ECE 209 Program 3: `phones`

Due Friday, April 24 @ 11:45pm

This programming assignment must be completed individually. Do not share your code with or receive code from any other student. Evidence of copying or other unauthorized collaboration will be investigated as a potential academic integrity violation. **The minimum penalty for cheating on a programming assignment is a grade of -100 on the assignment.** If you are tempted to copy because you're running late, or don't know what you're doing, you will be better off missing the assignment and taking a zero. Providing your code to someone is cheating, just as much as copying someone else's work.

DO NOT copy code from the Internet, or use programs found online or in textbooks as a "starting point" for your code. Your job is to design and write this program from scratch, on your own. Evidence of using external code from any source will be investigated as a potential academic integrity violation.

This program will search through a text file for phone numbers, and will then sort the phone numbers in ascending order.

The learning objectives of the program are:

- Use C functions to solve a problem.
- Read from a text file.
- Sort the elements of an array

Problem Description

The program's job is to search for phone numbers in a text file. The file may be of any format and of any length, but it will be a simple text file. For the purposes of this assignment, a phone number must be in one of the following formats, where 'x' represents any decimal digit (0-9):

```
x-xxxx
xxx-xxxx
xxx-xxx-xxxx
(xxx)xxx-xxxx
```

Note that there is never a space in the middle of a phone number. The phone number cannot be combined with another string -- in other words, except for the very beginning or end of a file, there must be at least one whitespace character before and after the string. (This means that a phone number that is part of a comma-separated list, or at the end of a sentence, must not be detected by this program. This simplifies the specification and the coding.)

A phone number has three components:

- *Area code:* first three digits in a 10-digit number, otherwise 0.
- *Prefix:* the three digits after the area code. For numbers less than 10 digits, this is 1- or 3-digit field before the hyphen.
- *Line number:* the last four digits of every phone number.

Program Specification

Your assignment will be to implement specific functions in a single source code file, named **phones.c**. You will be provided with 3 files: **main.c, phones.h** and **phones.c**. The **main** function will be provided, but you must implement at least three additional functions, described below. You may choose to implement additional functions as part of your implementation.

Important: Any code you implement should be written in the phones.c file. Failing to comply may cause your code to fail testing.

From a user's perspective, the program simply extracts all (actually, up to100) of the phone numbers from the file, according to the format described above, and prints them in sorted order. See the ***Details: Sorting*** section to learn exactly what "sorted order" means. (Don't assume that you know -- read the spec!)

There is a limit to the number of phone numbers, because we need to allocate space to store them. However, your functions must not assume that the limit is 100. The limit will be passed to your functions as an argument, so the code is more flexible. (And your functions will be tested using different limits.)

Within the main function, here's what happens:

1. An array of pointers (char *) is allocated and initialized, so that all pointers are NULL.
2. The array, limit, and file name are passed to the **findPhoneNumbers** function, which will open the file and find the phone numbers. When the function returns, each array elements points to a phone number string, in the order in which the numbers were read from the file. (Unused pointers remain NULL.) The return value is the number of phone numbers that were extracted.
3. The **main** function will verify that only phone number strings are in the array, by calling **isPhoneNumber** for each. (You should also call **isPhoneNumber** from within the **findPhoneNumbers** function, but this way I can be sure that the function is called.)
4. The string array is the passed to the **sortPhoneNumbers** function. This will rearrange the pointers in the array, so that the corresponding strings are sorted from smallest to largest. (See **Details: Sorting**.)
5. The main function calls **printPhones** to print the sorted array of phone strings. This function is provided for you.

As with Program 2, your grade will depend only on the correctness of your functions, not on how the **main** function works. Feel free to change the **main** function during your testing and development. But any changes to **main** will be ignored during grading.

Details: Sorting

The program must follow these rules for sorting the phone numbers:

- First, sort by area code. Seven- and five-digit numbers have an area code of zero. The format of the area code --- "(xxx)" vs. "xxx-" --- doesn't matter.
- Next, sort by prefix.
- Next, sort by line number.
- Finally, if all the components are equal, sort by string length. (E.g., 003-2508 should come after 3-2508.)

