CS69003: Computing Systems Lab I
Autumn 2008

Lab Test 1

Time: 2:00pm – 4:00pm, September 05, 2008 (Friday)

The built-in integer data types in C can handle integers of fixed sizes only. For example, a `long int` is capable of storing integers in the range -2^{31} to $2^{31}-1$. For several applications, we have to deal with larger integers (like 100-digit integers). One possibility to store such an integer is by using an array of characters (that is, strings) with each character storing one digit and with the null character `'\0'` indicating the termination of the string. Assume that we are interested in positive integers with ≤ 100 decimal digits.

Your task is to read two such big integers m and n (as character arrays), report which one of these two integers is larger and print the sum $m+n$.

In all the parts, you are allowed to use string library functions or manipulate the strings character by character. Note that the character `'0'` has ASCII value 48, `'1'` has ASCII value 49, ..., and `'9'` has ASCII value 57. If `c` is a character variable storing a decimal digit, then the value of the decimal digit is `c-48` or `c-'0'`. Conversely, you may convert a decimal digit d (an integer in the range $0 \leq d \leq 9$) to the corresponding character as `d+48` or `d+'0'`.

Part 1 **(30)**

Write a function `cmpBigInt()` with the following prototype.

```
int cmpBigInt ( char *m , char *n );
```

The function takes, as input, two character arrays, each representing a big integer in the form specified above. The function outputs 0 if $m = n$, it outputs 1 if $m > n$, and it outputs -1 if $m < n$. For simplicity, you may assume that there are no leading zero digits in the strings m and n.

Part 2 **(50)**

Write a function `addBigInt()` with the following prototype.

```
void addBigInt ( char *m , char *n );
```

The function accepts, as input, two positive integers stored as strings and prints the sum of these integers.

If the lengths of m and n are different, you may add the requisite number of leading zero digits to the shorter operand so as to make the lengths of the two operands equal. Next, you carry out a digit-by-digit sum with appropriate carry adjustments, starting from the least significant end.

As an example, take $m = 9988776655$ and $n = 23456789$. We first make their lengths equal by converting n to 0023456789. Then, we add $9988776655 + 0023456789 = 10012233444$. This example illustrates also the situation where the sum is one digit longer than the operands.

Part 3 **(20)**

Write a `main()` function that reads two positive big integers m, n and stores them as (null-terminated) strings in character arrays. For simplicity, assume that the user supplies a sequence of decimal digits (0–9) with the first digit being non-zero. The maximum number of digits is 100. The `main()` function then calls the functions described in Parts 1 and 2 for comparing and adding the two input integers.

Write your name and roll number (as comment) at the beginning of your C program. Send the file by e-mail with the subject specifying your name and roll number.