ECS30 **Programming Assignment #4** **Spring 2015**

Due: 5-27-2014 (May 27, 2015)

Filenames: linegraph.c, insertionsort.c, indexarray.c

Add your Name and UC Davis ID number as the first comment of each file to make it easier for TA's to identify your files.

All programs should compile with no errors. Warnings are okay, as long as execution is still correct. Compile programs using "gccx filename.c", then execute with "./a.out" or specify your executable filename during compilation using "gccx -o outputFile filename.c" and execute with "./outputFile".

Use *diff* with the *_output files to compare the output of your executable files with the official output files. (Syntax: "diff file1 file2") *diff* will compare two files line by line, and list the changes that need to be made to the first file to make it identical to the second. If there are no differences between the two files, then *diff* outputs nothing.

Submitting files
Usage: handin cs30 assignmentName files...

The assignmentNames are Proj1, Proj2, Proj3, Proj4, Proj5. **This is Proj4**. Do not hand in files to directories other than Proj4, they will not be graded. The files are the names of the files you wish to submit. For this assignment, your files will be linegraph.c, insertionsort.c, indexarray.c.

Therefore,

you would type from the proper directory:

>>handin cs30 Proj4 programName.c

Do NOT zip or compress the files in any way. Just submit your source code.
>> handin cs30 Proj4 linegraph.c, insertionsort.c, indexarray.c

If you resubmit a file with the same name as one you had previously submitted, the new file will overwrite the old file. This is handy when you suddenly discover a bug after you have used handin. You have up to 4 late days to turn in files, with a 10% deduction for each late day. The deadline for each project (and late day) is 11:59pm.

TA comments:
If it makes sense to put code into a function, use function definition. Otherwise, you can write your code in main, you do not HAVE to use functions for each of these problems. Just to be clear, this does not mean hard code the print statements into your main function. It just meant that if it makes sense to move part of your code into a separate function, do so, otherwise, it's okay to keep all the code in main.

Some of you have been enabling c99 in your compilers so you can do things like:

```
for (int i-0; i< NUM; i++)
```

instead of:

```
int i;
for (i=0;i<NUM;i++)
```

Do NOT do this. For consistency we will not be enabling c99 during your grading scripts, so this will result in an error and you WILL LOSE POINTS. Please just write the extra line of code.

OUTPUT FILES and DIFF:

For this project we are giving .out files for all the problems, and input files when needed. These are EXECUTABLE FILES, NOT TEXT files. You will need to pipe the outputs of these files to an output text file as well as the outputs of your programs into a separate output text file for diff comparison. For example:

If you are working on hello.c:

Assume the official provided file is **hello.out**, and your hello.c file generates **hello.**

To compare the outputs of the two .out files, you need to:

$ ./hello > myHello.txt
$ ./hello.out > hello.txt
$ diff myHello.txt hello.txt

If there were no differences, nothing will print when you run 'diff'. If you get a "Permission denied" error for any of the files (ex. hello.out), try *$ chmod 777 hello.out* in the console/terminal.

For the problem with required input file, do the following (ex. inchtocm.c):

$ ./inchtocm < inchtocm_input > myInchtocm.txt
$ ./inchtocm.out < inchtocm_input > inchtocm.txt
$ diff myInchtocm.txt inchtocm.txt

$ diff –w myInchtocm.txt inchtocm.txt (ignore differences of white spaces) - used for grading

Checking with diff is very important as we will be using scripts to grade your homeworks, and even the smallest difference will result in your not getting points for a problem. Also note that checking with diff does not automatically guarantee 100% score on an assignment. We give a sample of tests, you should check other tests yourselves as well.

Some comments about how to properly format printing are included below as "TA comments".

FOR THE GRAPHICS PROGRAMS: All the graphics programs are optional. You should still do them, but they will NOT be graded. Each program that is optional will have the word (OPTIONAL) next to it. ***Ch11: Problem 9 is a graphics program and is OPTIONAL.***