In other words, all numbers with the same area code will be grouped together, and then sorted by prefix within that group. All numbers with the same prefix and area code will be grouped together, and sorted by line number within that group. (See the Appendix for examples.)

This *does not* mean that your code should do multiple passes of the sorting algorithm. One pass will be sufficient. This simply specifies the rules for determining the order of the numbers: whether one number is less than, greater than, or equal to another.

Details: Functions

This section gives the prototype (declaration) for each function[1] that you must implement, as well as a description about what that function must do. You may not change the prototype, and the function must behave as specified. We will be testing each function individually. You may declare and define other functions.

```
int findPhoneNumbers(const char *filename, char * phones[], int limit);
```

The caller passes in the file name (a string), an array to store the phone numbers (an array of pointers), and the size of that array. The phone number array does not have storage for the phone number strings, just the pointers. So, your function must allocate an array for each string using **malloc**. (Hint: Make sure to allocate the proper number of characters to store the string!)

The return value is the number of strings that were extracted from the file.

If the file cannot be opened, the function returns zero. (The caller will not be able to distinguish a file error from a file with no phone numbers.)

Don't forget to close the file after reading it. You will lose points if you don't.

Call **isPhoneNumber** to check whether each string read from the file is or is not a phone number. Do not implement this decision in **findPhoneNumbers** directly.

Hint: Don't **malloc** an array until you know that the string is a phone number. Use a statically-allocated array to read the next string. Then check if it's a phone number. If it isn't, nothing else to do. If it is a phone number, then **malloc** a new array and add the pointer to the array. For the static array, assume that there will no ridiculously long strings in the file; a size of 50 or 100 characters will be plenty. (This is not safe in general, but ok for this assignment. Think about how your code might change if you have to deal with the possibility of arbitrarily-long strings.)

```
int isPhoneNumber(const char * str);
```

Returns 1 if the string is a phone number, and 0 otherwise.

Make sure your string meets ALL the requirements for a phone number, including no trailing non-whitespace characters. (E.g., "3-2508x" is not a phone number.)

```
void sortPhoneNumbers(char * phones[], int n);
```

[1] In case the prototype in the document does not match the prototype in the code that I provide, go with the code. This is a prime location for a specification error to creep in, because sometimes I adjust the code without coming back to change this spec. If there's a discrepancy, ask on Piazza, but assume that the code version is correct.

The caller passes in an array of pointers, each pointing to a phone number string. The second argument is the number of strings to sort. The function sorts the string pointers according to the rules described above. There is no return value.

Hint: Create a separate function to decide whether one phone number string is less than another. This will greatly simplify your sorting code, which should look very similar to the sorting code covered in class. (Use either selection sort or quicksort.)

You *may not* use the qsort function provided by the C standard library. You will get zero points for sorting if you do this. One learning objective of this assignment is for you to write your own sorting code.

Running the Program

For this program, the input file name must be provided as a command-line argument. Instead prompting the user for a file name, the user writes the file name on the command line when she invokes the program. For Linux, this looks just like it sounds:

```
    ./phones file.txt
```

For CLion, you have to go to "Edit Configuration" to set/change the command-line arguments. This can be found in two places: (1) Run > Edit Configurations... (2) Click on the pull-down menu next to the Run and Build icons, and select Edit Configurations.

In either case, type the file name in the box labelled Program Arguments.

You will also need to set the Working Directory to be your project directory, so that you can put the input files in the same directory as your source code.

Testing the Program

I have provided four different test files for your use.

File	Description
randomPhones.txt	Just phone numbers, with a mixture of formats.
airlines.txt	A directory of airlines and phone numbers. Mixed text and phone numbers. All phone numbers are in the same format.
gov.txt	A directory of contact numbers for the US government. More than 100 phone numbers. A mixture of text, various numbers, and phone numbers. All phone numbers are in a single format.
rhapsody.txt	A mix of normal words and phone numbers. Mixed phone number formats.

