A.Y. 2023-2024 | Second Semester

CIS 2101 (Data Structures and Algorithms)

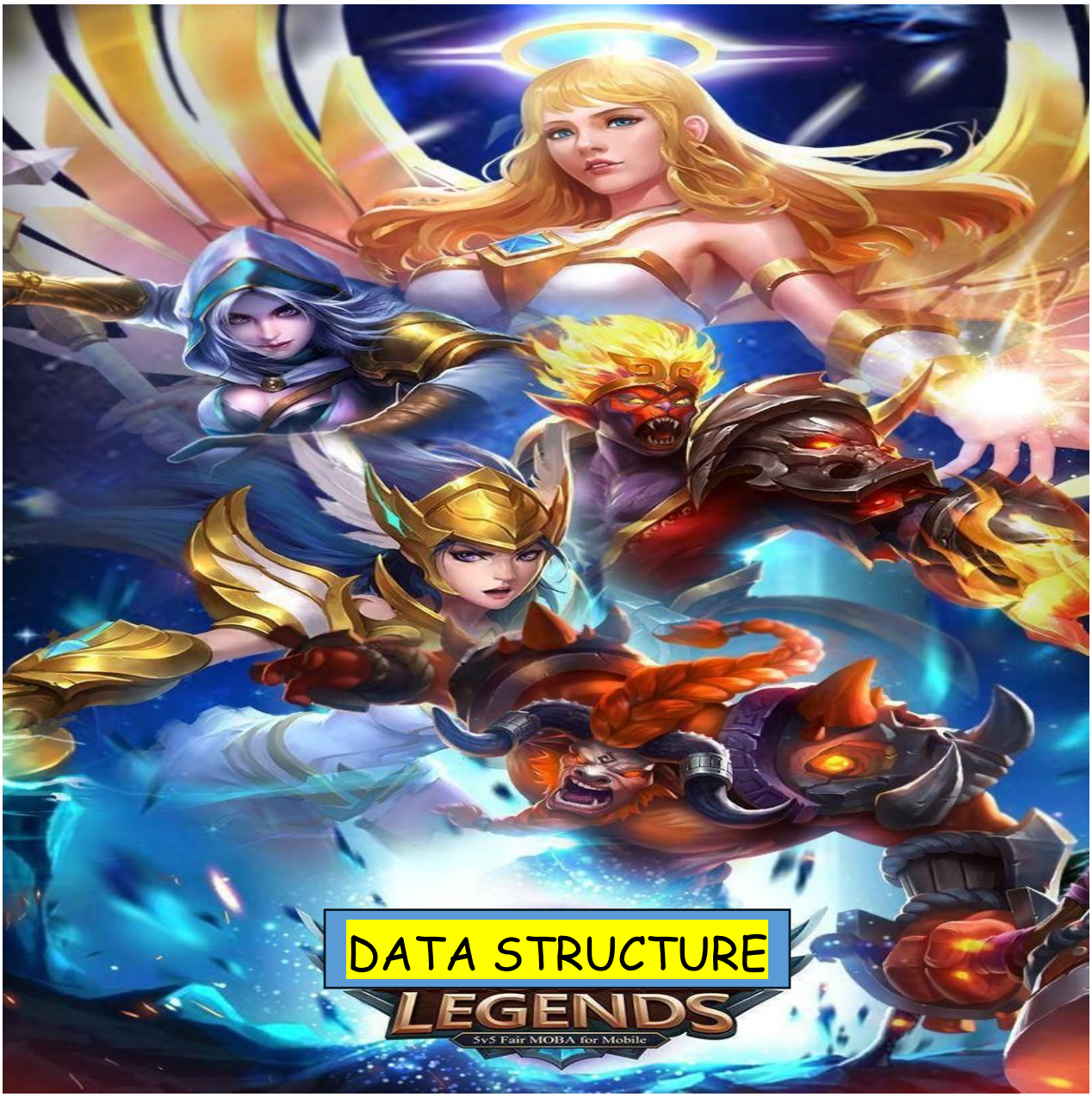
Hands-on Premidterm Exam

Name: _____

Program and Year: _____ Date: _____

SCORES		
Part 1		**35**
Part 2		**40**
Part 3		**60**
Part 4		**70**
Part 5		**95**
Total		**300**

THE DATA STRUCTURE LEGENDS GAME (vol. 1)

