CS265
Advanced Programming Techniques

Interfaces

The practice of Programing

Kernighan & Pike

Chapter 4

The practice of programming

Typical first questions

- What programming language should we use?
- What data structures should we use?
- What algorithms should we use?

The practice of programming – larger systems

Other important questions to ask

that help balance competing goals and constraints in larger systems

- Interfaces
  - What services should be provided by the various components?
- Information Hiding
  - What data and services should be private?
- Resource Management
  - Who should be responsible for managing memory and other resources?
  - Who allocates memory?
  - Who manages shared memory?
  - Who opens files?
- Error Handling
  - Who should detect errors?
  - Who should report them?
  - How should errors be reported? (printf not such a good idea!)
  - How should we recover from errors?

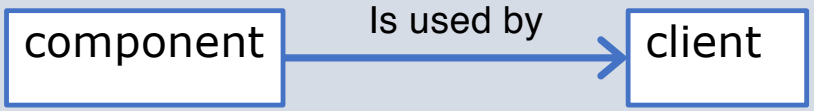
Software Systems and Modularity

One of the key concepts that are important in designing big systems is

modularity

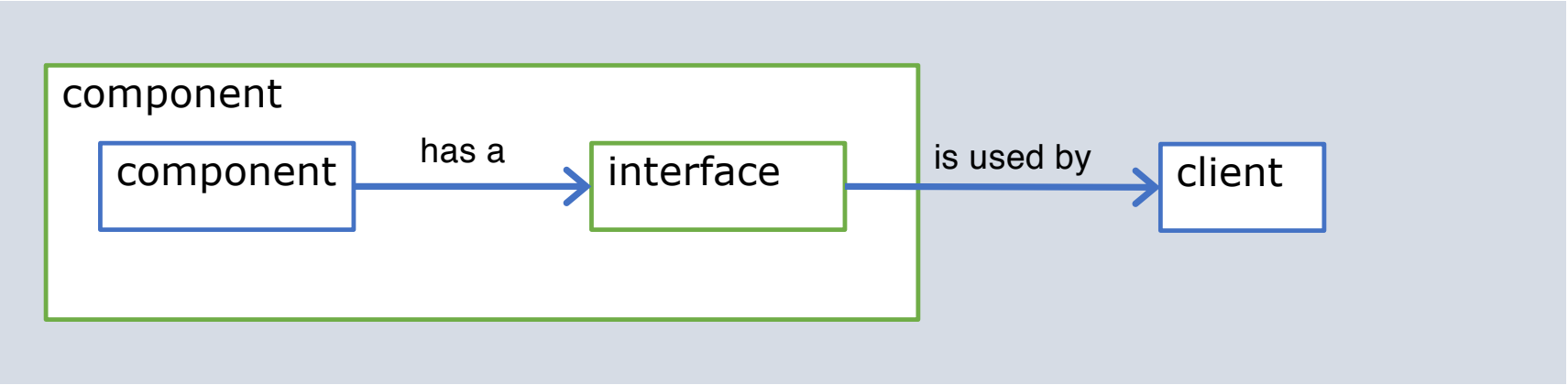
the idea and practice that subdivides a system into smaller parts called modules which can be independently created, modified, replaced or exchanged

Modularity in Software



- Breaks a big problem into smaller pieces
- Introduces dependencies (`client` depends on `module`)
- Every time a module changes, the client must be recompiled

Modularity and Interfaces



A better approach!

- The client uses the interface of a component
- The interface defines the functions exposed by the component
- The interface hides the implementation details

Interfaces – why should we care?

- We can change the implementation without affecting the clients
- We can hide implementation details (data hiding, encapsulation)
- We can support prototyping
- We can use stubs to test the interfaces without writing code, etc.

