APS105 Lecture 9
5.1-5.2

Dan Zingaro

February 8, 2010

Feedback from Reading Quiz

```
void doThis (int n, double x);
```

- We expect to call `doThis` with an `int` and a `double`
- We could also call it with a `char` and an `int` — why?

- "Is there any way to swap values (change the values of `i` and `j`) without pointers in the function?"
  - No. If we want a function to change the value of a variable, we must use pointers
- "I don't understand the difference between parameter and argument."
  - Parameter: `n` and `x` in the function definition above
  - Argument: what we pass to a function in a function call

Functions: Motivation

- Suppose you are writing a cookbook
- It has 12 cake recipes, and five of them involve a buttercream filling
- Do we have to explain how to make the filling five separate times?
- No — we can "define" it (write it out) once, and then "call" it from other recipes
- This type of reusability is why we use functions in programs

Why Functions?

- Helps break our problem into more easily-managed sub-problems
- Makes testing easier, because we can test each function separately
- Allows us to run the same code multiple times without having to copy-and-paste

Simple Functions

- The simplest functions take no parameters and return no value to the caller

```
void sayHi (void) {
  printf ("Hi!\n");
}
```

- The first `void` means "return nothing" to the caller; the second `void` means "obtain nothing" from the caller
- We call the function using `sayHi ();`
- A function call executes the code in a function, and then it returns to the point right after the function call
- Hmm. How about `printf (sayHi());`?

Function Prototypes

- For consistency in this course, we will define all functions above our `main` function
- Between the `#include` lines, and our function definitions, we will declare each function using a prototype
- The prototype is the first line of the function definition, with a semicolon at the end
- It tells C about the function's parameters and return value
- The purpose of prototypes is similar to the purpose of `#include`

Example: Printing a Line

```
#include <stdio.h>

void printLine (void);

void printLine (void) {
  int i;
  for (i = 1; i <= 40; i++)
    printf ("*");
  printf ("\n");
}

int main (void) {
  printLine();
  printLine();
  return 0;
}
```

ConcepTest

What is printed by the following program fragment?

```
void sayHi (void) {
  printf ("Hi! ");
  sayBye();
}

void sayBye (void) {
  printf ("Bye! ");
}

int main (void) {
  sayHi();
  sayHi();
  ...
```

A. `Hi! Bye! Hi! Bye!`
B. `Hi! Hi!`
C. `Hi! Bye!`
D. `Hi! Bye!` infinitely often

Function Parameters

- Without parameters, a function does the same thing each time it is called
- Some functions accept parameters when they are called, and can use the values of the parameters during their execution
- We must pass one value for each parameter that the function expects
- Each value is assigned to the corresponding parameter inside the function
- e.g. we must pass one value to the following function when we call it

```
void printRecip (float f);
```

ConcepTest

How would we call the following function to print 10 x's?

```
void printRow (char c, int n) {
  for (int i = 1; i <= n; i++)
    printf ("%c", c);
  printf ("\n");
}
```

- A. `printRow (10, x);`
- B. `printRow (x, 10);`
- C. `printRow (10, 'x');`
- D. `printRow ('x', 10);`

Pass-by-Value

```
void printRecip (float f) {
  f = 1 / f;
  printf ("%f\n", f);
}

int main(void) {
  float g = 5;
  printRecip (g);
  ...
```

- Here, we are passing value g to `printRecip`
- C executes code similar to the following assignment statement, just prior to execution of the `printf` inside of the function:

```
printRecip's f = main's g;
```

Pass-by-Value...

- Consequence: parameters (like `f` above) are copies of values, not the values themselves
- If we change a parameter inside the function, we are not changing the value outside of the function!
- It's just like an assignment statement
  - When we say `y = x;`, changes we make to one variable are not seen in the other