Data Structures in C

Prof. Georg Feil

Modular Programming in C

Summer 2018

Acknowledgement

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

Reading Assignments

- Our supplementary textbook "C For Programmers" considers a *function* to be a module
  - That's one way to define it
- In this course we'll think of a *source file (.c)* as a module
- … see this link:

http://www.embedded.com/design/prototyping-and-development/4023876/Modular-Programming-in-C

What are C programs made of?

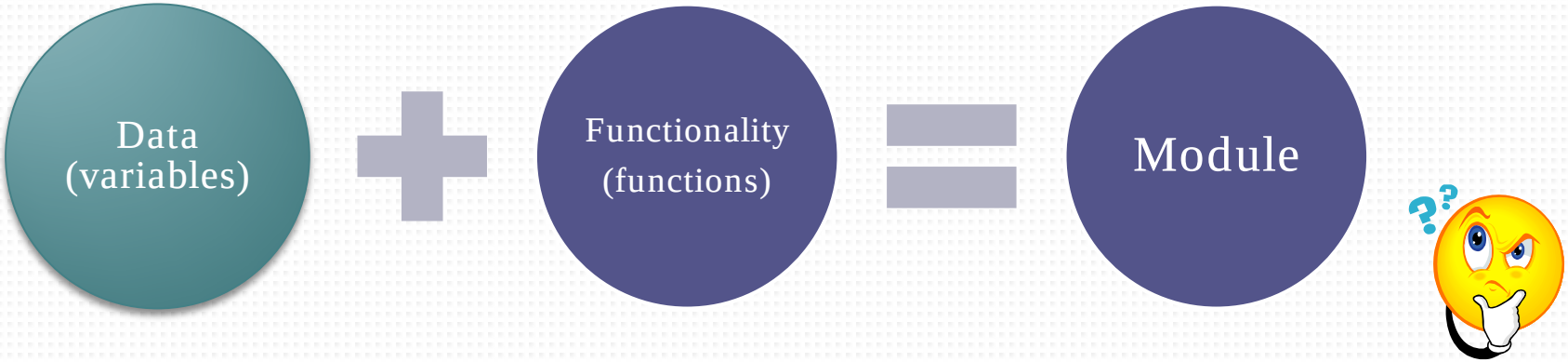
Modules

What defines a module?

A single source file (.c) and its associated header file (.h)

What's in a C module?

- Information or data (what the module stores or remembers) and functionality (what the module is able to do)
- The parts of a C module (source file) that contain information (data) are its variables.
- The parts of a C module that execute commands to implement functionality are its functions.



Modules lead to better programming

- A program that's properly divided into modules (*and* uses correct variable/function scope) is easier to
  - Understand
  - Develop
  - Maintain
  - Debug
- Makes it easier to reuse code, reduces duplication (DRY)
- Think of a single C source file (.c extension) as one module
  - The equivalent of a module in object-oriented programming is a class
  - The C static keyword allows us to encapsulate (hide) information in a module, similar to private fields & methods in Java
  - The associated header file (.h extension) can make parts of the module accessible to other modules, similar to public fields & methods in Java

Variable Scope

- Variables contain information of a specific type
- The scope of a variable can be
  - Global: Accessible from all .c files in the program
  - Module: Accessible within one .c file only, declared with static keyword
  - Local: Accessible only within a function scope
- You should use the most *restrictive* scope that is suitable for a variable
  - If a variable is only needed in one function, make it local
  - If a value can be passed as a parameter or returned as the return value of a function, then use a parameter / return value
  - If a variable is only needed in one source file (.c), make it a module variable using static

Function Scope

- Functions implement the functionality of a C program
- The scope of a function can be
  - Global: Callable from all .c files in the program
  - Module: Callable within its own .c file only, declared with static keyword
- You should use the most *restrictive* scope that is suitable for a function
  - If a function only needs to be called from one source file (.c), give it “module” scope using static

A module should include its own .h file

- Each module (.c file) should have a header (.h file) of the same name, unless the .c file is a completely standalone program
  - The header file should contain prototypes of publicly accessible functions
- Any other .c file can #include the header file
- Modules (.c files) should always include their own header file
  - This ensures that the declarations in the .h file and the .c file match (the compiler will give an error if not)
  - Eliminates problems calling functions that appear later in the .c file
- To write a "private" function that's not visible (callable) from other .c files, add the static keyword before the return type and *don't put the prototype in the header file*
  - Example: `static void printArea(void) {…}`

Never #include .c files

Only #include .h files

Exercise 1: Dividing a program into 2 modules

- When writing a program with more than one .c file in Dev-C++ you must create a project
  - File > New > Project...
  - Choose Empty Project and select C (not C++)
  - After the project is created you can create new source files or add existing files using the Project menu, or right-click on the project
- Add your program from Exercise 1 in the Functions slides ("The size affects the bark") to your project
  - Make sure all files are saved in the same folder
- Divide the program into two .c files
  - The main() function goes into a new .c file, bark() stays in dog.c
- Make changes needed so the program works!

Modular C Programming

- We'll now continue to learn about the basic elements of data structures and C programming
- We'll touch on key points about modular programming along the way
- This will allow you to apply some of your OOP knowledge to C!