TC2025 / Advanced Programming

Final Exam

Tecnológico de Monterrey

Professor: Miguel Angel Medina Pérez

Date: April 10th, 2018

ID and group: TC2025-201811.1

Name: Hector Manuel Takami Flores Student ID: A01377647

Commitment to academic integrity

Adhering to the Academic Integrity Regulations of the Instituto Tecnológico y de Estudios Superiores de Monterrey, I agree to obey the Academic Integrity rules in this exam. In congruence with the commitment acquired with the Academic Integrity Regulations, I will carry out this exam in an honest and personal way, to reflect, through it, my knowledge and to accept, later, the obtained evaluation.

Click on the checkmark to accept: ⬓

IMPORTANT: You must attach this document in the exam publication in Google Classroom entitled Final Exam with the data that was requested up to this point. You must also attach a text document with your source codes. If you do not submit this document with the requested data, do not submit the source or you submit codes and / or modify any of these documents once the submission time passed, your solution will be invalidated. This exam must be solved individually. Any sign of copy, cheating or fraud will be sanctioned according to the Academic Integrity regulations.

Exercises

1. Fingerprint Identification Systems use fingerprint features based on minutiae descriptors. One of the most efficient and accurate minutiae descriptor is Cylinder Codes (CC). Some applications represent CCs at low resolution (64 bits) to perform fast operations.
   Implement a function with variable number of parameters (*variadic*) that receives many CCs (each CC is represented with an unsigned integer of 64 bits) and returns a new CC which contains the mixture of the CCs passed as parameters.
   As an example, if the function receives the following CCs:

   ```
   0…01110
   0…11000
   0…01100
   ```

   the result will be:

   ```
   0…11110
   ```

Checklist

(Every item has a value of 30/8 points)

i. ☐ The program compiles without errors using the C compiler (C99 or C11).
ii. ☐ The name of the variables, functions, and parameters have a clear meaning in the program; except for those variables used for controlling iterations in the loops (e.g. `i`, `j` y `k`).
iii. ☐ The prototype and definition of the function specify the parameters and returning value as described in the exercise.
iv. ☐ The function mixes correctly the CCs passed as parameters.
v. ☐ The function returns an unsigned integer of 64 bits containing the mixture of the CCs passed as parameters.
vi. ☐ The function correctly declares the pointer that is used to iterate over the CCs specified as parameters.
vii. ☐ The function correctly reads each CC using the pointer that was declared for iterating over the CCs specified as parameters.
viii. ☐ The function guaranties to return correctly after iterating over the CCs specified as parameters.

2. Your company must develop a system for automatic detection of pneumatic failures in Temporary Immersion Bioreactors (TIB). TIB are widely used for increasing the plant multiplication rates and plant quality. These systems are actioned by a pneumatic system.

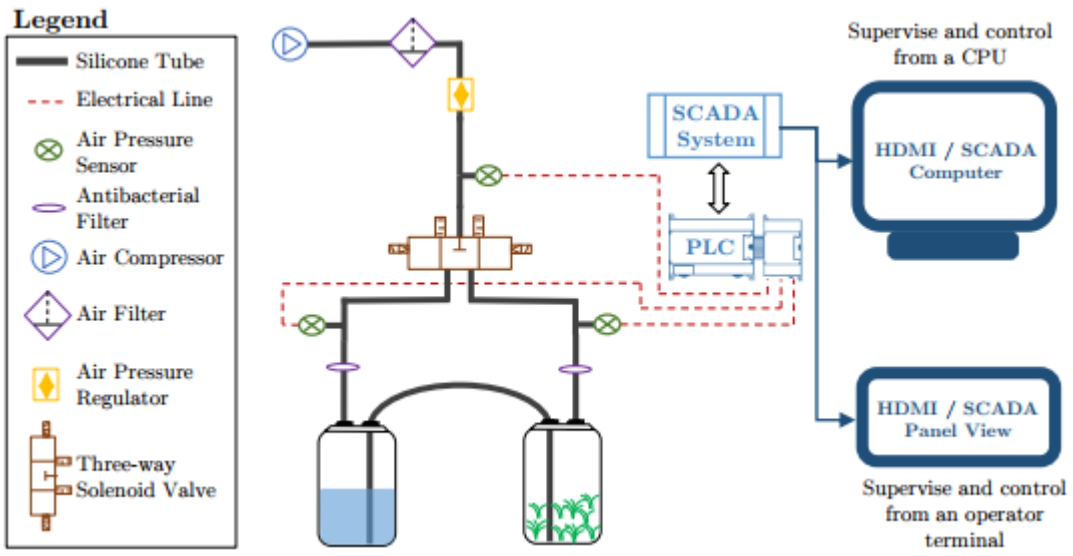

Figure 1. The general diagram of a Temporary Immersion Bioreactors. Taken from: O. Loyola-González, M. A. Medina-Pérez, J. F. Martínez-Trinidad, A. Carrasco-Ochoa, R. Monroy, "An Approach based on contrast patterns for detecting pneumatic failures on temporary immersion bioreactors," Knowledge Based Systems, (Under review).

