TC2025 / Advanced Programming

Second Exam

Tecnológico de Monterrey

Professor: Miguel Angel Medina Pérez, PhD

Date: October 25th, 2019

ID and group: TC2025-201913.1

Name: Eric Gómez Vázquez Student ID: A01378838

Commitment to academic integrity

Adhering to the Academic Integrity Regulations of the Technological Institute and Higher Education of 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 Exam 2 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

Your enterprise must develop an Automated Fingerprint Identification System (AFIS) for federal institutions such as INE, SAT, and INM. You must implement the following requirements which are the core of the system.

The AFISs use fingerprint representations based on minutiae. Minutiae are the points on the ridges where the continuity is broken. These points can be represented with four attributes: 1) horizontal coordinate of the image (integer in the interval [0, 1023]); 2) vertical coordinate of the image (integer in the interval [0, 1023]); 3) direction of the ridges (integer in the interval [0, 359]); 4) minutia type (integer value in the interval [0, 2]).

You must implement the following to model minutiae and perform operations with them:

1. The struct with alias `Minutia` has four members. The members `x` and `y` are unsigned integer of 16 bits. The member `angle` is a floating-point number of 32 bits. The member `type` is an enumeration with alias `MinutiaType` with one of the possible three values: `Ending`, `Bifurcation`, and `Unknown`.

2. The struct with alias `MinutiaArray` has two members. The member `minutiae` is an array of minutiae and it is represented with a pointer to `Minutia`. The member `length` (unsigned integer of 16 bits) indicates the number of minutiae inside the array `minutiae`.
3. The function `createMinutia` dynamically creates a minutia from the information passed as parameters (positions `x` and `y`, `angle`, and `type`) and returns a pointer to the created minutia.
4. The function `createMinutiaArray` dynamically creates an array with the number of minutiae specified by a constant parameter. It initializes every minutia with zero-valued members. The function returns a pointer to the created array (`MinutiaArray`).
5. The function `releaseMinutiaArray` releases all the memory occupied by an array of minutiae which is passed as a parameter (a pointer to `MinutiaArray`) of the function.
6. The function `findCentroid` receives a constant pointer to a constant `MinutiaArray` and a pointer to a function `computeDistance`; it returns a pointer to a `Minutia`. The function `computeDistance` receives two constant pointers to constant `Minutia` and returns a floating-point number of 64 bits. `findCentroid` iterates over the minutiae passed as the parameter pointer to `MinutiaArray`. The function returns a pointer to the minutia which accumulated distance to the others is minimum. The distance between two minutiae is computed evaluating the function `computeDistance`.
7. The function `testFindCentroid` has no parameters and returns an integer value. This function reserve memory dynamically for a `MinutiaArray` which will contain at least ten (10) minutiae with non-zero-member values. `testFindCentroid` tests the function `findCentroid` passing the new `MinutiaArray` as parameter. The test function returns `1` if `findCentroid` returns a pointer to the minutia which accumulated distance to the others is minimum; otherwise, it returns `0`. Ensure that this function has no memory leaks.

See checklist in the next page

(Every item has a value of 100/30 points)

Checklist

(Every item has a value of 100/30 points)

i. ☐ The program compiles without errors with the GCC compiler or the C compiler of Microsoft.
ii. ☐ The name of all the variables, parameters, and functions of the program have a clear meaning in the problem domain (except those variables used to iterate in the cycles).
iii. ☐ All the definitions (structs, enumeration, function prototypes) must be in a single header file (.h).
iv. ☐ All the function implementations must be in a single source file (.c).
v. ☐ The program includes a struct with alias `Minutia` and a struct with alias `MinutiaArray`.
vi. ☐ The program includes a struct with alias `Minutia` with two unsigned integer members `x` and `y` of 16 bits.
vii. ☐ The program includes a struct with alias `Minutia` with a member named `angle` that is a floating-point number of 32 bits.
viii. ☐ The program includes an enumeration with alias `MinutiaType` with three possible values: `Ending`, `Bifurcation`, and `Unknown`.
ix. ☐ The program includes a struct with alias `Minutia` with a member named `type` that is of type `MinutiaType`.
x. ☐ The program includes a struct with alias `MinutiaArray` with a member named `minutiae` that is a pointer to `Minutia`.
xi. ☐ The program includes a struct with alias `MinutiaArray` with a member named `length` that is an unsigned integer of 16 bits.
xii. ☐ The program includes the prototype and the implementation of the function `createMinutia` fulfilling the requirements described in the exercise.
xiii. ☐ The program includes a function `createMinutia` that dynamically reserves memory for a new minutia.
xiv. ☐ The program includes a function `createMinutia` that returns a pointer to a minutia which is created reserving memory dynamically.
xv. ☐ The program includes a function `createMinutia` that creates a new minutia and initializes the members of the new minutia according to the parameters of the function.
xvi. ☐ The program includes the prototype and the implementation of the function `createMinutiaArray` fulfilling the requirements described in the exercise.
xvii. ☐ The program includes a function `createMinutiaArray` that dynamically reserves memory for a new array of minutiae which amount is specified as a parameter of the function.
xviii. ☐ The program includes a function `createMinutiaArray` that dynamically reserves memory for a new array of minutiae which amount is specified as a parameter of the function. It that returns a pointer to the created array.
xix. ☐ The program includes a function `createMinutiaArray` that dynamically reserves memory for a new array of minutiae which amount is specified as a parameter of the function. It initializes every minutia with zero-valued members.
xx. ☐ The program includes the prototype and implementation of the function `releaseMinutiaArray` fulfilling the requirements described in the exercise.

xxi. ☐ The program includes a function named `releaseMinutiaArray` that releases all the memory occupied by the array of minutiae passed as a parameter.

xxii. ☐ The program includes the prototype and implementation of the function `findCentroid` fulfilling the requirements described in the exercise.

xxiii. ☐ The program includes a function named `findCentroid` that iterates correctly over the minutiae contained in the pointer to `MinutiaArray` passed as a parameter.

xxiv. ☐ The program includes a function named `findCentroid` that iterates correctly over the minutiae contained in the pointer to `MinutiaArray` passed as a parameter and, for each minutia, it computes correctly the accumulated distance (using the function `computeDistance` passed as a parameter) with respect to the other minutiae.

xxv. ☐ The program includes a function named `findCentroid` that iterates correctly over the minutiae contained in the pointer to `MinutiaArray` passed as parameter. For each minutia, it computes the accumulated distance with respect to the other minutiae and it returns a pointer to the minutia which accumulated distance is minimum.

xxvi. ☐ The program includes the prototype and implementation of the function `testFindCentroid` fulfilling the requirements described in the exercise.

xxvii. ☐ The program includes a function named `testFindCentroid` that reserve memory dynamically for a `MinutiaArray` which contains at least ten (10) minutiae with non-zero-member values.

xxviii. ☐ The program includes a function named `testFindCentroid` that reserves memory dynamically for a `MinutiaArray` which contains at least ten (10) minutiae with non-zero-member values. The function `testFindCentroid` has no memory leak.

xxix. ☐ The program includes a function named `testFindCentroid` that calls the function `findCentroid` passing a new `MinutiaArray` as a parameter.

xxx. ☐ The program includes a function named `testFindCentroid` that calls the function `findCentroid` passing a new `MinutiaArray` as a parameter. `testFindCentroid` returns `1` if `findCentroid` returns a pointer to the minutia which accumulated distance to the others is minimum; otherwise, it returns `0`.

Note: *The professor will include observations using the pdf tools for commenting documents.*

Tecnológico de Monterrey