Interfaces - How

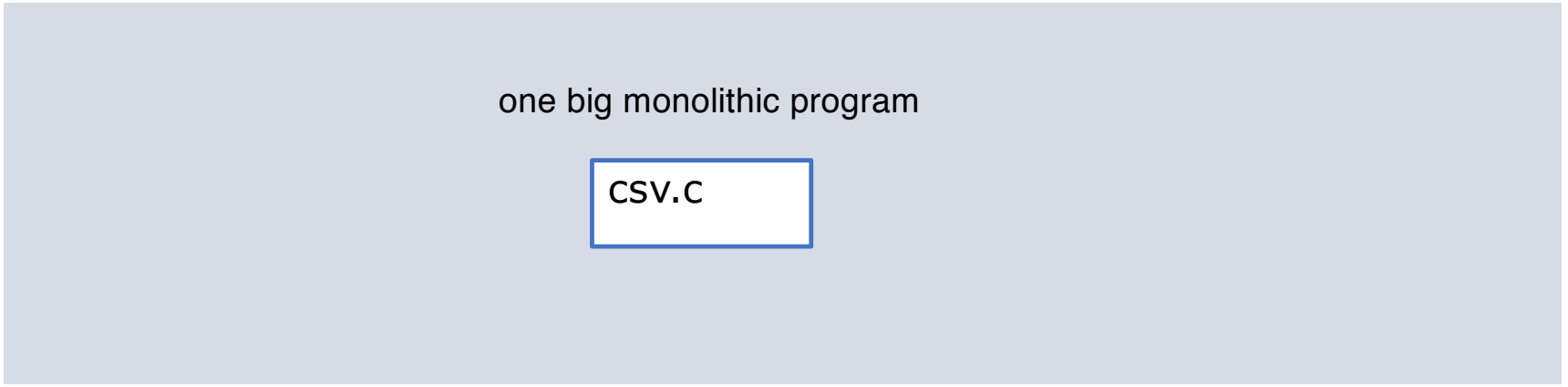
- In OO languages – super easy!
  - object classes support public interface methods which
  - define what methods the object must have
  - and describe the behavior of an object
- In C
  - we can use header files with similar results

Motivating Example - Requirements

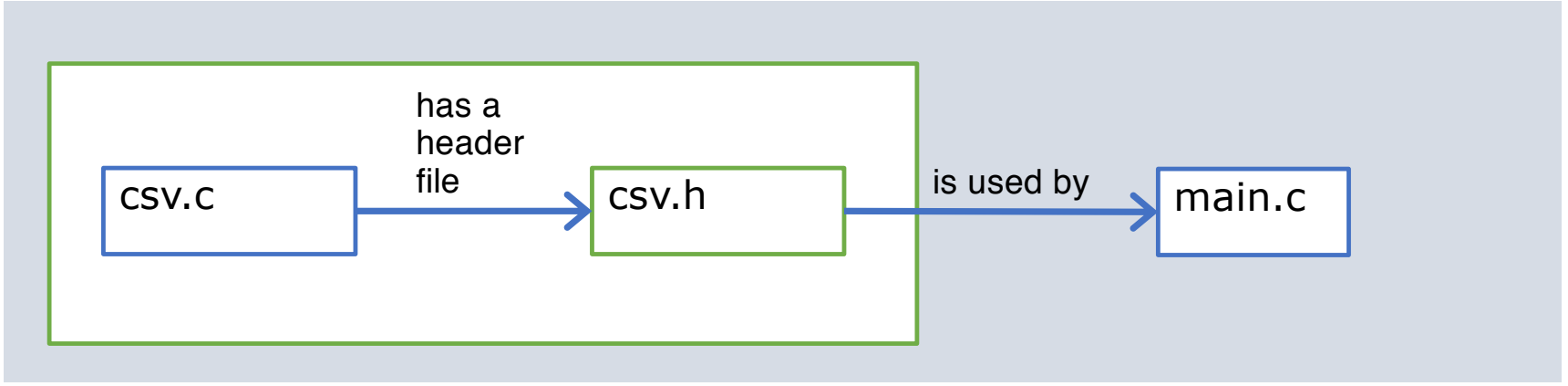
Let’s see a program that reads a CSV file

- Reads a line from a file
- Tokenizes the line into fields
- Uses commas to separate the fields (CSV format)
- Allows quoted fields

Motivating Example – we will go from this



to this



Option 1

one big monolithic program

Motivating Example (Page 1/3)

csv.c

```
#include <stdio.h>
#include <string.h>

char buf[200];   /* input line buffer */
char* field[20]; /* fields */

char* unquote(char *p) {
    if (p[0] == '"') {
        if (p[strlen(p)-1] == '”')
            p[strlen(p)-1] = '\0';
        p++;
    }
    return(p);
}
```

Design Decision #1:
Global variables!
An input line up to 199 chars + '\n'
Up to 20 fields per line

