Program Design (1)

Program Design (II)

2022 Spring

Fu-Yin Cherng
Dept. CSIE, National Chung Cheng University

All materials are provided by C Programming: A Modern Approach by K. N. King, 2nd edition, 2008, W. W. Norton & Company.

Storage Classes - Summary

- Ranking
- Select the group of three students with the heighest score; if having the same score, choose the faster group
  - 1st: 5 points 407530017, 407530022, 409210011
  - 2nd: 3 points 409220042, 409520056, 410410023
  - 3rd: 2 points 407420093, 407420079, 407530015
  - rest: 1 points

Final Project

- This is a team project that requires **four to five members**. Please find your teammates and register your team at this link: https://forms.gle/wdaGSv3BgQFSX3hj8
- **Please finish the registration before 2022/4/29**.
- Otherwise, you will not be able to obtain the score of the final project.
- Final Project: 20%

Outline

- Program Design
- Module
- Information Hiding

Program Design

- It’s obvious that the real-world programs are larger than the examples in this course.
- Most full-featured programs are at least 100,000 lines long.
- Although C wasn’t designed for writing large programs, many large programs have been written in C.
- Writing large programs is quite different from writing small ones.
  - building a small doghouse is not the same with building an apartment

Program Design

- Issues that arise when writing a large program:
  - Style
  - Documentation
  - Maintenance
  - Design
- Previous chapter of writing large programs focus on language details in C.
- This chapter focuses on design techniques that can make C programs readable and maintainable.

Program Design

- However, a complete discussion of program design issues is beyond the scope of this course.
- We will try to cover some important concepts and show how to use them to create C programs
- Complete techniques of program design will be introduced in the course like software engineering

“The only way to go fast is to keep the code clean.”

— Clean Code, Martin, Robert C, 2009

Modules

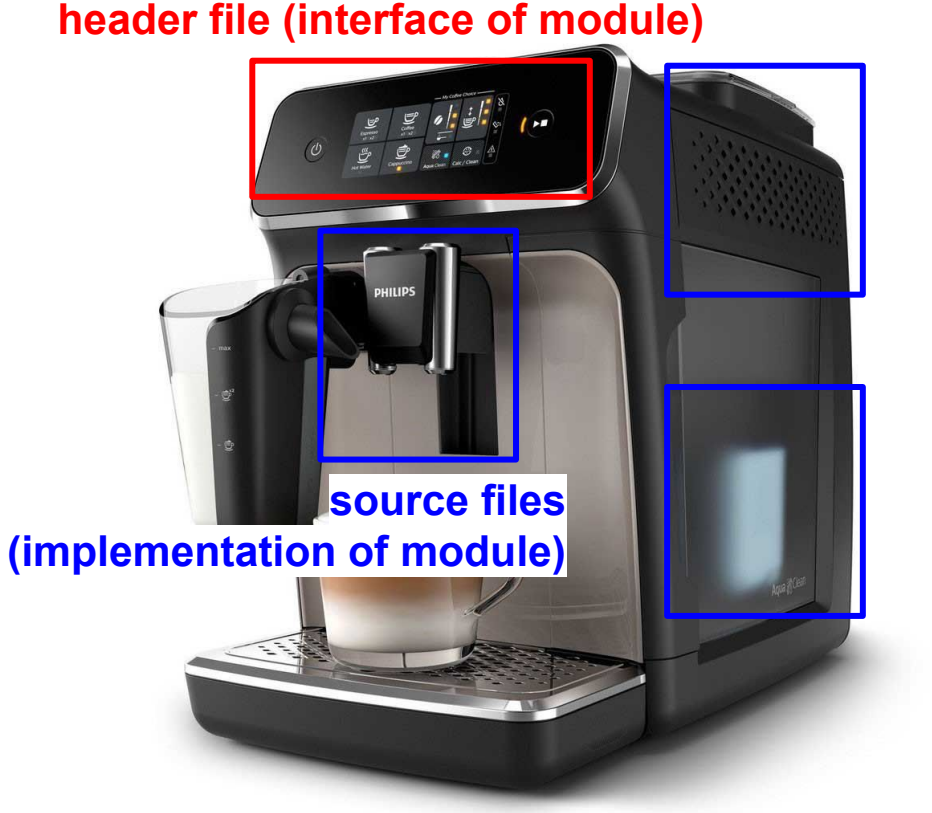
- It’s often useful to view a program as a number of independent ***modules.***
- A module is a collection of services, some of which are made available to other parts of the program (the ***clients***).

