Problem Statement

Develop a C library for the integers of arbitrary length (intal).

intal

An intal is a nonnegative integer of arbitrary length, but it is sufficient for your implementation to support up to 1000 decimal digits. The integer is stored as a null-terminated string of ASCII characters. An intal is represented as a string of decimal digits ('0' thru '9') that are stored in the big-endian style. That is, the most significant digit is at the head of the string. An integer 25, for example, is stored in a string s as '2' at s[0], '5' at s[1], and null char at s[2].

YourSRN.c

YourSRN.c is your implementation file, which has the definition of all the functions declared in the header file ***intal.h***. If your SRN is PES1201700000 then the filename should be ***PES1201700000.c***. You can use the ***intal_sampletest.c*** for the sanity check. Compile ***intal_sampletest.c*** with ***YourSRN.c*** to sanity-check your implementation. The following commands may give a clearer picture for you.

```
prompt> ls
intal.h          intal_sampletest.c    YourSRN.c

prompt> gcc -Wall YourSRN.c intal_sampletest.c -o mytest.out

prompt> ./mytest.out
```

After you submit ***YourSRN.c***, we are going to assess by running with a hidden test file. That is going to have an exhaustive set of test cases, some may fail even though you could not catch it with the sample tests. Feel free to modify the given ***intal_sampletest.c*** and play with it to make sure your implementation is robust enough. Do not modify ***intal.h*** at any point in time because that is the interface/contract between you and us!

All the functions, whenever they return an intal, it should have stripped off the leading zeros. For example, the difference of "98" and "103" should be returned as "5", not as "05" or "005". However, the integer value of zero is represented as “0”. That is, return “0” for the difference of “35” and “35”. A parameter intal is never invalid or null and has at least a digit and a null termination. Also, intal as a parameter is not going to have any leading zeros. However, the integer value of zero is represented as “0”.

Whenever you are returning an intal, make sure you have allocated memory using **malloc()**. The test function is going to call **free()** on the returned intal immediately after the validation. It is guaranteed even the returning intal value is going to be less than 10^1000. Any other memory allocated by you should be freed by you before returning to the test function. No global variables should be used.

You are allowed to use the library functions from the following header files.
#include <stdio.h>

```
#include <stdlib.h>
#include <string.h>
```

Do not include and use any other third-party library functions. Not even ***math.h*** functions. Other than the definitions of the functions declared in the header file, you can have your own helper functions. Make sure to keep the helper functions by making them "static". I hope you know when we need static functions in C (https://www.tutorialspoint.com/static-functions-in-c). That way your helper functions won't name conflict with any functions used by the test functions we are going to use for the assessment.

YourSRN.txt

Submit a report ***YourSRN.txt*** of the mini-project as a simple ASCII text file. If your SRN is PES1201700000 then the filename should be ***PES1201700000.txt***. The report should contain the approach in implementing all the functions. Try to keep it concise yet covering all the details. Do not include the source code, which is already available in the implementation file. The report should have the following sections.

1. Introduction
    a. The section should answer questions like
        i. What is an intal?
        ii. How is it different from an integer in general and integer data types supported by C?
        iii. Applications of intal.
2. Approach
    a. Your approach in implementing the functionalities of intal.
3. Future work (If you had time and interest)
    a. Any more functionalities you think that can be included in the intal library?
    b. If you had no restrictions, how would you implement a library to handle integers of arbitrary length?

Submission

Deliverables

1. ***YourSRN.c***: Your implementation file, which has the definition of all the functions declared in the header file ***intal.h***. If your SRN is PES1201700000 then the filename should be ***PES1201700000.c***.
2. ***YourSRN.txt***: The report of your implementation. Write the approach in implementing all the functions. Try to keep it concise yet covering all the details. Do not include the source code, which is already available in the implementation file.

Submission

- Submit at the Google Form: https://forms.gle/HFxYw49DGdo8nK6r7 (Submission of the APSSSE Mini-project)
- Deadline: 23:59 PM on 20th of May 2020

FAQs

1. Is there any restriction on the use of macros for example **`#define MAX`** (for finding the max of two elements)?
    a. There are no such restrictions. Make sure you write your own macros, don't use third-party code.
2. Can we use **`typedef`** in our program at a global level?
    a. I don't think there would be any issues. At the moment, I'm not entirely sure of any name conflicts with the tests.
3. Can we use int64_t, int32_t and other fixed sized types, defined in the <inttypes.h> and <stdint.h> library of C99, for our project to get fixed-sized integers?
    a. As you are not allowed to the mentioned header files, you don't get to use those data types. My suggestion is, avoid C99 and stick to basic functionalities of ANSI C level (aka C90).
4. Do we have any time/space constraints other than those specified in the comments of the intal.h file?
    a. Some hints are given in the intal.h itself, which is mostly at the asymptotic complexity level. Memory leaks are discouraged anyway. I don't have a well-defined limitation in terms of absolute time and space limitations at the moment. Try to implement a reasonably good algorithm for the problem without bugs.
5. Is it safe to assume in intal_diff function that intal1 >= intal2?
    a. No.
6. For all the functions can we assume that the inputs have no preceding '0's or should we right strip the inputs as well?
    a. There won't be any leading zeros in the arguments passed by the test functions. The integer zero is, however, will be "0".
7. Will **`intal_multiply()`** accept an O(n^2) solution or it has to be the Karatsuba algorithm?
    a. An O(n^2) solution is acceptable. I feel the Karatsuba algorithm is too much to ask for! Let me know if you have implemented the Karatsuba algorithm.
8. Could we get more edge cases in **`intal_sampletestcase.c`** to make sure our implementation guarantees to pass all the hidden test cases?
    a. No, sample tests are meant for understanding the problem along with a sanity check in terms of checking for compilation errors and making at least one call to each function. I'm not planning to cover all the edge cases which are going to be tested later in the assessment. You should play with the sample test file to test your code for the edge cases.
9.