Week 12

Modular Design and API

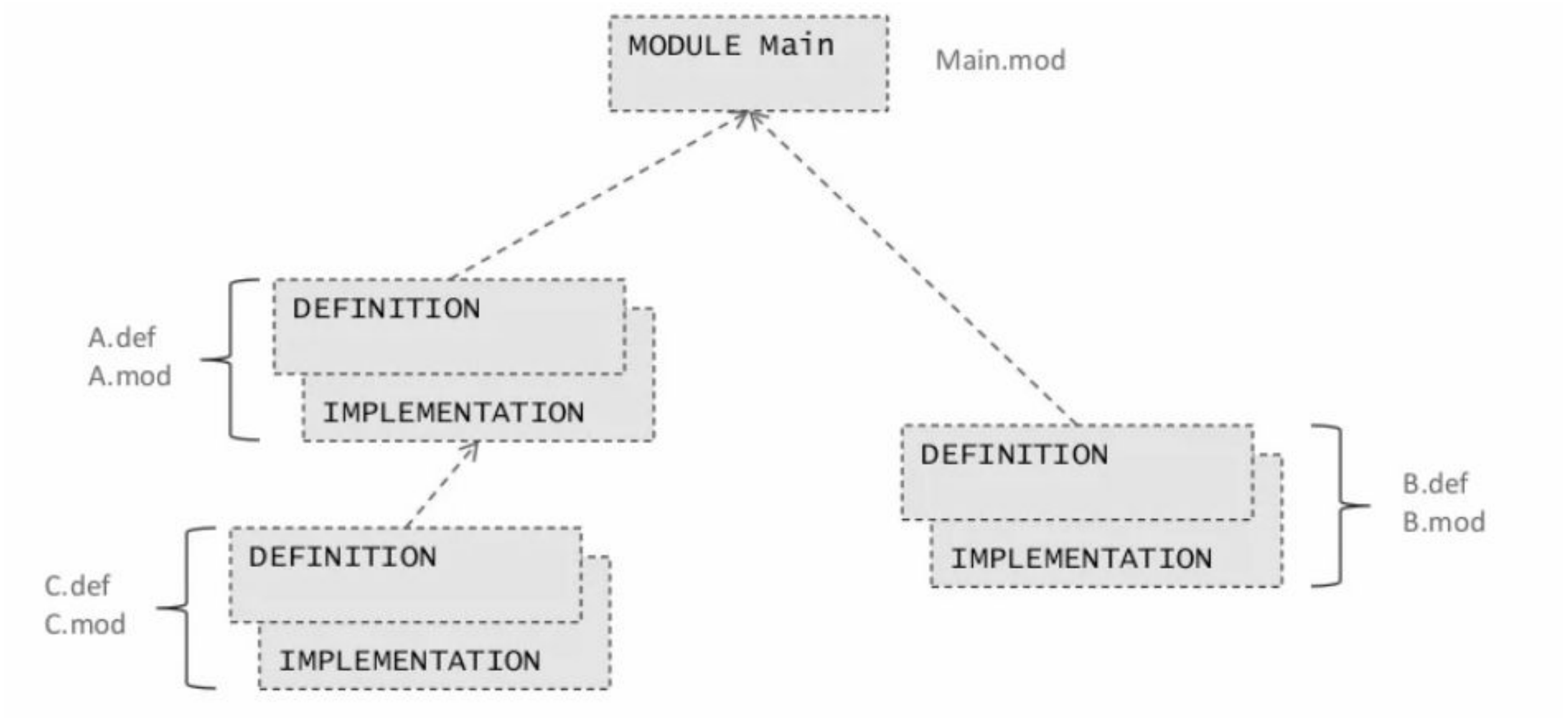
(first half of Unit 6!)

