Second assignment (2021/2022, this is the one you need to do), due 01-02-2022
Multi-ordered trees (part 1)

Sometimes it is necessary to store and process records of data and to access them using one of several possible keys (that is one of the things that databases do). In this second assignment we will consider records that contain the following items of information about a person:

- first and last name,
- zip code, and
- telephone number.

We want, for example, to list these records in ascending order of one of these items. In order to do so, we will construct ordered binary trees (one per item). There exist several ways to do this, but in this assignment we propose one: the binary tree nodes, besides storing the data items, will also have three left and three right pointers (one per data item), so that the three ordered binary trees coexist in the same data structure. In particular, the tree node will be declared as follows.

```
typedef struct tree_node_s
{
  char name[MAX_NAME_SIZE + 1];                         // index 0 data item
  char zip_code[MAX_ZIP_CODE_SIZE + 1];                 // index 1 data item
  char telephone_number[MAX_TELEPHONE_NUMBER_SIZE + 1]; // index 2 data item
  struct tree_node_s *left[3];                          // left pointers (one for each index) ---- left means smaller
  struct tree_node_s *right[3];                         // right pointers (one for each index) --- right means larger
}
tree_node_t;
```

It will be necessary to insert a given new item of information, already stored in a `tree_node_t` (but with the left and right pointers still uninitialized), in three ordered binary trees (so, there will exist three roots).

Multi-ordered trees (part 2, useful code)

In the archive A02.tgz you will find a partial implementation of the assignment. In particular,

- the AED_2021_A02.h file contains declarations of things used by the entire project;
- the random_knuth.h file contains an implementation of a pseudo-random number generator, done by Prof. Donald Knuth, that should produce the same results in GNU/Linux, Mac OS, and Windows;
- the random_number.c file uses Donald Knuth's code to produce a 30-bit pseudo-random integer;
- the elapsed_time.c file contains code to measure execution time;
- the random_data.c file contains code to generate random names, zip codes, and telephone numbers; and, last not not least,
- the multi_ordered_tree.c file contains a partial implementation of what needs to be done.

Of course, there is also a makefile, that you can use to compile the project.

Once you have a successful implementation you can use the program as follows:

```
Usage: ./multi_ordered_tree student_number number_of_persons [options ...]
Recognized options:
  -list[N]              # list the tree contents, sorted by key index N (the default is index 0)
```

The first argument is the student number, as that is used as the seed to the pseudo-random number generator. The second argument is the number of "random" persons that are going to be generated, The program has to place them all in the ordered binary trees, and then it has to measure some characteristics of the trees that were constructed. At least the option -list should be implemented; it lists in sorted order the contents of one of the trees.

Multi-ordered trees (part 3, examples)

Normal run:

```
> ./solution_multi_ordered_tree 2021 1000000
Tree creation time (1000000 persons): 4.367e+00s
Tree search time (1000000 persons, index 0): 1.169e+00s
Tree search time (1000000 persons, index 1): 1.351e+00s
Tree search time (1000000 persons, index 2): 1.172e+00s
Tree depth for index 0: 54 (done in 6.089e-02s)
Tree depth for index 1: 55 (done in 6.341e-02s)
Tree depth for index 2: 48 (done in 6.303e-02s)
```

Runs to list the tree contents (you are free to format the output in other ways, . . . denotes the output of a normal run):

```
> ./solution_multi_ordered_tree 2021 3 -list0
...
List of persons:
Person #1
  name ---------------- Helen Reyes
  zip code ----------- 22003 Annandale (Fairfax county)
  telephone number --- 4008 868 655
Person #2
  name ---------------- Kareem Johnson
  zip code ----------- 10463 Bronx (Bronx county)
  telephone number --- 2000 034 151
Person #3
  name ---------------- Luke Hall
  zip code ----------- 11215 Brooklyn (Kings county)
  telephone number --- 7362 997 722
```

```
> ./solution_multi_ordered_tree 2021 3 -list1
...
List of persons:
Person #1
  name ---------------- Kareem Johnson
  zip code ----------- 10463 Bronx (Bronx county)
  telephone number --- 2000 034 151
Person #2
  name ---------------- Luke Hall
  zip code ----------- 11215 Brooklyn (Kings county)
  telephone number --- 7362 997 722
Person #3
  name ---------------- Helen Reyes
  zip code ----------- 22003 Annandale (Fairfax county)
  telephone number --- 4008 868 655
```

```
> ./solution_multi_ordered_tree 2021 3 -list2
...
List of persons:
Person #1
  name ---------------- Kareem Johnson
  zip code ----------- 10463 Bronx (Bronx county)
  telephone number --- 2000 034 151
Person #2
  name ---------------- Helen Reyes
  zip code ----------- 22003 Annandale (Fairfax county)
  telephone number --- 4008 868 655
Person #3
  name ---------------- Luke Hall
  zip code ----------- 11215 Brooklyn (Kings county)
  telephone number --- 7362 997 722
```

Multi-ordered trees (part 4, written report)

The written report must have:

- A title page (front page) with the name of the course, the name of the report, date, the numbers and names of the students, and an estimate of the percentage each student contributed to the project (the sum of percentages should be 100%).
- A very short introduction describing how you got your results.
- Tables and graphs with the execution times for how long it took to create the trees, and how long it took to search all persons. This has to be done using each one of the trees, to see it there exist large differences in execution times. For example, there exist only 500 zip codes. Does that influence much the execution times for the construction and searches for the tree ordered by the zip codes?
- Tables and graphs of the maximum depth, and of any other tree characteristic you may choose do study, for each of the three trees. You may wish to do a statistical study of this. To that end, use many consecutive student numbers (say, from 999000 to 999999).
- Extra work (for a slightly better grade). Add a fourth field, for example, a social security number, and then work with four trees.
- Extra work (for a significantly better grade). List, for example, all persons with a given zip code.
- **Deliverable: one PDF file. No archives, please!** Put in an appendix the entire contents of the `multi_ordered_tree.c` file. If you made changes to any other file, place the changes you made also in the appendix, but do not place there the entire file (that applies in particular to the `random_data.c` file; it is BIG).