**ECS 30** **Homework #5 (59 points)** **Fall 2013**

Due: Wednesday, November 6th, Written: 4pm in 2131 Kemper. Program: 11:59pm using handin to cs30, p5 directory.
Filenames: checksum.c, polygon.c, election.c
gdb tutorial (10 points) Filenames: main.c prime.c typescript.

## Written (4 points): pp. 442-443: 1, 2, 3, 4

pp. 442-443

1. Identify an error in the following C statements:
```
int x[8], i;
for (i = 0;  i <= 8;  ++i)
  x[i] = i;
```
   Will the error be detected? If so , when?

2. Declare an array of type `double` values called **`exper`** that can be referenced by using any day of the week as subscript, where 0 represents Sunday, 1 represents Monday, and so on.

3. The statement marked `/* this one */` in the following code is valid. True or false?
```
int counts[10], i;
double x[5];
printf("Enter an integer between 0 and 4> ");
i = 0;
scanf("%d", &counts[i]);
x[counts[i]] = 8.384;  /* this one */
```

4. What are the two common ways of selecting array elements for processing?

## gdb Tutorial (10 points)

The interactive gdb tutorial is available online from the course website. The TAs will be in 67 and 75 Kemper 6-8pm on Thursday, Monday, and Tuesday to assist anyone with the tutorial. The tutors are also familiar with gdb. You should submit main.c, prime.c, and your typescript from this tutorial. Each person must do the tutorial individually.

## Programming (45 points)

All programs should be able to compile with no warnings when compiled with the –Wall option. You should put your name(s) in a comment on the first line of each file. The prompts, and output format of each program must match the examples exactly. main() should be the first function in each file. You may assume the user will enter valid values.

1.) p. 372 #12 (5 minutes, 5 points) Filename: checksum.c
"Since communications channels are often noisy, numerous ways have been devised to ensure reliable data transmission. One successful method uses a checksum. A checksum for a message can be computed by summing the integer codes of the characters in the message and finding the remainder of this sum divided by 64. The integer code for a space character is added to this result to obtain the checksum. Since this value is within the range of the displayable characters, it is displayed as a character as well. Write a program that accepts single-line messages ending with a period and displays the checksum character for each message. Your program should continue displaying checksums until the user enters a line with only a period." Include a period in the check sum calculation. Do NOT use arrays for this program.

```
[ssdavis@lect1 p5]$ checksum.out
Line: This is short.
Checksum: 2
Line: The quick brown fox jumped over the lazy dog.
Checksum: =
Line: A.
Checksum: O
Line: B.
Checksum: P
Line: C.
Checksum: Q
```

```
Line: .
[ssdavis@lect1 p5]$
```

2.) pp. 444-445 #2 (15 minutes, 15 points) Filename: polygon.c

"If *n* points are connected to form a closed polygon as shown below, the area A of the polygon can be computed as

$$A = \frac{1}{2}\left|\sum_{i=0}^{n-2}(x_{i+1} + x_i)(y_{i+1} - y_i)\right|$$

(See text for drawing of polygon)

Notice that although the illustrated polygon has only six distinct corners, *n* for this polygon is 7 because the algorithm expects that the last point, $(x_6, y_6)$ will be a repeat of the initial point, $(x_0, y_0)$.

Represent the $(x, y)$ coordinates of the connected points as two arrays of at most 20 type double values. For one of your tests, use the following data set, which defines a polygon whose area is 25.5 square units.

| *x* | *y* |
|---|---|
| 4 | 0 |
| 4 | 7.5 |
| 7 | 7.5 |
| 7 | 3 |
| 9 | 0 |
| 7 | 0 |
| 4 | 0 |

Implement the following functions:

**`get_corners`**—Takes as parameters an input file, arrays x and y, and the arrays' maximum size. Fills the arrays with data from the file (ignoring any data that would overflow the arrays) and returns as the function value the number of (*x*, *y*) coordinates stored in the arrays.

**`output_corners`**—Takes as parameters an output file and two type double arrays of the same size and their actual size, and outputs to the file the contents of the two arrays in two columns.