Instructors: E. Estrada, M. Ponce, P. Gawde

Quick recap on modules

- Modular programming is one way of managing the complexity of large C (or any other programming language) programs.
- It groups related sets of functions together into a module.
- The module is divided into an Interface and an Implementation
- Interface
  - Is exported by the module
  - Is imported by the client
- Implementation of functions
  - Is hidden from clients
- Modular programming promotes **Abstraction**

Modula-2 and -3 The archetypal modular programming language



Modular Programming

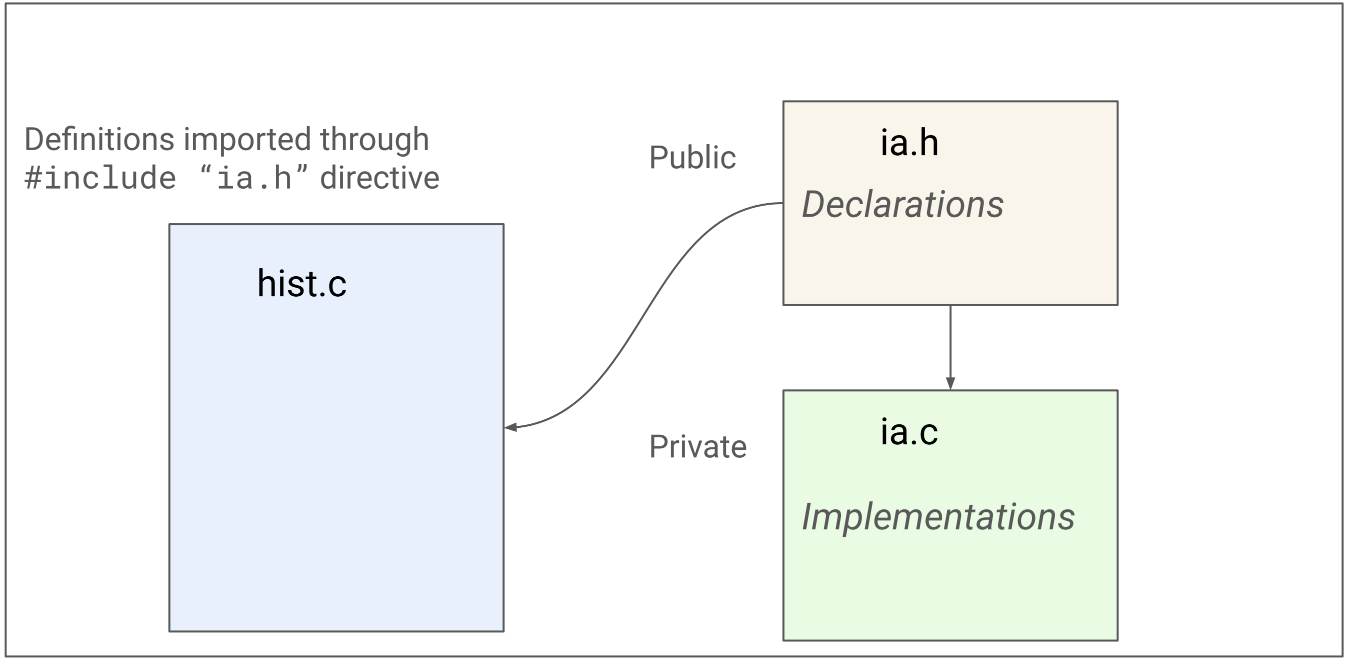
- Rules of modular programming
    - Client using a module sees the module's interface definition.
    - Client should not use anything in the implementation that is not defined in the interface.
    - **Information Hiding**: Module must ensure that the client cannot access the internals of implementations
        - Protect the integrity of its data and guarantee the module's correctness
        - Allows module's programmer to substitute the current implementation with improved implementation without affecting anything outside the module
- Syntax for creating a module:
    - .h file (Interface):
        - List of functions and data structures to export
        - declarations of those functions and data
    - .c file (Implementation):
        - Lists its dependencies on other module
        - Defines the exported functions
        - May use hidden helper functions and data

Example - Infinite arrays

- Example taken from : Practical C Programming 3rd Edition, by Steve Oualline
- Goal:
    - Write a module that allows the user to store the data into the array without worrying about the size of the array
    - Infinite array grows as needed
    - The array stores data for histogram

Public and Private

- Modules are divided into two parts: **public** and **private**.



