Complex Programs

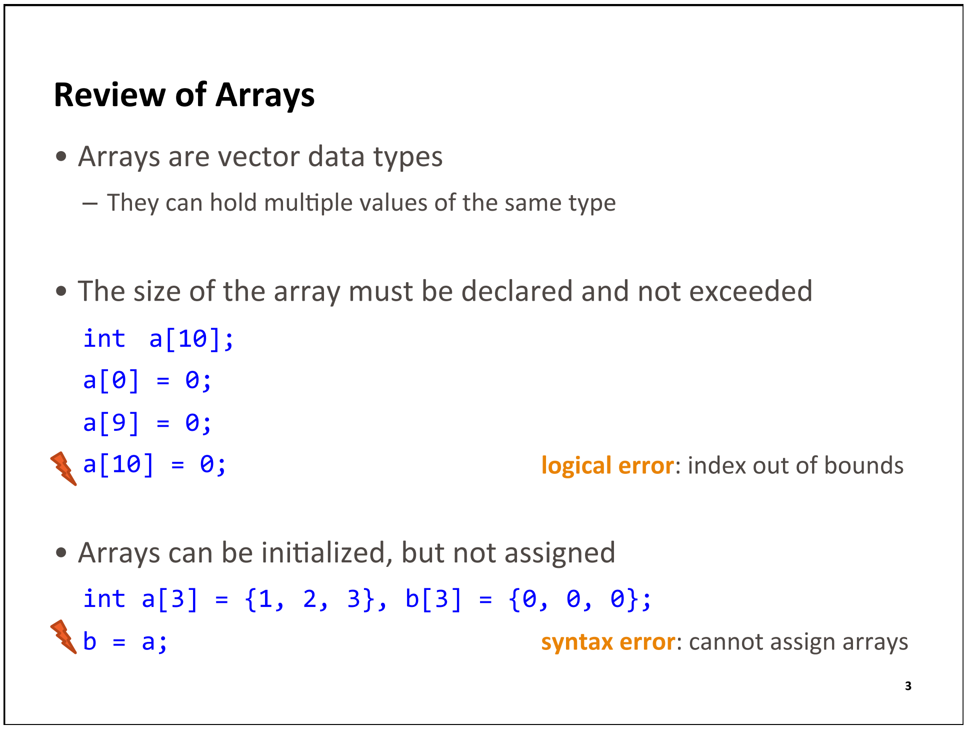
ENEE 140

Prof. Tudor Dumitraș
Assistant Professor, ECE
University of Maryland, College Park

http://ter.ps/enee140

Today's Lecture

- Where we've been
  - Scalar data types (`int, long, float, double, char`)
  - Basic control flow (`while` and `if`)
  - Functions
  - Random number generation
  - Arrays and strings
- Where we're going today
  - Structuring complex programs
  - Project 2
- Where we're going next
  - Control flow



Review of Arrays

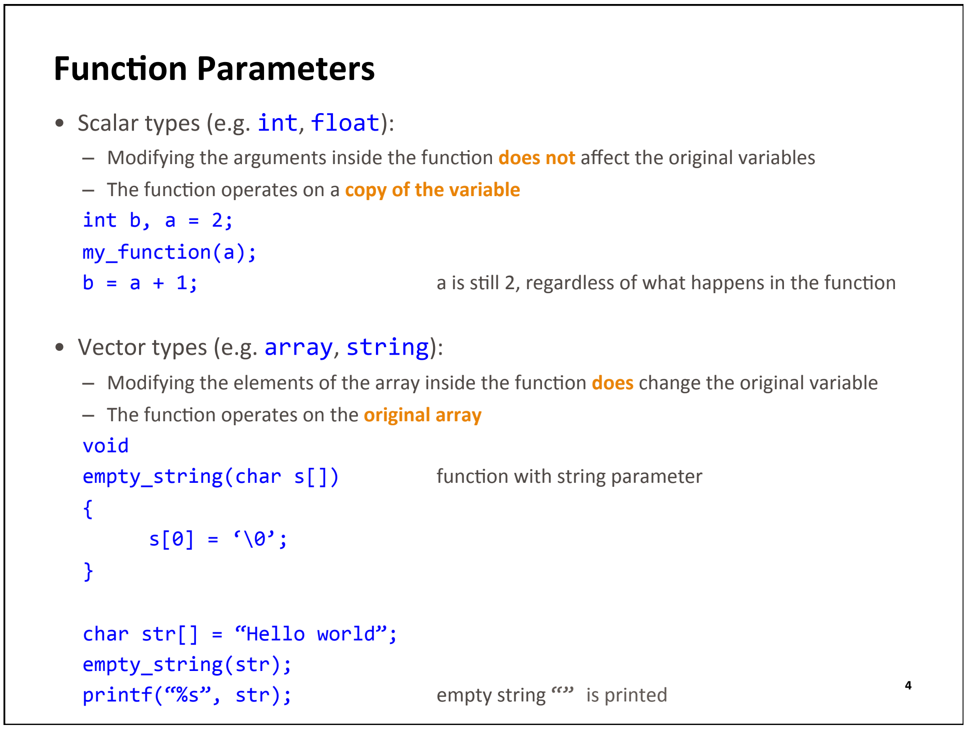
- Arrays are vector data types
  - They can hold multiple values of the same type

