Arithmetic and Decision-making in C — Partitioning Digits of an Integer

Given the integer value N = 73427347, and the pivot value 4, you could separate all digits of N that are less than 4 from all digits of N that are larger than 4, with any occurrences of 4 in the middle, yielding 32344777. Of course, you could also put the digits that are smaller than the pivot value after the pivot values instead, yielding 77744323.

For this assignment, you will use very basic C techniques to implement a C function to partition the digits of the base-10 representation of a nonnegative integer, relative to a given pivot value, as described above. Some examples:

N	ordering	Pivot	Result
954645	LOFIRST	6	544569
954645	HIFIRST	6	965445
333	LOFIRST	3	333
333	HIFIRST	3	333
84417	LOFIRST	4	14487
84417	HIFIRST	4	87441

You will provide an implementation for a C function that performs this calculation:

```
// Rather than use a nondescriptive label, or none at all, we will use
// an enumerated type to make the logic of of the code clearer:
enum FilterAction {HIFIRST, LOFIRST};

/**  Computes a new integer from N by separating all digits of N that
 *   are smaller than a specified value from those that are larger, and
 *   placing the sets of digits (less, equal, larger) in an order
 *   specified by one of the FilterAction values.
 *
 * For example:
 *
 *     {954645,  LOFIRST,  6}  -->   544569
 *     {954645,  HIFIRST,  6}  -->   965445
 *     {   333,  LOFIRST,  3}  -->      333
 *     {   333,  HIFIRST,  3}  -->      333
 *
 * Pre:  N is initialized
 *       action is HIFIRST or LOFIRST
 *       pivot is between 0 and 9, inclusive
 * Returns:  integer obtained by separating the digits of N as described
 *
 * Restrictions:
 *   - uses only its parameters and local automatic variables
 *     (i.e., no global variables)
 *   - does not make any use of character variables
 *   - does not make any use of arrays
 *   - does not read input or write output
 */
uint32_t FilterDigits(uint32_t N, enum FilterAction action, uint8_t pivot);
```

Make useful comments in your implementation of `FilterDigits()`. The same general guidelines for commenting that you have been taught in your Java courses should provide sufficient guidance.

Getting Started

A tar file is available, containing the testing/grading code that will be used to evaluate your solution:

`driver.c`	test driver... read the comments!
`FilterDigits.h`	header file declaring the specified function... do not modify!
`FilterDigits.c`	C source file for implementing the specified function
`Generator.h`	header file declaring the test case generator... do not modify!
`Generator.o`	64-bit object file containing the test case generator
`checkAnswer.h`	header file declaring the result checking code... do not modify!
`checkAnswer.o`	64-bit object file containing the result checker

Download the tar file `C01Files.tar` from the course website and save it on your CentOS 7 installation (or on rlogin), in a directory created for this assignment. Unpack the tar file there. You can do this by executing the command:

```
CentOS> tar xf C01Files.tar
```

The file `FilterDigits.c` contains a trivial, nonworking implementation of the specified function. You will edit this file to complete the function.

Implementation and Testing

You can compile the given code by executing the command:

```
CentOS> gcc -o driver -std=c11 -Wall driver.c FilterDigits.c Generator.o checkAnswer.o
```

You can run the test/grading code by executing the command:

```
CentOS> ./driver <name for test case file> <name for results file>
```

Read the comments in `driver.c` for more information. `driver` will create a file holding test data using the file name form the command line, and a file showing the results of comparing your results to correct results also using the file name from the command line, which might be something like this once you've completed your solution:

Test data

```
LOFIRST          0  4
HIFIRST          0  4
LOFIRST   54561559  9
HIFIRST   54561559  9
LOFIRST  236246306  0
HIFIRST  236246306  0
LOFIRST        111  1
HIFIRST        111  1
LOFIRST        842  5
HIFIRST        842  5
LOFIRST   58222212  4
HIFIRST   58222212  4
LOFIRST   51507663  3
HIFIRST   51507663  3
LOFIRST    3895190  4
HIFIRST    3895190  4
LOFIRST   32272487  5
HIFIRST   32272487  5
LOFIRST  200308482  4
HIFIRST  200308482  4
```