The extern modifier

The **extern** modifier is used to indicate that a variable or function is defined outside the current file

main.c

```c
#include <stdio.h>
/* number of times through the loop */
extern int counter;
/* routine to increment the counter */
extern void inc_counter(void);
int main()
{
      int index; /* loop index */
      for (index = 0; index < 10; index++)
            inc_counter();
      printf("Counter is %d \n", counter);
      return (0);
}
```

count.c

```c
/* number of times through the loop */
int counter = 0;
/* trivial example */ void
inc_counter(void)
{
      ++counter;
}
```

Modifiers

- extern
  - Variable/function is defined in another file.
- (none)
  - Variable/function is defined in this file (public) and can be used in other files.
- static
  - Variable/function is local to this file (private).

Small Example for Static and Extern

Example of Implementation

```
/* mod1.c */
/* Import needed */
#include “a.h”
#include “b.h”
/* Implements this interface: */
#include “mod1.h”

int var1;
static int var2;

void fun1(int *p) { … }
static void fun2(void) { … }
```

- Achieving abstraction
  - Create private data and functions
  - C makes every function (and data) defined in a file accessible from functions defined in any other file.
  - Solution?
    - C's static keyword to limit the scope of functions and data to a single file.

Example of Interface

```
/* mod1.h */

extern int var1;
extern void fun1(int *);
```

- Interface declares the external, publicly accessible parts of the module.
- A client module can access the integer variable var1 .
- Or, it can call function fun1 with an argument that is a pointer to an integer.

Continue Infinite Array Example

- Goal:
  - Write a module that allows the user to store the data into the array without worrying about the size of the array
  - Infinite array grows as needed
  - The array stores data for histogram

Headers

- Contains **public** information:
    - A comment section describing clearly what the module does and what is available to the user
    - Common constants
    - Common structures
    - Prototypes of all the public functions
    - extern declarations for public variables

Programming Example Link:

ia-2

How do we run it?

Compiling + Linking = Building

Makefile

- The program make is designed to aid the programmer in compiling and linking programs.
- The program make was created to make compilation dependent upon whether a file has been updated since the last compilation.
- The program allows you to specify:
  - dependencies of the program file and the source file,
  - the command that generates the program from its source.
- The file Makefile contains the rules used by make to decide how to build the program.
- The Makefile contains the following sections:
  - Comments
  - Macros
  - Explicit rules
  - Default rules

Makefile

```
#----------------------------------------------#
#    Makefile for unix systems          #
#    using a GNU C compiler             #
#----------------------------------------------#
all: ia.o hist.o
     gcc ia.o hist.o

ia.o:
     gcc -Wall -Wextra -std=c11 -c ia.c

hist.o:
     gcc -Wall -Wextra -std=c11 -c hist.c

clean:
     rm -f a.out hist.o ia.o
```

Compilation in c

Command: Open the door.

In general the compiler takes four steps when converting a .c file into an e

1. pre-processing - textually expands `#include` directives and `#def`
2. compilation - converts the program into assembly
3. assembly - converts the assembly into machine code
4. linkage - links the object code to external libraries to create an exec