- The size of the array must be declared and not exceeded

```
  int  a[10];
  a[0] = 0;
  a[9] = 0;
⚡ a[10] = 0;                    logical error: index out of bounds
```

- Arrays can be initialized, but not assigned

```
  int a[3] = {1, 2, 3}, b[3] = {0, 0, 0};
⚡ b = a;                         syntax error: cannot assign arrays
```



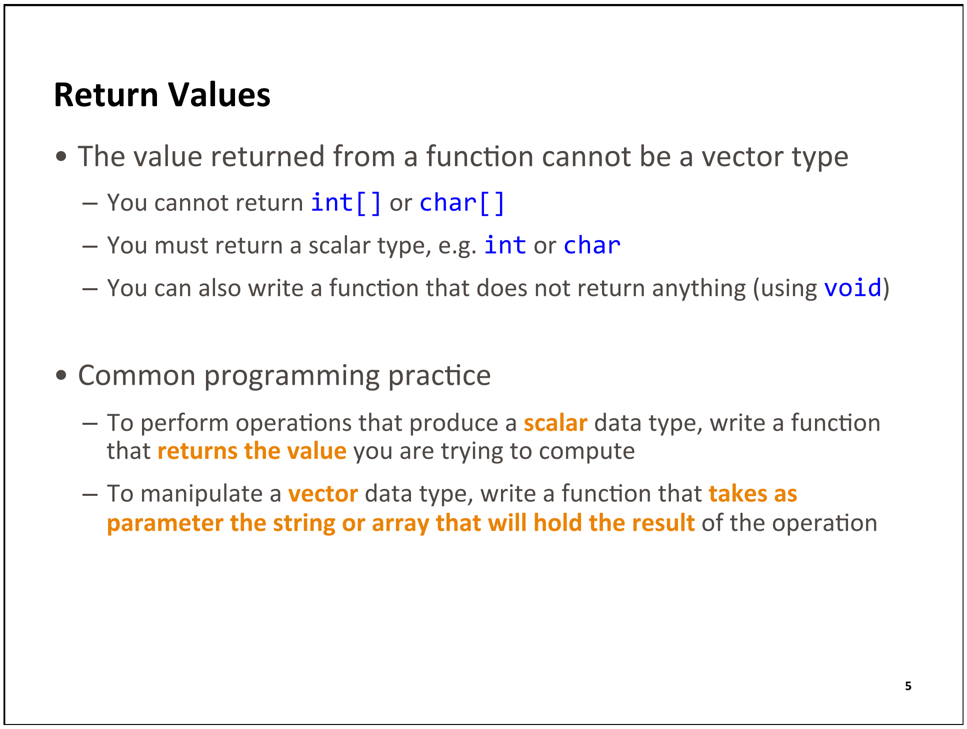
Function Parameters

- Scalar types (e.g. `int`, `float`):
  - Modifying the arguments inside the function **does not** affect the original variables
  - The function operates on a **copy of the variable**

```
int b, a = 2;
my_function(a);
b = a + 1;                     a is still 2, regardless of what happens in the function
```

- Vector types (e.g. `array`, `string`):
  - Modifying the elements of the array inside the function **does** change the original variable
  - The function operates on the **original array**