Modules

- Each module has an ***interface*** that describes the available services.
- The details of the module—including the source code for the services themselves—are stored in the module's ***implementation.***
- For example, the components of computer don't have to know how each other work or be implemented.
- They can still connect to each other to form a wrokable computer

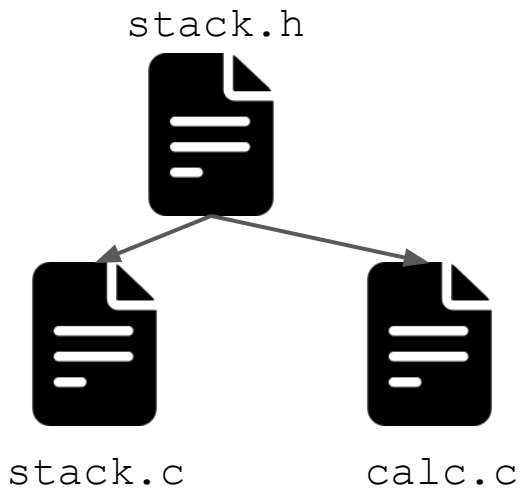
Modules

- In the context of C, “services” are functions.
- The interface of a module is a header file containing prototypes for the functions that will be made available to clients (source files).
- The implementation of a module is a source file that contains definitions of the module’s functions.



