Data Structures in C

Prof. Georg Feil

Functions

Summer 2018

Acknowledgement

- These lecture slides are based on slides and other material by Professor Magdin Stoica
- Additional sources are cited separately

Reading Assignments

Required

- C for Programmers (supplementary textbook)
  - Sections 5.4 - 5.8: Functions

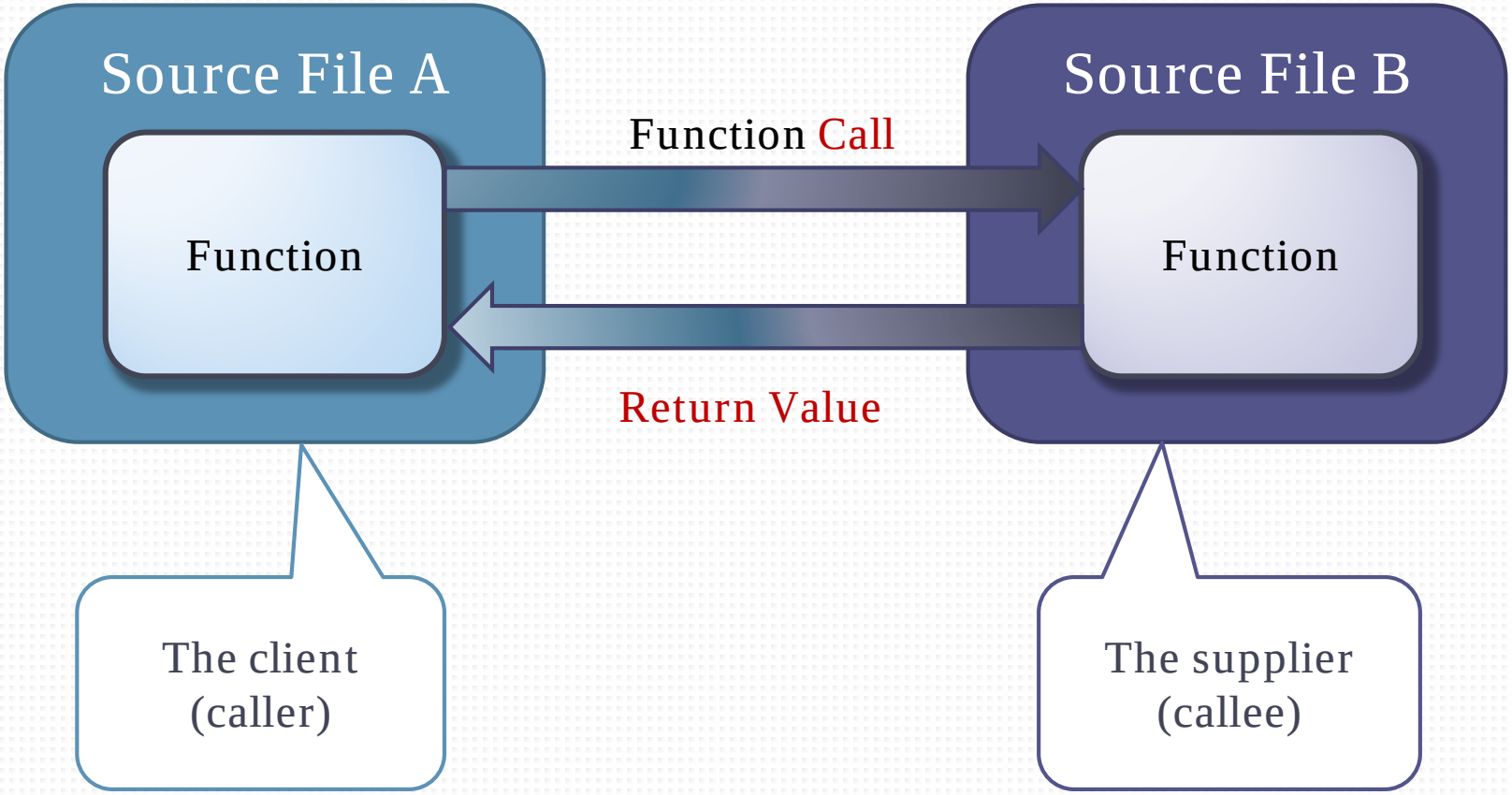
Functions

- Functions in software were inspired by the mathematical idea of a function, for example $y = f(x)$
- A function in C has 4 main properties:
  - Name
  - Return data type
  - Parameters (names and data types)
  - Scope
- A function in C is a lot like a Java method, but not declared inside a class

What is a function?

