Modeling With Data: The Cheat Sheet

Basic C

Types

`int`	Integer
`double`	Real Number
`char`	character

Creating a structure:

```
typedef struct{
  char* name;
  int number;
} person;
```

Declare & initialize:

```
int i = 8;
int an_array[] = {2, 4, 6, 8}
char string[50] = "Hi there.";
int *array_ptr = malloc(sizeof(int)*20)
```

The three laws of pointers

(1) To declare a pointer to an integer, use `int *k`.
(2) After its declaration, to refer to the int being pointed to, use `*k`.
(3) After its declaration, to refer to the pointer itself, use `k`.
For example, if `p1` is an existing pointer, and `p2` is being declared, set `p2`'s address equal to `p1`'s with `int *p1 = p2`, because `p1` abides by rule 1 and `p2` abides by rule 3.

Comparisons

`(a > b)`	a is greater than b
`(a < b)`	a is less than b
`(a >= b)`	a is greater than or equal to b
`(a <= b)`	a is less than or equal to b
`(a == b)`	a equals b
`(a != b)`	a is not equal to b
`(a && b)`	a and b
`(a || b)`	a or b
`(!a)`	not a

A minimal program

```
#include <apop.h>
int main(){
    printf("Hi.\n");
}
```

If you haven't built a makefile yet, name this `hello.c` and compile with

```
gcc -g -Wall -std=gnu99 hello.c
```

Flow

Conditional evaluation:

```
if (condition){
    do_if_true;
} else {
    do_if_false;
}
```

Three types of loop: while, do-while, and for.

```
while (condition){
    do_while_true;
}
```

```
do {
   do_while_true;
} while (condition);
```

```
for (pre-loop; condition; at_each_loop){
    do_while_true;
}
```

For example, the standard means of operating on every element of an array declared with `array[n]`:

```
for (int i=0; i<n; i++){
    operation(array[i]);
}
```

Printf family format specifiers

`%i`	integer
`%g`	general real number
`%s`	string
`%f`	`float`
`%lf`	`double` (formerly long float)
`%Lf`	`long double`
`%-`	left-justify
`%`n	allow n spaces for the number
`%.`n	allow n spaces after the decimal
`%%`	plain old percent sign
`\n`	begin a new line
`\t`	tab
`\"`	quotation mark
\(newline)	continue string on the next line

Matrices and data

Per-element operations

```
gsl_matrix *a, *b;
gsl_vector *av, *bv;
double x;

gsl_matrix_add (a,b);
gsl_matrix_sub (a,b);
gsl_matrix_mul_elements (a,b);
gsl_matrix_div_elements (a,b);
gsl_vector_add (av,bv);
gsl_vector_sub (av,bv);
gsl_vector_mul (av,bv);
gsl_vector_div (av,bv);
gsl_matrix_scale (a,x);
gsl_matrix_add_constant (av,x);
gsl_vector_scale (av,x);
gsl_vector_add_constant (av,x);
apop_vector_log(av);
apop_vector_exp(av);
```

Linear algebra

```
int *gsl_blas_ddot(gsl_vector *L, gsl_vector *R, double *out);
apop_data *apop_dot(apop_data *L, apop_data *R, ...);
double apop_matrix_determinant(gsl_matrix *in);
gsl_matrix* apop_matrix_inverse(gsl_matrix *in);
```

