C Header File Guidelines

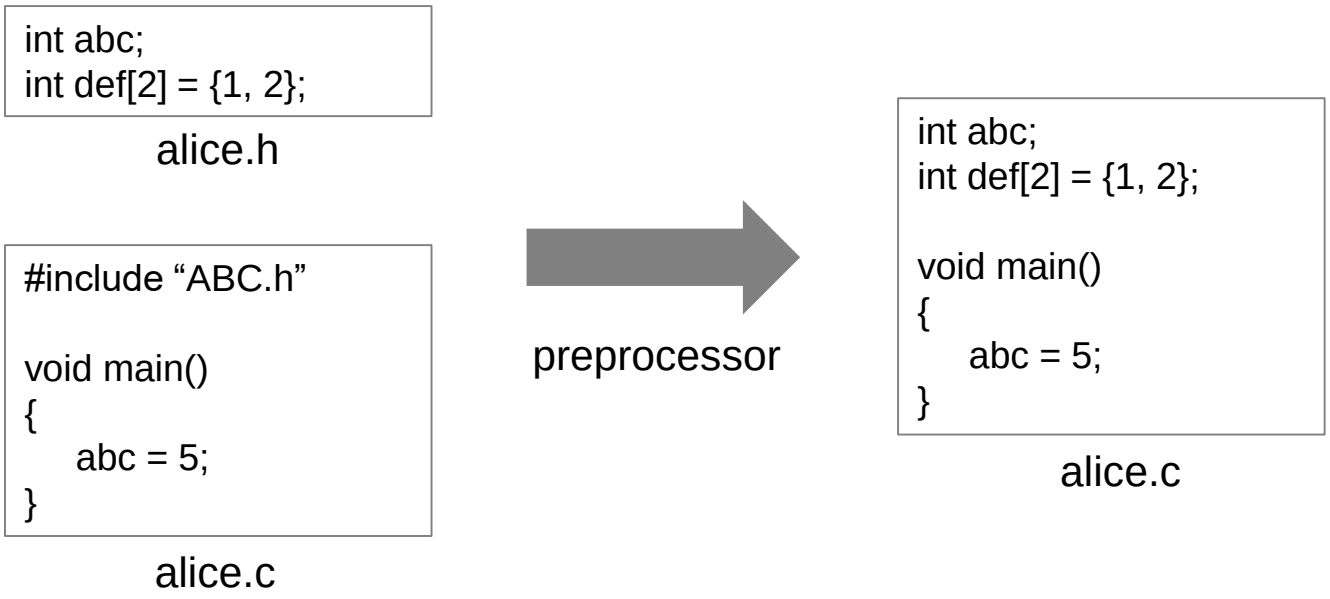
Minsoo Ryu

Operating Systems and Distributed Computing Lab.
Hanyang University

msryu@hanyang.ac.kr

Meaning of “#include” Directive

- A #include directive causes the preprocessor to replace the directive with the contents of the specified file



Declarations and Definitions

- A "declaration" specifies the interpretation and attributes of a set of identifiers
  - A declaration also specifies where and when an identifier can be accessed (the "linkage" of an identifier)

- A declaration that also causes storage to be reserved for the object or function named by the identifier is called a "definition"

Rules in C++ and C

One Definition Rule in C++

- **In any translation unit, a template, type, function, or object can have no more than one definition**
  - However, uninitialized variables or functions without body implementation may be declared more than once in a translation unit
- **In the entire program, an object or non-inline function can have no more than one definition**
  - However, some things, like types and templates, can be defined in more than one translation unit
  - For a given entity, each definition must have the same sequence of tokens

Undefined behavior in C

- **There may be more than one external definition for the identifier of an object, with or without the explicit use of the keyword extern; if the definitions disagree, or more than one is initialized, the behavior is undefined**

Example: Single File Case

```
int abc;                    /* First declaration */
int abc;                    /* Second declaration */
int abc = 3;                /* Assignment makes it
                               definition */
//int abc = 4;              /* Multiple Definition */

int def[2];                 /* First declaration */
int def[2];                 /* Second declaration */
int def[2] = {1, 2};        /* Assignment makes it
                               definition */
//int def[2] = {4, 5};      /* Multiple Definition */
```

alice.c (compilable)

```
struct abc {
    int member;
};

/* struct abc {             /* Multiple Definition */
    int member;
}; */

enum localEnum {
    ready,
    running
};

/* enum localEnum {         /* Multiple Definition */
    ready,
    running
}; */
```

bob.c (compilable)

- Note that we often use typedef for struct, but typedef simply gives an alias name to a data type
- Note that you can make an incomplete definition of struct X, also called "forward declaration", before the first use of X (particularly when we want to refer to X by a pointer)

Example: Hierarchical Inclusion

```
int abc;
int abc;
int abc = 3;

int def[2];
int def[2];
int def[2] = {1, 2};
```

grandparent.h

```
#include “grandparent.h”
```

parent.h

```
#include “grandparent.h”
#include “parent.h”
```

alice.c (not compilable)

