# CSC 101 Lecture Notes Week 2

## Let’s Start Programming

## I. Another sample C program.

A. Bit more complicated than last week.

B. Introduces topics from Chs 2 and 3 of the book.

C. Similar in structure and complexity to programming assignment 1.

## Another sample C program, cont’d

D. Here’s the spec:

```
Compute simple statistics for three real
numbers read from standard input.  The
statistics computed are the sum of the
numbers, the arithmetic mean, and the
standard deviation.  Output the results to
standard output, in the following form:

Sum =
Mean =

Standard Deviation =
```

## Another sample C program, cont’d

E. Here’s the program:

```
/* Libraries we’ll need  */
#include <stdio.h>
#include <math.h>

/* Program constant for number of data points */
#define NUM_DATA_POINTS 3
```

```
/*
 * The main function
 */
int main () {

   double x1, x2, x3;   /* Input variables */
   double sum;          /* Computed sum */
   double mean;         /* Computed mean */
   double std_dev;      /* Computed standard dev */
```

```
/*
 * Prompt the user for the input.
 */
printf("Enter three real numbers,
                     separated by spaces: ");
```

```
/*
 * Input the numbers.
 */
scanf("%lf%lf%lf", &x1, &x2, &x3);
```

```
/*
 * Compute the sum.
 */
sum = x1 + x2 + x3;
```

```
/*
 * Compute the mean.
 */
mean = sum / NUM_DATA_POINTS;
```

```
/*
 * Compute the standard deviation.
 */
std_dev = sqr(
    (pow(x1 - mean, 2) +
     pow(x2 - mean, 2) +
     pow(x3 - mean, 2)) / NUM_DATA_POINTS);
```

```
/*
 * Output the results.
 */
printf("Sum = %f\n", sum);
printf("Mean = %f\n", mean);
printf("Standard Deviation = %f\n\n", std_dev);
```

```
    /*
     * Make the compiler happy.
     */
    return 0;

}
```

## II. How about some testing?

A. OK, so I’ve implemented my program (I think).

B. Let’s compile and run it to see.

```
gcc -ansi -pedantic -Wall -Werror stats.c
a.out
Enter three real numbers, separated by spaces: 1 2 3
Sum = 6.000000
Mean = 2.000000
Standard Deviation = 0.816497
```

## How about some testing?, cont'd

C. Hmm, is this correct?

1. From what I know about standard deviation, I think it's supposed to be 1.0 in this case.

2. I better go look it up, and then ask the program specifier if what I'm doing is correct.

## III. OK, so the spec wasn’t precise enough.

A. We’ll fix it by citing an authoritative reference.

B. Then fix the program to agree with the spec.

## The spec wasn’t precise enough, cont’d

C. The fix entails changing this

```
std_dev = sqrt(
    (pow(x1 - mean, 2) +
     pow(x2 - mean, 2) +
     pow(x3 - mean, 2)) /
         NUM_DATA_POINTS);
```

## The spec wasn't precise enough, cont'd

to this

```
std_dev = sqrt(
    (pow(x1 - mean, 2) +
     pow(x2 - mean, 2) +
     pow(x3 - mean, 2)) /
        (NUM_DATA_POINTS - 1));
```

## The spec wasn’t precise enough, cont’d

D. Here’s the updated code:

```
/****
 *
 * This program computes simple statistics for three real numbers read from
 * standard input.  The statistics computed are the sum of the numbers, the
 * arithmetic mean, and the standard deviation.  The results are output to
 * standard output, in the following form:
 *
 * Sum =
 * Mean =
 * Standard Deviation =
 *
 * The precise formulae for mean and standard deviation are as defined here:
 *
 *    http://www.gcseguide.co.uk/statistics_and_probability.htm
 *
 * Author: Gene Fisher (gfisher@calpoly.edu)
 * Created: 31mar11
 * Last Modified: 4apr11
 *
 */
```

• • •

## IV. OK, let’s re-compile, and run it again.

```
gcc -ansi -pedantic -Wall -Werror stats.c
a.out
Enter three real numbers, separated by spaces: 1 2 3
Sum = 6.000000
Mean = 2.000000
Standard Deviation = 1.000000
```

