SE 2S03 — Assignment 1

Ned Nedialkov

30 September 2019

Due date: 11 October

Problem 1 (10 points) Implement the body of the functions `add` and `sub` given below.

```
void add(const char a[], const char b[], char res[]);
/* Adds two decimal numbers stored as strings and returns their sum.
   Input
     a: an array of characters storing decimal digits; a[0] != ’0’.
     b: an array of characters storing decimal digits; b[0] != ’0’.
   Output
     res: an array of characters storing a decimal number that is
        the sum of the numbers represented in a and b.
 */
void sub(const char a[], const char b[], char res[]);
/* Subtracts two decimal numbers stored as strings and returns their
    difference.
   Input
     a: an array of characters storing decimal digits; a[0] != ’0’.
     b: an array of characters storing decimal digits; b[0] != ’0’.
   Output
     res: an array of characters storing a decimal number that is
        the difference of the numbers represented in a and b.
 */
```

When the following main program is executed

```
#include <stdio.h>
#include <string.h>
#define N 80
char a[N], b[N], res[N + 1];
int main()
{
    char op;
    scanf("%s %s %c", a, b, &op);
    switch (op) {
    case ’+’:
        add(a, b, res);
        break;
    case ’-’:
        sub(a, b, res);
    }
    // print result
    char buf[20];
    int l1 = strlen(a), l2 = strlen(b), l3 = strlen(res);
    int m = l1 > l2 ? l1 : l2;
    m = m > l3 ? m : l3;
    sprintf(buf, "%% %ds\n%% %ds\n%% %ds\n", m, m, m);
    printf(buf, a, b, res);
}
```

it produces (with input on the first line):

```
111 99 +
111
 99
210
```

```
11 9 -
11
 9
 2
```

```
9 121 -
  9
 121
-112
```

```
 99 10000 -
   99
10000
-9901
```

When implementing these two functions, you are not allowed to use

- any additional arrays
- pointer arithmetic
- the string functions from `string.h` except `strlen`

Store `add` and `sub` in file `calc.c`.

Problem 2 (5 points) Given an $n \times n$ matrix, you need to determine if each diagonal of the matrix contains the same value. One such matrix is for example

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 7 & 1 & 2 & 3 \\ 8 & 7 & 1 & 2 \\ 9 & 8 & 7 & 1 \end{bmatrix}$$

Implement the function

```
int is_same_diagonals(int n, int a[]);
/*
  Input:
    n: number of rows and columns
    a: array of size n x n storing an integer matrix
       row wise
  Returns:
    1: if each diagonal of the matrix
       contains the same value
    0: otherwise
 */
```

Store your function in file `matrix.c`.

Note: The matrix

$$\begin{bmatrix} 4 & 3 & 2 & 1 \\ 3 & 2 & 1 & 7 \\ 2 & 1 & 7 & 8 \\ 1 & 7 & 8 & 9 \end{bmatrix}$$

does not qualify as having diagonals with the same value.

Problem 3 (10 points) Consider an $n \times m$ matrix A with entries 0's and 1's. Two zero entries $a_{i,j} = 0$ and $a_{k,l} = 0$, $0 \le i, k \le n-1$ and $0 \le j, l \le m-1$, are *connected* if $a_{k,l}$ can be reached from $a_{i,j}$ (and vice versa) by moving horizontally and vertically in the matrix through 0 entries.

For example if

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix},$$

then $a_{1,1}$ and $a_{0,3}$ are connected through the blue entries. Entries $a_{0,0}$ and $a_{1,1}$ are not connected.

A *path* from (i, j) to (k, l) is a sequence of pairs of indices such that (i, j) and (k, l) are connected, and there are no repeated pairs. The path from $(1, 1)$ to $(0, 3)$ is the sequence

$$(1, 1), (1, 2), (0, 2), (0, 3)$$

If, for example,

$$A = \begin{bmatrix} 0 & 1 & \mathbf{0} & \mathbf{0} \\ 1 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 1 & 0 & 1 & 0 \end{bmatrix}, \tag{1}$$

then there are 2 paths from $(1, 1)$ to $(0, 3)$:

$$\begin{array}{ll} (1,1),(1,2),(0,2),(0,3) & \text{and} \\ (1,1),(1,2),(1,3),(0,3) & \end{array}$$

but

$$(1,1),(1,2),(1,3),(2,3),(1,3),(0,3)$$

is not a path as $(1, 3)$ is repeated.

The number of pairs of indices in a path is its *length*.

Implement the function

```
int find_path(int n, int m, int A[], int i, int j, int k, int l,
              int path[][2])
/*
Input:
  n: number of rows, n>0
  m: number of columns, m>0
  A: an array of size m*n storing an n x m matrix row wise. Each
     entry of A is either 0 or 1.
  i,j,k,l: indices
Returns:
 -1: if entry (i,j) is 1 or entry (k,l) is 1
 -2: if i < 0 || i >= n
 -3: if j < 0 || j >= m
>=0: the length of a path between (i,j) and (k,l)
Output:
  path: if this function returns a value L>0, then
     path[0][0]=i,path[0][1]=j,
     path[L-1][0]=k,path[L-1][1]=l, and
     path[r][0],path[r][1] store the r-th pair of indices on
     this path.
     If the return value is -1, path is ignored, i.e. nothing is
        assigned to it.
 */
```

If this function is called with the matrix in (1) as

```
  int l = find_path(3, 4, A, 1, 1, 0, 3, path);
```

it must produce `l=4` and the array `path` should store either

1	1
1	2
1	3
0	3

or

1	1
1	2
0	2
0	3

Since there can be more than one path, you can return any of them.

- You are not allowed to use any additional arrays in `find_path`.
- The input array `A` should be the same on output. You can change `A` in your function, but then you have to revert the changes.
- Store the implementation of your `find_path` in file `path.c`.

Submit

- The files `calc.c`, `matrix.c`, and `path.c` to Avenue. They must not contain the corresponding main programs.
- Hard copy of these files in class.