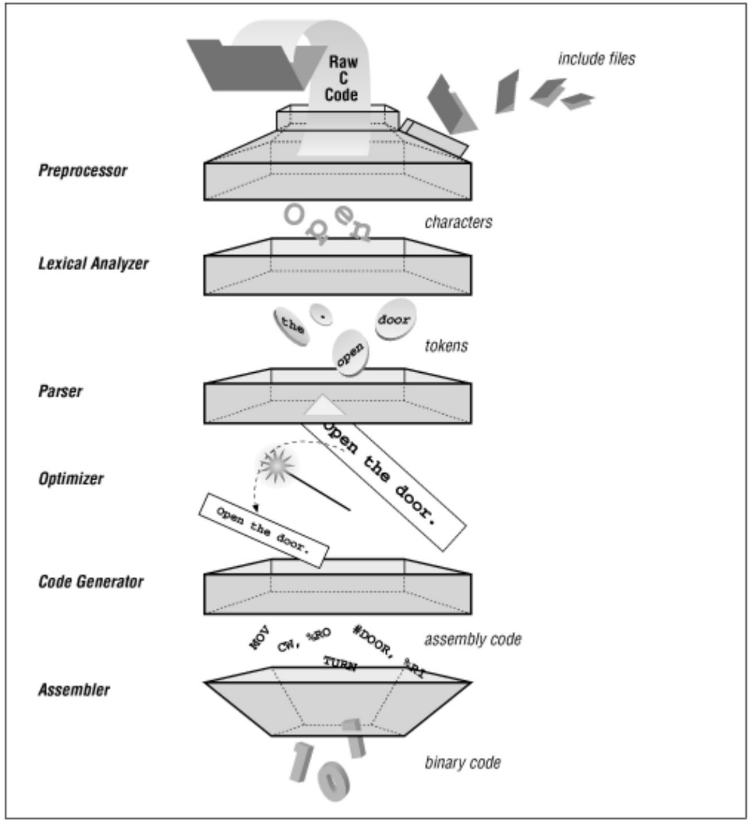


Image taken from: Practical C Programming 3rd Edition, by Steve Oualline

File Types in C

1. Source file:

   These files contain function definitions, and have names which end in .c by convention.

2. Header file:

   These files contain function prototypes and various preprocessor statements

3. Object file:

   These files are produced as the output of the compiler. They consist of function definitions in binary form, but they are not executable by themselves. Object files end in .o by convention

4. Binary executables:

   These are produced as the output of a program called a "linker". The linker links together a number of object files to produce a binary file which can be directly executed.

5. Libraries:

   A library is a compiled binary but is not in itself an an executable

Use of Modules

What we need to work with the modules?

- How will other programs interact with our module?
  - How to set up an interface
  - Which functions to make available to the user
  - Which data and functions are hidden from the user
  - How information is passed between various functions
- Application Programming Interface (API)
  - Specification required to make use of module

What is an API

- Blackbox for software module, code library, subsystem, and even working components of an operating system or a remote server.
- Allows us to use a module, library, or service, without having to know the details of how it's implemented
- All we need to know is how to pass information to the components of API
- APIs in C
    - function declarations (in the .h file), along with any constants and other important values defined there
    - documentation (at the top of each function) that describes what each of the functions does and their parameters and return values, and
    - any documentation that is maintained externally ()

Examples of APIs

- Google Maps: Allows you to make use of the Google maps framework for plotting locations and for finding paths between points in maps - among many other things!
  - https://developers.google.com/maps/documentation/javascript/overview
- TensorFlow: Allows you to set up, train, test, evaluate, and operate a deep neural network for solving a task that requires learning from a very large dataset. Nowadays this is behind some of the most useful applications in A.I.
  - https://www.tensorflow.org
- Amazon AWS: Amazon's cloud-based AWS runs a large portion of internet-hosted services, and powers all kinds of applications from on-line trade to providing computing power for large simulations.
  - https://docs.aws.amazon.com/index.html#lang%2Fen_us
- Unity: Possibly the most popular API for creating, manipulating, and rendering 3D content, from graphical user interfaces and simulations, to interactive programs and games.
  - https://docs.unity3d.com/ScriptReference/

How to design an API?

- Design decisions include following certain rules:
  - APIs (Public) are like arrows that leave the bow
    - The arrow that has left the bow never returns.
  - Understand the requirements carefully and develop “use case”
  - Declared functions should be self-contained and accessible only via well-defined interfaces
  - Should not expose function internals
  - Function dependence should be avoided or minimized
  - A function should perform a single specific task
  - Function interfaces should be minimal.
  - A good interface should be intuitive to use.

Software error that costed $200 million

- Martian Climate Orbiter was launched in 1999
- September 23, 1999, communication with the spa
- the primary cause of this failure was that one piec produced results in a United States **customary uni** expected those results to be in **SI units**.