## Re-compile, and run it again., cont’d

A. We’ll definitely need to do some more testing before we’re sure it’s correct.

B. For now, let’s do some program refinement, in particular by adding some *functions*.

## V. Restructuring the program using functions.

A. A *function* in a C program is a piece of computation that has the following properties:

1. It has a ***mnemonic name***.

2. It has a zero or more ***inputs***.

3. It has an ***output***.

## Restructuring using functions., cont’d

B. Here’s a simple function that computes the sum:

```
/*
 * Return the sum of the given three numbers.
 */
double compute_sum(double x1, double x2, double x3) {
    return x1 + x2 + x3;
}
```

## Restructuring using functions., cont’d

C. We’ll dissect this definition fully in class;
chapter 3 of the book provides an in-depth intro.

## Restructuring using functions., cont’d

D. Here’s a refined version of the stats program with three functions, one for each stat computation:

```
/* Libraries we’ll need  */

#include <stdio.h>
#include <math.h>

/* Program constant for number of data points */

#define NUM_DATA_POINTS 3
```

```
/*
 * Declare prototypes for functions.
 */
double compute_sum(double x1, double x2, double x3);
double compute_mean(double x1, double x2, double x3);
double compute_std_dev
              (double x1, double x2, double x3);
```

```
/*
 * The main function
 */
int main () {

    /*
     * Declare the variables used in main.
     */
    double x1, x2, x3;  /* Input variables */
    double sum;         /* Computed sum */
    double mean;        /* Computed mean */
    double std_dev;     /* Computed standard deviatio
```

```
/*
 * Prompt the user for the input.
 */
printf("Enter three real numbers,
                     separated by spaces: ");
```

```
/*
 * Input the numbers.
 */
scanf("%lf%lf%lf", &x1, &x2, &x3);
```

```
/*
 * Compute the sum.
 */
sum = compute_sum(x1, x2, x3);
```

```
/*
 * Compute the mean.
 */
mean = compute_mean(x1, x2, x3);
```

```
/*
 * Compute the standard deviation.
 */
std_dev = compute_std_dev(x1, x2, x3);
```

```
/*
 * Output the results.
 */
 printf("Sum = %f\n", sum);
 printf("Mean = %f\n", mean);
 printf("Standard Deviation = %f\n\n", std_dev);
```

```
    /*
     * Make the compiler happy.
     */
    return 0;

}
```

```
/*
 * Return the sum of the given three numbers.
 */
double compute_sum(double x1, double x2, double x3) {
   return x1 + x2 + x3;
}
```

```
/*
 * Return the arithmetic mean of the given
 * three numbers.
 */
double compute_mean(double x1, double x2, double x3) {
   return compute_sum(x1, x2, x3) / NUM_DATA_POINTS;
}
```

```
/*
 * Return the standard deviation of the given
 * three numbers.
 */
double compute_std_dev(double x1, double x2, double x3) {

   double mean = compute_mean(x1, x2, x3);

   return sqrt(
      (pow(x1 - mean, 2) +
       pow(x2 - mean, 2) +
       pow(x3 - mean, 2)) / (NUM_DATA_POINTS - 1 ));
}
```

## VI. One more program refinement.

A. Functions can be used wherever their value is needed, without having to store it in a variable.

B. Here’s what this idea looks like in `stats.c`:

***... Same as in previous version up to here.***

```
/*
 * Input the numbers.
 */
scanf("%lf%lf%lf", &x1, &x2, &x3);
```

***... Same as in previous version up to here.***

```
/*
 * Input the numbers.
 */
scanf("%lf%lf%lf", &x1, &x2, &x3);

/* The assignment statements are now gone. */
```

***... Same as in previous version up to here.***

```
/*
 * Input the numbers.
 */
scanf("%lf%lf%lf", &x1, &x2, &x3);

/* The assignment statements are now gone. */

/*
 * Compute and output the results.
 */
 printf("Sum = %f\n", compute_sum(x1, x2, x3));
 printf("Mean = %f\n", compute_mean(x1, x2, x3));
 printf("Standard Deviation = %f\n\n",
               compute_std_dev(x1, x2, x3));
```

