Lecture 05 Notes

Derek Harter

2015-02-13

1 First Session (11 - 11:40)

1.1 Scope Rules

- Variables declared in functions are local to the function.
- Variables (and constants) declared outside of functions are global.
- You probably should NOT be using global variables. Global variables allow state information to leak between functions.
- Global constants, however, are often a good idea and useful.
- Understanding the scope of variables is important. Limiting scope of variables is important.

1.2 Passing Parameters by Reference

- Variables are passed to functions by value by default.
- This means, the value is copied, and if you change value in the function, the value is not changed in the caller.
- The only way to get a value back to the caller, is to return it as the return result from the function.
- However, sometimes we need to return more than 1 value, or sometimes for efficiency reasons (e.g. need to return 1 million values, we probably don't really want to copy them all back).
- In this case, we can pass in values by reference.

- A reference parameter, if changed in the function, will be changed for the caller who provided it. In effect, a reference parameter is not a copy, but it IS the actual variable from the caller, so changes to it will be accessible to the caller when the function returns.

1.3 Random Number Generation

- `rand()` and `srand()` functions
- Included in the C standard library `<cstdlib>` `<stdlib.h>`
- `RAND_MAX`
- `rand()` generates an integer from 0 to max
- `srand()` setting the seed.
- Random number generator function is **pseudo random**, a sequence
- Can repeat a sequence of random numbers.
- How do we flip a coin sided die?
- Write a function that flips a coin.
- Write a function that returns random number in arbitrary range.
- How do we generate a floating point number randomly?
- Write a function generate a float over range 0.0 to 1.0 with uniform probability.

2 Second Session (11:45 - 12:30)

2.1 Recursion

- A function that calls itself (either directly or indirectly).
- Some problems are much more easily (succinctly) stated or solved as a recursive relationship.
- **base case(s)** are extremely important. All recursive functions must have 1 or more base cases. The base cases are the cases that are directly solvable. These are the pieces we know the answer to, or can determine easily.

- The recursive cases then are those that we don't know directly, but that we solve usually by breaking apart the problem, and specifying/solving in terms of easier subparts. We can call our function itself on these smaller subproblems, and combine the answer in some way to solve the more complex problem. This is the **recursive call** or the **recursion step**.
- Example, calculate the factorial of a number
- Write an iterative function to calculate factorial first.
- Write recursive function to calculate factorial.
- Example, the Fibonacci sequence
- Write an iterative function implementation first.
- Write a recursive function to calculate Fibonacci sequence

3 Third Session (12:40 - 1:40)

3.1 Function Call Stack

- Example, functions A(), B() and C()
- Recursive
- Some advice/examples on using the Visual Studio debugger.