CS106X Handout
Winter 2017 Feb 17th, 2017

Functions and Pass by Reference

Function Prototypes

As we learned in lecture, C++ computations are carried out in the context of functions. A **function** is a named section of code that performs a specific operation. Function **prototypes** tell the compiler everything it needs to know about how to call each function when it appears in code. C++ requires prototype declarations so the compiler can check whether calls to functions are compatible with the function definitions.

Function Definitions

Programs should be broken down into several smaller functions. Good decomposition leads to code that is clear, logical and easy to understand. All functions have a **body** that consists of a **return**, a **name** and potentially **parameters**. Here is a simple function example:

```
int absoluteValue(int n) {
  if (n < 0) {
    return -n;
  }
  return n;
}
```

```
return: int
name: absoluteValue
parameters: n
```

Pass by Value

In C++, whenever you pass a variable from one function to another as a parameter, the function gets a copy of the variable. Assigning a new value to the parameter as part of the function changes the local copy but has no permanent effect on the variable. Consider the following example:

```
void setToZero(int n) {
  n = 0;
}
```

After the function finishes, *n* will no longer equal zero. It will only equal zero in the scope of this function.

Pass by Reference

In C++, if you declare a parameter with an ampersand (&) after its type, instead of passing a copy of its value, it will link the caller and callee functions to the same variable in memory. Passing a variable by reference ensures that any changes made to the variable will persist outside of the scope of the function. This style of parameter passing is generally used when a function needs to return more than one value to the calling program. Let's revisit the setToZero function:

```
void setToZero(int& n) {
  n = 0;
}
```

After the function finishes, *n* will still be equal to zero. The caller of setToZero will see *n* as zero after the function returns.

Benefits:

- Allows functions to ‘return’ multiple values
- Often used with large objects to avoid making copies (can be time-consuming)

Downsides:

- Hard to determine from caller if a variable is passed by reference or by value
  ```
  foo(a, b) // will foo change a or b?
  ```
- Can’t pass literal values to a reference parameter
  ```
  foo(10) // error
  ```