- A function is a named block of statements
  - The statements have a common purpose, solving a unique problem
  - The statements may work with global/module variables declared at the top of the file
- The statements inside the function are executed when the function is called or invoked
  - The program "remembers" where the call came from and returns to the statement right after the call when the function completes
- A function has two parts
  - Declaration or Prototype: Introduces the function and provides the function's name, return data type, and each parameter's name along with its data type
  - Definition: Defines the function statement block, the collection of statements that execute when the function is called

Concept: Clients and Suppliers



Example A: Function declaration and definition

(Does not return any value, does not take any parameters)

Notice 'void' here!

```
static double radius = 5;

void printArea(void)
{
   printf("%f\n", radius * radius * 3.14);
}
```

Function declaration

Function Definition

Function Properties

- A function has a name, return type, list of parameters, and scope
- Name identifies the function (same naming rules as variables)
- Return type
  - The data type of the information returned by the function (e.g. int, double, char, etc.)
  - Used when the function calculates or generates a value
  - If the function does not return a value, its return type is defined as void
- List of parameters declares the input variables needed by the function
  - In C you can also do output parameters using pointers or references
  - Recall scanf had variables with '&' in front to get data *from* the function
- Scope is global by default. If you add the static keyword before the return type then the scope of the function is that .c file (module) only.

Function Parameters

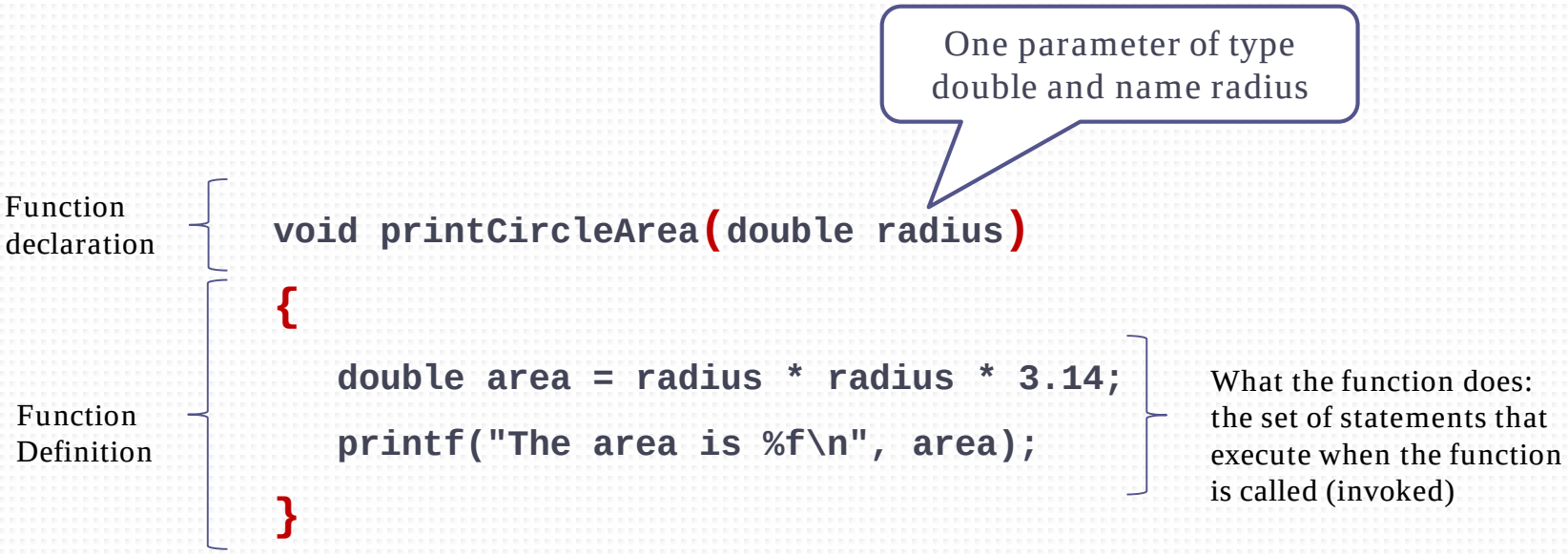
- Function parameters are usually input variables whose values are provided by the caller
- The list of function parameters is enclosed in (parentheses)
- A parameter is declared like a local variable
- When multiple parameters are needed they are separated using a comma
- It is common for functions to *not* need any parameters
  - The function declaration should use (void)
  - This is different from Java and C++, where you just use ()
  - Empty parentheses are needed when calling, e.g. printArea()
- Warning: Declaring parameters as () will work, means "undefined"
  - Not good C programming style, use -Wstrict-prototypes to catch this

