Tip 31: Use the database for configuration info

Ben Klemens

2 December 2011

level: writing programs with lots of options
purpose: avoid writing yet another config system

An easy text-to-database conduit isn't just for data sets. Your project may also need an extensive set of configuration details. Simulations are especially prone to this: how many periods should run, how many agents should we start with, what percent of agents will be type 1, et cetera. These can all be expressed as plain text.

You could thus write a text file where each line is a key, followed by a colon, followed by the key's value. Read in the text file using `apop_text_to_db` with the delimiter set to `:`, and you now have a key/value database with all of your configuration info.

Let's re-do the Fibnoacci example from Tip #19 (Entry #069). First, a config file, here-document-ified for your cutting and pasting convenience:

```
cat > fib_config << "."

title: The Fibonacci sequence
how many: 20

#In my example, I started the sequence with 1, 1, but the
#Fib. sequence formally starts with 0, 1.
first: 0
second: 1

#Or uncomment these to try Lucas numbers, which start with:
#first: 2
#second: 1
#title: The Lucas sequence

#The setup below will use only the first instance of any given key.
.
```

Our keys can have spaces (and basically any other odd characters that make the names human-friendly), because they're text to be read into the database, not variable names. Apophenia follows the shell convention that #s indicate comment lines.

Now for the program that uses the config file. The big change from the version in Tip #19 (where this code was used to introduce static variables for state machines) is the `Get_float_key` macro used throughout, and that `main` starts by reading in the database and setting defaults. If you recall the macro tips from earlier (Entry #079), the macros should be easy to read.

```
#include <apop.h>

#define Get_float_key(k) \
    apop_query_to_float("select value from config where key='" #k "'")

//Get_text_key leaks. New School Tip: the leak is too small to be worth caring a
#define Get_text_key(k) \
    apop_query_to_text("select value from config where key='" #k "'")->text[0][0

#define Check_key(k, default)   \
    if (!apop_query_to_float("select count(*) from config where key='" #k "'"))
        apop_query("insert into config values ('" #k "', '" #default "')");

#define Staticdef(type, var, initialization) \
    static type var = 0; \
    if (!(var)) var = initialization

long long int fibonacci(){
    Staticdef(long long int, first, Get_float_key(first));
    Staticdef(long long int, second, Get_float_key(second));
    long long int out = first+second;
    first=second;
    second=out;
    return out;
}

int main(){
    sprintf(apop_opts.input_delimiters,":");
    apop_text_to_db("fib_config", "config", .field_names = (char*[]){"key", "val
    Check_key(title, Fibbo sequence);
    Check_key(first, 0);
    Check_key(second, 1);
    Check_key(how many, 10);

    printf("%s\n", Get_text_key(title));
    int max=Get_float_key(how many);
    for (int i=0; i< max; i++)
        printf("%Li\n", fibonacci());
}
```

Great, now you can endlessly tweak your program's parameters without recompiling. If you have to keep a dozen different variant runs organized, you can write a single script that precedes each run of your program with a new config file generated via here document.

There are lots of other ways to read in config data; have a look at *Modeling with Data* pp 203–210 for several. But here's the short version: if you're using `fscanf` or `getchar`, there are higher-level tools that will do the work for you.