Design Decision #2:
Remove the leading and trailing quotes – no embedded quotes

Motivating Example (Page 2/3)

CSV.C

```
int csvgetline( FILE *fin )
{
    int nfield;
    char *p, *q;

    if (fgets(buf, sizeof(buf), fin) == NULL)
        return(-1);

    nfield = 0;

    for (q=buf; (p = strtok(q, ",\n\r")) != NULL; q=NULL)
    field[nfield++] = unquote(p);

    return(nfield);
}
```

Design Decision #3:

- Use `strtok` instead of `scanf`.
- `strtok(p,s)` returns a pointer to the first token within `p` consisting of characters not in `s`.
- Overrides the original string / destroys input file. Input line not preserved.

Design Decision #4:

No data saved from one line to the next

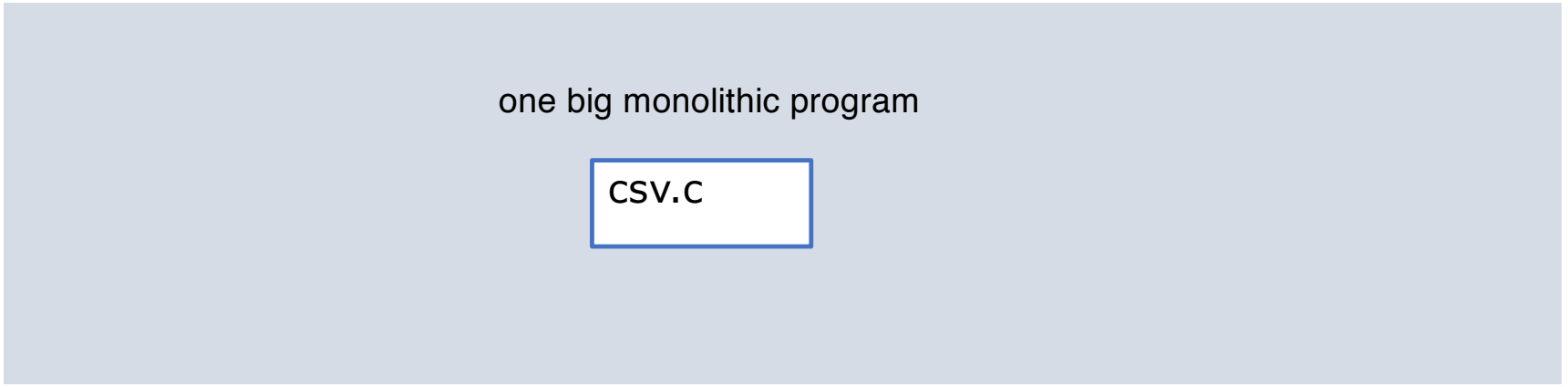
Motivating Example (Page 3/3)

CSV.C

```
int main( void ) {

    int i, nf;

    while ((nf = csvgetline(stdin)) != -1 )

        for( i=0; i<nf; i++)
        printf( "field[%d] = %s\n", i, field[i]);

    return(0);
}
```

Design Decision #5:
Test with stdin instead of a file

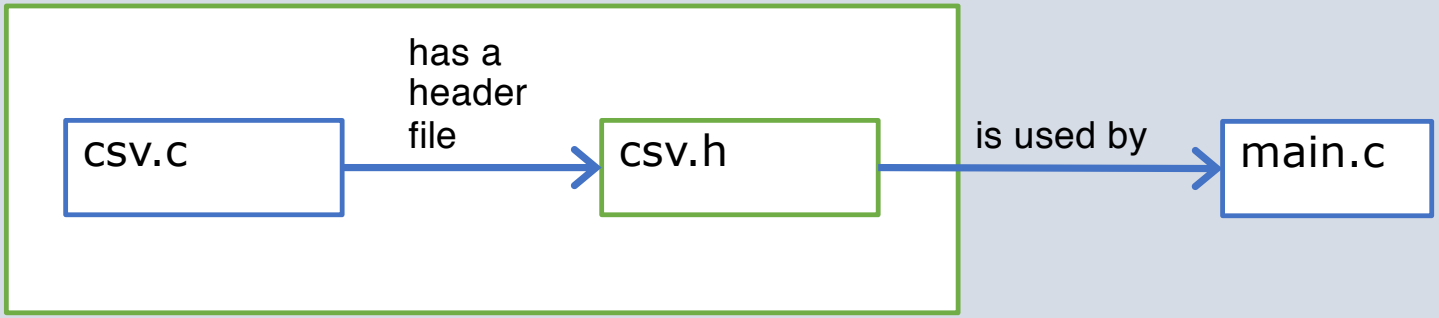
Option 1 – How to Compile




```
gcc csv.c –o csv
```

