Design and Analysis of Algorithms (DAA) - UE16CS251
Assignment 1
Develop a C library of an integer of arbitrary length (intal)

FAQs

1. If the inputs to a certain function yield a value that is not in the domain of intal, i.e. nonnegative integers (for example, decrement 1), what is to be done?
    a. Decrement function won't yield a negative number because it returns '0' on decrement of '0' by definition of the function.
    b. There are no functions, which result in a negative number. Even the difference is of two intal 'a' and 'b' is nonnegative (it's the absolute value of a-b). There is no "subtract" function.
2. Is "difference" function just another name for "subtract"?
    a. No, "difference" is always nonnegative. Difference of two numbers 'a' and 'b' is, essentially max{a, b} - min{a, b}. There is no explicit "subtract" function in our library. We are dealing with only nonnegative integers.
3. How do we deal with NaN (not a number) like division by zero?
    a. A null pointer represents a NaN. So, division by zero returns a null pointer. Even intal_create() returns a null pointer if the string is not representing a nonnegative integer.

Problem Definition:

Develop a C-library of an integer of arbitrary length, let us call it as "intal" in short. The functionalities to be implemented in the library are declared in the header file given at the end of the document.

Library "intal", short for integer of arbitray length, a library of nonnegative integers of arbitrary length. The given header file "intal.h" declares the functionalities the library is expected to provide except that there is no definition of the "intal" itself. That is left to the implementation file, which should declare the structure of the intal along with defining the functionalities declared in intal.h. Don't modify intal.h, all of your contribution must be limited to one file; intal.c. When you submit the intal.c, we are going to compile it with intal.h and a client file of our own to test the functionalities.

Client treats an intal (an integer of arbitrary length) as an object pointed by a pointer "void*". An intal can be created by intal_create() by providing a char string of a nonnegative integer provided in decimal digits. Some intals are created out of some functionalities like intal_add(), which creates a new intal. A new intal created must have allocated a dynamic memory (may be by a malloc() call). Responsibility of destroying the intals created lies with the client by calling intal_destroy(), which will free whatever memory allocated during the creation of intal. Client sees an intal as a "void*". It could be a pointer to char array, int array, long int array, double array, or a struct array. There is no theoretical limit to the size of the integer, but memory limitations of the process (Operating System). If the OS allows, your library should be able to hold the largest prime number known, which is 23,249,425 digits long (as of Feb 2018).

Due Date

1st April, 2018 (we are not fooling about it even though it’s a Sunday!).

Deliverables

The only deliverable is **“intal.c”**, which implements all the functionalities declared in the header file **“intal.h”**. After you submit, we will test against multiple clients (test-cases). The sample client file provided should clarify the problem statement.

How to submit?

You will be asked to upload a copy of the “intal.c” file you have written along with printed copy.

Team

It’s an individual effort. Each student needs to work on his/her own code. Plagiarism will be dealt strictly, but you are welcome to discuss the approach to solve a problem.

Assessment

The library source code you submit (intal.c) will be compiled with our test-cases (client files). Assessment will be done on working of each functionality, coverage of border cases, duration of execution, and coding style (readability of the code).

Associated documents:

intal.h

```
// Library "intal" - Integer of arbitray length
// intal is a nonnegative integer of arbitrary length.
// The way the integer is stored is specific to the
//  implementation as long as the interface (this header file) is intact.

// DO NOT modify this header file.
// As usual, an implementation file implements all the functionalities decalred here
//  and a client file uses the functionalities declared here.

//String (array of chars with a null termination) of decimal digits converted to intal
type.
//Input str has most significant digit at the head of the string.
//"void *" abstracts out the format of intal.
//The returned pointer points to the intal "object". Client need not know the format
of the intal.
//Even if you happen to use "char*" as the format of the intal, just like the input
string,
// it's expected to a create a new copy because the intal object should be modifiable,
but
// the input could be a constant literal (that's why parameter is "const").
//The intal created here obviosuly needs some memory allocation, which would be freed
in intal_destroy().
//The memory allocated by this function is pointed by the pointer it returns. The
client has no idea
// what kind of object it is. It could be a pointer to char array, int array, long int
array, double array, or
// a struct array. There is no theoretical limit to the size of the integer, but
memory limitations of the
// process (Operating System). If the OS allows, your library should be able to hold
the largest prime number
// known, which is 23,249,425 digits long (as of Feb 2018).
//Returns "null" is str is not representing a valid nonnegative integer. A "null"
pointer
// represents a NaN (not a number).
void* intal_create(const char* str);

//Destroy the created "object".
//It mainly frees the memory allocated by intal_create().
void intal_destroy(void* intal);

//Converts intal to a string of decimal digits for mostly display purpose.
//Returned string has most significant non-zero digit at the head of the string.
char* intal2str(void* intal);

//Increments the integer by one and returns the incremented intal.
//In most cases, it'll return the same object. But in some cases, it may create a
```

```
// new object to accommodate the incremented value. In that case, this function
// destroys the older intal and returns the new one.
void* intal_increment(void* intal);

//Decrements the integer by one and returns the decremented intal.
//No change if the intal is zero because it is nonnegative integer.
//In most cases, it'll return the same object. But in some cases, it may create a
// new object to accommodate the decremented value. In that case, this function
// destroys the older intal and returns the new one.
void* intal_decrement(void* intal);

//Adds two intals and returns their sum.
void* intal_add(void* intal1, void* intal2);

//Returns the difference (obviously, nonnegative) of two intals.
void* intal_diff(void* intal1, void* intal2);

//Multiplies two intals and returns the product.
void* intal_multiply(void* intal1, void* intal2);

//Integer division
//Returns the integer part of the quotient of intal1/intal2.
//Returns "null" if intal2 is zero. A "null" pointer represents a NaN (not a number).
void* intal_divide(void* intal1, void* intal2);

//Returns -1, 0, +1
//Returns 0 when both are equal.
//Returns +1 when intal1 is greater, and -1 when intal2 is greater.
int intal_compare(void* intal1, void* intal2);

//Returns intal1^intal2.
//It could be a really long integer for higher values of intal2.
//0^n = 0. where n is any intal.
void* intal_pow(void* intal1, void* intal2);
```

Sample client file:

```
//A sample client for intal.h

//Expected output for this client:
/*
First intal: 4999
Second intal: 2001
Two intals after increment and decrement:
5000
2000
Max of two intals: 5000
Sum: 7000
Diff: 3000
Product: 10000000
Quotient: 2
```

```
5000 ^ 2: 25000000
*/

#include <stdio.h>
#include "intal.h"

int main(int argc, char const *argv[]) {
	char *str1 = "4999";
	char *str2 = "2001";
	void *intal1;
	void *intal2;
	void *sum;
	void *diff;
	void *product;
	void *quotient;
	void *exp;

	intal1 = intal_create(str1); //4999
	intal2 = intal_create(str2); //2001

	printf("First intal: %s\n", intal2str(intal1)); //4999
	printf("Second intal: %s\n", intal2str(intal2)); //2001

	intal1 = intal_increment(intal1); //5000
	intal2 = intal_decrement(intal2); //2000

	printf("Two intals after increment and decrement:\n");
	printf("%s\n", intal2str(intal1)); //5000
	printf("%s\n", intal2str(intal2)); //2000

	printf("Max of two intals: %s\n", //5000
		(intal_compare(intal1, intal2) > 0) ? intal2str(intal1) :
intal2str(intal2));

	sum = intal_add(intal1, intal2); //7000
	printf("Sum: %s\n", intal2str(sum));

	diff = intal_diff(intal1, intal2); //3000
	printf("Diff: %s\n", intal2str(diff));

	product = intal_multiply(intal1, intal2); //10000000
	printf("Product: %s\n", intal2str(product));

	quotient = intal_divide(intal1, intal2); //2
	printf("Quotient: %s\n", intal2str(quotient));

	exp = intal_pow(intal1, quotient); //5000^2 = 25000000
	printf("%s ^ %s: %s\n", intal2str(intal1), intal2str(quotient),
intal2str(exp));

	//Make sure you destroy all the intals created.
	intal_destroy(sum);
	intal_destroy(diff);
```

```
        intal_destroy(product);
        intal_destroy(quotient);
        intal_destroy(exp);
        intal_destroy(intal1);
        intal_destroy(intal2);
        return 0;
}
```