Functions
ENEE 140

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

http://ter.ps/enee140

Today’s Lecture
• Where we’ve been
  – Variables and constants
  – Variable assignment and operators
  – int, float and char
  – Iterating (while, for) and branching (if)

• Where we’re going today
  – Functions

• Where we’re going next
  – Integer and floating point arithmetic
Formed Input and Output

• We’ve seen:
  ```c
  int a = 0;
  printf(“The value of a is %d\n”, a);
  ```

• You can output data with printf and read data from the input with scanf

• printf format specifiers
  – %d: int
  – %ld: long
  – %f: float, double
  – %E: float, double in scientific notation, e.g. 1.5E3
  – %c: character
  – %: the ‘%’ character
  – See Table 7.1 in K&R for a complete specification
    • Or type man 3 printf on the command line

• Read data from the input
  ```c
  int a, b;
  scanf(“%d %d”, &a, &b);
  ```

Prompting the User for Input

• Print a message indicating the input expected
• Then read the input

  ```c
  int sec;
  float gpa;

  printf(“Enter your section number: ”);
  scanf(“%d”, &sec);

  printf(“Enter your GPA: ”);
  scanf(“%f”, &gpa);
  ```
Functions

- Functions allow you to encapsulate computation
  - You don’t care how a job is done; you know what is done

- Examples of functions we’ve seen so far
  
  ```
  printf("The value of a is %d\n", a);
  c = getchar();
  ```

- You can use these functions in your programs without knowing how they are implemented

- You can also define your own functions
  - Example:
  ```
  int main() {
      ... return 0;
  }
  ```

Functions

- Function declaration (prototype)
  ```
  int square(int param);
  ```

- Function definition (implementation)
  ```
  int square(int param) {
      int result; result = param * param;
      return result;
  }
  ```

- Function invocation (calling the function in your program)
  ```
  int a = 1+square(2)+square(3);
  ```

- You must declare or define a function before you invoke it
Function Parameters and Local Variables

- Function **parameters** (arguments)
  - Parameters must have types (e.g. `int`, `float`) and are specified in the function declaration and definition:
    
    ```
    int pow(int x, int y);
    ```
    - the function takes 2 int parameters
  - When you call a function, you must pass as many parameters as in the prototype
    
    ```
    z = pow(2, 3);
    ```
    - the types must match as well
  - Modifying the arguments inside the function **does not** affect the original variables
  - The function operates on a copy of the variable
    
    ```
    int a = 2;
    my_function(a);
    ```
    - a is still 2, regardless of what happens in the function

- **Variables local to the function**
  - You can declare variables inside the function, like you do in `main()`

- Parameters are local variables **cannot be accessed outside the function**

---

Return Values

- The type return value is specified in the prototype, before the name of the function
  
  ```
  int pow(int x, int y);
  ```
  - the function returns an int

- It is also possible to write a function that does not return anything
  
  ```
  void err_msg(int code)
  {
    printf(“Encountered an error with code %d\n”, code);
  }
  ```
  - return statement is not needed
Modularity

- Functions allow you to break down your program’s functionality into smaller pieces

- Programs that are made up of many small functions are called **modular**
  - In such programs it’s easy to modify one function, without affecting how the rest of the program works
  - Modular code is also easier to read

- Modular programs are the result of **top-down problem solving**
  - Break down the problem you need to solve into smaller sub-problems
  - For each sub-problem, write the prototype of a function that would solve it
  - Write your program by invoking these functions, assuming that they are implemented
  - Then figure out how to implement each function

Modularity – cont’d

- Example of top-down problem solving
  - You are asked to write a program that prints a Celsius-Fahrenheit conversion table
  - Imagine that you have a function, which takes a float argument representing the temperature in Fahrenheit degrees, and returns a float with the corresponding Celsius value
  - Write the loop that prints the conversion table
  - Then look up the conversion formula and implement the function

- Helper functions
  - In your assignments, you will often be asked to implement functions that provide a certain functionality
  - It is often a good idea to write additional **helper** functions that you use in your program
  - For example, such helper functions may provide functionality that is useful for several tasks
Mathematical Functions Available in C

- These functions typically accept and return variables of type double
  
  ```c
  #include <math.h>  // must include this header to use the math functions (more on this later)
  
  sin(x);  // sine of x (in radians)
  cos(x);  // cosine of x (in radians)
  exp(x);  // e^x
  log(x);  // natural logarithm of x
  log10(x);  // base 10 logarithm of x
  sqrt(x);  // square root of x
  ...
  ```

Aside: Manual Pages

- You can get help on most functions from the C standard library using the man command on the GRACE machines
  
  ```bash
  man printf  // manual page of printf() function
  man scanf  // manual page of scanf() function
  ```
Review of Lecture

• What did we learn?
  – Declaring a function (i.e. deciding the parameter list and return type)
  – Defining a function (i.e. implementing the function)
  – Invoking a function (i.e. using the function in your code)
  – `printf`, `scanf` and math functions
  – Modulearity
  – Debugging

• Next lecture
  – Integer and floating point arithmetic

• Assignments for this week
  – Read K&R Chapters 2.5, 2.7, 2.8, 2.10, B2, B11
    • Note: some of these chapters refer to strings (e.g. `char s[]`), which we’ll cover later
    • For now, think of `s[1]` as a character variable
    • Read man pages for `rand()` and `srand()`; try to understand the implementations on page 46
  – Weekly challenge: `read_divide_ints.c`
  – Homework: `enee140_lab04.pdf`, due on Friday at 11:59 pm
  – Quiz 4, due on Monday at 11:59 pm