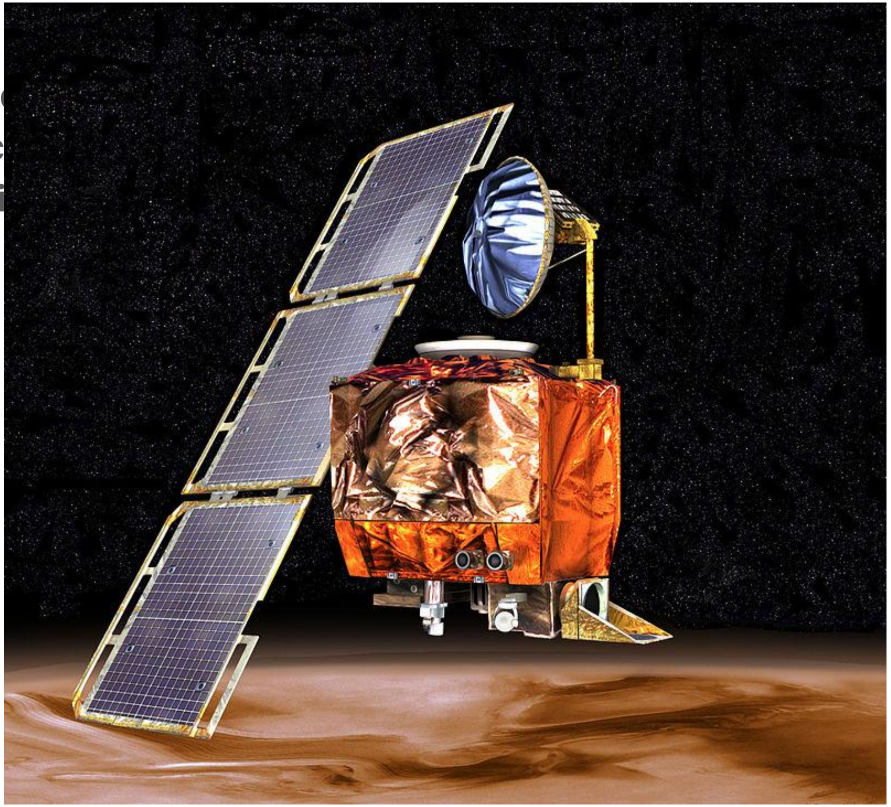


Image Source https://commons.wikimedia.org/wiki/File:Mars_Climate_Orbiter_2.jpg#filelinks

Obstacles in designing good API

- Requirements keep on changing
    - We might make certain changes to given function to support a specific use case
    - That increases the amount of work that needs to be done when bug-fixing and maintaining the module.
    - Leads to **breaking changes**
    - Example:
        - Changing types in functions arguments or return types will break the applications using a given API

Some solution to avoid breaking the chain?

- Carefully plan your API in advance
  - importance of carefully planning your APIs cannot be understated.
- Test your code for accidental breaking changes.
  - Find out the breaking points in API
- Future proof documentation
  - For instance, you can warn clients in advance of some added function parameters. You can instruct clients to expect this and build their applications against it.

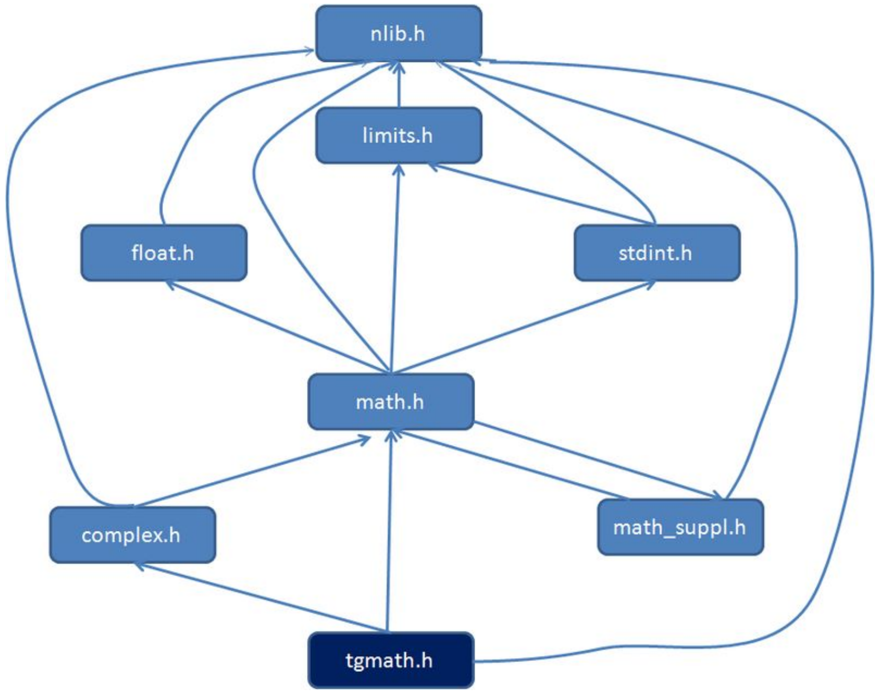
Obstacles in designing good API - Updating APIs