- With every change we compile the whole project
- All implementation details visible everywhere

Option2 – Using C Header Files



csv.c Module (Page 1/2)

CSV.C

```
#include <stdio.h>
#include <string.h>


char buf[200];   /* input line buffer */
char* field[20]; /* fields */



char* unquote(char *p) {

    if (p[0] == '"') {
        if (p[strlen(p)-1] == '”')
            p[strlen(p)-1] = '\0';
        p++;
    }
    return(p);
}
```

as before

csv.c Module (Page 2/2)

CSV.C

```
int csvgetline( FILE *fin )
{
    int nfield;
    char *p, *q;

    if (fgets(buf, sizeof(buf), fin) == NULL)
        return(-1);

    nfield = 0;


    for (q=buf; (p = strtok(q, ",\n\r")) != NULL; q=NULL)
    field[nfield++] = unquote(p);


    return(nfield);
}
```

as before

Header File

csv.h

```
extern char buf[]; /* input line buffer */
extern char *field[]; /* fields */



char* unquote(char *p);
int csvgetline( FILE *fin );
```

Define Global Variables as Exernal
No array bounds given
Careful here, can't use
`char **field`, must use `char *field[]`

Just Function Definitions
No bodies, no implementation

main.c

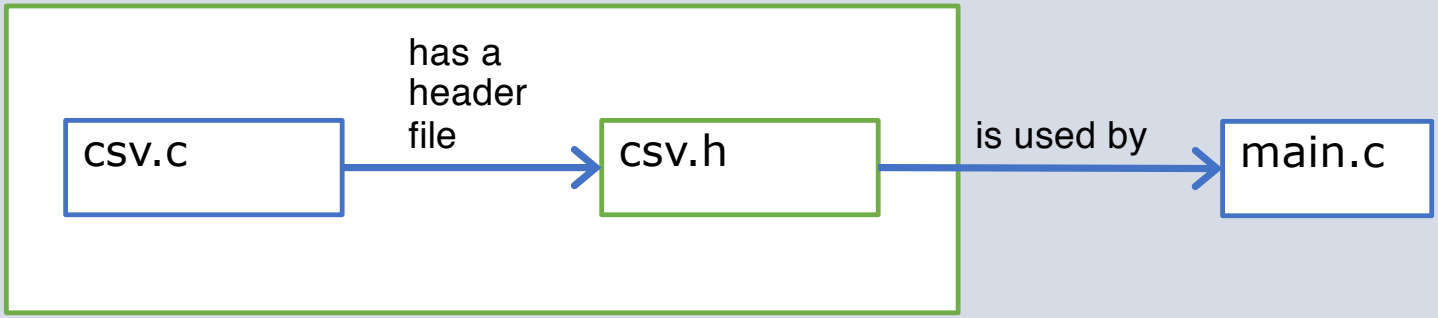
main.c

```
#include <stdio.h>
#include <string.h>
#include "csv.h"


int main( void ) {
    int i, nf;

    while ((nf = csvgetline(stdin)) != -1 )
        for( i=0; i<nf; i++)
        printf( "field[%d] = %s\n", i, field[i]);

    return(0);
}
```

How to use a custom header file
Note the quotes

Still uses global variables
Still a bad design

Option2 – Using C Header Files – How to compile



- If the implementation changes without the interface changing, no need to recompile the client

`gcc -c csv.c` (produces `csv.o`)

`gcc -c main.c` (produces `main.o`) **OPTIONAL**

`gcc *.o -o client` (links all object files into executable `client`)

Note

- The book (Chapter 4) shows the option of using C++ to implement an interface for this motivating example

Lessons

- Lesson 1: Interfaces are great
- Lesson 2: Interfaces are natural with OO languages (C++, Java,..)
- Lesson 3: Interfaces in C are possible via header files

Resources

- These notes
- K&R Book http://tinyurl.com/yaemm9vh (it covers an older version of C but it is a great way to learn the basics)