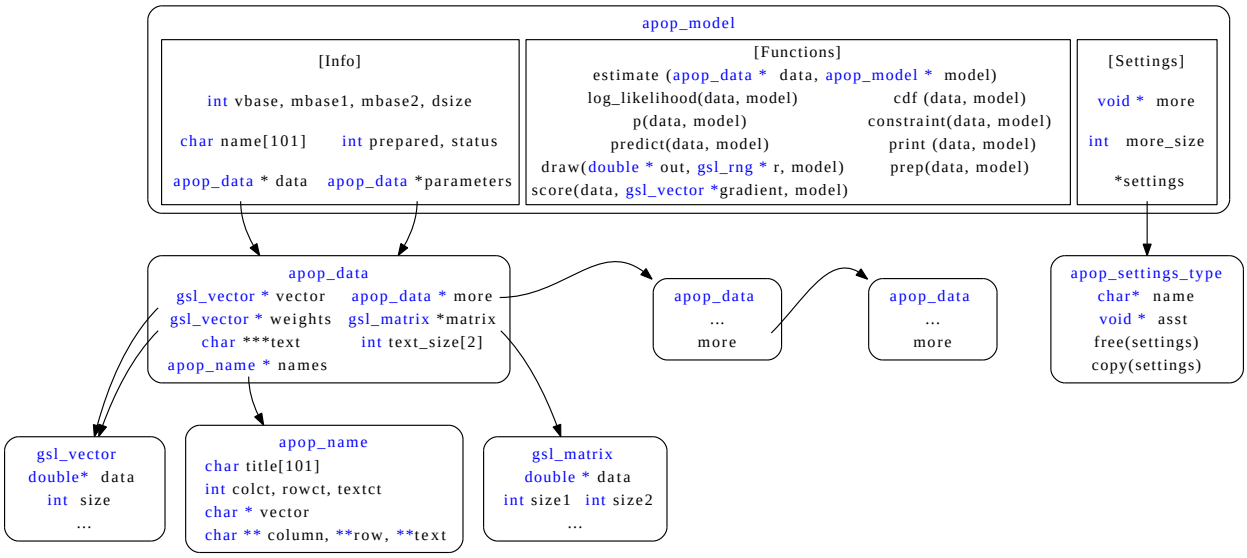

Figure 1: Key structures are built using other key structures. From the bottom up:
➢ The GSL provides simple vector and matrix structures.
➢ The `apop_data` struct has one of each, plus a text grid, weights vector, and metadata in an `apop_name` struct.
➢ Also, the `more` element points to an `apop_data` struct as well, allowing for a list of data sets (retrieved via, e.g., `apop_data_get_page(basedata, "infopage")`).
➢ The `apop_model` includes three types of item: information like data and parameters (which are `apop_data` structs) functions that act on the model, and settings. See the online refs on using the settings.

SQL

The basic query format:

```
select columns
from tables
where row_conditions
```

Specifying columns (select)

All cols: `select *`
Only a few: `select col1, col2`
with table name: `select t1.col1, t2.col2`
with an alias: `select col1+col2 as sum`

Specifying data sources (from)

just list tables: `from table1, table2`
or alias them: `from table1 t1, table2 t2`
or use a subquery:

```
from table1, (select a, b from a_tab) qtab
```

Specifying rows (where)

The usual conditions:

```
where col1+col2 > 3 and (col1=0 or col2!=1)
```

join two tables via a common column: `where t1.name = t2.name`

Vertical query merging

Put on table on top of another, deleting dups: `union`
Put on table on top of another, do not delete dups: `union all`
Find rows that appear in two queries: `intersect`
Find rows that appear only in first query: `except`
All four of these must go between two queries (that return the same number of columns), e.g.:

```
select * from t1
union
select * from t2
```

Table maintenance

create: `create table t(row1, row2, row3)`
add data: `insert into t values(1, 2, 3)`
add via query: `insert into t select ...`
delete: `drop table t`

Gnuplot

Basic 2D: `plot 'datafile'`
Basic 3D: `splot 'datafile'`
Inline data:

```
plot '-'
1 2
3 4
5 6
e
```

Settings

`set term ...`: set the output format, one of
`postscript, postscript color, png, gif, latex`
To return to screen display, use `windows` or `X11`.
`set out outfile`: use in conjunction with `set term`.
`unset border`: Delete the border of the plot.
`unset grid`: Make the plot even more minimalist.
`set size square`: Set axes to have equal length on screen or paper.
`set format y "%.3g"`: Use printf strings to format axis labels.
`set format y ""`: Or just turn off printing on the Y axis entirely.

Comment Corner

C, one line	`//two slashes`
SQL, one line	`--two dashes`
Gnuplot & shell languages	`#number sign`
C & SQL, multiline	`/* star-slash to slash-star*/`