UNIVERSITY OF MALTA
FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE

CPS1011 Programming Principles (in C)

Assignment 2018/9

Instructions:

1. This is an **individual assignment**.
2. You may be required to demonstrate and present your work to an exam board.
3. The hard copy of the report must be handed in to the Computer Science Dept. clerk's office by the assignment deadline. A soft-copy of the same report along with all digital artefacts must be uploaded to the VLE upload area by the same deadline. All files must be archived into a single .zip file. It is the student's responsibility to ensure that the uploaded zip file and all contents are valid. You must include all source, make files, any scripts and instructions required to compile the code.
4. Reports (and code) that are difficult to follow due to low quality in the writing-style/organisation/presentation will be penalised.
5. Assignment queries are to be made strictly during the beginning of lectures or posted to the assignment's forum on VLE, as of when individual tasks are announced in class till one week before the deadline (excluding recess).
6. This assignment comprises 100% of the final CPS1011 assessment mark.
7. You are to allocate 50 to 60 hours for this assignment.
8. The report must be organized according to the task sequence, and for each clearly explaining the relevant code fragments as well as the program's output listings. The full source code files must also be uploaded to VLE (as explained above) along with 2 `CMakeLists.txt`, one for each of tasks 1 and 2. A `readme.txt` file must be included in the root directory of the archive file describing the content's organization.

9. While it is strongly recommended that you start working on the tasks as soon as they are announced in class, the firm submission deadline is **Thursday 17th January 2019 at NOON**.

1. **Problem solving.** (Total-45 marks)

   For each of the following tasks, the allocated marks are equally split between:

   - correct application of programming principles;
   - source code explanation; and
   - working solution as demonstrated for all tasks through task (c).

   Tasks: -

   (a) Write a function that makes a copy of the contents of a 3D array of integers. The function should support any 3D array size.

   [15 marks]

   (b) Write a function that takes an array of strings as argument and reverses the order of individual words in each string. For e.g. the array {`"This is a first sentence", "and this is a second"`} is transformed to {`"sentence first a is This", "second a is this and"`}.

   [15 marks]

   (c) Write a program that presents the end-user with a command-line menu, and that repeatedly asks the user to execute either of the functions above, or to quit. The program should prompt for function arguments accordingly. Proper validation of user input is expected.

   [15 marks]

2. **A *Tuples* library.** (Total-55 marks)

   One common usage of C programming is that of creating runtimes for higher level languages. These runtimes at minimum consist of a language interpreter/compiler, as well as a number of readily-available functions and data types. Python's *tuples*, for e.g, are native to the language itself and are loosely based on the mathematical definition. An *n-tuple* is defined as an ordered list of elements, and is typically written as a sequence of elements enclosed in non-curly brackets: e.g. `(2, 8, 200, 7)` or `<"physics", 75, "chemistry", 80>` or `[a, 1, b, 2, c, 3]`. As shown in these examples, individual elements may

be of a different type. Also, as per definition, `(2, 8, 200, 7)` $\neq$ `(7, 200, 8, 2)`.

Designing and implementing a full programming language requires more knowledge and time than what is made available by a primer course in programming, however you should be able to write a library that makes tuples available to C programmers. The end result takes the form of a new data type called `tuple_t` with the following associated operations:

- `createTuple()` - Creates a tuple, as identified by a string identifier and a sequence of element type/values. In the case of an existing identifier, the new tuple replaces the existing one.
- `getTupleByID()` - Returns a pointer to a `tuple_t` object through its identifier.
- `getTupleID()` - Returns a `tuple_t` identifier through its pointer.
- `showTuple()` - Displays the tuple's content through its pointer to `tuple_t`.
- `deleteTuple()` - Deletes a `tuple_t` object through its identifier.
- `cmpTuples()` - Compares two compatible tuples based on their in-order comparison of their elements, returning a positive integer in case the first non-equal element pair is larger for the first tuple, a negative integer in the opposite case, or 0 if all elements are equal. Compatible tuples are those with the same element number and types, or where the element number/types of the first is a prefix of the second.
- `joinTuples()` - Creates a new tuple, as identified by its string identifier, out of two pointers to existing `tuple_t` objects.
- `saveAllTuples()` - Saves all existing (created but not deleted) tuple objects to disk.
- `loadAllTuples()` - Loads all tuples from a saved file.

For each of the following tasks, the allocated marks are equally split between:

- Correct application of programming principles, including the provision of an API header file in each task.

- Source code explanation.
- Working solution as demonstrated by a test driver developed for each task, each comprehensive of the implemented functions.

Tasks: -

(a) Implement a simplified version of the data type and functions described above. In this first step, the `tuple_t` data type has a fixed structure (i.e. number and type of elements) of your choice, but consisting of at least three elements. The created tuples should all be contained within a dynamic array stored on the heap, and which should be continuously resized in a way not to waste memory space. At this stage, the implementation of `joinTuples()` should simply create a copy of the first tuple argument.

[20 marks]

(b) Extend the implementation in 2(a) so that this time individual tuple objects may take any structure, and which implies the need to internally store information about their structure, e.g a colon delimited string that identifies the number of and type of each element. So, `"5:s:i:l:c:c"` would be a 5-tuple where the first element is a string, the second an integer, the third a long integer, while the last two elements are both single byte characters. At this point you need to revise the API and implementation of the simpler version. Marks allocated for this task only correspond to the enhancements made over the simpler version.

[20 marks]

(c) Compile the full implementation as a shared library and provide a proper Abstract Data Type interface. The test driver this time should link with the shared library. In case of difficulties with task 2(b), you may opt to compile the library out of the simpler version developed in 2(a), with a possible maximum of 8 marks.

[15 marks]

---

end of assignment