Suggested Development Plan

To implement this program, I suggest the following plan.

1. Create a small test file with only a few random phone numbers, and only one format type (e.g., x-xxxx). Implement **findPhoneNumbers**, and set **isPhoneNumber** to always return 1. Don't sort. Make sure that the strings are being read and printed.
2. Implement **sortPhoneNumbers** for the one format that you chose in Step 1. Use the same test file to check whether the output is sorted properly. (Make sure your input file is not in sorted order.)
3. Add in different phone number formats, adjusting **findPhoneNumbers** and **sortPhoneNumbers** to handle the different formats. Use the **randomPhones.txt** file to test with a larger number of inputs.
4. Create a new test file, with phone number of one format and also non-phone text. Implement **isPhoneNumber** to distinguish phone numbers (in this one format) from other text. If you pick the xxx-xxx-xxxx format first, you can use **airlines.txt** and **gov.txt** to test your code.
5. Incrementally add new formats to your small test file, and enhance **isPhoneNumber** to recognize the new formats. When all four formats are implemented, use **randomPhones.txt** and **rhapsody.txt** to test your code.

Hints and Suggestions

- Don't overcomplicate the program.
- Work incrementally. See the suggested development plan.
- You must include **stdio.h** for **printf/scanf** and **stdlib.h** for **malloc**.
- For compiler errors, look at the source code statement mentioned in the error. Try to figure out what the error is telling you. Try to fix the first error first, and then recompile. Sometimes, fixing the first error will make all the other errors go away. (Because the compiler got confused after the first error.)
- Use a source-level debugger to step through the program if the program behavior is not correct. If you are using CLion on your own computer, the debugger is integrated with the editor and compiler, so there's no excuse for not using it.
- For general questions or clarifications, use the Message Board, so that other students can see your question and the answer. For code-specific questions, email your code (as an attachment) to one of the TAs.

Administrative Info

Updates or clarifications on Piazza:

Any corrections or clarifications to this program spec will be posted on Piazza. It is important that you read these postings, so that your program will match the updated specification.

What to turn in:

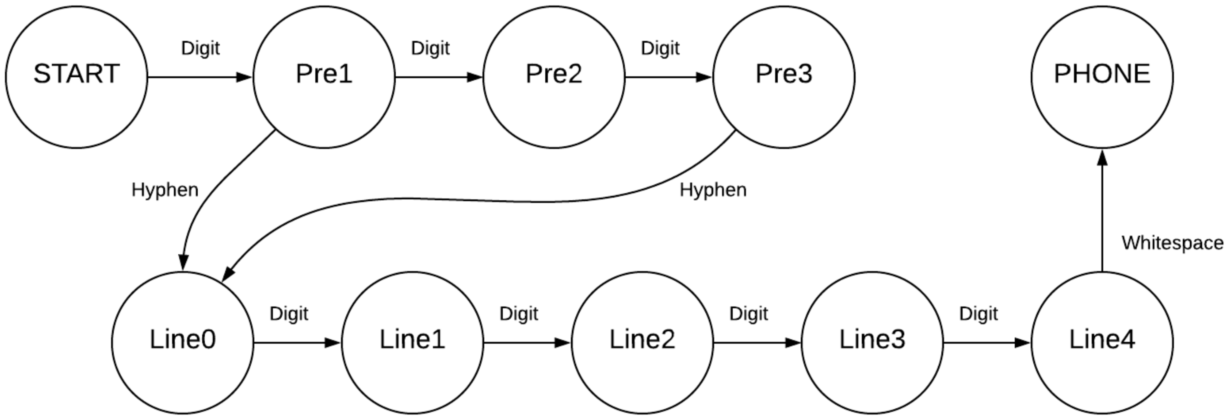
- Source file – it must be named **phones.c**. Submit via zyBook to the Program 3 - Phones assignment.

