Function Pointers in C

Computing Lab
https://www.isical.ac.in/~dfslab

Indian Statistical Institute

Function pointers

- Declaring function pointers

```
<return type> (* <function name>) ( <parameter list> )
```

Example:

```
int *aFunction(int), *(*aFunctionPointer)(int);
```

- Using function pointers

```
(*f)(...)
```

- Setting function pointer variables / passing function pointers as arguments: simply use the name of the function

Example:

```
aFunctionPointer = aFunction;
```

Generic sort/search routines

```
#include <stdlib.h>
```

Sorting

```
void qsort(void *base, size_t nmemb, size_t size,
           int (*compar)(const void *, const void *));
```

Searching

```
void *bsearch(const void *key, const void *base,
              size_t nmemb, size_t size,
              int (*compar)(const void *, const void *));
```

Comparator routine: examples

```
int compare_int (void *elem1, void *elem2)
{
    int *ip1 = elem1;
    int *ip2 = elem2;
    return *ip1 - *ip2;
    /* Or more explicitly:
       int i1 = *((int *) elem1);
       int i2 = *((int *) elem2);
       return i1 - i2;
     */
}

int compare_strings (void *elem1, void *elem2)
{
    char **s1 = elem1; // Alt.: char *s1 = *((char **) elem1);
    char **s2 = elem2; // Alt.: char *s2 = *((char **) elem2);
    return strcmp (*s1, *s2);   // Alt.: return strcmp(s1, s2);
}
```

Using `qsort` and `bsearch`

```
char **strings;
int *a;
int num_strings, N;

qsort(a, N, sizeof(int), compare_int);
qsort(strings, num_strings, sizeof(char *), compare_strings);
```

Programming problems – I

1. Download `bubble-sort.c` and `student-data.txt` (and `common.h`, if necessary). Save all 3 files in one directory. Compile and run the program.
   Modify the program so that it uses the `qsort` function to sort the student data in alphabetical order by name, or by aggregate percentage, or by attendance, depending on the user's choice.
2. Write a program to compute a function of the form

$$\sigma\left(\sum_{i=0}^{N-1} w_i \cdot x_i + b\right).$$

   where $\sigma : \mathbb{R} \to \mathbb{R}$ is one of the following:
   - (*sigmoid*) $\sigma(x) = \frac{1}{1+e^{-x}}$
   - (*tanh*) $\sigma(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$

Programming problems – II

- (*LRELU*) $\sigma(x) = \begin{cases} x & \text{if } x \geq 0 \\ x/100 & \text{if } x < 0 \end{cases}$

Your program should all input values from a file named `params.txt`, which should have the following format:

- it should have four lines;
- the first line will contain a single integer N which can be stored in a variable of type `int`;
- the second and third lines will each contain N floating point numbers corresponding to $w_0, w_1, \ldots, w_{N-1}$ and $x_0, x_1, \ldots, x_{N-1}$, resp.;
- the fourth line will contain one of the three strings `sigmoid`, `tanh` or `LRELU`, specifying the form of σ. Note that the string may be in upper, lower or mixed case; thus, `SIGMOID`, `lReLU` are also valid inputs.

Programming problems – III

3. Many programming languages provide a *map*() function that works as follows. Given two sets X, Y and a function $f : X \to Y$, *map*() takes as input a list $L = [l_0, l_1, \ldots, l_{N-1}]$ of elements from X, and returns $f(L) \triangleq [f(l_0), f(l_1), \ldots, f(l_{N-1})]$, a list of elements from Y. Implement a `map` function in C. The function should have the following prototype:

```
void *map(void *L, unsigned int N,
          size_t domain_elt_size, size_t range_elt_size,
          void (*f)(void *input, void *output))
```

where

- `L` is the input list;
- `N` is the number of elements in `L`;
- `domain_elt_size` (and `range_elt_size`, resp.) correspond to the size of each element of the domain and range of $f : X \to Y$; and

Programming problems – IV

- `f` is a pointer to a C function that implements f (`input` is a pointer to an element of the domain X, and `output` is a pointer to an existing chunk of memory that is just big enough to store an element of the co-domain Y).