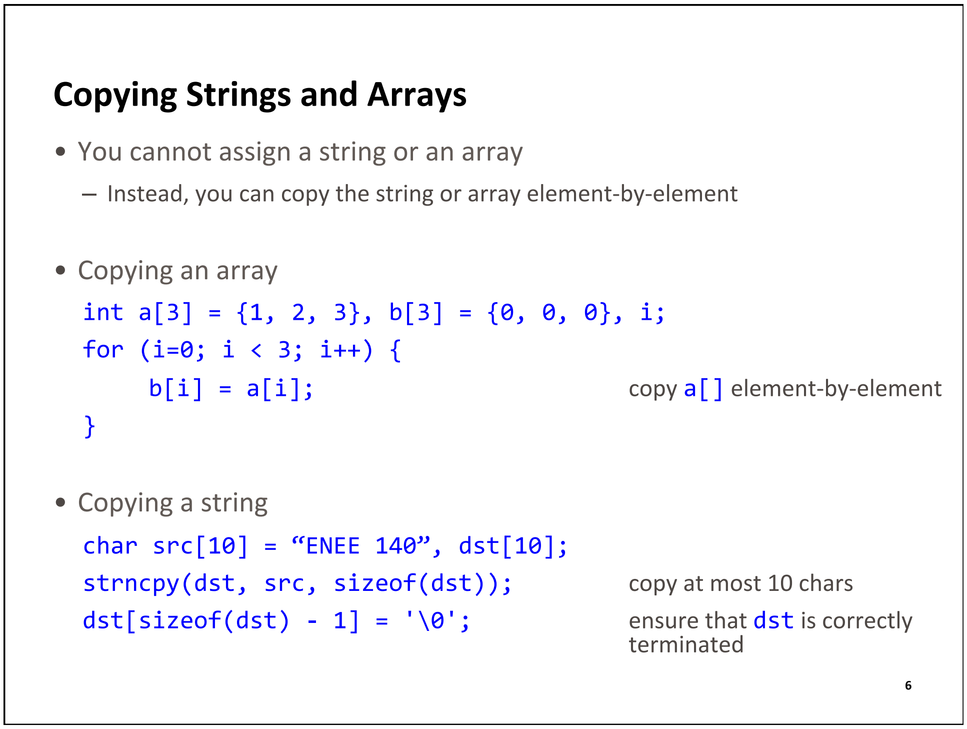
```
void
empty_string(char s[])         function with string parameter
{
      s[0] = ‘\0’;
}

char str[] = “Hello world”;
empty_string(str);
printf(“%s”, str);             empty string “”  is printed
```



Return Values

- The value returned from a function cannot be a vector type
  - You cannot return `int[]` or `char[]`
  - You must return a scalar type, e.g. `int` or `char`
  - You can also write a function that does not return anything (using `void`)

- Common programming practice
  - To perform operations that produce a **scalar** data type, write a function that **returns the value** you are trying to compute
  - To manipulate a **vector** data type, write a function that **takes as parameter the string or array that will hold the result** of the operation



Copying Strings and Arrays

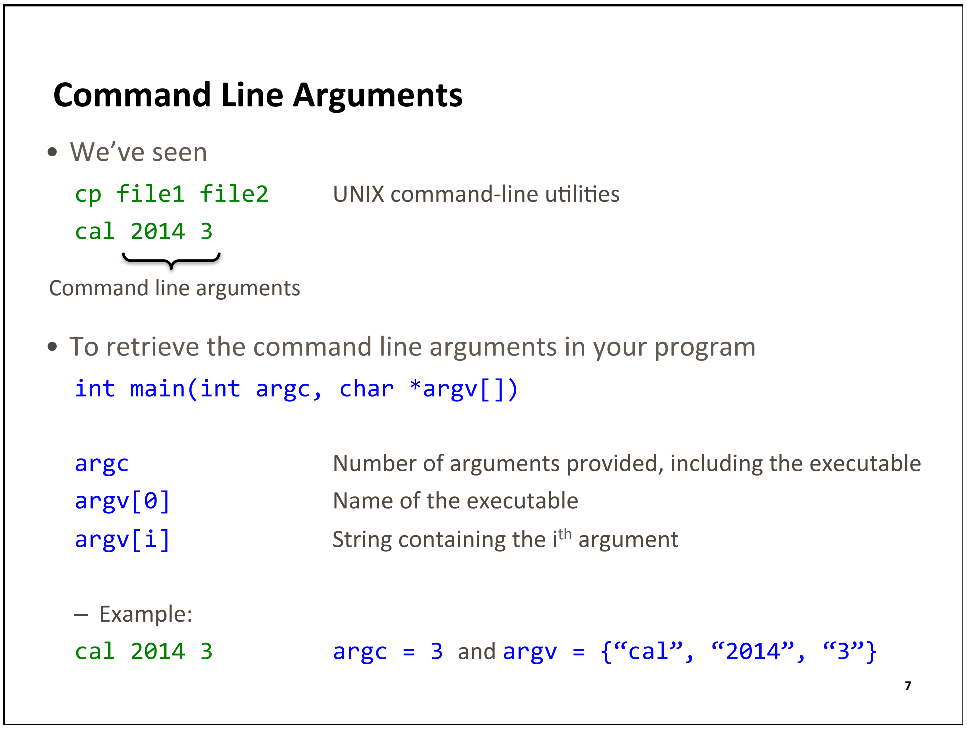
- You cannot assign a string or an array
  - Instead, you can copy the string or array element-by-element