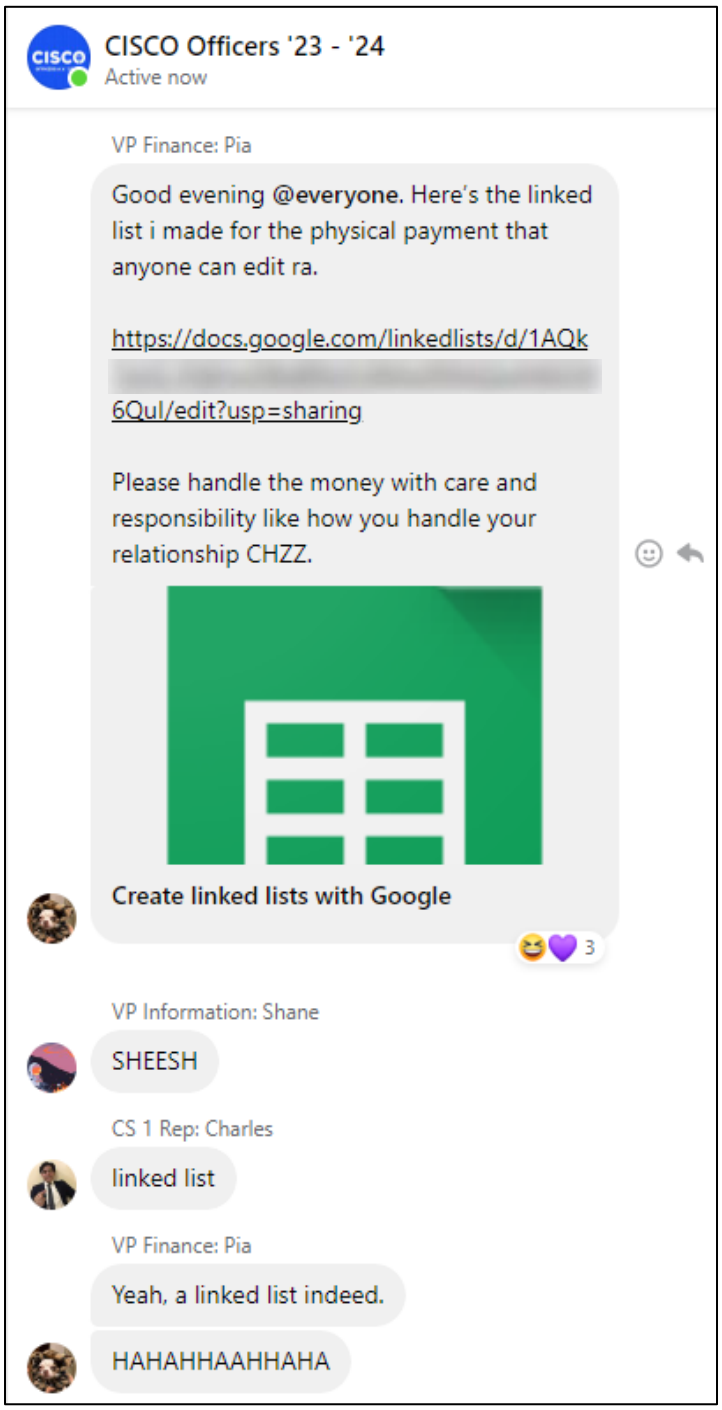


The story: Pia, the current VP Finance for CISCO, is currently addicted to the new release of the game "Data Structure Legends" as she kept on playing the practice mode of linked list simulation battles to gain the XP she needed to unlock the premidterm exam. As soon as she resumed her council duties which was to create a tracker for payments for the upcoming DCISM General Assembly, she made a mistake that instead of creating just a Google sheet for the officers to use, she created a Google Linked List instead (see photo).

Now the rest of the officers were struggling to understand its implementation. But as a data structure student, you are now approached by Pia's friend Lorraine, to help automate the tracking of payments using the linked list Pia created, since Pia has no more time to edit her output and message as she will be busy practicing linked lists.

The struggle for Pia unfortunately did not just end there. When the student adviser of CISCO (Master Wayne) found out about it, he decided to tweak this problem further by transforming the data set for the DCISM students and the collected transactions so far into array lists and cursor-based lists respectively. With these crazy ideas brewed together, Pia and her classmates now receive a reviewer that would put their data structure skills into an ultimate test.

Would you be that one person who can conquer this challenge and save Pia, Lorraine, and the council before the General Assembly commences within a week and ensure all the participants have their payments properly settled?