Grading criteria:

20 points:	File submitted with the proper name. (This is all or nothing. Either you will get all 20 points, or you will get zero. Be sure your file is named correctly, and remember that case matters. No exceptions!!!)
10 points:	Compiles with no warnings and no errors (using –Wall and –pedantic). (If the program is not complete, then only partial credit is available here. In other words, you won't get 20 points for compiling a few trivial lines of code.)
10 points:	Proper coding style, comments, and headers. No unnecessary global variables. No goto. (See the Programming Assignments section on Moodle for more information.) For this assignment, there is no reason to use any global variable.
16 points:	**findPhoneNumbers** works correctly. Must close file for full credit. 10 pts -- reads until end of file OR limit is reached 6 pts -- uses **isPhoneNumber** to read ONLY phone numbers. (You will get credit for doing this correctly, even if your **isPhoneNumber** function does not work.)
20 points:	**sortPhoneNumbers** works correctly. 8 pts -- correctly sorted using a simple string comparison. 12 pts -- correctly sorted as specified, by area code, prefix, and line number. No credit if qsort function (from standard C library) is used.
24 points:	**isPhoneNumber** works correctly. 4 pts for each format. 8 pts for recognizing all non-phone-number strings.

Appendix: Identifying a Phone Number

There are, of course, many different ways to solve the problem of verifying a phone number string. But one suggestion is to think of the problem as a state machine -- each character in the string is an input, and the state transitions get you closer and closer to the goal. Below is an example of a state diagram, which would identify either the 5-digit or the 7-digit format. Any input character that does not match the transition would result in a failure -- no phone number detected.



The state can be kept in a single variable, or as a combination of variables (e.g., a counter for the number of digits, flags to track whether hyphen has been seen, etc.) The idea is to have some representation of what has been seen, and then act upon the next character accordingly.

Appendix: CLion Tips

If you are using CLion as your development environment, here are some tips to make your life easier for this program.

Using the provided source code

I am giving you two source code files (**main.c** & **phones.c**) and a header file named **phones.h**. The main function is within **main.c** and performs the general flow of the program. All required function declarations and descriptions are located in **phones.h. phones.c** will contain all function definitions. Initially **phones.c** will contain dummy implementations of all the required functions. Your task is to rewrite their definition. You are allowed to implement any additional function you may need to complete the task. **DO NOT WRITE YOUR CODE IN main.c OR phones.h.**

To use these in CLion:

1. Create a new CLion project (C executable, C90 standard). You can call the project anything you want. The project name is also the name of the directory that will be used to hold the project files.
2. Open the project in CLion.
3. Download **main.c, phones.c and phones.h** into the project directory.

4. Edit the **CMakeLists.txt** file by adding “phones.c” to your executables, eg add_executable(PROJECTNAME main.c phones.c). Now the project will compile **phones.c**, and you won’t have to remember to change the file name when you submit your code.
5. While you’re in there, you should also set the compiler flags:
   `set(CMAKE_C_FLAGS "-Wall -pedantic")`

Downloading text files

To keep things simple, you want to download the text files into the same directory as your source code files -- the main project directory. The problem: If you do that, then your program won’t be able to find the file when you open it, because it is looking in the directory where the executable resides.

To fix this, you should set the working directory for your project. That will tell the program to look in that directory, instead of the default.

1. With your project open, select Run > Edit Configurations...
2. Next to Working Directory, click the box with ...
3. Navigate to and select your project directory.

You can choose a different directory, if you want, but to me it’s convenient to have the files in the same directory as the source code for this program.