If I see ( ) it must be a function

Parameters or not, function calls ALWAYS use parentheses

( )

Example B: Function with one parameter



Example C: Function with two parameters

Two parameters. They have the type double and are named length and width. Divided using a comma.

Function declaration

```
void printRectArea(double length, double width)
```

Function Definition

```
{
    double area = length * width;
    printf("The area is %f\n", area);
}
```

What the function does: the set of statements that execute when the function is called (invoked)

Calling (Invoking) a function

- A function is called using a function call statement which specifies
  - The name of the function to be called
  - The list of values to be passed as parameters. Values are expressions which can be made of variables, literals or combinations of variables and literals using operators.
    - The list of values is enclosed in parentheses
    - If there is more than one parameter the values are separated by commas
- If the function returns a value, the function call can be used as a value in an expression
  - The value returned may be stored in a variable, used in a calculation, printed out, passed as a parameter etc.
  - If the caller is not interested in the value they are free to ignore it

Function call examples

```
setRadius(20);
printSum(3,5);
drawRectangle(2,6,num, 16);
setFirstName("Barack");

double area = calculateArea();
double circleDiameter = getRadius() * 2;
printf(getName());
printf("%s", getName());
```

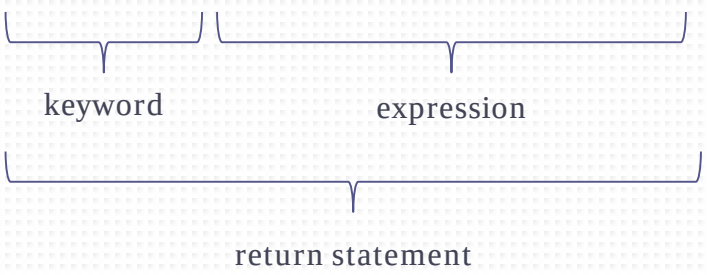
Functions that perform an action but do not return a value

Functions that return a value

The return statement (keyword)

- How does a function return a value it calculated to the caller?
  - Using the return statement (like Java)
- In C you can also use parameters as output (return) values
  - You can return a value by passing a pointer or reference (unlike Java)
  - We'll learn about pointers and references soon
  - For now we'll use parameters only as inputs to a function, except scanf

```
int cube(int num)
{
    return num * num * num;
}
```



Example D: Function with two parameters that returns a value

```
double calculateRectArea(double length, double width)
{
   double area = length * width;
   return area;
}
```

Example Program with more than one function

- The next two slides show an example program that has more than one function
- It also demonstrates if-else and a while loop with 'break'

Example Prog: “The size affects the bark” (pg 1)

```
#include <stdio.h>
#include <stdbool.h>   // So we can use 'true'

// Prints the sound a dog might make.
// 'dogNum' is the dog number, 'dogWeight' is weight in pounds
void bark(int dogNum, float dogWeight)
{
    if (dogWeight < 10) {
        printf("Dog #%d says Yip! Yip!\n", dogNum);
    }
    else if (dogWeight < 50) {
        printf("Dog #%d says Ruff! Ruff!\n", dogNum);
    }
    else {
        printf("Dog #%d says Woof! Woof!\n", dogNum);
    }
}

// (Continued on next page)
```