Additional notes:

- **The data structure definition for the problems can be found at the last page.**
- **The list of students currently enrolled in DCISM is implemented using an array implementation.**
    - **Each array element contains:** Email, name (uncleaned), and program/year (uncleaned).
- **The list of payments will be initially presented through an array, which you will be transforming into a linked list filtered with valid transactions and grouped by program and year.**
    - **Each array element contains:** ID and transaction number.
    - **Each node contains:** Email, name (cleaned), transaction number, and timestamp (cleaned).
- **The list of valid GCash transactions extracted from the application is implemented using cursor-based implementation.**
    - **Every node in the virtual heap stores:** Transaction ID, timestamp (uncleaned), array of pointers representing a list of nodes that reference the transaction ID, and count of referenced nodes.

Upon performing all the tasks, you will be able to properly eliminate all the invalid payment entries in the payment list and retain the valid ones. During the process, you will be also doing data cleaning such as string manipulation to fix the data formatting inconsistencies and implement the cleaned content into the filtered list.

This filtered list will help CISCO identify the names and emails of the participants with valid registrations for the upcoming General Assembly. The identified students will then be notified via email of their registration statuses.

Perform all the following tasks below. For each number, modify the code file (.c) in the given folder for your output. Optimize your code for efficiency and readability by avoiding unnecessary statements and control structures that would degrade the program's performance.

Program files that are not running due to any kind of error will not be checked, and a score of zero (0) will be immediately assigned for that section, so review your codes properly before submitting. Only correctly implemented functions will be awarded points which are indicated at the rightmost column of the table of functions.

Note: This shall serve as a practice for you to better prepare for the difficulty adjustments expected for Data Structures and Algorithms. You are free to search for additional resources to help you understand new or unfamiliar concepts. DO NOT ATTEMPT TO USE ANY AI TOOL TO DIRECTLY GENERATE OR ADJUST YOUR CODE, NOR ASK FOR ANSWERS DIRECTLY FROM ANY OTHER PEOPLE!

Task 1 (initializations – students and payments)

- Objective: To perform all the initialization operations for the student info and payment info data structures.
- Implement the functions below:

Function header	Function definition	Points
`void initStudentList( StudentList* L)`	Initializes the student array list by setting its fields appropriately.	7

`void displayStudentList( StudentList L)`	Displays the student array list with proper alignments. See sample format below:
	
	`=== STUDENT LIST ===`
	`Index Email Name Program and Year`
	`0 XXXX XXXXXX XXXXXXXXX`
	`1 XXXX XXXX XXXXXXXXXXX`
		5

`void initTempArrayList( StudentPaymentTempArrayList *L)`	Initializes the student payment temporary array list by setting its fields appropriately.	4

`void displayTempArrayList( StudentPaymentTempArrayList L)`	Displays the student payment temporary array list with proper alignments. See sample format below:
	
	`=== PAYMENT ARRAY LIST ===`
	`Index ID TransactionID`
	`0 XXXX XXXXXX`
	`1 XXXX XXXX`
		4

`void initPaymentLinkedLists( PaymentLinkedLists *L)`	Initializes the payment linked lists by setting its fields appropriately.	5

`void displayLinkedList( StudentPaymentList L)`	Displays the payment linked list (single chain) with proper alignments. See sample format below:
	
	`Index Email LN FN MI TransID Timestamp`
	`0 XXXX XXX XXX X XXXXXX XXXX-XX-XX XX:XX:XX`
	`1 XXXX XX XX X XXXXXXX XXXX-XX-XX XX:XX:XX`
		5

`void displayPaymentLinkedLists( PaymentLinkedLists L)`	Displays the payment linked list (all chains). This function must invoke `displayLinkedList()`. See sample format below:
	
	`=== BS COMPUTER SCIENCE YEAR 1 ===`
	`Index Email LN FN MI TransID Timestamp`
	`0 XXXX XXX XXX X XXXXXX XXXX-XX-XX XX:XX:XX`
	`1 XXXX XX XX X XXXXXXX XXXX-XX-XX XX:XX:XX`
		5

TOTAL POINTS (TASK 1)		**35**

Task 2 (initializations and management operations – transactions)

- Objective: To perform all the initialization and virtual heap management operations for the transaction data structures.
- Implement the functions below:

Function header	Function definition	Points
`void initVirtualHeap( VirtualHeapForTransactions *VH)`	Initializes the virtual heap containing the transaction records by linking the next fields and setting avail to an appropriate value.	8

`void displayVirtualHeap( VirtualHeapForTransactions VH)`	Displays the virtual heap of the transaction records including the avail field. See sample:
	`=== VIRTUAL HEAP ===`
	`Index of next available node: X`
	`Index TransID Amount Timestamp # Nodes Next`
	`0 XXXX XXX.XX XXXX-XX-XX XX:XX:XX X XX`
	`1 XXXXXX XX.XX XXXX-XX-XX XX:XX:XX XX XX`
		4