- Copying an array

```
int a[3] = {1, 2, 3}, b[3] = {0, 0, 0}, i;
for (i=0; i < 3; i++) {
     b[i] = a[i];                         copy a[] element-by-element
}
```

- Copying a string

```
char src[10] = “ENEE 140”, dst[10];
strncpy(dst, src, sizeof(dst));           copy at most 10 chars
dst[sizeof(dst) - 1] = '\0';              ensure that dst is correctly
                                          terminated
```



Command Line Arguments

- We've seen

```
cp file1 file2        UNIX command-line utilities
cal 2014 3
```

$\underbrace{\texttt{2014 3}}_{\text{Command line arguments}}$

- To retrieve the command line arguments in your program

```
int main(int argc, char *argv[])
```

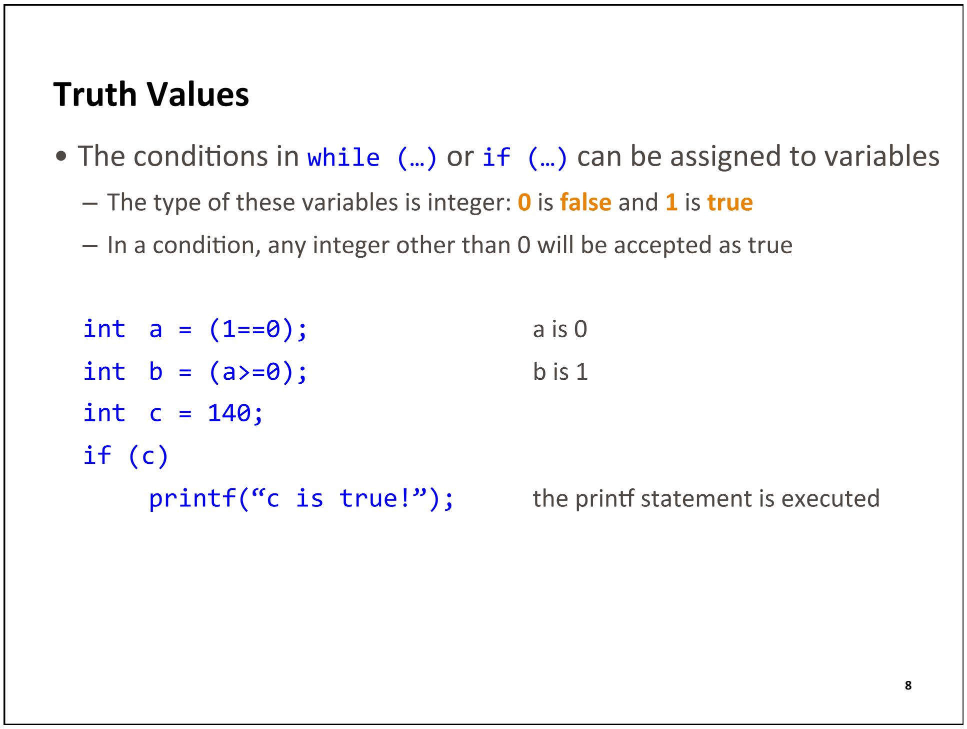
`argc` Number of arguments provided, including the executable

`argv[0]` Name of the executable

`argv[i]` String containing the ith argument

  – Example:

