Notes for Tuesday 18 January

23 January 2011

- Meeting 1
- read §2.1–2.2

Introduction, motivation, tools, identifiers, `main`, and `printf`. Hello, world!

Motivation

Programming a computer is a tremendously valuable skill, but it takes a lot of practice and patience. Try to work a little every day (and a **lot** some days!) and don't get discouraged if it starts to feel difficult. Although some people will make

In particular don't get discouraged by mistakes. Mistakes are how we learn. Remember that compiler errors are routine – sure, there's a mistake, but it doesn't reflect badly on **you**. Just systematically start finding and fixing it. Professional programmers with decades of experiences see hundreds of errors every single day.

If you feel like you're getting lost, **do not hesitate** to seek help from me. I try to be very accessible, in office hours, by appointment, or over IM. Because everything we learn **builds on** everything that came before, any problem you're having will get better **only** with additional practice and expert assistance. It will not get better on its own!

Getting "hello world" to run

Here we will keep instructions on getting a simple program to run on various compiler systems. For Windows users, I'm recommending Visual C++ Express Edition. In the lab, we have Visual Studio 2008. For the Mac, get Apple's XCode developer tools. Other configuratinos are possible. If you find directions for some system not listed here, do send them to me.

Here is the sample program we'll use. You can also start with this for Assignment 1.

```
#include <stdio.h>

int main()
{
    printf("Hello, world!\n");
    return 0;
}
```

Dissecting the sample program

Here are some of the concepts we covered about the sample program today. They are probably incomplete.

- `include` means to bring in functionality (called a **library**) from somewhere else. In this

case, `stdio.h` refers to the standard input/output library, which is the home of `printf`.

- `main()` is the name of a function, or procedure, which is a sequence of steps to execute. The parentheses indicate that it's a function, which is why we need them even though, in this case, they're empty. The name `main` is special, because it is used as the "entry point" of the program — it all starts here. Later on we'll have additional functions with other names.
- The **curly brackets** `{}` (also called **braces**) indicate the beginning and end of a **block** of code. They show the extent of the `main` procedure.
- The `printf` is how we do formatted output. It is a function (you can tell because of the parentheses) from the standard input/output library.
- The double quotes indicate a **string** or sequence of characters. Because they are provided to `printf`, these are the characters that will appear on the screen when it runs.
- One of the characters is special: it consists of the sequence backslash, then lowercase 'n': `\n` This is called a **new line** character. It tells the output to go on to the next line, much like hitting return or enter in a word processor.
- Finally, `return` is a keyword (there are no parentheses, so it cannot be a function) that means we are finished with the current function. At this point, because the function we're talking about is `main`, the program ends. If you were to include extra code after `return`, it would not be executed.
- Note also the semi-colons. There are semi-colons on statements *within* the block (between the curly braces), but not on the curly braces themselves, and not on the `main()` declaration, and not on the `#include`. If you forget a semi-colon, the compiler will definitely let you know. However, if you put one where it is not supposed to go, it could happen that the compiler won't notice but the program also won't work right! We'll see examples of that insidious problem later.

Identifier rules

Identifiers are how we *name* things in a program. `main` and `printf` are examples of identifiers. They both name functions: one defined in this program, and one included from `stdio.h`.

Identifiers must start with a letter (upper or lower case) or an underscore character (shift-hyphen). Following the first character, they can contain letters, *numbers*, or underscore. They may **not** contain spaces. However, we often use the underscore to separate words that are part of the same identifier, for readability, as in: `quiz_2_average` or `parking_ticket_amount`.

Identifiers are case-sensitive, which means that it matters whether upper- or lowercase letters are used. `Quiz` and `QUIZ` and `quiz` and `qUiZ` are *all* valid identifiers but they are **different** identifiers. You cannot type `Quiz` in one place and then expect `quiz` to work in another.

Fonts

Because programming languages are picky, it's important to configure a good font, and make it big enough, so that characters like `0` (zero), `O` (capital O), `1` (one), `l` (lowercase

l), ; (semi-colon), and : (colon) are sufficiently distinguishable.

In Visual Studio, you can configure this under Tools » Options » Fonts & Colors. It's also recommended to use a fixed-width (monospace) font (they were the bold ones in Visual Studio). A reasonable choice is "Consolas".