UNIVERSITY OF PUERTO RICO AT BAYAMON

**COTI 4039 – COMPARATIVE PROGRAMMING LANGUAGES**

**ASSIGNMENT #3 – 100 points**

**NOTE**: It is no allowed to use features of the programming language that have not been discussed in class, assigned to read or explicitly authorized.

1. (15 points) Write a C program (diving\_competion.c) that reads a file with the data of a diving competition (competition.txt) and displays a report of the results with the following format:

Diving Competition Results

Id Name Trunc. Average

-- ---- --------------

1234 Doe, John 8.625

5678 Doe, Jane 8.625

7483 Almighty, Zeus 9.813

8752 Pitt, Brad 8.375

1235 Jolie, Angelina 8.500

The winner is Zeus Almighty!

Your program must read each record from the file and place it in a variable that is an instance of the following data structure:

#define ID\_SIZE 4

#define NAME\_SIZE 10

#define NUM\_SCORES 10

typedef struct {

char diver\_id[ID\_SIZE + 1],

first\_name[NAME\_SIZE + 1],

last\_name[NAME\_SIZE + 1];

float scores[NUM\_SCORES];

} Diver;

In addition to displaying the report on the screen, your program must also create a file with a copy of the report (report.txt). As you can see, the program computes the truncated average for each diver. The truncated average is calculated by discarding the maximum and minimum score.

2. (15 points) Write a C program (employee\_payroll.c) that creates an array of at least five different kinds of employees and displays their data, including their weekly salary with the appropriate format. Each employee has a kind, an id, and a full name. There are two kinds of employees: hourly employees and sales employees. Each hourly employee has a pay rate and a number of hours worked. Each sales employee has a commission rate and a sales amount. Thus, you should use the following data structure:

#define ID\_SIZE 4

#define NAME\_SIZE 20

typedef enum {HOURLY, SALES} EmployeeKind;

typedef struct {

float pay\_rate,

hours\_worked;

} HourlyEmployee;

typedef struct {

float commission\_rate,

sales\_amount;

} SalesEmployee;

typedef struct {

EmployeeKind kind;

char employee\_id[ID\_SIZE + 1],

full\_name[NAME\_SIZE + 1];

union {

HourlyEmployee hourly;

SalesEmployee sales;

};

} Employee;

The weekly salary of an hourly employee is calculated as the product of the pay rate and the hours worked. If the hours worked is greater than 40, an overtime rate of 1.5 is applied. On the other hand, the weekly salary of a sales employee is calculated as the product of the sales amount and the commission rate.

3. (10 points) Modify linked\_list.c to add the following:

* A function that returns the number of elements in the given list. Its prototype should be:

int length\_list(const List \*lst);

* A function that returns the minimum element in the given list. Its prototype should be:

int minimum\_list(const List \*lst);

4. (15 points) Modify binary\_tree.c to add the following:

* A function that returns the number of elements in the given tree. Its prototype should be:

int size\_tree(const Tree \*bst);

* A function that returns the number of levels in the tree. Its prototype should be:

int depth\_tree(const Tree \*bst);

* A function that returns the minimum element in the tree. Its prototype should be:

int minimum\_tree(const Tree \*bst);

5. (20 points) Write a C source file (queue.c) that implements Queue as an opaque data type using a linked list as its data structure. A queue is a list in which elements are added in one end and removed from the opposite end. The interface for this data type is located on queue.h and you can test your implementation using queue\_menu.c. When implementing the queue, use the following data structure:

typedef struct node Node;

struct node {

void \*data; // pointer to generic data

Node \*next; // pointer to the next node in internal linked list

};

struct queue {

Node \*front; // pointer to the front element

Node \*rear; // pointer to the rear element

};

**Hint:** The following algorithm describes how to add a new rear element to the queue:

*Create a new node*

*If front is null:*

*Set front to point to the new node*

*else*

*Link the new node next to the current rear node*

*Set rear to point to the new node*

The following algorithm describes how to remove the current front element from the queue:

*If queue is empty:*

*Return null*

*else*

*Save the data of the current front node*

*Set front to point to the node next to the current front node*

*If front is null:*

*Set rear to null*

*Return the saved data*

6. (25 points) Write a C source file (vector.c) that implements Vector as an opaque data type using a dynamically allocated array as its data structure. A vector is a kind of array where elements can be added or removed but are kept contiguous. The interface for this data type is located on vector.h and you can test your implementation using vector\_test.c. When implementing the vector, use the following data structure:

struct vector {

int capacity; // current capacity of internal array

void \*\*data; // pointer to array of pointers to generic data

int size; // number of elements

};

The following is a running of vector\_test.c:

An empty vector has been created.

This is the vector: [ ].

It has 0 elements.

After adding some elements...

This is the vector: [ cat tree house woman man ].

It has 5 elements.

After inserting some elements...

This is the vector: [ cat dog tree bicycle house boat woman man ].

It has 8 elements.

The element at index #5 is boat.

After modifying an element...

This is the vector: [ cat dog tree bicycle house ship woman man ].

It has 8 elements.

The element at index #5 is ship.

Searching for the ship...

The element was found at index #5.

After sorting the vector...

This is the vector: [ bicycle cat dog house man ship tree woman ].

It has 8 elements.

Searching for the ship...

The element was found at index #5.

After removing the element at index #5...

The ship was removed.

This is the vector: [ bicycle cat dog house man tree woman ].

It has 7 elements.

The vector has been deallocated.

**Note:**

Do not forget to document your programs with the name of the source file, your name and student number, the date of creation and the purpose. Document each function with a sentence that summarizes what it does, what it receives and what it returns.