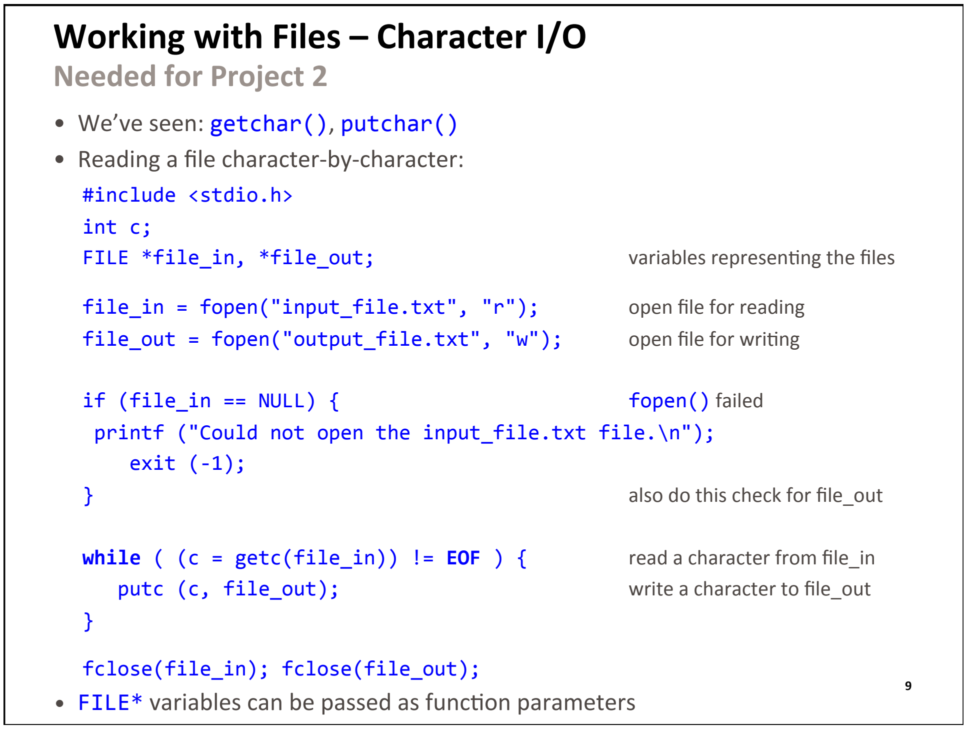
`cal 2014 3` `argc = 3` and `argv = {“cal”, “2014”, “3”}`



Truth Values

- The conditions in `while (…)` or `if (…)` can be assigned to variables
  - The type of these variables is integer: **0** is **false** and **1** is **true**
  - In a condition, any integer other than 0 will be accepted as true

```
int  a = (1==0);              a is 0
int  b = (a>=0);              b is 1
int  c = 140;
if (c)
     printf(“c is true!”);    the printf statement is executed
```



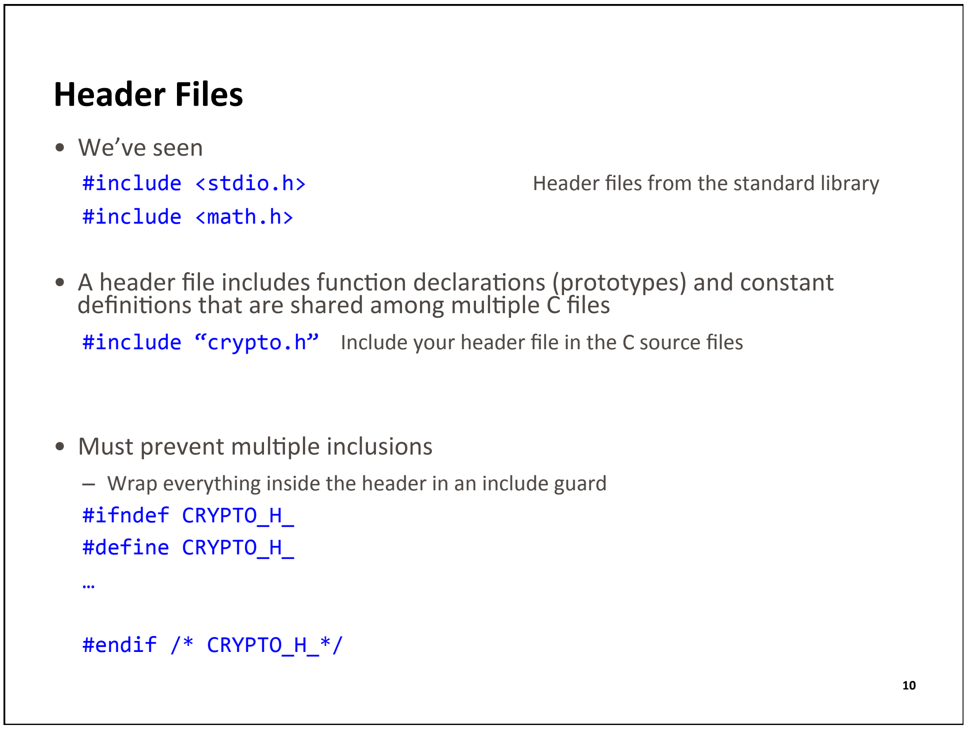
Working with Files – Character I/O

Needed for Project 2

- We’ve seen: getchar(), putchar()
- Reading a file character-by-character:

```
#include <stdio.h>
int c;
FILE *file_in, *file_out;                       variables representing the files

file_in = fopen("input_file.txt", "r");         open file for reading
file_out = fopen("output_file.txt", "w");       open file for writing

if (file_in == NULL) {                          fopen() failed
 printf ("Could not open the input_file.txt file.\n");
    exit (-1);
}                                               also do this check for file_out

while ( (c = getc(file_in)) != EOF ) {          read a character from file_in
   putc (c, file_out);                          write a character to file_out
}

fclose(file_in); fclose(file_out);
```

