Dynamic Memory Allocation

- How do you allocate an array of variable length?
- Example: reading in different data sets of different lengths
- Answer: Allocate as a pointer, then later allocate the necessary memory for the length of the array
- Use function `malloc` in stdlib to allocate
- or `calloc` to allocate and initialize to zero
- `malloc` has one argument — the amount of memory to allocate
- `calloc` has two arguments, the number of items and the amount of memory each one takes

`malloc.c`

Command Line Arguments

- Can pass values to the program when you run it
- Example: the name of the datafile and its length
- `main` takes two arguments: `argc` and `argv`
- `argc` is the count of the number of arguments in the unix call of the program (including the name of the executable)
- `argv` is an array of character strings, the values of the arguments, with `argv[0]` being the name of the executable
- If you want to use an argument as a number, you need to convert it from a string to an int or float/double with `atoi` or `atof` from stdlib
- The `main` call in the program must specify its arguments, e.g., `int main(int argc, char *argv[])`

The `qsort()` function

- To sort in C, use the `qsort()` function from stdlib.
- `qsort()` takes four arguments:
  - a pointer to the array to be sorted
  - the length of the array
  - the size of an element of the array
  - a comparison function, `fcmp(a,b)`, that returns -1 if a<b, 0 if a=b, 1 if a>b
- It is complicated so that the same function can be used for different types of variables

`sortvec.c`

Allocating Variable Matrices

- This is easiest with the *Numerical Recipes* code
- See `setup.c` and `setup.h`
- A matrix must be treated as a pointer to pointers

`boxing.2.c`

Calling Fortran Functions

- What if you can't find a library routine in C, but you can find one in Fortran?
- You can call Fortran code from C.
- Fortran functions must be declared and called with a trailing underscore
- You must include the fortran libraries, e.g.,
  `gcc -o code.exe code.c -limsl -lfor -lUfor -lm`
- All arguments must be passed as pointers
- All matrices must be transposed

IMSL

- One large set of library routines is IMSL
- Look at the manuals for syntax
- Functions are complicated because they are general
- Also complicated because they are written to save memory

`ttest.c, linreg.c`

Other Libraries

- BLAS — Basic Linear Algebra Subroutines
- LAPACK —- Linear Algebra Package
- other things also available at `www.netlib.org`

`eigen.c`

Notes on Homework 7

- $\log(x_i!) = \text{lgamma}(x_i + 1)$, but it's constant wrt λ
- $\log(\text{pow}(\lambda, x_i)) = x_i * log(\lambda)$
- reading in `turns.dat` is easy
- emacs should auto-indent
- avoid global variables when possible
- print sample every iteration
- don't forget to initialize variables, and do so outside the MCMC loop
- supposed to find that prior isn't important when $N = 300$

Additional Notes for Homework 8

- Question 1: You are expected to get the t value for the width of the interval from IMSL.
- Question 2: You may use either true dynamic memory allocation (malloc/calloc) or the gcc trick. You may modify the data file to specify the size of the data, or you may use command line inputs. The IMSL function for inverting a matrix is easier to use than the LAPACK one, because LAPACK will require you to first get the LU decomposition. You may use either.
- IMSL will not work on the stat department BSD machines, you will need to log onto an alpha. As of the time of printing, the IMSL license still had not been renewed on ACPUB. Non ISDS students may want to work with ISDS students on this homework.
- BLAS and LAPACK work on both the ISDS BSD machines and on ACPUB. The compiling command on an ISDS BSD machine is:

```
gcc -O2 -o eigen.exe eigen.c -L/usr/local/lib -lblas -llapack -lg2c -lm
```

Pseudo-Dynamic Memory Allocation with GCC

- gcc can do run-time memory allocation
- Not truly dynamic, so "pseudo-dynamic"
- gcc will let you declare arrays and matrices in terms of previously defined variables
- This is **NOT** ANSI standard

`sortvec2.c`

Declaring local variables

- Variables can be declared in a local environment by declaring them inside of curly braces `{}`
- In C, you can declare new variables at the beginning of such an environment
- You can combine this with gcc's pseudo-dynamic memory allocation to approximate dynamic memory allocation

`sortvec3.c`