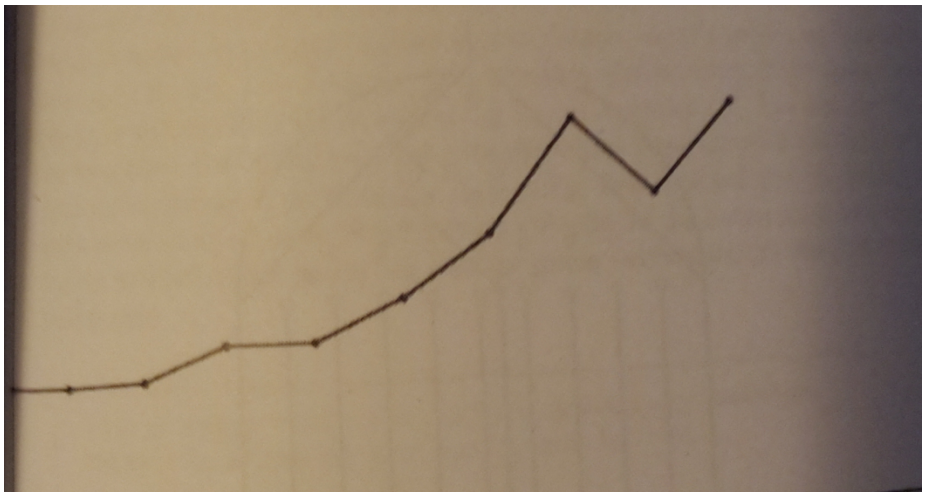
Programming project #4: Ch. 11: 9; Ch. 12: 12; Ch. 13: 5
Programs reproduced below for your convenience: (from the Programming Exercises section of each chapter)

CH.11: (pg. 416-424)

9. filename: linegraph.c *(OPTIONAL)* **(sample files given: linegraph.out, linegraph_input, linegraph.png)**
When you are trying to represent the behavior of some quantity that varies over time, one of the usual tools is the **line graph**, in which a set of data values are plotted on an x-y grid with each pair of adjacent points connected by a straight-line. For example, given the following set of 10 points:

(0.0, 0.67)
(0.4, 0.68)
(0.8,0.71)
(1.2,0.86)
(1.6,0.86)
(2.0,1.04)
(2.4,1.30)
(2.8,1.81)
(3.2,1.46)
(3.6,1.86)

The line graph that represents them looks like this:



Write a function *DrawLineGraph* that generates a line graph given an array of x-coordinate values, a second array of corresponding y-coordinate values, and the number of data points.

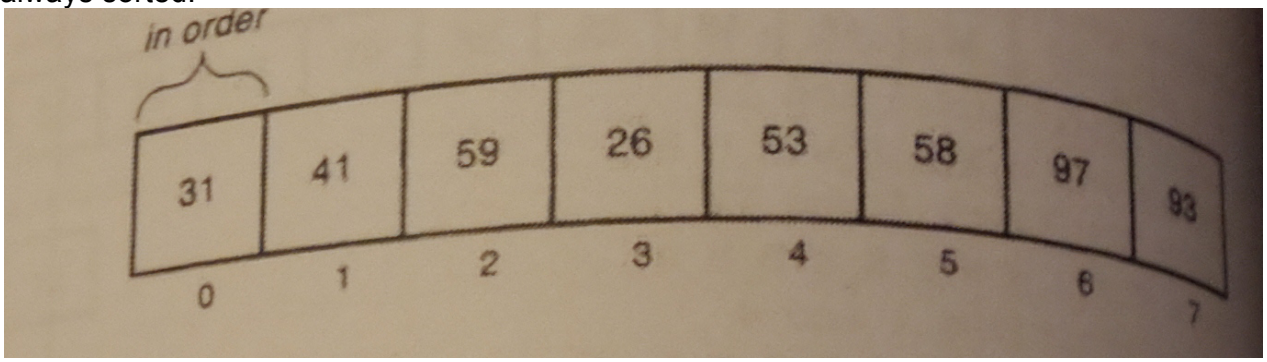
TA comment: try using the StringToReal(arg) function from strlib.h.

CH.12: (pg. 447-452)

12. filename: insertionsort.c (insertionsort.out, insertion_input provided)
Please use the insertionsort.out file to figure out the different formatting scenarios. Make sure to press enter after each number in the list you want to sort.