- FILE* variables can be passed as function parameters



Header Files

- We’ve seen

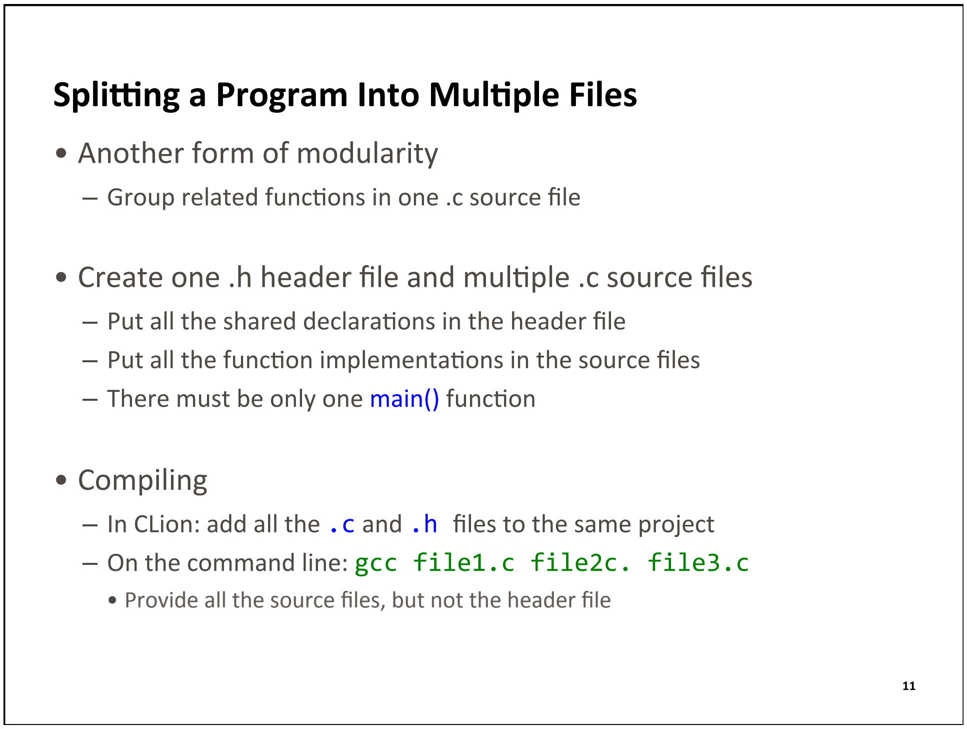
```
#include <stdio.h>                   Header files from the standard library
#include <math.h>
```

- A header file includes function declarations (prototypes) and constant definitions that are shared among multiple C files

```
#include “crypto.h”    Include your header file in the C source files
```

- Must prevent multiple inclusions
  - Wrap everything inside the header in an include guard

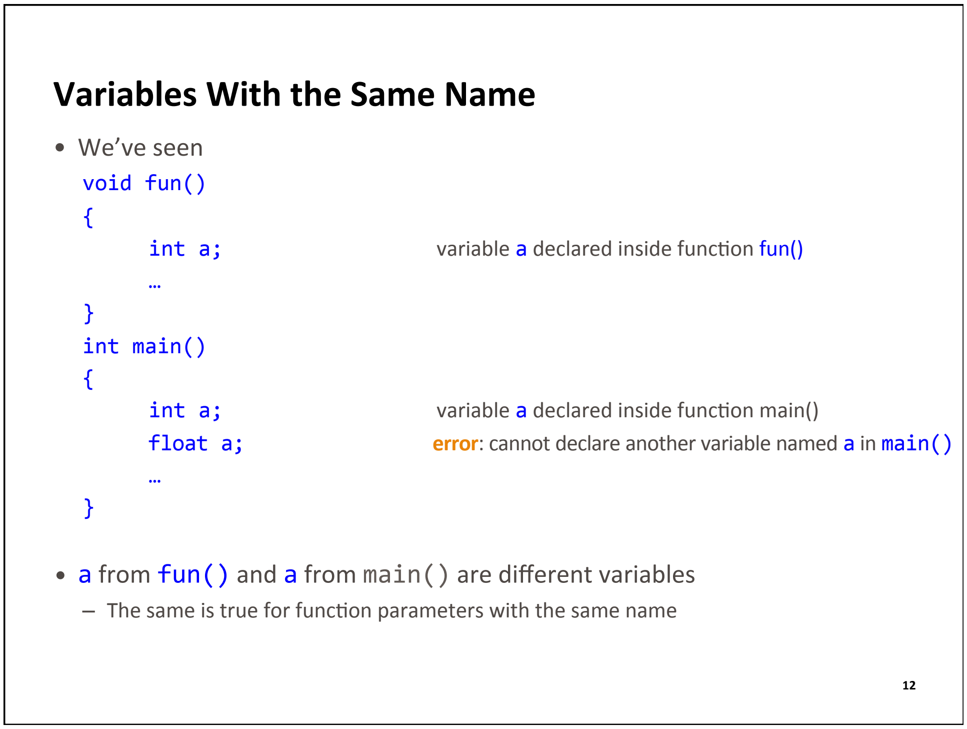
```
#ifndef CRYPTO_H_
#define CRYPTO_H_
…

#endif /* CRYPTO_H_*/
```



