

# CME 212: Basic Statistics

January 18, 2012

In this assignment you will implement several basic statistics functions to compute the mean, variance, median and a histogram of a given real number array. The due date for this assignment is February 1, 2012.

## 1 Assignment summary

The major components of the assignment are

- `stats.h` and `stats.c`: header and implementation files for computing statistics.
- `runstats.c`: a driver file to compute statistics for input data.
- `makefile`: a GNU Make file to build the executable and compile the documentation.
- `git`: we expect you to use `git` for version control.
- `readme.tex` and `readme.pdf`: L<sup>A</sup>T<sub>E</sub>X documentation for the assignment with answers to assignment questions.

## 2 Assessment

The assignment will be graded out of 100 points. A total of 50 points will be given for perfect functionality of code; 40 points will be awarded for

- design and implementation choices,
- code readability and commenting,
- well formatted and written readme with answers to questions,
- and correctly following `make` and `git` guidelines.

The final 10 points are assigned to the challenge part of the assignment.

### 3 Basic statistics

Your task is to implement code computing the mean, variance, and median of an array of doubles. In addition you are to construct a histogram from an array of doubles and a given number of bins. This section describes the function prototypes and the histogram structure. All function and type declarations will reside in `stats.h` with definitions in `stats.c`. Make sure to use a proper header guard.

#### 3.1 Computing the mean

```
// mean takes a pointer to an array of doubles with length n
// and returns the mean
double mean(const double *a, size_t n);
```

#### 3.2 Computing the variance

```
// var takes a pointer to an array of doubles with length n
// and returns the variance
// if samp_var == 0, then compute the population variance
// if samp_var != 0, then compute the sample variance
double var(const double *a, size_t n, int samp_var);
```

#### 3.3 Computing the median

For this part it is fine to use an algorithm from some reference as long as that reference is cited in the readme document.

```
// median takes a pointer to an array of doubles with length n
// and stores the median value at med
// median returns 0 for any runtime exception or failure
// and 1 on successful completion
int median(const double *a, size_t n, double *med);
```

#### 3.4 Constructing a histogram

We will create a histogram structure to store the results of the computation. Let's call this structure `hist_t`, which is short for "histogram type".

```

// histogram type
typedef struct {
    size_t nbin; // number of bins
    size_t *count; // pointer to array of length nbin intended for counts
    double *bin; // pointer to array of length nbin intended for bin centers
} hist_t;

```

The histogram type contains two pointers intended for arrays that will store the histogram data. You will implement two simple memory management routines for the histogram type. The first, called `hist_alloc`, will allocate space for the histogram. The second, called `hist_free`, will free memory associated with a histogram.

```

// hist_alloc allocates memory for the hist_t structure and corresponding
// arrays based on the input number of bins. If successful, it returns a
// pointer to a hist_t structure. If it is not, it returns a null pointer.
hist_t *hist_alloc(size_t nbins);

```

```

// hist_free frees memory that was allocated with hist_alloc
void hist_free(hist_t *h);

```

Finally, the histogram will be computed with the `hist` function.

```

// hist constructs a histogram from an array of doubles of size n. It stores
// the result in the preallocated hist_t structure pointed to by h. The number
// of desired bins is specified by the nbin field in h (h->nbin).
// When finished, h->count contains the histogram counts and h->bin stores the
// bin centers. The function returns 1 if successful or 0 if unsuccessful.
int hist(const double *a, size_t n, hist_t *h);

```

## 4 The driver file: `runstats.c`

The main driver for the assignment will be called `runstats`. It will be invoked from the command line in the following manner:

```
$ ./runstats N file_name
```

### 4.1 Input

- The first argument is an integer indicating the desired number of bins for the histogram.

- The second argument is a file name of a binary file containing data of interest. The first 8 bytes will store an unsigned integer indicating the length of the array. Each subsequent 8 byte block will contain a 64-bit IEEE floating point number (a C `double`). It will be a simple matter to read the data with `fread` function in `stdio.h`.