Appendix: Example Runs

Here are the sorted outputs for each of the test files. Output is shown in multiple columns to save space, but of course it will not be printed this way on your console.

```
Enter file name:
airlines.txt
  204-888-2665
  702-505-8888
  800-227-3247
  800-237-6639
  800-267-1247
  800-359-6786
  800-361-2965
  800-388-1105
  800-433-7300
  800-435-9792
  800-455-2720
  800-538-2583
  800-547-9308
  800-567-6567
  800-661-0407
  800-665-0212
  800-803-9943
  800-839-2256
  800-864-8331
  801-401-2222
  801-401-9000
  801-401-9100
  807-577-1141
  855-865-2747
  866-285-8307
  866-516-1685
  866-847-1112
  867-874-3333
  877-359-8474
  877-426-4537
  888-247-2262
  888-247-2262
  888-545-6794
  888-619-8622
  888-935-9848
  888-937-8538
```

```
Enter file name: gov.txt
 202-357-5098
 202-357-5100
 202-357-5182
 202-357-5205
 202-357-5250
 202-357-5263
 202-357-5287
 202-357-5300
 202-357-5313
 202-357-5464
 202-357-5696
 202-357-5900
 202-357-5900
 202-357-7458
 202-357-7467
 202-741-5771
 202-741-6002
 202-741-6025
 202-741-6057
 202-741-6100
 206-336-5124
 206-336-5129
 206-336-5143
 215-305-2011
 301-512-1617
 301-778-1553
 301-837-0294
 301-837-0366
 301-837-0366
 301-837-0366
 301-837-0407
 301-837-0450
 301-837-0475
 301-837-0643
 301-837-0730
 301-837-0760
 301-837-0855
 301-837-0880
 301-837-0923
 301-837-0939
 301-837-0976
 301-837-1539
 301-837-1539
 301-837-1546
 301-837-1668
 301-837-1706
 301-837-1710
 301-837-1740
 301-837-1740
 301-837-1744
 301-837-1744
 301-837-1795
 301-837-1895
 301-837-1948
 301-837-1981
 301-837-1981
 301-837-2010
 301-837-2029
 301-837-2039
 301-837-2039
 301-837-2043
 301-837-2929
 301-837-3000
 301-837-3022
 301-837-3026
 301-837-3064
 301-837-3109
 301-837-3110
 301-837-3115
 301-837-3270
 301-837-3557
 301-837-3754
 301-837-3754
 303-604-4763
 314-801-0512
 314-801-0582
 314-801-0587
 314-801-0602
 314-801-0647
 314-801-0797
 314-801-0871
 314-801-1909
 314-801-2505
 314-801-9141
 404-736-2825
 404-736-2888
 413-236-3606
 618-935-3005
 618-935-3010
 773-948-9007
 781-663-0139
 816-268-8149
 816-994-1702
 817-551-2003
 817-551-2003
 913-825-7809
 937-425-0601
 937-425-0661
 951-956-2015
 951-956-2016
```

```
Enter file name:
randomPhones.txt
       2-8593
       3-0376
       4-6920
       5-3321
       5-6883
       5-9016
       6-6163
       6-8297
       7-6903
       8-1373
       8-8106
       8-9515
       9-4699
     200-2391
     216-8948
     218-5084
     238-0793
     307-7205
     346-4168
     460-1504
     590-5121
     649-8410
(127)336-0590
 170-114-1277
 202-929-2415
 227-510-0024
(459)860-7174
(464)066-1219
 476-716-3746
 581-400-3465
(628)765-2458
(666)421-1210
(731)918-2575
 750-886-5830
 755-562-8290
 773-200-5465
(783)646-0592
 883-936-5595
 930-102-8350
(930)538-9149

Enter file name:
rhapsody.txt
       2-9408
       3-2148
       4-7041
       5-0874
       5-6788
       6-5102
       8-2266
       8-4306
       8-4823
     042-8776
     113-6227
     129-6276
     260-6925
     268-5675
     416-2361
     429-4890
     882-6026
     975-5487
(155)531-7641
(162)837-1058
(165)429-4249
(182)267-4402
(200)217-8522
 340-784-2550
 353-340-3091
 358-769-5692
 515-906-4384
 591-398-0096
(639)767-3943
(690)321-8008
 733-995-8047
 748-278-4207
(751)243-5185
 762-572-7237
(901)537-8597
(907)759-2445
(965)604-8505
```