Test results

```
OK?           Test Case                    Response
-----------------------------------------------
Correct!              0  LOFIRST  4           0
Correct!              0  HIFIRST  4           0
Correct!       54561559  LOFIRST  9    54561559
Correct!       54561559  HIFIRST  9    95456155
Correct!      236246306  LOFIRST  0    23624636
Correct!      236246306  HIFIRST  0   236246360
Correct!            111  LOFIRST  1         111
Correct!            111  HIFIRST  1         111
Correct!            842  LOFIRST  5         428
Correct!            842  HIFIRST  5         842
Correct!       58222212  LOFIRST  4    22221258
Correct!       58222212  HIFIRST  4    58222212
Correct!       51507663  LOFIRST  3    10355766
Correct!       51507663  HIFIRST  3    55766310
Correct!        3895190  LOFIRST  4     3108959
Correct!        3895190  HIFIRST  4     8959310
Correct!       32272487  LOFIRST  5    32224787
Correct!       32272487  HIFIRST  5    78732224
Correct!      200308482  LOFIRST  4   200302488
Correct!      200308482  HIFIRST  4   884200302

Score:  200 / 200
```

If you try this with the original version of `FilterDigits.c`, the code will compile, but the results will be (mostly) incorrect. Each execution of driver will produce a different set of test data, unless you use the `-repeat` switch:

```
CentOS> ./driver <name of test case file> <name of results file> -repeat
```

In that case, the test data file created by a previous run will be reused. That allows you to focus on a fixed set of test cases while you are debugging.

You should test your solution thoroughly; the given testing code generates random test data, and there is no guarantee that it will cover all cases unless you run it a sufficient number of times.

Submission and Grading

You should not submit your solution to the Curator until you can correctly pass tests with the given testing/grading code.

Submit your completed version of `FilterDigits.c`, after making changes and testing. Your submission will be compiled, tested and graded by using the supplied code, but that will be done manually after the due date. A TA will also check to see if your solution violates any of the restrictions given in the header comment for the function; if so, your submission will be assigned a score of zero (0), regardless of how many tests it passes.

If you make multiple submissions of your solution to the Curator, we will grade your last submission. If your last submission is made after the posted due date, a penalty of 10% per day will be applied.

The *Student Guide* and other pertinent information, such as the link to the proper submit page, can be found at:

http://www.cs.vt.edu/curator/

Pledge

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the following pledge statement in the submitted file:

```
//  On my honor:
//
//  - I have not discussed the C language code in my program with
//    anyone other than my instructor or the teaching assistants
//    assigned to this course.
//
//  - I have not used C language code obtained from another student,
//    the Internet, or any other unauthorized source, either modified
//    or unmodified.
//
//  - If any C language code or documentation used in my program
//    was obtained from an authorized source, such as a text book or
//    course notes, that has been clearly noted with a proper citation
//    in the comments of my program.
//
//  - I have not designed this program in such a way as to defeat or
//    interfere with the normal operation of the grading code.
//
//    <Student Name>
//    <Student's VT email PID>
```

We reserve the option of assigning a score of zero to any submission that is undocumented or does not contain this statement.

Implementation Notes

The Standard Library includes a vast library of mathematical functions. One you may be tempted to use is:

```
double pow(double base, double exp);   // returns base to power exp
```

However, this function is inappropriate, since it computes values of type `double`, and we are concerned only with integer values. Aside from that, the function is also unnecessary. We will compile your submission using the command listed earlier in this specification; the use of the C mathematical library requires an additional switch, `-lm`, which we will not use.

Change Log

Any changes or corrections to the specification will be documented here.

Version	Posted	Pg	Change
1.0	Aug 20		Base document.