# *Homework 1*

*100 Points*

# *One Dimensional Arrays*

**Project A: Lucky Numbers and Insertion Sort**

The input file numbers.txt contains one line for each student in CIS22B. Each line contains two integers representing the student ID followed by their lucky number, an integer. The lucky numbers are either positive or negative, but not zero:

```
1234 22
2200 -5
7784 17
9288 18
```

Read data from file in two parallel array: one or the IDs and the other one for the lucky numbers.Use an array of maximum size 50. In case the input file contains data for more than 50 students, print a message such as "The file contains more than 50 lines!" and terminate the program.

Change the Insertion Sort function to sort the parallel arrays in descending order (from the largest to the smallest ID).

Write the sorted arrays to another file named sorted.txt, 8 numbers per line in 4 columns (id / lucky number), as shown below:

```
9288 18    7784 17    2200 -5    1234 22
1200 -3    1150 29    1100 -8    1010 99
1000 13
```

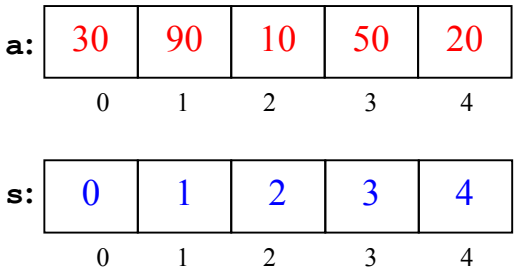
Place the even lucky numbers in an array called **`eList`**, the odd lucky numbers in an array called **`oList`**, and the negative lucky numbers in an array called **`nList`**.

Write the three arrays to a file named lucky.txt, as shown below

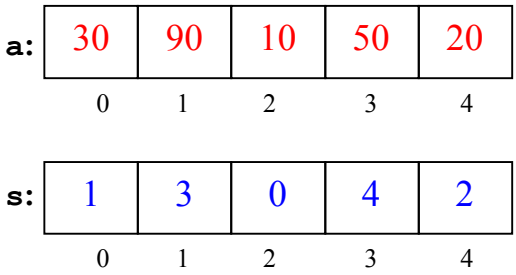
```
EVEN  ODD   NEGATIVE
  18   17         -5
  22   -5         -3
  -8   -3         -8
       29
       99
       13
```

**Project B: An unusual sort**

Write a function that sorts an array of integers in descending order using a variation of the Insertion Sort algorithm. What is unusual about this sorting algorithm is the requirement to make no changes to the original array and make no copies of the original array. You will need another array that stores the indices of the original array, as shown below.




After sorting, the original array, a, has not changed, but the other array did change:




```
k = 0   i = s[k] = s[0] = 1 => a[i] = a[1] = 90
k = 1   i = s[k] = s[1] = 3 => a[i] = a[3] = 50
k = 2   i = s[k] = s[2] = 0 => a[i] = a[0] = 30
k = 3   i = s[k] = s[3] = 4 => a[i] = a[4] = 20
k = 4   i = s[k] = s[4] = 2 => a[i] = a[2] = 10

90, 50, 30, 20, 10 // SORTED!
```

Write another function that displays data in the original order and sorted order too, as shown below:

```
ORIGINAL     SORTED
   30          90
   90          50
   10          30
   50          20
   20          10
```

You might think that it would be much easier to create a copy of the original array and call the regular insertion sort twice. Why is the other solution (described above) preferred, although it is more difficult and does not reuse the insertion sort as it is?

In a real application that deals with a data base we avoid replicating the data to

1. save space and also to
2. keep the updating simple (update in one place).

You will learn later how to process an array of objects (such as books or movies, etc.) With such an array this special sorting algorithm is very useful.

**Grading**

**Project A**

| | |
|---|---|
| Read data from file in two parallel array | – 10Points |
| Insertion sort | – 10 |
| Write the sorted arrays to a file | – 10 |
| Place the even, odd, negative numbers in other arrays | – 20 |
| Write the three arrays to a file | – 20 |
| main() | – 10 |

**Project B**

| | |
|---|---|
| Special sort | – 10Points |
| Display data in original order and sorted order | – 10 |

Run each program once and save the output at the end of the source file as a comment. Compress the source files, input and output files, and upload the compressed file: 22B_LastName_FirstName_H1.zip