## One more program refinement., cont’d

C. We’ve eliminated the three program variables `sum`, `mean`, and `std_dev`.

D. The print statements call the functions directly.

E. Illustrates nicely how functions can be used.

## VII. So what’s so great about functions, really?

A. In this particular example, you may be asking

*"What’s the point of using functions when the program is pretty much as simple without them?"*

## So what's so great about functions?, cont'd

B. This question is fair enough, and the best immediate answer is this --

*"Very few C programs are as simple as this; we're introducing functions this way to avoid being overwhelmed with details."*

## So what's so great about functions?, cont'd

C. We'll see very soon that using functions in larger C programs is a must for good program design.

## So what’s so great about functions?, cont’d

D. Here are a couple quick questions to stimulate your thinking in this area:

1. *Suppose we want to write a program that computes statistics on thousands of numbers?*

2. *Suppose we want to share our statistical program with other people, so they can use it like other programs from the C library?*

## So what’s so great about functions?, cont’d

E. The answer to both these question involves designing the statistical program using functions.

## VIII. **Introduction to Conditional Statements**

A. Brief intro in week 1.

B. It’s a fundamental aspect of programming.

C. Allows decision to be made, with outcome affecting what a program does.

## IX. Extending spec for stats program.

### A. Add the following:

```
Input numbers must be non-negative.  If a
negative number is input, treat it as 0.
The number of data points remains 3.  For
example, with inputs 1 -2 3, results are:

Sum = 4.000000
Mean = 1.333333
Standard Deviation = 1.527525
```

## B. The updated program looks like this:

## *Up to here, same as before ...*

```
/*
 * Input the numbers.
 */
scanf("%lf%lf%lf", &x1, &x2, &x3);
```

## *Up to here, same as before ...*

```
/*
 * Input the numbers.
 */
scanf("%lf%lf%lf", &x1, &x2, &x3);

/*
 * Consider any negative input to be 0.
 */
```

## *Up to here, same as before ...*

```
/*
 * Input the numbers.
 */
scanf("%lf%lf%lf", &x1, &x2, &x3);

/*
 * Consider any negative input to be 0.
 */
if (x1 < 0) x1 = 0;
if (x2 < 0) x2 = 0;
if (x3 < 0) x3 = 0;
```

```
/*
 * Compute and output the results.
 */
```

***same as before ...***

C. We’ll dissect these uses of **`if`** in class;
chapter 4 of the book has in-depth discussion.

## X. **Yet another change to the spec.**

A. Treat negative inputs as zero, and drop them out by subtracting 1 from the number of data points.

B. The revised spec looks like this:

## Yet another change to the spec, cont'd

```
Input numbers must be non-negative.  If a
negative number is input, treat it as 0.
For each negative input, the number of data
points is decremented by 1.  For example,
with inputs 1 -2 3, the results are

Sum = 4.000000
Mean = 2.000000
Standard Deviation = 1.000000
```

## Yet another change to the spec, cont’d

C. The updated program looks like this:

## ***Up to here, same as before ...***

```
#include <stdio.h>
#include <math.h>
```

***Up to here, same as before ...***

```
#include <stdio.h>
#include <math.h>
```

***Remove this constant declaration:***

```
#define NUM_DATA_POINTS 3
```

```
int main () {

  /*
   * Declare the variables used in main.
   */
  double x1, x2, x3;
```

```
int main () {

  /*
   * Declare the variables used in main.
   */
  double x1, x2, x3;
  int num_data_points;   /* NEW */
int main () {

  /*
   * Declare the variables used in main.
   */
  double x1, x2, x3;
  int num_data_points;   /* NEW */

  /*
   * Initialize number of data points to 3.
   */
  num_data_points = 3;
```

```
/* Same as before ... */

/*
 * Prompt the user for the input.
 */
printf("Enter three real numbers, ... : ");

/*
 * Input the numbers.
 */
scanf("%lf%lf%lf", &x1, &x2, &x3);
```

```
/*
 * Consider any negative input to be 0
 * Drop it from stats by decrementing
 * number of data points.
 */
if (x1 < 0) {
    x1 = 0;
    num_data_points = num_data_points - 1;
}
```

```
/*
 * Consider any negative input to be 0.
 * Drop it from stats by decrementing
 * number of data points.
 */
if (x1 < 0) {
    x1 = 0;
    num_data_points = num_data_points - 1;
}
if (x2 < 0) {
    x2 = 0;
    num_data_points = num_data_points - 1;
}
if (x3 < 0) {
    x3 = 0;
    num_data_points = num_data_points - 1;
}
```

```
/*
 * Compute and output the results.
 */

/* Same as before ... */
```

