What about static functions?

`inline` is introduced in **C99**, this could prevent generating duplicate functions and even improve performance in some situations.

Make it work, but also make it good!

Make C program link correctly

How to write header files

Debugging and Compiling

Q. & A.

GDB: Locate runtime errors.

Using -g flag to generate debug information for gdb. Using command `gdb` to invoke debugger.

- `b`: set break point.
- `c`: Continues running the program until the next breakpoint or error.
- `s`: Runs the next line of the program.
- `bt`: show the current stack back trace.
- `p`: print variable.
- `info stack full`: show the current stack along with all variables on stack.

Mordern Compilers: Faster than assembly.

Nowadays compiler can generate highly optimized code probably better than handwritten assembly.

- different optimization levels: `-O1`, `-O2`, `-O3`, `-Os`.
- link time optimization: `-flto`.

Make it work, but also make it good!

Make C program link correctly

How to write header files

Debugging and Compiling

Q. & A.