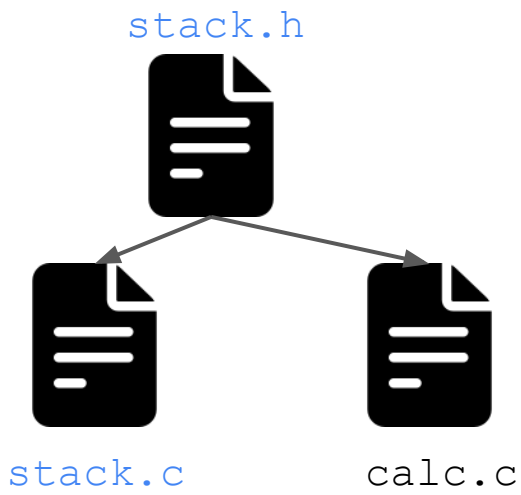
Modules

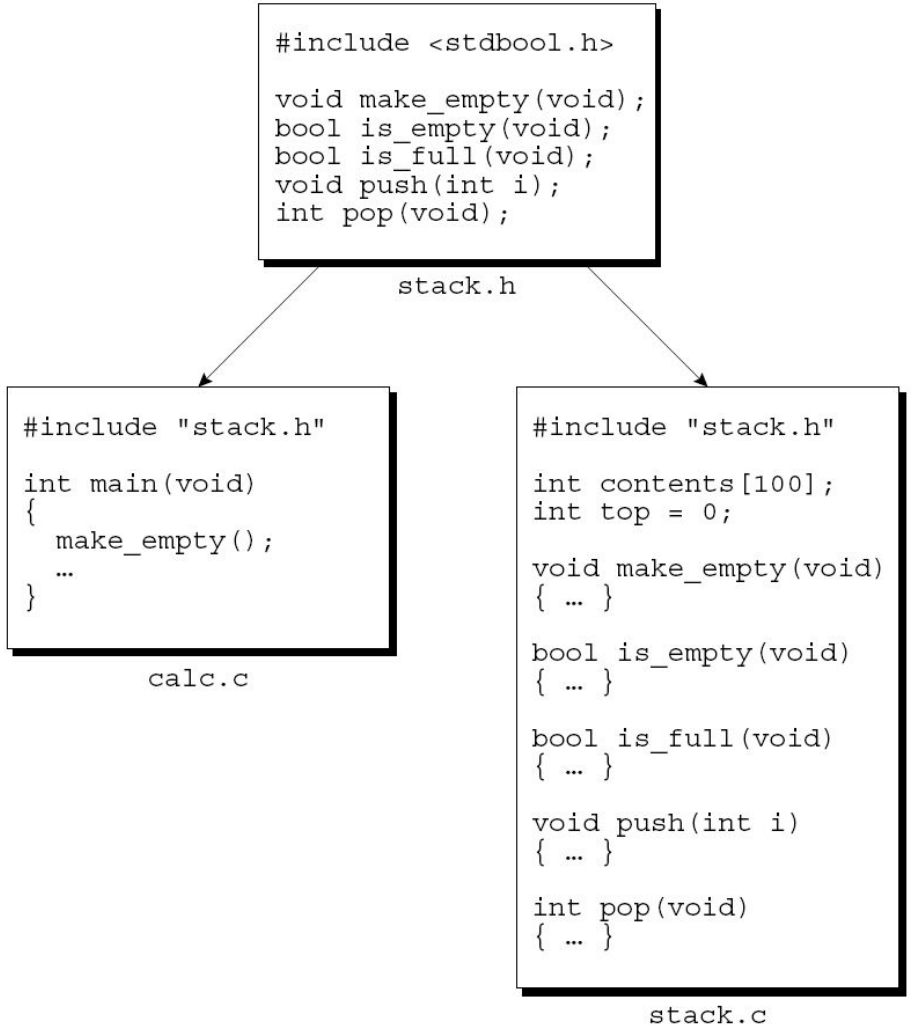
- The calculator program sketched when introducing Reverse Polish Notation (RPN) calculator example consists of:
  - `calc.c`, which contains the `main` function
  - A stack module, stored in `stack.h` and `stack.c`



Modules

- `calc.c` is a *client* of the stack module.
- `stack.h` is the *interface* of the stack module.
- `stack.c` is the *implementation* of the module.




#include <stdbool.h>
void make_empty(void);
bool is_empty(void);
bool is_full(void);
void push(int i);
int pop(void);
stack.h
#include "stack.h"
int main(void)
{
make_empty();
…
}
calc.c
#include "stack.h"
int contents[100];
int top = 0;
void make_empty(void)
{ … }
bool is_empty(void)
{ … }
bool is_full(void)
{ … }
void push(int i)
{ … }
int pop(void)
{ … }
stack.c

Modules

- The C library is itself a collection of modules.
- Each header in the library serves as the interface to a module.
  - `<stdio.h>` is the interface to a module containing I/O functions.
  - `<string.h>` is the interface to a module containing string-handling functions.

Let’s Take a Break!

Modules

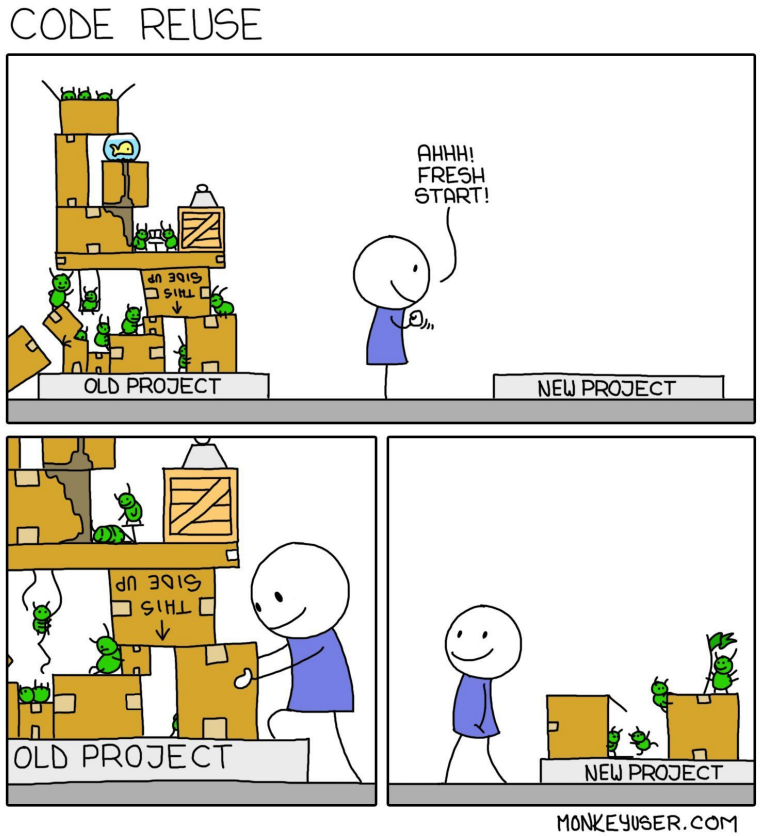
- Advantages of dividing a program into modules:
  - Abstraction
  - Reusability
  - Maintainability
- Let’s see the meaing of each one

