COSC 301: Wednesday, 29 August 2012

Learning goals

- Solve with moderate expertise problems with C programs that use pointers, structures, arrays, and other fundamental language features.
- Describe how C programs are compiled, including preprocessing, compilation, linkage, and loading, and how the operating system is invoked for each of these steps.

Why C?

- All OSes are written in C
- Great low-level language to learn and know
- Broaden your skillset

C program pieces and compiling

`.c` C source code file. Java syntax borrowed heavily from C, so a lot of what we see will look (superficially) similar.

`.h` A "header" file: contains data type and function declarations, among other things. More on these when we talk about functions and prototypes.

`Makefile` or `SConstruct` Configuration file for describing how the program should be compiled. (More on these later.)

`.a` or `.so` libraries Files that contain function object code (compiled code) to be linked into an executable program.

We'll be using `gcc` as a C compiler, usually invoked with a `Makefile` or with `scons`. The C compilation process is a little different than in Java and other compiled languages:

```
Source code -> Preprocessor -> Compiler -> Linker -> Executable
                                                    /
                                            Libraries
```

`gcc` can do all the phases in one shot:

```
gcc -o program sourcecode.c
```

We'll talk a little more about the preprocessor and linker stages later.

C types

Basic C data types are all fundamentally numeric:

- Integer types: `int` (32 bits), `short` (16 bits), `char` (8 bits)
  - By default these are *signed*, but they can be declared as unsigned:

```
unsigned char x; // what's the range?
                 // what's stored in x?
x = 130;
x += 2;          // what's stored in x?
```

  - Also `long long`, which is 64 bits
  - `#include <stdint.h>` to get explicitly sized integral types
    * `int8_t`, `uint8_t`, `int16_t`, `uint16_t`, etc. (32 and 64 bit types, too)
- Floating point: `float` (32 bits), `double` (64 bits)
- No booleans, really. Use `int`: zero is false, non-zero is true.
- No automatic initialization of variables as in Java! Remember to set variables to a known state.
- The built-in C function `sizeof` takes a type expression as a parameter and returns the number of bytes that the type will occupy in memory.

Output with `printf`

Use it to print output to console:

```
printf ("Hello world\n");
```

Stuff with `printf`:

- Must explicitly print newline characters to move to next line of output
- Use format strings to tell C how to display a particular value
- Follow the format string with all the items to fill in for each `%` format placeholder
  - `%d`: integer (printed in decimal format)
  - `%s`: string (char array - more on strings later)
  - `%c`: character (char)
  - `%x`: integer (printed in hexadecimal format)

```
int x = 100;
printf("The value of x is %d\n", x);
printf("The value of x is %x\n", x);
```

`man` pages

Manual pages are your best friend in Linux and C (as are Google and stackoverflow, I suppose):

```
$ man printf
```

You can use `q` to get out of the man page; `d` or space to go down, `u` to go up. `h` for help.

Sometimes there are multiple manual pages for the same name (`printf` has this problem). You can type `man -aw printf` to get the various man pages that apply, or `man -k printf` to do a search for `printf` in the titles of man pages:

```
$ man -aw printf
/usr/share/man/man1/printf.1
/usr/share/man/man3/printf.3
```

The output shows that `printf` appears in sections 1 and 3 of the manual pages. To get a specific section, you can just type the section as the second argument: `man 3 printf`.