Tip 1: Use a makefile

Ben Klemens

1 October 2011

level: Start here
purpose: Worry about compilation details once and only once

On the scale from *works out of the box* on one end to *infinitely configurable and tweakable* on the other, the C compiler is far on the tweakability side. But correctly tweaking the compiler is a problem you need to solve once, and then get on with your life. The solution is the *makefile*, which is basically an organized set of variables and shell scripts.

If you just want to go from `.c` files to an executable in least time, here's the `makefile` for you:

```
CFLAGS = -g -Wall -std=gnu99 -O3
LDLIBS=
OBJECTS=
CC=gcc

$(P): $(OBJECTS)
```

Usage:

- Once ever: save this (with the name `makefile`) in the same directory as your `.c` files.
- Once per session: Set the variable `P` as the name of your program from the command line: `export P=your_program` (not `your_program.c`).
- Every time you need to recompile: type `make` .

Tweaks:

- If you have a second (or more) C file, add `second.o third.o`, et cetera on the `OBJECTS` line (no commas, just spaces between names).
- If, when you run the debugger, you find that too many variables have been optimized out for you to follow what's going on, then change the `-O3` part (which sets Optimization level three) to `-O0`.

- If you are using a not-entirely-standard library of functions, then you will need to add the library on the `LIBS` line and the include path on the `CFLAGS` line. Try typing `pkg-config` on your command line; if you get an error about specifying package names, then great, you have pkg-config and can use it like:

  ```
  CFLAGS=[everything above plus:] `pkg-config --cflags apohenia glib-2.0`
  LDLIBS=`pkg-config --libs apohenia glib-2.0`
  ```

  If you get an error about `pkg-config` not being found, you'll have to specify each library and/or its locations:

  ```
  CFLAGS=[everything above plus:] -I/home/b/root/include
  LDLIBS=-L/home/b/root/lib -lweirdlib
  ```

  Tune in next time for an extended example.

- After you add a library to the `LIBS` and `CFLAGS` lines and you know it works on your system, there is little reason to ever remove it. Do you really care that the final executable might be 10 kilobytes larger than if you customized a new makefile for every program?

- If you are using a compiler other than GCC, set `CC` accordingly.

OK, you're done! After you `export P=your_program`, you can run `make` and watch the compiler run and/or spit out errors, and never worry about what all that junk in the makefile actually means.

There will be limited plugs for *Modeling with Data* in this tip-a-day series, but I think it's worth mentioning that Appendix A goes into great detail about tweaking the makefile to do great things. Some notes for now:

- Threading? Then add `-pthreads` to both `CFLAGS` and `LDLIBS`.

- The `make` system has several automatic rules. To generate an object file, it runs

  ```
  $(CC) $(CPPFLAGS) $(CFLAGS) -c
  ```

  and to generate a final program from a set of object files,

  ```
  $(CC) $(LDFLAGS) yourtarget.o $(LOADLIBES) $(LDLIBS)
  ```

  (though it also does the right thing when you have multiple objects). With those rules in hand, you know what variables to set if you find that you need more tricks. [Yes, that is how `LOADLIBES` is spelled.]

To do:
Here's the world-famous `hello.c` program, in two lines:

```
#include <stdio.h>
int main(){ printf("Hello, world.\n"); }
```

Save that and the `makefile` to a directory, and try the above steps to get the program compiled and running.