**`polygon_area`**—Takes as parameters two arrays representing the (*x*,*y*) coordinates of the corners of a closed polygon and their actual size and returns as the function value the area of the closed polygon"

Additional specifications: The input filename will be passed as the first command line parameter (*argv[1]*), and the output filename will be passed as the second command line parameter (*argv[2]*). By "input file" and "output file", the authors meant FILE*. You should use a macro for the arrays' maximum size. To determine the absolute value of the summation you will need fabs() from math.h. Remember to use –lm when you write your compile line in Makefile.

```
[ssdavis@lect1 p5]$ cat poly1.txt
4 0
4 7.5
7 7.5
7 3
9 0
7 0
4 0
[ssdavis@lect1 p5]$
[ssdavis@lect1 p5]$ polygon.out poly1.txt poly1b.txt
The area is 25.5.
[ssdavis@lect1 p5]$
[ssdavis@lect1 p5]$ cat poly1b.txt
 4.0  0.0
 4.0  7.5
 7.0  7.5
 7.0  3.0
 9.0  0.0
 7.0  0.0
 4.0  0.0
```

```
[ssdavis@lect1 p5]$
```

3.) pp.448-449 #10 (30 minutes, 25 points) Filename: election.c Executable name: election.out
“The results from the mayor’s race have been reported by each precinct as follows:

| Precinct | Candidate A | Candidate B | Candidate C | Candidate D |
|---|---|---|---|---|
| 1 | 192 | 48 | 206 | 37 |
| 2 | 147 | 90 | 312 | 21 |
| 3 | 186 | 12 | 121 | 38 |
| 4 | 114 | 21 | 408 | 39 |
| 5 | 267 | 13 | 382 | 29 |

Write a program to do the following:

a. Display the table with appropriate labels for the rows and columns.
b. Compute and display the total number of votes received by each candidate and the percentage of the total votes cast.
c. If any one candidate received over 50 percent of the votes, the program should display a message declaring that the candidate the winner.
d. If no candidate received 50 percent of the votes, the program should display a message declaring a runoof between the two candidates receiving the highest number of votes; the two candidates should be identified by their letter names.
e. Run the program once with the data shown and once with candidate C receiving only 108 votes in Precinct 4.”

Additional specifications: The data will be stored in a file that will be passed as the only command line parameter. The format of the file will match that of the table above, except that the labels will not be in the file. Instead of presenting the candidates in alphabetical order, they should be presented from the largest total to smallest. You must write a sort() function that sorts the election matrix based on the total votes of each candidate. Your main will call four functions: read_file(), sort(), show_table, and declare_winner(). Your main() may have nothing in it but variable declarations, the four function calls, and the return statement. There will always be five precincts and four candidates. Your output should match mine exactly. As a reminder, here is a way to test for differences between your output and mine.

```
election.out election1.txt > mine.txt
~davis/30/p5/election.out election1.txt > seans.txt
diff mine.txt seans.txt
```

Hints: By making the number of rows in your matrix one larger than needed, you can store the total votes for each candidate in the extra row. Write show_table() before writing sort() so that you can make sure you are reading in the values correctly. Because you will re-ordering the columns in the matrix, you need a way to keep track of the candidate letters while you are sorting so your labels are correct. Have a char array, *names*, that contains 5 elements that are initially “ABCD”. As you swap the columns in the matrix in your sort() function, swap the letters in *names* at the same time. You can then use your *names* array to help print the Candidate labels.

```
[ssdavis@lect1 p5]$ cat election1.txt
1 192 48 206 37
2 147 90 312 21
3 186 12 121 38
4 114 21 408 39
5 267 13 382 29
[ssdavis@lect1 p5]$ election.out
[ssdavis@lect1 p5]$ election.out election1.txt
           Candidate  Candidate  Candidate  Candidate
Precinct       C          A          B          D
   1          206        192        48         37
   2          312        147        90         21
   3          121        186         12         38
   4          408        114         21         39
   5          382        267         13         29
Total:       1429        906        184        164

Candidate C is the winner.
[ssdavis@lect1 p5]$
[ssdavis@lect1 p5]$ cat election2.txt
1 192 48 206 37
2 147 90 312 21
3 186 12 121 38
4 114 21 108 39
5 267 13 382 29
[ssdavis@lect1 p5]$ election.out election2.txt
           Candidate  Candidate  Candidate  Candidate
Precinct       C          A          B          D
   1          206        192         48         37
   2          312        147         90         21
   3          121        186         12         38
   4          108        114         21         39
   5          382        267         13         29
Total:       1129        906        184        164

Candidates C and A will have a runoff.
[ssdavis@lect1 p5]$
[ssdavis@lect1 p5]$ cat election3.txt
1 192 48 206 37
2 147 90 312 21
3 186 12 121 38
4 114 21 108 39
5 267 13 82 29
[ssdavis@lect1 p5]$
[ssdavis@lect1 p5]$ election.out election3.txt
           Candidate  Candidate  Candidate  Candidate
Precinct       A          C          B          D
   1          192        206         48         37
   2          147        312         90         21
   3          186        121         12         38
   4          114        108         21         39
   5          267         82         13         29
Total:        906        829        184        164

Candidates A and C will have a runoff.
[ssdavis@lect1 p5]$
```