Splitting a Program Into Multiple Files

- Another form of modularity
  - Group related functions in one .c source file
- Create one .h header file and multiple .c source files
  - Put all the shared declarations in the header file
  - Put all the function implementations in the source files
  - There must be only one main() function
- Compiling
  - In CLion: add all the `.c` and `.h` files to the same project
  - On the command line: `gcc file1.c file2c. file3.c`
    - Provide all the source files, but not the header file

11



Variables With the Same Name

- We’ve seen

```
void fun()
{
      int a;                  variable a declared inside function fun()
      …
}
int main()
{
      int a;                  variable a declared inside function main()
      float a;                error: cannot declare another variable named a in main()
      …
}
```

- a from `fun()` and a from `main()` are different variables
  - The same is true for function parameters with the same name

12

Variable Scope

- Variable scope (where is the variable visible)
  - Inside the block where it is declared
    - A block is enclosed in `{ }`
  - Can also declare variables at the start of `if`, `while`, `for`, etc. blocks

```
while (condition) {
     int a = 1;            variable a visible only inside while loop
     …
}
```

Global Variables

- Variables declared outside any function

```
int a;                     global variable
int main()
{
     …
}
```

- Global variable scope
  - Globally accessible in all the files compiled and linked together

Static Variables Declared Outside Any Function

- Declared using keyword `static`

```
static int a;           variable local to current .c file
int main()
{
        …
}
```

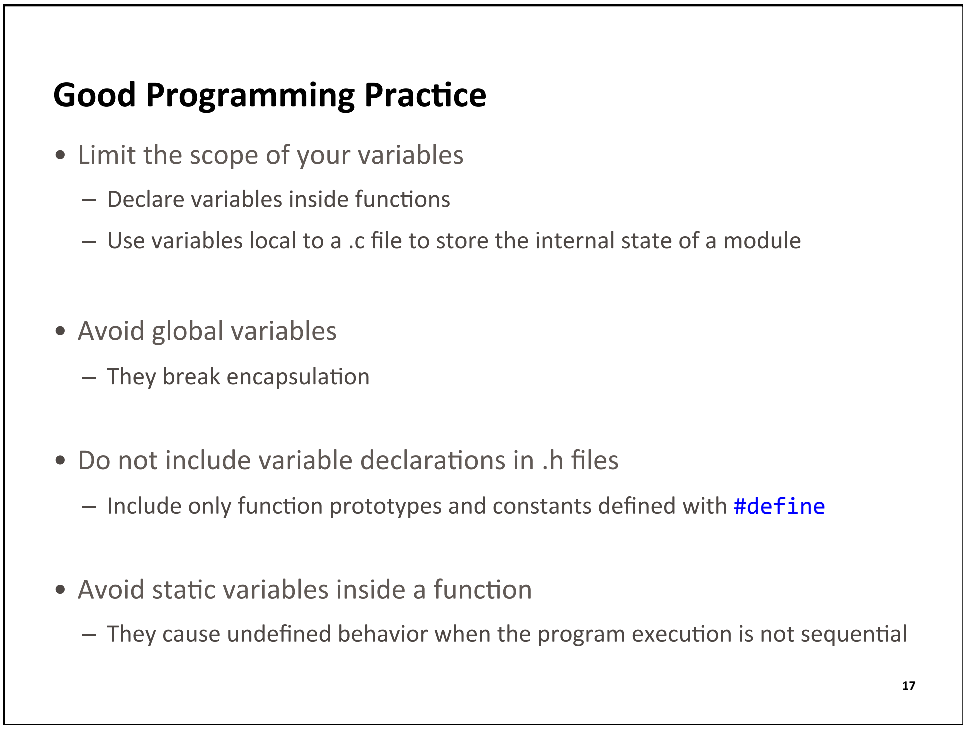
- Variable scope
  - Visible only inside the .c file where they are declared
  - Can be used to hold the internal state of a library