Example: Multiple File Case

```
int abc;
int abc;
int abc = 3;

int def[2];
int def[2];
int def[2] = {1, 2};
```

alice.c

```
int abc;
int abc;
int abc = 3;

int def[2];
int def[2];
int def[2] = {1, 2};
```

bob.c

alice.o and bob.o cannot be linked

```
#define PI 3.14

struct abc {
    int member;
};
```

alice.c

```
#define PI 2.15

struct abc {
    int member;
};
```

bob.c

alice.o and bob.o can be linked

You can define the same macro multiple times using different values, but the value of macro might not be what you expect

Avoiding Multiple Definitions

- Define global variables and functions in only one .c file, contain extern declarations of them in header files
- Declare types (struct, union, enum, ...) and macros in only one .h file and include it in every .c file that needs them

```
#include “alice.h”

int glob;
int foo() {return 0;}
```

alice.c

```
extern int glob;
extern int foo();

struct abc {
    int member;
};
```

alice.h

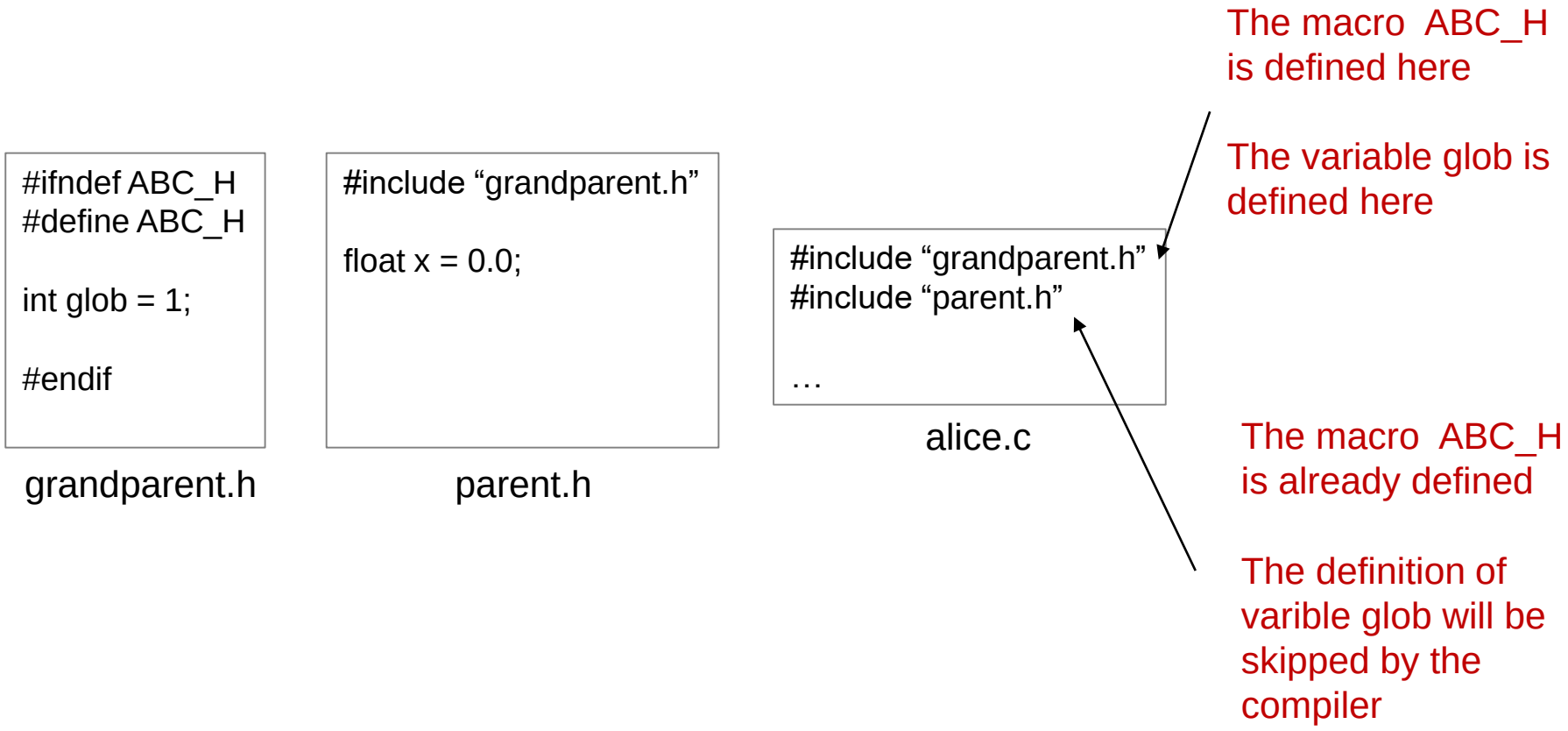
```
#include “alice.h”

int main() {foo();}
```

bob.c

Include Guards

- Use “include guard” when you are not sure
  - It is the most widely used construct to avoid multiple definitions



More Guidelines for Header Files

- The header file "alice.h" should #include every other header file that "alice.h" requires to compile correctly, but no more
  - Make sure that the module "bob.c" using "alice.c" only needs to include "alice.h"
  - Never include the internal declarations or definitions of "alice.c" in the header file "alice.h"

- The "alice.c" file should first #include its "alice.h" file, and then any other headers required for its code
  - It will be a nice defensive check that "alice.c" and "alice.h" are consistent
  - It helps to keep your list of include files leaner and cleaner
    - Your module's header file will include anything that it needs
    - Therefore, including that header file means that you don't have to explicitly include all those extra header files

thank you!