Exercise 5.1 — Top 10 Lists (55% of Midterm 3, 2003)

A computer game maintains its top 10 list in two arrays, declared globally by:

```
#define TOPLEN 10
int top10scores[TOPLEN];
char * top10names[TOPLEN];
```

(These are global arrays and need not be passed as arguments to the functions below.)

Scores in this game are always non-negative, so negative entries in `top10scores` indicate empty positions, i.e., positions that have not been claimed yet. For example, after the first player plays this game, achieving the non-negative score $s_1$ only the entry for the “top position” will be occupied by $s_1$; all other entries in the array `top10scores` will be negative.

Players who provide their name will have their name listed in the array `top10names` in the same positions that their scores occupy in `top10scores`. Players can play anonymously; instead of their names, the `NULL` pointer value will be stored in their positions in the array `top10names`.

(a) \(\approx 10\%\) Some possible states of the two arrays `top10scores` and `top10names` make no sense. For example, there should be no “empty” entries between real scores. Define precisely which states of the two arrays `top10scores` and `top10names` you consider as legal, and how you interpret these states. In particular, where will the best score be stored?

(b) \(\approx 20\%\) Define the interface of a function `insertIntoTop` that attempts to insert a new score into the top 10 lists — it will insert only if the new score deserves it, and it will inform the caller of the following:
- whether insertion was successful,
- whether the score of a different non-anonymous player was expunged from the list, and if yes, who this was, and what their score was (so the system can, for example, send them an e-mail to ask them to play again),
- the difference between the supplied score and the previous best score.

Document how the caller of the function `insertIntoTop` will be able to access all this information after a call, and document the arguments the function `insertIntoTop` accepts and which assumptions it makes about those arguments. — Hint: Pass-by-reference may be useful.

(c) \(\approx 25\%\) Implement the function `insertIntoTop` from (b).

(d) (not on the original midterm — independent from (b) and (c))

Implement the function `displayTop10` that produces a sensible display of legal states — see (a) — of the top 10 list.

(e) (not on the original midterm) Implement an appropriate `main` program to test your functions.
Solution Hints

Legal states:
- Specify where the highest score is stored, for example at index 0 — we assume this throughout the following.
- The sequence of scores is monotonically decreasing
- Decision: negative scores are all -1
- Decision: positions with negative score have NULL as name
- Possible decisions (not implemented in the example solution to (b,c) below): (Consecutive) entries with the same score must not have the same name
- Possible decision (not implemented in the example solution to (b,c) below): equal names are represented by equal pointers.
- Intuitively desirable: Correspondence between names and scores.
  However, instead of a static condition on states, this is a condition relating the states before and after insertion. Mentioning this is therefore not expected in (a).

The interface of a function consists of prototype and specification of behaviour. Beyond the insertion aspect covered in the question (insertion of key-value-pair into list of key-value-pairs sorted by keys), the arguments and return values need to be documented, here as comments in the code:

```c
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#define TOPLEN 10
int top10scores[TOPLEN];
char *top10names[TOPLEN];

bool insertIntoTop( /* return value: success (Boolean) */
                int score,
                char *name,
                int *diff,
                int *outscore,
                char **outname)
                /* in: new score — non-negative */
                /* in: name of score winner, or NULL */
                /* out: difference, always set */
                /* out: expunged score: -1 if none, or if the expunged score belongs to */
                /* out: expunged name — non-NULL possible only if *outscore ≠ -1 */
{
    int i = 0, j;

    if (top10scores[0] > 0)
        *diff = score - top10scores[0];
    else
        *diff = score; /* maximum of empty set of non-negative scores is 0. Alternative choice: *diff = 0; */

    while (i < TOPLEN && top10scores[i] > score) { i++; }
    if (i == TOPLEN)
```

return false; /* score does not belong into top 10 */

/* Now, for all \( j \) with \( 0 \leq j < i \), we have \( \text{top10scores}[j] > \text{score} \).
and for all \( j \) with \( i \leq j < \text{TOPLEN} \), we have \( \text{top10scores}[j] \leq \text{score} \).
We decide that in the case of equal scores, the latest comer is top.
Therefore, \( i \) is the position where score and name have to be entered. */

/* shift scores that are not better than new score */
for (j=i; j < \text{TOPLEN} ; j++) {
    /* swap \((\text{score},\text{name})\) with \text{top10}[j] using \text{*out...} as temporary variables;
this way, \text{*outscore} and \text{*outname} contain the expungend entry at the end, or are empty */
    *outscore = \text{top10scores}[j];
    *outname = \text{top10names}[j];
    \text{top10scores}[j] = \text{score};
    \text{top10names}[j] = \text{name};
    \text{score} = \text{*outscore};
    \text{name} = \text{*outname};
}
/* clear \text{*out...} if \text{name} pushed out their own score */
if (*outscore > 0 && /* not necessary since otherwise \text{*outname} = NULL */
    *outname && \text{name} && !\text{strcmp(*outname,name)}) /* *outname == \text{name} is acceptable in the
test */
    { *outname = NULL; *outscore = -1; }
return true;
}
**Exercise 5.2 — Find Errors (15% of Midterm 3, 2003)**

Find and describe the error in each of the following program segments. If the error can be corrected, explain how.

(a) `char *s; printf("%s\n", s);`
(b) `char s[] = "Some string."; printf("%s\n", &s[1]);`
(c) `float *x, y; x = y;`
(d) `char s[4] = {'a', 'b', 'd', 'e'}; printf("%s\n", s);`
(e) `int z = 5; int *p, q; /* integer pointers p and q */ p = &z; q = *p;`

**Solution Hints**

(a) The pointer s is not initialised — initialise it by assigning the start address of some string, e.g., `s = ""`.  
(b) The period “.” should be a comma “,”.  
(c) Type error — change to `x = &y;`.  
(d) There is no terminating zero character in the array s — when changing to string initialisation, take care to allow enough space for the terminating zero character, e.g., `char s[5] = "abde"`  
(e) The comment is wrong — change it!

**Exercise 5.3 — Typing (8% of Midterm 2, 2004)**

Let the following declarations be given:  

`char z[100]; char * c[15]; int ** p;`

Give the types of the following expressions:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>p[42]</code></td>
<td>(a) int *</td>
</tr>
<tr>
<td><code>z + 4</code></td>
<td>(b) int</td>
</tr>
<tr>
<td><code>*(c+5)</code></td>
<td>(c) char *</td>
</tr>
<tr>
<td><code>&amp;(c[1])</code></td>
<td>(d) char **</td>
</tr>
</tbody>
</table>

**Solution Hints**

(a) A pointer to T cab ne treated as an array of T elements, so p is treated here as an array p : int * [] , and we have `p[42] : int *`  
(b) As argument to pointer addition, z is considered as a pointer, i.e., `z : char *`, so `z + 4 : char *`  
(c) Analogously, `(c+5) : char **`, so `*(c+5) : char *`  
(d) `(c[1]) : char *`, so `&(c[1]) : char **`