Static Variable Declared Inside A Function

- Initialized only the first time when the block is executed

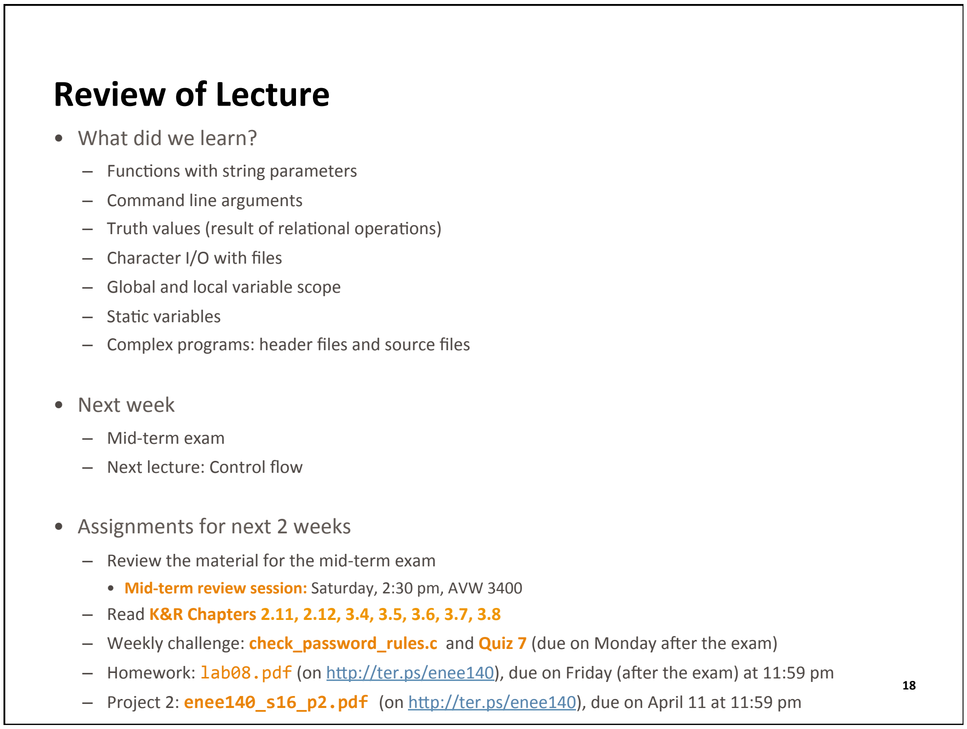
```
void fun()
{
        static int count_invocations = 0;   static variable
        count_invocations++;
        …
}
```

- Static variables preserve their value across function invocations
  - Same as global variables
- Variable scope
  - Visible only inside the function where they are declared



Good Programming Practice

- Limit the scope of your variables
  - Declare variables inside functions
  - Use variables local to a .c file to store the internal state of a module
- Avoid global variables
  - They break encapsulation
- Do not include variable declarations in .h files
  - Include only function prototypes and constants defined with `#define`
- Avoid static variables inside a function
  - They cause undefined behavior when the program execution is not sequential



Review of Lecture

- What did we learn?
  - Functions with string parameters
  - Command line arguments
  - Truth values (result of relational operations)
  - Character I/O with files
  - Global and local variable scope
  - Static variables
  - Complex programs: header files and source files
- Next week
  - Mid-term exam
  - Next lecture: Control flow
- Assignments for next 2 weeks
  - Review the material for the mid-term exam
    - **Mid-term review session:** Saturday, 2:30 pm, AVW 3400
  - Read **K&R Chapters 2.11, 2.12, 3.4, 3.5, 3.6, 3.7, 3.8**
  - Weekly challenge: **check_password_rules.c** and **Quiz 7** (due on Monday after the exam)
  - Homework: `lab08.pdf` (on http://ter.ps/enee140), due on Friday (after the exam) at 11:59 pm
  - Project 2: **`enee140_s16_p2.pdf`** (on http://ter.ps/enee140), due on April 11 at 11:59 pm