Modules - Abstraction

- A properly designed module can be treated as an ***abstraction;***
- We know what it does, but we don't worry about how it works.
- Thanks to abstraction, it's not necessary to understand how the entire program works in order to make changes to one part of it.
- Abstraction also makes it easier for several members of a team to work on the same program.

Modules - Reusability

- Any module that provides services is potentially reusable in other programs.
- Since it's often hard to anticipate the future uses of a module, it's a good idea to design modules for reusability.



Modules - Maintainability

- A small bug will usually affect only a single module implementation, making the bug easier to locate and fix.
- Rebuilding the program requires only a recompilation of the module implementation (followed by linking the entire program).
- An entire module implementation can be replaced if necessary.

Modules - Maintainability

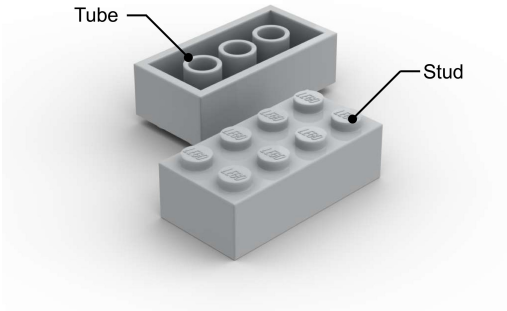
- Maintainability is the most critical advantage.
- Most real-world programs are in service over a period of years
- During this period, bugs are discovered, enhancements are made, and modifications are made to meet changing requirements.
- Designing a program in a modular fashion makes maintenance much easier.

Modules

- Decisions to be made during modular design:
  - What modules should a program have?
  - What services should each module provide?
  - How should the modules be interrelated?

Cohesion and Coupling

- In a well-designed program, modules should have two properties.
- ***High cohesion.***
  - The **elements** of each module should be **closely related to one another.**
  - High cohesion makes modules easier to use and makes the entire program **easier** to **understand**.



Cohesion and Coupling

- In a well-designed program, modules should have two properties.
- ***Low coupling.***
  - Modules should be as **independent** of each other as possible.
  - **Low coupling** makes it easier to modify the program and reuse modules.

Types of Modules

- Modules tend to fall into certain categories:
  - Data pools
  - Libraries
  - Abstract objects
  - Abstract data types