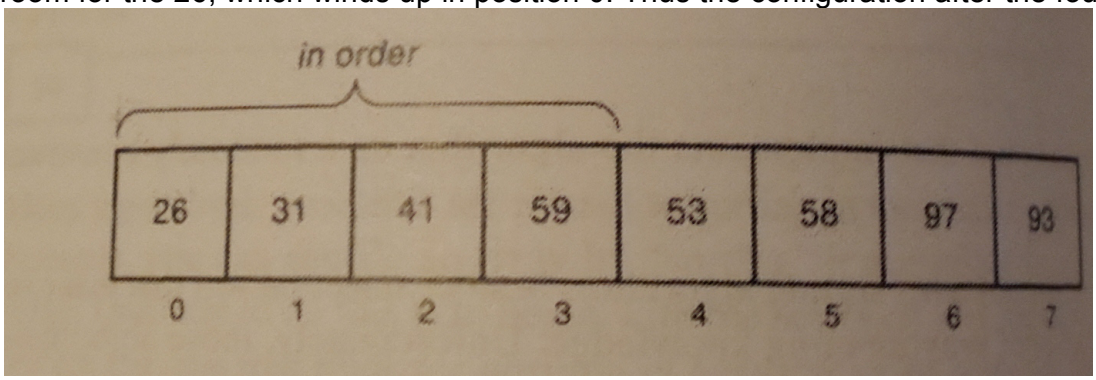
Another sorting algorithm – insertion sort – operates as follows. You go through each element in the array in turn, as with the selection sort algorithm. At each step in the process, however, your goal is not to find the smallest value remaining value and switch it into its correct position, but rather to ensure that the values you've covered so far in the array are correctly ordered with respect to each other. Although those values may shift as more elements are processed, they form an ordered sequence in and of themselves.

For example, if you consider again the data used in the selection sort discussion, the first cycle of the insertion sort algorithm requires no work because an array of one element is always sorted:



The next two cycles of the main loop also require no rearrangement of the array, because the sequence 31-41-59 forms an ordered subarray.

The first significant operation occurs on the next cycle, when you need to fit 26 into this sequence. To find where 26 should go, you need to move backward through the earlier elements, which you know are in order with respect to each other, looking for the position where 26 belongs. At each step, you need to shift the other elements over one position to make room for the 26, which winds up in position 0. Thus the configuration after the fourth cycles is



On each subsequent step, you again insert the next element in the array into its proper position in the initial subarray, which is always sorted at the end of each step.

The insertion sort algorithm is particularly efficient if the array is already more or less in the correct order. It therefore makes sense to use insertion sort to restore order to a large array I which only a few elements are out of sequence.

Reimplement the *SortIntegerArray* function using the insertion sort algorithm.

TA comment: Assume that you will only input integers, you do not have to worry about floating point values. Therefore, when you print out the unsorted and sorted integer arrays, use %d.

TA comment: the program should accept is list of integers, with an "Enter" after each number, and a blank line with "Enter" indicating end of list. Then print out the two statements as follows:
The input array is: a, c, b
The sorted array is: a, b, c
…with a,b,c representing 3 input values. Please see the sample .out file for more details.

CH.13: (pg. 486-489)
5. filename: indexarray.c (indexarray.out, indexarray_input provided)
Please use the indexarray.out file to figure out how your output should be formatted.

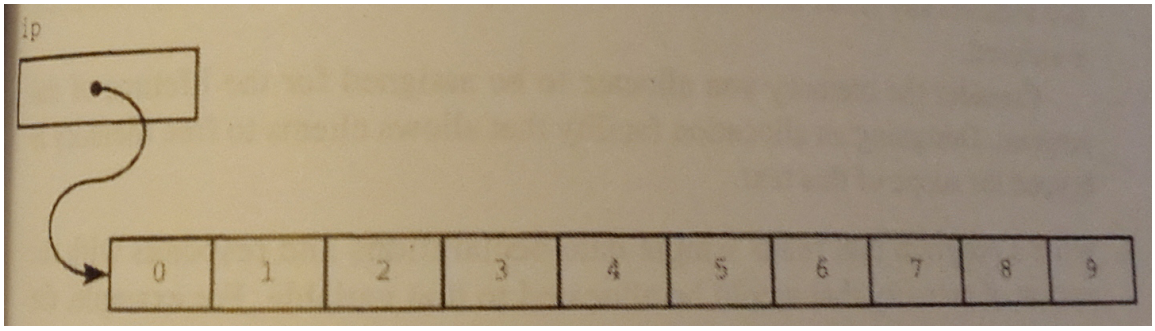
Write a function *IndexArray(n)* that returns a pointer to a dynamically allocated integer array with *n* elements, each of which is initialized to its own index. For example, assuming that *ip* is declared as

*int *ip;*

the statement

ip = IndexArray(10);

should produce the following memory configuration:



TA comment: You can, of course, assume that each index is an integer and use %d to print each value of the array. You need to print out the value of each datum in the array. Again use the sample .out file for exact specifications.