Functions

William Hendrix

Lecture 10

Outline

- Function syntax
- Variable scoping
- The call stack
- Recursion

Functions

- Block of code used to accomplish a single purpose
  - E.g., calculating a value, outputting a message, reading input
  - Often return a value such as a calculated value or error indicator
  - Imagine a black box that takes in some input and transforms it into output
  - Can be called multiple times during a program
    - If you find yourself copy+pasting code, consider turning it into a function
- Code with shorter, modular functions can be easier to debug
- Advantage over copy+paste: only need to make changes in one place
- Three aspects to using functions
  - Definition
  - Implementation
  - Invocation (calling functions)

Defining functions

- Syntax: `<type> <func_name>(<param_list>);`
- Should be declared in preamble (other than `main`)
  - Definition not strictly necessary if implementation appears before `main`, but very good practice
- Type represents the value that the function computes
  - Can be any variable type or `void` (no value)
    - **Exception:** cannot return a static array (e.g., `int[]`)
  - `void` functions are used for repetitive tasks like input or output
- Name follows same rules as variables
  - **Good practice:** name should correspond with purpose
  - **Convention:** all lowercase, underscores or caps for multiple words
- Parameter list: comma separated list of inputs
  - Each parameter should list type and name, like a declaration

Implementing functions

- Syntax: very similar to declaration

```
<type> <name>(<param_list>)
{
    //Function code goes here
    return value;
}
```

- First line is called function prototype
  - Should match definition
- `return` keyword
  - Ends function immediately and outputs a given value
  - Can be used by itself (`return;`) for `void` functions
  - Return value will be essentially random if you forget to `return`
    - Like an uninitialized variable
    - `void` functions can end at end brace, but better style to use `return`

Commenting your functions

- All of your functions in assignments should have a descriptive comment
- Comments generally appear above implementation

- Should provide:
  - Overall description of purpose
  - The meaning of and restrictions on all parameters
    - `@param <name> <description>`
  - The value being returned (if not `void`)
    - `@return <description>`
  - The author, along with possibly copyright and organization information
    - `@author <you>`
    - `@copyright <year>`

Calling functions

- Syntax: `<name>(<parameters>)`
  - E.g., `int num = findMax(10, 5);`
  - Make sure that number and type of parameters match declaration and implementation
  - Parameters must be in correct order
- On calling:
  - Current function pauses
  - Called function is executed until a `return`
  - Function call is replaced with return value and function resumes
  - Debugging: use "step into" to go to first line of a called function
- Functions can be called anywhere their output values could appear
  - On a line by itself (esp. `void` functions)
  - Assignment statements
  - If/while conditions
  - For loop initialization or update

Function exercise

- Implement a function findMax that takes two integer values and returns the larger of the two

Variable scoping

- All variables have a scope, or region of code for which they are defined
  - Generally, innermost set of braces in which declaration appears
  - Variables declared outside of a function (usually in preamble) have scope for the entire program
    - Overuse of global variables is considered bad style
- After leaving block (function, `if`, `while`, `for`, etc.), variable disappears and memory is returned to OS
  - **Key point:** function variables cannot be accessed by calling function
  - Use parameters and return values to pass information in and out of a function, everything else is inaccessible
- **Coupling:** bad software practice in which the calling function relies on implementation details of the called function in order to work properly
  - E.g., relying on global variables
  - If function is changed, your code may break

Reference parameters

- When function is called, values in function call are copied into variables before function starts
  - If you pass a variable as a parameter, the original value will not change
  - Exceptions: global variables and arrays
  - Considered bad style to modify parameter values in the function
- What if we want to change parameter values?
  - E.g., `void swap(int a, int b)`

- Reference parameters
  - Denoted by appending `&` to type
  - E.g., `void swap(int& a, int& b)`
    - When calling, a variable must be specified for this parameter
  - Useful when you need to return multiple pieces of information

Tonight

Lab 2 is due Tuesday at noon

Lab 3 is due next Monday at 11:59pm