Example Prog: “The size affects the bark” (pg 2)

```
int main(int argc, char** argv)
{
    float weight = 0;
    int count = 0;

    while (true)   // Loop until 'break'
    {
        // Prompt for input and read the weight in pounds
        printf("Enter the weight of dog %d: ", ++count);
        scanf("%f", &weight);
        if (weight == 0)
            break;  // Quit if weight is 0

        bark(count, weight);
        printf("\n");          // Leave a blank line
    }

    return 0;
}
```

Exercise 1: “The size affects the bark”

- Copy / paste the example program into Dev-C++
  - Copy both pages, one after the other, into one source file
- Compile and try running the program with different input
  - Do while loops, break, and if-else seem to work like Java?
  - What happens if you enter zero?
  - What happens if you enter something that’s not a number, for example enter your name? (explain the results)
    - Also try the “bad” input as the *second* input value
  - Fix the program so it can handle bad input [Hint: to “get rid of” bad input sitting in the input buffer use `fflush(stdin)`]

Function Prototypes and Header Files

The order of function declarations matters in C

Function Prototypes and Header Files

- To call a function without getting warnings or errors the compiler must "see" the function declaration before it encounters any calls to the function
- Prototype for function with no parameters and no return value:

```
void printArea(void);
```

- We put the prototype (declaration with semicolon) in a header file, then #include the header file in source files which call the function
  - Header file names end with .h
  - Don't put implementations in the header file, only declarations (1st line)
- You can also put the prototype right in the .c file, usually near the top, if you need to call a function that appears later in the file
  - Or put the entire function being called before the function that calls it
  - In all cases the compiler must "see" the function declaration before the call

Including Header Files

- We've already seen how to include "system" header files

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

- The angle brackets < > tell the compiler to look in system folders where the C library header files are located
- When including *your own* header files you should use double quotes around the header file name or you may get errors

```
#include "secret_code.h"
```

- This looks for header files in the source file folder
  - Put your header files in the same folder as the source file

Header Files – possible problems

- Using #include incorrectly can cause major problems
- If you put function prototypes in a header file then the source file containing those functions should include the header file
  - Allows the compiler to check that the function declarations match
  - If you don't do this, there is no compile-time check and unusual runtime errors may occur
- Do not include .c files!
  - Leads to incorrect program structure, linker errors
- You can include header files from other header files
  - But do this only when needed to compile the header file properly

Exercise 2: Prototypes and a Header File

Starting with your program from Exercise 1:

- Try moving the bark() function after the main() function and compile the program
  - Do you get errors or warnings?
  - Note: This depends on the C compiler and options!

Can cause bad execution! Try removing -pedantic-errors

- Add the prototype for the bark() function (function declaration with semi-colon) near the start of the .c file
  - Do the errors/warnings go away?
  - Does the program work properly?
- Remove the bark() prototype at the start of the .c file and move it to a header file, then #include the header
  - Ensure the program compiles & runs properly

Exercise 3: Guess the Function Declaration

- Suppose you see some code that calls a function like this:

```
int count = 3;
double result = calcAve(73.9, 85.1, 20.0, count);
```

- You haven't seen the function declaration. By examining how the function is called, can you guess what the function declaration looks like?
  - In fact you can't be 100% certain, however it should be possible to say what the function declaration *probably* looks like
- Write down your "guess"!

Summary: Functions

- A C source file can contain data in the form of variables, and functionality in the form of functions
- A function is similar to a Java method and has 4 main properties: return data type, name, parameters, scope
- Use static to make the function visible in the current source file only
- To call a function you write its name followed by parentheses containing values for the parameters (if any)
- If a function returns a value, the call can be used in an expression
- A function prototype is its declaration with a semicolon at the end
  - Put the prototype in a header file (.h) to #include it from other modules
  - The header file should have the same name as the .c file
  - Put the header file in the same folder as the .c file
- The order of function declarations matters in C... the compiler must see a function's prototype before the first call to it