Advanced Programming: First Exam

ITESM - Campus Estado de México

Professor: Miguel Angel Medina Pérez

Date: February 15th, 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 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.

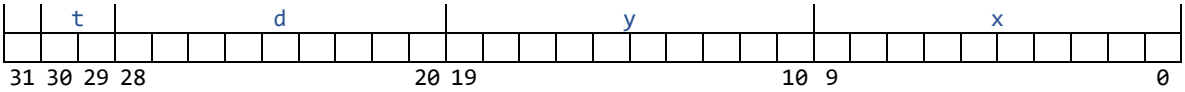
Click on the checkmark to accept: ☑

IMPORTANT: You must attach this document in the exam publication in Google Classroom entitled Exam 1 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

Automated Fingerprint Identification Systems (AFIS) use fingerprint representations based on minutiae. Minutiae are 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]).

The minutiae are coded in a 32-bit integer in the following way: the horizontal coordinate (`x`) occupies the least significant 10 bits; the vertical coordinate (`y`), the next 10 bits; the direction of the minutia (`d`), the next 9 bits; while the minutia type (`t`) occupies two bits after the direction of the minutia as shown in the following figure:



1. **Decoding minutia (50%).**

Implement a function called `decodeMinutia` that receives a 32-bits unsigned integer, decodes the values of `x`, `y`, `d`, and `t`; and prints the values in the console. Write a complete program in C that 1) reads a 32-bit unsigned integer from the console; 2) invokes the `decodeMinutia` function by passing the parameter read from console; and 3) prints the decoded `x`, `y`, `d`, and `t` values in the console.

Checklist

i. ☐ The program compile without errors.
ii. ☐ The program includes the prototype besides the definition of the function `decodeMinutia`.
iii. ☐ The prototype and the definition of the function `decodeMinutia` have no implementation errors and they follow the restrictions of the problem: it receives a parameter of type unsigned integer, it returns no value, and the function name is `decodeMinutia`.
iv. ☐ The name of all the variables, parameters, and functions of the program have clear meaning in the problem domain (except those variables used to iterate in the cycles).
v. ☐ Values are read from the console without any error.
vi. ☐ The function `decodeMinutia` is invoked in the main program without errors (it receives the parameter read from the console).
vii. ☐ The function `decodeMinutia` decodes correctly the value `x`.
viii. ☐ The function `decodeMinutia` decodes correctly the value `y`.
ix. ☐ The function `decodeMinutia` decodes correctly the value `d`.
x. ☐ The function `decodeMinutia` decodes correctly the value `t`.
xi. ☐ The function `decodeMinutia` shows in the console (without errors) the decode values `x`, `y`, `d`, and `t`.
xii. ☐ The program passes at least one of the tests in omegaup.
xiii. ☐ The program passes at least two of the tests in omegaup.
xiv. ☐ The program passes the three tests in omegaup.
xv. ☐ The main function returns the proper integer value which indicates the successful execution of the program.

2. **Encoding minutia (50%).**

Implement a function called `encodeMinutia` that receives the integer parameters `x`, `y`, `d`, and `t`; and returns the encoding of the minutia in a 32-bit integer. Write a complete program in C that 1) reads four integer values from the console; 2) invokes the `encodeMinutia` function by passing the parameters read from console; and 3) prints the encode minutia value in the console.

Checklist

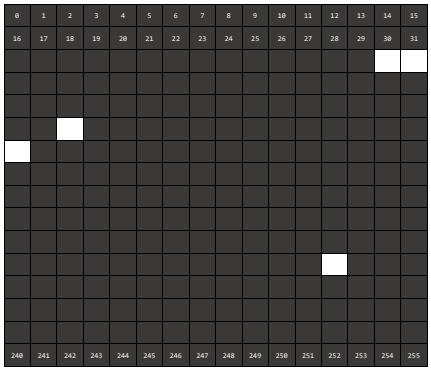
i. ☐ The program compile without errors.
ii. ☐ The program includes the prototype besides the definition of the function `encodeMinutia`.
iii. ☐ The prototype and the definition of the function `encodeMinutia` have no implementation errors and they follow the restrictions of the problem: it receives four parameters of type integer, it returns an unsigned integer of 32 bits, and the function name is `encodeMinutia`.
iv. ☐ The name of all the variables, parameters, and functions of the program have clear meaning in the problem domain (except those variables used to iterate in the cycles).
v. ☐ Values are read from the console without any error.
vi. ☐ The function `encodeMinutia` is invoked in the main program without errors (it receives the parameters read from the console and the returned value is stored in a variable).
vii. ☐ The function `encodeMinutia` encodes correctly the value `x`.
viii. ☐ The function `encodeMinutia` encodes correctly the value `y`.
ix. ☐ The function `encodeMinutia` encodes correctly the value `d`.
x. ☐ The function `encodeMinutia` encodes correctly the value `t`.
xi. ☐ The function `encodeMinutia` shows in the console (without errors) the encoded minutia.
xii. ☐ The program passes at least one of the tests in omegaup.
xiii. ☐ The program passes at least two of the tests in omegaup.
xiv. ☐ The program passes the three tests in omegaup.
xv. ☐ The main function returns the proper integer value which indicates the successful execution of the program.

3. Changing resolution (Additional 5 points).

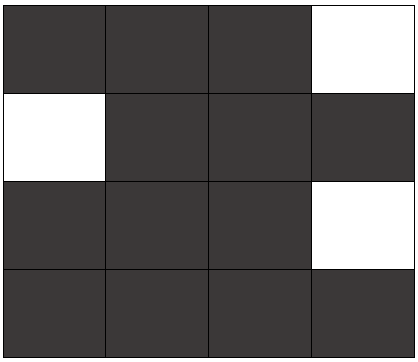
This is an all-or-nothing evaluation. You must solve it without any error in order to achieve the additional 5 points.

To find local matching minutiae, AFISs use information additional to the minutiae, i.e. minutiae descriptors. A recent minutiae descriptor which has proved to be highly accurate is cylinder code. Basically, a cylinder code contains five slices of 256 bits each.

In this exercise you must create a function which receives a slice (represented by 4 integers of 64 bits each) and returns a 16-bits integer. This function maps a high-resolution slice into a low-resolution slice as in the following example:



maps to:



The bits' sequences must be exactly the one used in the example.

You must create five test functions that shows that your mapping function works fine. The test functions must test different outputs each.

The professor will create its own test function that your code must pass in order to be considered as a correct solution.