- Adding new features or changing implementation details is sometimes inevitable
- But all the dependencies are also required to be updated
- Each time that the API is updated, applications using it need to be re-compiled
    - especially important if your API has any security-sensitive components
- Solutions:
    - Figure out how to organize our software so it gets updated whenever the APIs are updated.
    - Possible problems:
        - Once the API is table and live, it is used by growing number of applications
        - Application dependency should also be considered

The more your API is used, the harder it is to change fundamental aspects of it because of the impact to all the software that relies on the API

Concepts

- Use case:
  - this is simply a specific situation or problem that a module or software component may be used for.
- Dependency:
  - is a relation between software modules where the dependent module requires that a library or module it depends on be available and have the correct version in order to compile and run.



Source: https://en.wikibooks.org/wiki/C_Programming/tgmath.h#:~:text=h%20is%20a%20Standard%20C.h%20.
Image Dependency Graph for tgmath.h.png CC BY-SA 3.0

Criteria for building a good API

https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/32713.pdf

- A good API should:
  - Easy to learn and use
  - Difficult to use incorrectly
  - Easy to maintain (the code in it is readable and well written)
  - Easy to extend and improve
  - Suitable for those who will be using it

A good API

- API should do one thing, and do it well
  - the functionality should be easy to explain and make sense of
- API should be as small as possible, but not smaller
  - you can add to it later, but once it’s in there, it’s hard to remove functionality
- API should be implementation independent
- Maximize information hiding
  - Only make available to the user the functionality and data that the user needs
- Names Matter–API is a Little Language
  - Use naming consistently, the same goes for the use of underscores and capitalization
- Good documentation, every component (every class, interface, method, constructor, parameter, and exception) must be documented properly.
- Consider Performance Consequences of API Design Decisions

Limitations of our Programming Language

Good side of C

- C is often called a **middle level programming language** because it supports the feature of both high level and low level language.
- C being a mid level language doesn't mean that, it is less powerful or harder to use than any high level language.
- C combines the best elements of high level language with the control and flexibility of low-level language(assembly language).
- Like assembly language, C provide support for manipulation of bits, bytes and memory pointers at the same time it provides abstraction over hardware access.

Cheating Client

list.h

```
struct List;

struct List * new(void);
void insert (struct list *l, int key);
void concat (struct list *l, struct list *m);
int search_nth_key (struct list *l, int n);
```

user.c

```
#include "list.h"
struct List {int data; struct Node *next;};
int f(void) {
      struct List *l, *m;
      l = new();
      m = new();
      insert (l,6);
      …
      }
```

- Last lecture,
  "Putting struct List; in list.h instead of struct List {fields...}; enforces the abstraction: it prevents user.c from accessing the fields of the struct."
- The enforcer has limits. Clients can always find a way to get around it. That leads to brittle, buggy programs!

What’s missing in C?

- Some concepts such as Information hiding:
  - where a stable interface is open to the rest of the program while its implementation details are hidden.
- Example:
  - C has only structs and their fields are always public, so the rest of the program always has access to the struct fields.
  - This introduces the danger of higher coupling between modules, by directly manipulating the fields of a foreign data structure.
- Example 2:
  - Designing API for linked list of strings:
    - The pointer to the head of the linked list is kept and managed by the function(s) that need the list!
  - Problem: One of the key components of the linked list, the head pointer, is declared and kept by a function outside our API.
  - C doesn’t provide any way to protect the functions and data in your module
- Solution?
  - find a different way to organize our code and data so that we can implement software modules that are truly self-contained, and that are able to control access to the data managed by the module in a way that prevents users from unintentionally, or intentionally, misusing the module and its functionality.
  - This is called information hiding, and is one of the fundamental principles of good software design.
  - The model of programming we will explore next is called **Object Oriented Programming (OOP).**