Types of Modules - data pool

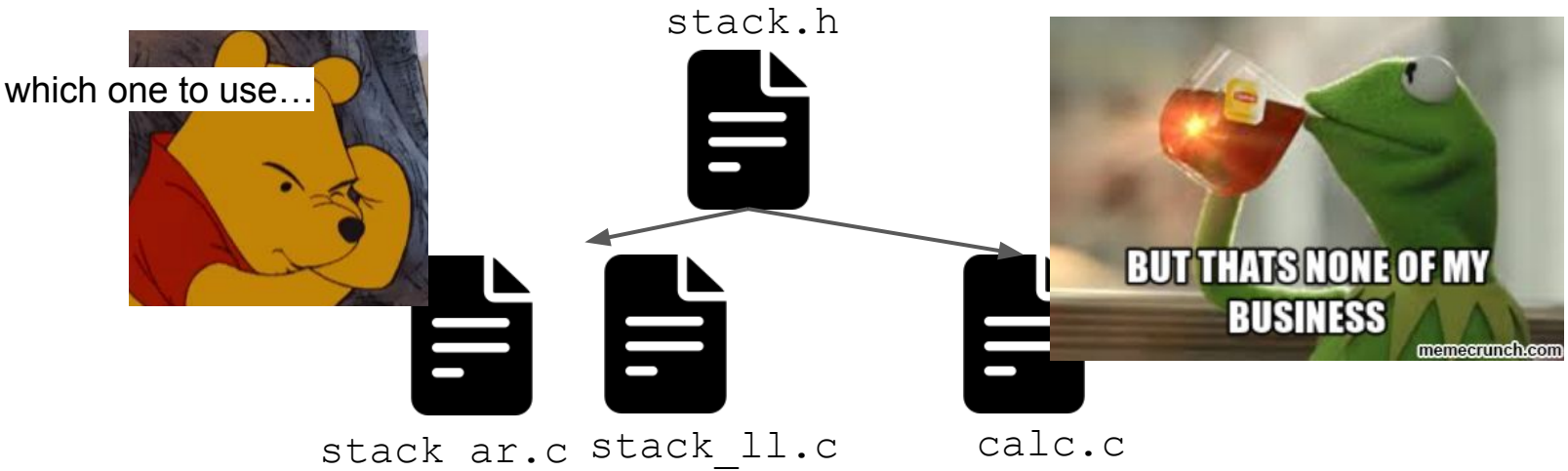
- A ***data pool*** is a collection of related variables and/or constants.
  - In C, a module of this type is often just a header file.
  - `<float.h>` and `<limits.h>` are both data pools.
- A ***library*** is a collection of related functions.
  - `<string.h>` is the interface to a library of string-handling functions.

Types of Modules - Abstract Object & ADT

- An ***abstract object*** is a collection of functions that operate on a hidden data structure.
- An ***abstract data type (ADT)*** is a type whose representation is hidden.
  - Client modules can use the type to declare variables but have no knowledge of the structure of those variables.
  - To perform an operation on such a variable, a client must call a function provided by the ADT.

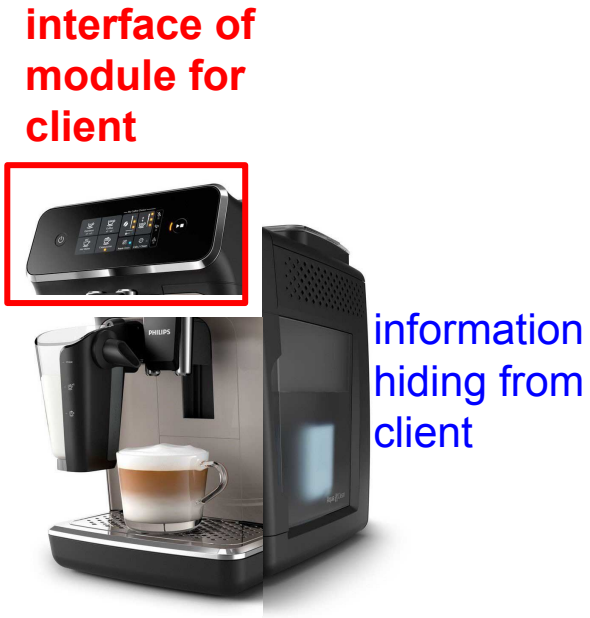
Information Hiding

- A well-designed module often keeps some information **secret** from its **clients**.
  - Clients of the stack module have no need to know whether the stack is stored in an array, in a linked list, or in some other form.



Information Hiding

- Deliberately concealing information from the clients of a module is known as ***information hiding.***
- Primary advantages of information hiding:
  - ***Security.*** If clients don't know how a module stores its data, they won't be able to corrupt it by tampering with its internal workings.
  - ***Flexibility.*** Making changes—no matter how large—to a module's internals won't be difficult.



Information Hiding

- In C, the major tool for enforcing information hiding is the `static` storage class.
  - A `static` variable with file scope has internal linkage, preventing it from being accessed from other files, including clients of the module.
  - A `static` function can be directly called only by other functions in the same file.

Summary

- Program Design
  - Why?
- Module
  - Abstraction
  - Reusability
  - Maintainability
  - Cohesion and Coupling
  - Types of Modules
- Information Hiding
  - Security and Flexibility

Leave some time for you to find team member for the final project and discuss the topic of final project