## 4.2 Expected output

The driver will write to standard out in the following manner:

```
$ ./runstats 5 data.dat
mean:    4.00e+00
var:     1.00e+01
median: -1.00e+00
histogram:
| bin      | count
| -1.00e+00 | 10
| -5.00e-01 | 4
| 0.00e+00 | 5
| 5.00e-01 | 2
| 1.00e+00 | 100
```

The numbers shown in the above example are arbitrary. Specifically, floating point numbers must be aligned and printed in a 9 character field with 2 decimals of precision. It is acceptable to use a 10 character field if the number is large and negative (for example, `-4.00e+100`) even though this will disrupt the formatting of the histogram table. In case of an unsuccessful computation for any statistic print `n/a` in place of a numerical result.

## 5 The makefile

The `makefile` must satisfy the following requirements:

- Executing `make` in the directory must compile all code, produce the `runstats` executable, and generate the `readme.pdf` documentation.
- Executing `make clean` must remove all object files, executables, pdfs, and `LATEX` temporary files.

## 6 Version control

You are expected to use `git` for version control in this and all subsequent programming assignments. When you start working on the assignment first initialize your local `git` repository with:

```
$ git init .
```

After you create some files, add and commit them to the repository:

```
$ git add foo.c bar.c
$ git commit -m "added foo.c"
```

After you make a change to a file, add and commit the changes:

```
$ emacs bar.c
$ git add bar.c
$ git commit -m "fixed nasty bug in bar.c"
```

You can check the history of your work with:

```
$ git log
```

We expect you to commit often as you work on the project. It is not acceptable to just have one commit at the completion of the assignment.

Some good references on the basic use of `git` are:

- <http://git-scm.com/documentation>
- <http://gitref.org/>

## 7 Documentation

All of the documentation for the assignment will be presented in a `readme.tex`, which compiles to `readme.pdf`. The document shall contain:

- Your name and SU-Net ID.
- A description of each file in the directory.
- Instructions how to build and run your code.
- An indication of any external code you used.
- A list of people you worked with or consulted besides course staff.
- Answers to the questions in a following section.

## 8 Questions

**Question 1** The function prototypes specified in this assignment use `size_t` where it might seem natural to use `int`. Why is this important?

**Question 2** Quickly describe the procedures you used to compute the median and construct the histogram.

**Question 3** The following piece of code is not safe. Why? Please assume that the number of integers in array `a` is greater than or equal to `n`.

```
void zero_array(int *a, int n) {
    for (size_t i = 0; i < n; ++i)
        a[i] = 0;
}
```

## 9 Challenge

Challenge points will be awarded for completion of the following task:

- It is often desirable to produce figures from the result of a computation. One way to do this is write a data file for plotting with another package like Matlab. This challenge is to produce a nice figure directly from the C code. There are several libraries one could use for this purpose. One example is `PLplot` (<http://plplot.sourceforge.net/>); there are also many APIs available to the popular `gnuplot` package. The `corn.stanford.edu` system may not have the libraries you'd like to use and you are welcome to build and include a library archive and header file in your project.
- For this challenge you are to implement a function `plot` which given a `hist_t` type variable creates a plot of the histogram, i.e. a plot of the bin value vs the count. The look of the plot and any other additional functionality is up to you. Your Makefile should generate an executable called `plot` which runs your plot function on the input data outputting a plot image. You must also describe and document your method and include a plot figure in your readme.

## 10 Submitting the assignment

All of your files should be in your `CME212` work directory on `corn.stanford.edu` under the subdirectory `stats`. Make sure that

```
$ ls -a ~/CME212/stats
```

lists all of your files as well as the `.git` directory. Then,

1. Make sure your code compiles and produces an executable and a readme document when the makefile utility is run as

```
$ make
```

2. Submit your assignment via the command

```
$ /afs/ir/class/cme212/bin/submit stats
```