ECS 30 **Homework #4 (50 points)** **Winter 2013**

Due: Wednesday, February 6th. Written: 4pm in 2131 Kemper. Programs: 11:55pm using handin to cs30, p4 directory. Filenames: prime.c, stats.c, range.c.

Written (7 points): pp. 303-304: 1, 5, 6, 7 pp. 363-364: 3, 4, 6.
pp.303-304:
1. In what ways are the initialization, repetition test, and update steps alike for a sentinel-controlled loop and an endfile-controlled loop? How are they different?

5. Rewrite the program segment that follows using a for loop:

```
count = 0;
i = 0;
while (i < n) {
  scanf("%d", &x);
  if (x == i)
    ++count;
  ++i;
}
```

6. Rewrite this for loop heading, omitting any invalid semicolons.

```
for (i = n;
  i < max;
  ++i;);
```

7. Write a do-while loop that repeatedly prompts for and takes input until a value in the range 0 through 15 inclusive is input. Include code that prevents the loop from cycling indefinitely on input of a wrong data type.

pp. 363-364:
3. Explain the allocation of memory cells when a function is called. What is stored in the function data area for an input parameter? Answer the same question for an output parameter?

4. Which of the functions in the following program outline *can* call the function grumpy()? All function prototypes and declarations are shown; only executable statements are omitted.

```
int grumpy(int dopey);
char silly(double grumpy);
double happy(int char greedy);

int main(void) {
  double p, q, r;
  …
}

int grumpy(int dopey) {
  double silly;
  …
}

char silly(double grumpy) {
  double happy;
  …
}

double happy(int goofy, char greedy) {
  char grumpy;
  …
}
```

6. Present arguments against these statements:

a. It is foolish to use function subprograms because a program written with functions has many more lines than the same program written without functions.

b. The use of function subprograms leads to more errors because of mistakes in using argument lists.

Zyante: 4.1-4.7, 8.1, 8.2, 9.5

Programming (43 points)

All programs should be able to compile with no warnings when compiled with the –Wall option, e.g. gcc –Wall taxes.c. Beginning this week we will be taking points off for warnings. You should put your name(s) in a comment on the first line of each file. Unless specified otherwise, your program need only contain a main() function. When you are to write your own functions, since main() should be the first function in each file, you will need to provide a prototype above main() for each function you write. You are not to use arrays for any program. Unless specified otherwise, you may assume that all data is valid. Remember that if you use functions from math.c, then you need to add –lm to your compile line.

You will find my executables as well testing files in ~ssdavis/30/p4. The prompts and output format of each program must match the examples exactly. User inputs are in **bold**.

#1. Filename: prime.c (9 points, 9 minutes) Write a program that prints out all the prime numbers between two numbers entered by the user in ascending order. A prime number is any integer greater than one that is only evenly divisible by itself and one. Since the modulo operator returns the remainder of division, if it returns zero, then the number is evenly divisible by the divisor. Therefore, a number is prime if none of the numbers between two and the square root of that number evenly divide into the tested number. For this program you must write a function is_prime() that takes a number as its sole parameter, and returns true if the number is prime. Your program must use a do-while loop to check that the numbers entered by the user are greater than one and in ascending order, and if not, then ask for a new pair. "Please try again." may be in only one printf statement. You may assume that the user will enter integers.

```
[ssdavis@lect1 p4]$ prime.out
Please enter two integers greater than one, and in ascending order: 1 20
The numbers must both be greater than one.
Please try again.

Please enter two integers greater than one, and in ascending order: 4 2
The numbers must be in ascending order.
Please try again.

Please enter two integers greater than one, and in ascending order: 4 77
5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73
[ssdavis@lect1 p4]$ prime.out
Please enter two integers greater than one, and in ascending order: 5700 6200
5701 5711 5717 5737 5741 5743 5749 5779 5783 5791 5801 5807 5813 5821 5827 5839
5843 5849 5851 5857 5861 5867 5869 5879 5881 5897 5903 5923 5927 5939 5953 5981
5987 6007 6011 6029 6037 6043 6047 6053 6067 6073 6079 6089 6091 6101 6113 6121
6131 6133 6143 6151 6163 6173 6197 6199
[ssdavis@lect1 p4]$
```

#2. Filename: stats.c (24 points, 25 minutes) Write a program that reads a file named stats_data.txt, and determines the minimum value of each line, maximum value of each line, average of each line, average of the whole file, and standard deviation for the numbers in the file.

1. The first line of the file holds an integer that indicates how many values there are on every line of the file. The number of lines is indeterminate.
2. The output of the program will print out the minimum, maximum, and average for each line with the statistics for two lines on each line of the screen.

