**Department of Computing Sciences**

**SUNY Brockport**

**CSC 219 (Programming in C)**

**Lab Exercise 6**

This assignment is expected to provide you with some experience in writing some reasonably complex C programs. For each of the problems indicated below, turn in a file with the error-free listing of your program, and at least three test runs for **each** program.

Each of your C programs should have a comment at the top indicating your name and a brief description of the program's purpose. Additional comments in the body of the program are also encouraged.

1. The maximum integer storable in C has 10 digits or so (even if long int is used). In many applications, such numbers are not big enough. For example, the federal government has to deal with figures in the trillions of dollars (or, more probably, quadrillions!). In this program we want to explore how two *arbitrarily large integers* can be added in a primitive fashion.

The basic strategy you have to follow is to store each digit of the arbitrarily large integer as an array element. These digits are stored *backwards* - therefore, the 0th element of the array will contain the last digit of the integer, the 1st element the second last, and so on. Since we want to add two numbers, we will have two such arrays. A third array will hold the sum of the numbers in these two arrays. The corresponding elements of each of these two arrays will be added to provide the value of the digit of the sum array. Note that the additions of two digits generates a *carry* - if 9 and 9 are added, for example, the result is 18 and the digit in the sum array will be 8 and we will have a carry of 1. This carry will then be added to the sum of the next two elements.

Write a function "add" that will carry out addition in the above manner. Integrate your function with the code below.

#include <stdio.h>

#define N 20

void add(int sum[], int a[], int b[]);

void prn\_num(int a[]);

int main(void)

{

int a[N] = {7, 5, 9, 8, 9, 7, 5, 0, 0, 9, 9, 0, 8, 8};

int b[N] = {7, 7, 5, 3, 1, 2, 8, 8, 9, 6, 7, 7};

int sum[N];

printf("Integer a: ");

prn\_num(a);

printf("Integer b: ");

prn\_num(b);

add(sum, a, b);

printf(" Sum: ");

prn\_num(sum);

return 0;

}

void prn\_num(int a[])

{

int i;

for (i = N - 1; i >= 0; --i) {

if (a[i] == 0)

putchar(' ');

else

break;

}

for (; i >= 0; --i)

printf("%d", a[i]);

putchar('\n');

}

Note that the digits are stored in array elements going from element 0 to element N-1, but they are printed in reverse order.

You have to write the function add. The output, after you have written this function and made a complete program which you have compiled and executed, should look like

Integer a: 88099005798957

Integer b: 776988213577

Sum: 88875994012534

You should then edit this program to try another test case. Present the output of running this second test case in your output file as well. You need not redisplay the (changed) source code in your script file.

FINALLY,

Write an additional function called mul that will effect the *multiplication* of such arbitrarily long integers. Note that the result of this multiplication must be held in a product array whose size is *twice* the size of the operand arrays (arrays a and b above). The multiplication is also done on a digit by digit basis, just like the addition, with the multiplicand (first operand) multiplied with each digit of the multiplier (second operand) to get a *partial product*. This partial product is then *long added* to the "product obtained so far" (which is initially all zeros). This approach is repeated for all digits of the multiplier, except that the partial product obtained using each digit of the multiplier must be *shifted* by the appropriate number of digits –i.e., the partial product obtained using the last digit of the multiplier will not be shifted at all, that obtained using the next to last digit will be shifted by one (meaning one additional 0 will be added as the last digit of the partial product) before being long add-ed, that obtained using the next to next to last digit will be shifted by two (meaning two additional 0s will be added as the last digit of the partial product) before being long add-ed, etc. Note that the function "mul" will use the function "add" above.

Submit a *separate* program that does the multiplication. The output generated by this program should be similar to the one for the long addition program above.

Hint: You may duplicate some of the functions from the addition program in the code of the multiplication program (e.g., the "add" and printing functions), though you have to make *some* changes to them so that they work properly in the multiplication program - they will not be absolutely identical!