TIBs have three sensors that measure the air pressure in three different modules of the system (see Figure 1). You must implement the following requirements which will be the bases for developing the algorithms for failure detection (there are more requirements in the checklist at the end of this exam).

a. Implement a struct with alias `SensorValues` with the following members:
   i. `values`: a pointer to `double` that will contain an array.
   ii. `length`: indicates the number of elements in the array `values`.
b. Implement a struct with alias `AirPressureSensorsValues` with the following members:
   i. `ps`: a pointer to `SensorValues`. This member contains the measures of the air pressure captured by the sensor located in the air filter of the liquid medium. (i.e., the sensor located at the top of the left container in Figure 1).
   ii. `pfv`: a pointer to `SensorValues`. This member contains the measures of the air pressure captured by the sensor located in the air filter of the culture container. (i.e., the sensor located at the top of the right container in Figure 1).
c. Implement the function `createAirPressureSensorsValues` that receives two constant parameters indicating the length of the structs (`SensorValues`) pointed by `ps` and `pfv`. The function also receives a parameter by reference named `output` that will contain the value `ARG_OUT_OF_RANGE` if any of the first two parameters of the function contains a value lower or equal than `0`; otherwise, `output` will contain the value `OPT_SUCCESS`. `OPT_SUCCESS` and `ARG_OUT_OF_RANGE` are symbolic constants that you must define with the values `1` and `-1` respectively.
   If any of the first two parameters of the function contains a value lower or equal than `0`, then the function returns `NULL`; otherwise, the function dynamically reserves memory and returns a pointer to a new `AirPressureSensorsValues` initializing the members `ps` and `pfv` according to the values passed as parameters of the function.
d. Implement a test function with no parameters that returns `1` if the function `createAirPressureSensorsValues` returns `NULL`, while the parameter `output` is outputted with value `ARG_OUT_OF_RANGE` when the parameter indicating the length of `ps` has a value lower or equal than zero (otherwise, the test function returns `0`). Notice that the parameter indicating the length of `pfv` must have a value greater than `0`.
e. Implement a test function with no parameters that returns `1` if the function `createAirPressureSensorsValues` returns a pointer to a `AirPressureSensorsValues` which memory was dynamically reserved with values greater than `0` and different to `NULL`, while the parameter `output` is returned with value `OPT_SUCCESS`; otherwise, the function returns `0`. In order to return `1`, the function must also verify correctly that the members `length` of `ps` and `pfv` have the same values that you specified as parameters when calling the function `createAirPressureSensorsValues` inside the test function.
f. Implement the function `freeAirPressureSensorsValues` that receives a pointer to `AirPressureSensorsValues` and releases the memory to which it points to and the memory occupied by its members. This function returns no value.

g. Implement the function `applyFilter` that receives the parameters: a) a pointer to a struct `AirPressureSensorsValues`; b) a constant array to `double`; c) a constant integer of 8 bits that indicates the length of the array (b). This function returns nothing. The function applies the array of `double` as a filter over the respective arrays stored in the members `ps` and `pfv` of `AirPressureSensorsValues`. This kind of functions are common in signal processing to mitigate the noise in the signals.
Let's assume that you will apply the filter [0.3, 0.4, 0.3] over the array [1, 3, 2, 1]. You will perform the following steps:

i. Superimpose the center of the filter over the first element of the array as follows:
[0.3, 0.4, 0.3]
[1, 3, 2, 1].
Now, the elements of the filter are multiplied by the corresponding elements in the lower array and the results are accumulated: 0.3*0+0.4*1+0.3*3=1.3.

ii. Superimpose the center of the filter over the first element of the array as follows:

[0.3, 0.4, 0.3]

[1, 3, 2, 1].

Now, the elements of the filter are multiplied by the corresponding elements in the lower array and the results are accumulated: 0.3*1+0.4*3+0.3*2+0*1=2.1.

iii. Superimpose the center of the filter over the first element of the array as follows:

[0.3, 0.4, 0.3]

[1, 3, 2, 1].

Now, the elements of the filter are multiplied by the corresponding elements in the lower array and the results are accumulated: 0*1+0.3*3+0.4*2+0.3*1=2.

iv. Superimpose the center of the filter over the first element of the array as follows:

[0.3, 0.4, 0.3]

[1, 3, 2, 1].