3. After printing the information for each line, the last line on the screen will be the average for the whole file (including the first line) followed by its standard deviation. The formula for standard deviation is $\sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}}$ where $\bar{x}$ is the mean (average) of all the items in the file, and *n* is the number of items in the file.
4. Your main() must contain only variable declarations, function calls, and a return statement. You should have separate functions to 1) open the file and determine the first set of stats for the file, 2) deal with the stats for one line and print them, and 3) determine the standard deviation for the file.
5. Hints: To determine the standard deviation you will need to rewind() the file. During that run your program does not care how many rows and columns there are in the file. To avoid passing the address of FILE pointer (hence a pointer to a pointer) into the function that opens the file, have that function use its "return" to return the value of the FILE* it fopen()ed.
6. If you wish to generate your own stats_data.txt file, I have written a program do that called generate_stats_file.out, that was compiled from generate_stats.c. These files are in ~ssdavis/30/p4. Note that to keep multiple stats_data.txt files on hand, you should either rename them or store them in separate directories.

```
[ssdavis@lect1 p4]$ generate_stats_file.out
Please enter the number of rows and columns: 5 4
Please enter the maximum number: 100
Please enter the seed for the random number generator: 2
[ssdavis@lect1 p4]$ cat stats
stats_data.txt  stats.out*
[ssdavis@lect1 p4]$ cat stats_data.txt
4
81 7 0 59
89 52 50 100
70 40 23 54
30 53 1 58
23 34 83 41
[ssdavis@lect1 p4]$ stats.out
min: 0   max: 81  avg: 36.75  min: 50  max: 100 avg: 72.75
min: 23  max: 70  avg: 46.75  min: 1   max: 58  avg: 35.50
min: 23  max: 83  avg: 45.25
Average: 45.33 Standard deviation: 29.45
[ssdavis@lect1 p4]$ mv stats_data.txt stats_data1.txt
[ssdavis@lect1 p4]$ generate_stats_file.out
Please enter the number of rows and columns: 4 7
Please enter the maximum number: 1000
Please enter the seed for the random number generator: 9
[ssdavis@lect1 p4]$ cat stats_data.txt
7
905 177 738 891 912 853 271
607 894 958 848 636 691 309
953 888 153 485 969 483 379
151 196 106 387 115 569 788
[ssdavis@lect1 p4]$ stats.out
min: 177 max: 912 avg: 678.14 min: 309 max: 958 avg: 706.14
min: 153 max: 969 avg: 615.71 min: 106 max: 788 avg: 330.29
Average: 562.72 Standard deviation: 320.27
[ssdavis@lect1 p4]$
```

#3 Filename: range.c (10 points, 10 minutes) Write a program that asks for minimum double value, a maximum double value, and then a value between the two values. The maximum value should be greater than or equal to the minimum value entered. In each of the three cases, the program should detect errors in the entry from the user, explain the error, and then ask the again until they provide a legitimate double within the range specified. Possible errors are: 1) entry is not a number; and 2) extra characters between the number and the end of line, and 3) entered number is outside the specified range. On page 275 in the textbook they have the following code that works fairly well for ints. You may wish to adapt it to doubles, though your function error messages will need to be more specific, and your function must be able to detect all

three errors. Since the first number entered is unrestricted, you will want to use DBL_MAX that is available by #including <float.h>. By using the "%g" format string in your printfs, the program will choose between providing the boundary values in either fixed point or scientific notation formats based on the magnitude of the boundary number. I used "%0.8g." Please be aware that you will have to make significant changes to how get_int() handles skip_ch.

```
// Returns the first integer between n_min and n_max entered as data.

int get_int(int n_min, int n_max)
{
  int in_val,  // input  - number entered by user
    status;    // status value returned by scanf
  char skip_ch;    // character used for skipping balance of line
  int error;    // error flag for bad input

  do
  {
    error = 0; // No errors yet
    printf("Enter an integer in the range from %d ", n_min);
    printf("to %d inclusive> ", n_max);
    status = scanf("%d", &in_val);

    if(status != 1) // in_val didn't get a number
    {
      error = 1;
      scanf("%c", &skip_ch);
      printf("Invalid character >>%c>>. ", skip_ch);
      printf("Skipping rest of line.\n");
    }  // if scanf() failed to read anything
    else
      if(in_val < n_min || in_val > n_max)
      {
        error = 1;
        printf("Number %d is not in range.\n", in_val);
      } // if number out of range

    do  // skip rest of line
      scanf("%c", & skip_ch);
    while(skip_ch != '\n');
  } while(error);

  return in_val;
} // get_int()
```

```
[ssdavis@lect1 p4]$ range.out
Enter a double in the range from -1.8e+308 to 1.8e+308 inclusive> a 4
Invalid character >>a>>. Skipping rest of line.
Enter a double in the range from -1.8e+308 to 1.8e+308 inclusive> 4.0 $
Invalid character >> >>. Skipping rest of line.
Enter a double in the range from -1.8e+308 to 1.8e+308 inclusive> -3.334
Enter a double in the range from -3.3 to 1.8e+308 inclusive> -67.7
Number -68 is too small.
Enter a double in the range from -3.3 to 1.8e+308 inclusive> 99
Enter a double in the range from -3.3 to 99 inclusive> -25.234
Number -25 is too small.
Enter a double in the range from -3.3 to 99 inclusive> 101.2
Number 1e+02 is too large.
Enter a double in the range from -3.3 to 99 inclusive> 77.23411
min: -3.334, max: 99, input: 77.23 are all OK.
[ssdavis@lect1 p4]$
```