## XI. A picture of computer memory in the stats program.

A. We’ve discussed the idea in previous lectures, and it’s introduced in book chapter 1.

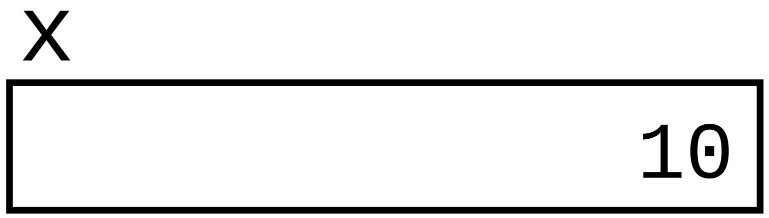
B. Drawn as boxes, labeled with variable names.

## A picture of computer memory, cont'd

C. For example,

```
int x;
x = 10;
```

can be illustrated like this:

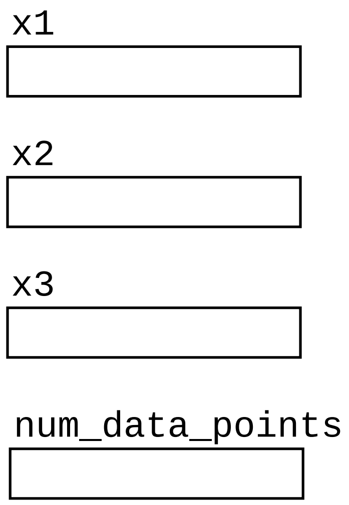


## A picture of computer memory, cont’d

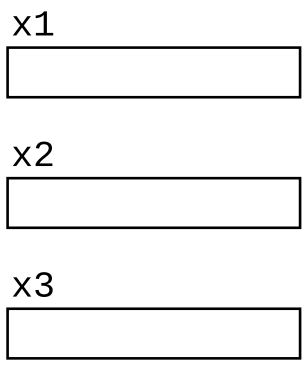
D. Program memory is organized into separate pieces, one piece for each function.

E. Here’s what it looks like in the final version of `stats`:

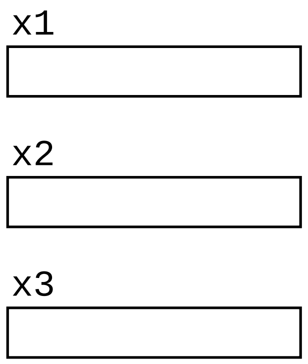
**main memory:**



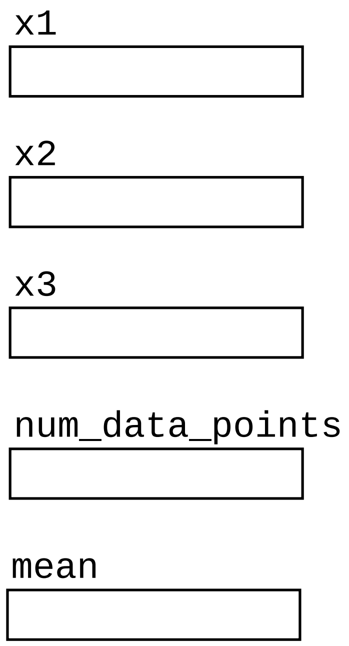
**compute_sum memory:**



**compute_mean memory:**



**compute_std_dev memory:**



## A picture of computer memory, cont’d

F. You may find pictures like this helpful.

G. We’ll use them a lot to help explain program execution.

H. Try filling in the picture at the following points:

## A picture of computer memory, cont’d

1. Before the program starts.
2. After the `scanf` has executed, but before any of the `if`s.
3. After `if`s, but before any of the functions is called.
4. After `compute_sum` is called, but before it returns.
5. After `compute_sum` is called, and it’s returned.
6. After the program is entirely finished.