# Assignment 2 - Software I, Summer 2003 (0368-2157-20)

`http://www.cs.tau.ac.il/~efif/courses/software1`

Due: Aug. 13, 2003

Before starting to answer the questions, please read very carefully the "Submission Guidelines"[1]. Make sure your programs detect invalid input data, and print out appropriate error messages. Do not add "friendly" messages to your programs, as they are tested automatically by other programs.

## Ex 2.1 reverse1

Write a program that reads input lines and prints each line backwards immediately when the `'\n'` character is reached. A line terminates with the `'\n'` character. The entire input terminates with the EOF. Implement a function with the prototype below that reverses a given input string in place.

```
void reverse(char line[], int len)
```

The program should process only the first 80 characters of each input line.
For example, input:

```
abcd
```

output:

```
dcba
```

## Ex 2.2 reverse2

Write a program that reads input lines and prints each line backwards as in the previous exercise. This time the program should be able to process infinitely long lines using recursion without allocating memory explicitly.

## Ex 2.3 point

Write a program that computes a final intersection point as follows. The program reads a sequence of $1 + 3k$ points for some integer $k$. Each point is given by a pair of $x$ and $y$ floating point coordinates. The program reduces 4 points to a single point repeatedly as described below until all input points are consumed. Each pair of points define a line. The program reduces a set of 4 points to the intersection point of the 2 lines defined by the 4 points respectively.

[1] `http://www.cs.tau.ac.il/~efif/courses/software1`

Verify that the number of points in the input is indeed $1 + 3k$ for some integer $k$.
Use the following type to hold the $x$ and $y$ point coordinates:

```
typedef double Point[2];
```

Print out the result with 6 digits of precision for the fraction part. Print negative zero (-0.000000) as zero (0.000000).
For example, input:

```
0.0 1.0
2.0 1.0
1.0 0.0
1.0 2.0
-1.0 -1.0
-1.0 1.0
1.0 -1.0
```

output:

```
0.000000 0.000000
```

## Ex 2.4 apd

Write a program that computes all pair distances between nodes in an unweighted undirected graph using the version of Seidel's algorithm described below. Let $G$ be an undirected, unweighted, connected graph with vertex set $\{1, 2, \ldots, n\}$ and adjacency matrix $A = (a_{ij})$, where $a_{ij}$ is 1, if nodes $i$ and $j$ are connected with an edge, and 0 otherwise. The program reads an integer $n$ followed by the $n^2$ boolean numbers of an adjacency matrix $A$. It prints a matrix $D = (d_{ij})$, where $d_{ij}$ is the length of the shortest path between nodes $i$ and $j$.

**Algorithm 1** APD - All Pairs Distances

**Require:** $A : n \times n$ boolean matrix
**Ensure:** $D : n \times n$ distance integer-matrix
1: $B \leftarrow (A^2 \cup A) \cap \neg I$ {Boolean Product}
2: **if** $B \cup I = J$ **then**
3: return $D \leftarrow 2B - A$
4: **end if**
5: $C \leftarrow \mathrm{APD}(B)$
6: **for** $r \leftarrow 0, 1, 2$ **do**
7: $C^r \leftarrow (c^r_{ij})$, where $c^r_{ij} = \begin{cases} 1 & c_{ij} + 1 \pmod 3 = r \\ 0 & c_{ij} + 1 \pmod 3 \neq r \end{cases}$
8: $X^r \leftarrow C^r \cdot A$ {Boolean Product}
9: **end for**
10: return $D \leftarrow (d_{ij})$, where $d_{ij} = \begin{cases} 2c_{ij} & x_{ij}^{c_{ij} \pmod 3} = 0 \\ 2c_{ij} - 1 & x_{ij}^{c_{ij} \pmod 3} = 1 \end{cases}$

$I$ is the identity matrix, and $J$ is a matrix of all 1's. You really don't need to understand the algorithm, but if you do want to, keep reading.

The boolean matrix $B$ computed in (1:) is the adjacency matrix of the graph $G'$, obtained by connecting every two vertices $i$ and $j$ by an edge iff there is a path of length 1 or 2 between $i$ and $j$ in $G$. $G'$ is a complete graph iff $G$ has diameter at most 2. In this case $d_{ij} = 2$ if $a_{ij} = 0$ and $d_{ij} = 1$ if $a_{ij} = 1$. Thus, the algorithm returns the correct values in (3:) for graphs of diameter at most 2.

$c_{ij}$ computed in (5:) is the shortest path between $i$ and $j$ in $G'$. If $i = i_0, i_1, \ldots, i_{2s-1}, i_{2s} = j$ is a shortest path in $G$ between $i$ and $j$ for some $s$, then $i = i_0, i_2, \ldots, i_{2s-2}, i_{2s} = j$ is a shortest path between $i$ and $j$ in $G'$. On the other hand, if $i = i_0, i_1, \ldots, i_{2s-1} = j$ is a shortest path in $G$ between $i$ and $j$ for some $s$, then $i = i_0, i_2, \ldots, i_{2s-2}, i_{2s-1} = j$ is a shortest path between $i$ and $j$ in $G'$. Therefore, $d_{ij}$ even implies $d_{ij} = 2c_{ij}$, and $d_{ij}$ odd implies $d_{ij} = 2c_{ij} - 1$.

After $c_{ij}$'s are computed recursively in (5:), we determine the parities of the $d_{ij}$'s in statements (5:) through (10:), in order to deduce their values from the respective $c_{ij}$'s.

Let $k$ be adjacent to $j$ in $G$. If $k$ is on the shortest path between $i$ and $j$, then $d_{ij} = d_{ik} + 1$. Otherwise, $d_{ij} \in \{d_{ik}, d_{ik} - 1\}$. Therefore, once $d_{ik}$ is established, $d_{ij}$ must be in $\{d_{ik} - 1, d_{ik}, d_{ik} + 1\}$. Therefore, we can compute $C^r$ in (7:) for 3 values only, and still have the following property for the Boolean product computed in (8:):

$$x_{ij}^{c_{ij}} \pmod 3 = \begin{cases} 1 & \exists k, c_{ik} = c_{ij} - 1 \text{ and } a_{kj} = 1 \\ 0 & \text{no such } k \text{ exists} \end{cases}$$

Moreover, if $d_{ij} = 2c_{ij} - 1$ is odd, then there is least one $k$, (the one on the shortest path), such that $d_{ik} = 2c_{ij} - 2 = 2c_{ik}$. In other words, $c_{ik} + 1 = c_{ij}$. Similarly, if $d_{ij} = 2c_{ij}$ is even, there is no such a $k$. In other words, $c_{ik}$ must be in $\{c_{ij}, c_{ij} + 1\}$. Therefore, $x_{ij}^{c_{ij}} \pmod 3 = 0$ if and only if $d_{ij}$ is even.

# Good Luck!

## More Information on the Submission

### Files Name

The files for the exercises should be located under `~/software1/assign2`, and their names should match the name of the exercise. For example, the `C` source file and corresponding executable for exercise 2.1, namely reverse1, should be `~/software1/assign2/reverse1.c` and `~/software1/assign2/reverse1` respectively. Note that names are case sensitive (i.e. `reverse1.C` is different than `reverse1.c`).

### Giving Permission to the Files

Before submitting the solution set, please give permission to the files by executing the following command:

```
chmod 705 ~ ~/software1 ~/software1/assign2 ~/software1/assign2/*
```