Notes for Thursday 20 January

23 January 2011

- Meeting 2
- Assignment 1 due
- read §2.3–2.5

Integer and floating-point data types, arithmetic operations, variable declaration and initialization.

Syntax

Syntax refers to the form a program takes, and **semantics** refers to the meaning (what it does). There are lots of ways to change the syntax without affecting the semantics: add or reduce the spacing, put statements all on one line or spread it out, indent more or less.

However, not all of these changes are equally good. Since we write programs for humans to read, not just for computers to execute, you want to write your programs to maximize readability. A little extra space is almost always better. Put separate ideas on separate lines. Proper indentation is **crucial**: any statement inside a *block* (the curly braces) should be indented using the tab key (or a few spaces, that's up to you). The braces themselves should align with each other vertically.

Comments also improve the readability of your program. They are ignored by the compiler, so they can contain plain English, or even C code that you don't want to execute right now. There are two kinds: single-line and multi-line:

```
/* This is a
   multi-line comment
   that is started by slash-star,
   and ended with star-slash.
   Code can come right after: */  printf("This is code\n");

printf("Still code.\n"); // This is single-line comment.
// Another single-line. Starts with double-slash,
// ends at end of line.
```

Types

We learned two numeric types: `int` for whole numbers (including zero and negatives), and `float` for numbers that can have decimal parts.

Variables

A variable is an identifier that stands for a location in memory where we can store a value of some type. You must declare all variables before they are used. A declaration looks

like this:

```
int quiz2;
float width;
```

That is, the type followed by the identifier, then semi-colon. If you have multiple identifiers of the same type, you can merge them in one declaration with commas:

```
int quiz3, quiz4, midterm;
float height, length;
```

Variables get their values from assignment statements, which use an equals sign:

```
quiz3 = 98;
quiz4 = 83;
height = 1.25;
width = 2.8;
```

However, unlike in mathematics, this equals sign is a **one way street**: it goes right-to-left. On the left side, there must be a variable name. The right side can be any expression that produces a value of the correct type.

```
quiz2 = 100 - 5;
midterm = quiz2 * 3;
length = height * 1.2;
```

Operators

Some of the operators that work on `int` and `float` values or addition (`+`), subtraction (`-`), multiplication (`*`), and division (`/`). There are many more that we'll learn as we go.

Format strings

Finally, we want to be able to print these values on the screen. We know `printf` can print text, but how does it retrieve the value of the variable? It uses *format codes*, which start with a percent sign.

code	purpose
`%d`	substitute next value as a (base ten) integer
`%f`	substitute next value as a (floating-point) real number

Here is how they work:

```
printf("The score is %d.\n", quiz2);
printf("The room is %f by %f.\n", length, width);
```

Practice

Try putting together all these elements into a full program you can run on your system. Below is an example. Predict what the program will output on paper first, and then run it, fix any syntax errors, and see if you were correct.

```
#include <stdio.h>

int main()
{
    int a,b;
    float f, g;
    f = 3.8;
    g = f - 5.2;
    printf("%f,%f", f, g);
    return 0;
}
```