Functions (2)

Advanced function concepts
Goals

• By the end of this lesson you will be able to:
  – Understand what global variables and global constants are
  – Understand static variables and what makes them different from regular variables
  – Understand the rules for overloading functions
  – Understand the rules for default function parameters
  – Understand how to write a recursive function
Global variables and scope

- In C++ we desire to declare variables as close as possible to when they are used for both readability purposes and issues relating to scope.
- Variables declared outside of the main body (typically before function prototypes) makes them global meaning they can be used in any function.
- Using global variables is a bad practice, because if those variable identifiers are re-used it is ambiguous to the programmer as to which is being used.
Avoiding global variables

- It is often easier to use global variables than adhere to proper programming practices, but just follow these rules:
  - If you need access to a variable in multiple functions, pass it as a parameter
  - Pass by value when you do not need or care about the function modifying the value/variable
  - Pass by reference when you do need the function to modify the variable
Global constants

- One thing that is generally acceptable is creating a global constant – this is a good idea for values you may want to quickly edit and recompile and/or when you need them in multiple functions.

- When you get into software development, using a global constant is not the best practice, and you may want to place it in a namespace instead – but this is a more advanced concept.
Static variables

- Static variables are unique and rarely used, but they can be very useful.
- To use them, place the `static` keyword in front of the data type.
- Static variables have the following rules:
  - They are initialized once.
  - Their value stays between calls of the loop.
- Note that you cannot access the static variable outside of the function (same as with normal function variables).
- It should be intuitive that `rand()` uses a static variable.
Function overloading

- You can write multiple functions with the same name as long as the parameter lists are different.
- This is OK:
  ```
  int fcn(int, int);
  int fcn(string);
  ```
- This is not OK:
  ```
  int fcn(int, int);
  double fcn(int, int);
  ```
Default function parameters

- You can specify default parameters for your functions, resulting in many ways you can call the same function.
- Default values are specified in the prototype and once a parameter has a default value, the remaining parameters on the right must also have default values.
- If you do not provide a parameter to the function when calling it, the default value is used.
Default function parameters

- This is OK:
  ```
  void fcn(int, int = 3, int = 4);
  ```

- You may call the above function in any of these forms:
  ```
  fcn(5);
  fcn(5, 6);
  fcn(5, 6, 7);
  ```

- You may not call the function like this:
  ```
  - fcn();  // no default parameter for the first variable
  - fcn(5, ,8);  // you cannot “skip” a variable
  ```

- This function is not OK:
  ```
  void fcn(int = 3, int, int = 4);
  ```
Recursion

• Recursion is a technique in which a function calls itself
• Many problems can be modeled in a recursive manner, for instance the factorial operation from math: \( x! = x \times (x - 1) \times (x - 2) \times \ldots \times 2 \times 1 \)
• Importantly, \( x! \) can also be written like this: \( x! = x \times (x - 1)! \)
• Recursive functions consist of two cases:
  - The recursive case in which the function calls itself
  - The base case in which the function does not call itself
• Appropriate logic must be used to correctly stop the recursion
Recursion – factorial example

- Step 1: We call fact(3)
- Step 2: fact(3) returns the value of 3 * fact(2) – so we need to evaluate fact(2) before we can return from fact(3)
- Step 3: fact(2) returns the value of 2 * fact(1) – so we need to evaluate fact(1) before we can return from fact(2)
- Step 4: fact(1) evaluates to 1
- Step 5: fact(2) thus evaluates to 2 * 1
- Step 6: fact(3) thus evaluates to 3 * 2 * 1 = 6
Exploration

- Write your own function using a static variable that when called the first time will return 1 then for successive calls will return increasing powers of 2
- Overload a function called print such that one version takes a string and another version that takes an integer – output them as appropriate
- Write a function called areavolume using default parameters that will take either two or three parameters to calculate the volume of a box or the area of a rectangle (if the third parameter is not provided)
- Write a recursive function that will calculate the nth number in the Fibonacci series (1, 1, 2, 3, 5, 8, 13, 21, 34, … – each term is the sum of the two previous terms)