Now, the elements of the filter are multiplied by the corresponding elements in the lower array and the results are accumulated: 0*1+0*3+0.3*2+0.4*1+0.3*0=1.

v. Substitute the original values of the array [1, 3, 2, 1] by the new values [1.3, 2.1, 2, 1].

Important: Notice that, despite the example above applies a filter of three elements over a vector of four elements, your code must apply filters of any length (odd number) over vectors of any length greater or equal than that of the filter.

h. Implement a function with no parameters that test that the function `applyFilter` applies correctly a filter equal to the above exercise. The test function dynamically allocates memory for a `AirPressureSensorsValues` where the members `ps` and `pfv` (which are pointers to `SensorValues`) must contain three elements and of four

elements in their respective member `values` (all the elements of the member `values` must be greater or equal than `0`). In order to return `1`, the function must verify that the filter was applied correctly (as specified in the above exercise) over the values that you specified as parameters when calling the function `applyFilter` inside the test function; otherwise, the test function returns `0`.

Checklist

(Every item has a value of 70/29 points)

ix. ☐ The program compiles without errors using the C compiler (C99 or C11).
x. ☐ There are no global variables in the source code.
xi. ☐ The name of the variables, functions, and parameters have a clear meaning in the program; except for those variables used for controlling iterations in the loops (e.g. `i`, `j` y `k`).
xii. ☐ The program includes a struct with alias `SensorValues`.
xiii. ☐ The program includes a struct `SensorValues` with a member `values` which is a pointer to `double`.
xiv. ☐ The program includes a struct `SensorValues` with a member `length` which is an unsigned integer of 16 bits.
xv. ☐ The program includes a struct with alias `AirPressureSensorsValues`.
xvi. ☐ The program includes a struct `AirPressureSensorsValues` with a member `ps` which is a pointer to `SensorValues`.
xvii. ☐ The program includes a struct `AirPressureSensorsValues` with a member `pfv` which is a pointer to `SensorValues`.
xviii. ☐ The program includes the symbolic constants `OPT_SUCCESS` and `ARG_OUT_OF_RANGE` with values `1` and `-1` respectively.
xix. ☐ The program includes the prototype and definition of the function `createAirPressureSensorsValues`.
xx. ☐ The prototype and definition of the function `createAirPressureSensorsValues` specifies the parameters and returning value as described in the exercise.
xxi. ☐ The function `createAirPressureSensorsValues` dynamically reserves memory and returns a pointer to a new `AirPressureSensorsValues`. The function correctly initializes the members `ps` and `pfv` from the values passed as parameters of the function. Notice that the function must allocate memory dynamically for the structs to which the members `ps` and `pfv` will point to.
xxii. ☐ In the function `createAirPressureSensorsValues`, the parameter by reference `output` takes the value `ARG_OUT_OF_RANGE` if any of the constant parameters has a value lower or equal than `0`; otherwise, `output` takes the value `OPT_SUCCESS`.
xxiii. ☐ The function `createAirPressureSensorsValues` returns `NULL` if any of the constant parameters has a value lower or equal than `0`.
xxiv. ☐ The program includes the prototype and a correct implementation of the test function described in 2.d.

xxv. ☐ The test function described in 2.d has no memory leaks.
xxvi. ☐ The program includes the prototype and a correct implementation of the test function described in 2.e.
xxvii. ☐ The test function described in 2.e has no memory leaks.
xxviii. ☐ The program includes the prototype and definition of the function `freeAirPressureSensorsValues`.
xxix. ☐ The prototype and definition of the function `freeAirPressureSensorsValues` specifies the parameters and returning value as described in the exercise.
xxx. ☐ The function `freeAirPressureSensorsValues` releases all the memory occupied by the struct (`AirPressureSensorsValues`) passed as a parameter.
xxxi. ☐ The program includes the prototype and definition of the function `applyFilter`.
xxxii. ☐ The prototype and definition of the function `applyFilter` specifies the parameters and returning value as described in the exercise.
xxxiii. ☐ The function `applyFilter` iterates correctly over the arrays of the members `ps` and `pfv` of the struct `AirPressureSensorsValues` passed as parameter.
xxxiv. ☐ The function `applyFilter` iterates correctly over the arrays of the members `ps` and `pfv` of the struct `AirPressureSensorsValues` passed as parameter and, for each element in the array, the function applies the filter (passed ad parameter) as described in the exercise.
xxxv. ☐ The function `applyFilter` has no memory leaks.
xxxvi. ☐ The program includes the prototype and a correct implementation of the test function described in 2.h.
xxxvii. ☐ The test function described in 2.h has no memory leaks.

Note: *The words "correct" and "correctly" in the checklist means that the code must do what it is stated in the exercises. The professor will include observations of your solution using the pdf tools for commenting documents.*

Tecnológico de Monterrey