`void initTransactionSet( TransactionSet *S, VirtualHeapForTransactions *ptr)`	Initializes the transaction set by setting its fields appropriately denoting an empty set.	4

`void displayTransactionSet( TransactionSet S)`	Displays the transaction set. See sample format below:
	`=== TRANSACTION SET ===`
	`Index TransID Amount Timestamp # Nodes`
	`0 XXXX XXX.XX XXXX-XX-XX XX:XX:XX X`
	`1 XXXXXX XX.XX XXXX-XX-XX XX:XX:XX XX`
		8

`int allocSpace( VirtualHeapForTransactions *VH)`	Removes the first of the chain of available nodes in the virtual heap and returns its index.	8
`void deallocSpace( VirtualHeapForTransactions *VH, int node)`	Inserts the node with the given index into the start of the chain of available nodes in the virtual heap.	8
TOTAL POINTS (TASK 2)		**40**

Task 3 (cleaning and helper functions):

- Objective: To implement the data cleaning operations which will be invoked when creating the filtered linked list of valid transactions.
- Implement the functions below:

Function header	**Function definition**	Points
`int emailContainsID( char email[], char ID[])`	Returns 1 if the email string contains the ID string and 0 if otherwise.	5
`NameType extractNames( char name[])`	Returns a NameType structure containing the extracted first name, last name, and MI from a given name string which can be either in a last name first format or first name format.	15
`program extractProgram( char progYear[])`	Determines and returns the program of a given progYear string. Possible variations of the string include "BSCS", "BS Computer Science", "BS CS", "computer science" (abbreviation or complete title of program).	15
`int extractYear( char progYear[])`	Returns the year level from the given progYear string.	3
`int compareTimestamps( int time1[], int time2[])`	Returns -1 if the first timestamp is earlier than the second timestamp, 1 if the first timestamp is later than the seconf timestamp, and 0 if both timestamps are equal.	7
`char* toDateTimeString( int timestamp[])`	Returns the string equivalent (YYYY-MM-DD HH:MM:SS) of a given timestamp array. Note: There must be leading zeroes if the values are one digit i.e. 01, 02, …, 09.	15
TOTAL POINTS (TASK 3)		**60**

Task 4 (data storage and searching):

- Objective: To populate the list of students, payments, and transactions into their respective data structures initialized in tasks 1 and 2. Refer to the data structure definition to how the items in each list should be sorted.
- Implement the functions below:

Function header	Function definition	Points
`void insertToStudentList( StudentList *list, char email[], char name[], char progYear[])`	Inserts a student record into its appropriate position in the given list.	10
`void populateStudentList( StudentList *list)`	Populates the student list with 15 items by repeatedly calling **`insertToStudentList()`**.	5
`void insertToPaymentTempArray( StudentPaymentTempArrayList *list, char ID[], char transactionID[])`	Inserts a payment record in the last position of the given list.	5
`void populatePaymentTempArray( StudentPaymentTempArrayList *list)`	Populates the payment list with 15 items by repeatedly calling **`insertToPaymentTempArray()`**.	5
`void insertToTransactionSet( TransactionSet *set, char transactionID[], float amount, int timestamp[])`	Inserts a transaction record into its appropriate position in the given set. The amount can be any multiple of 750.00 indicating group transactions, or a value lower than that indicating an invalid transaction during filtering.	20
`void populateTransactionSet( TransactionSet *set)`	Populates the transaction set with 15 items by repeatedly calling **`insertToTransactionSet()`**.	5
`StudentType* searchStudentByID( StudentList list, char studID[])`	Returns the address of the student record where its email string contains the given studID string, and NULL if there is no match.	8
`TransactionType* searchTransactionByID( TransactionSet set, char transactionID[])`	Returns the address of the transaction record where its transaction ID string contains the given transactionID string, and NULL if there is no match.	12
TOTAL POINTS (TASK 4)		**70**

Task 5 (generating and filtering results):

- Objective: To create the filtered linked list of valid transactions and segregate the valid and invalid transactions in the transaction set. The output files generated from the last function will be used by CISCO to notify the students with valid and invalid payments for the DCISM General Assembly.
- Implement the functions below:

Function header	Function definition	Points
`void createPaymentLinkedLists( StudentPaymentTempArrayList payments, StudentList students, TransactionSet transactions, PaymentLinkedLists *output)`	Populates the payment linked lists *output* by scanning for matching student record in *students* from the given *ID* and matching transaction record in *transaction* from the given *transactionID* for each item of the *payments* list.	30

	Only proceed with the node creation and insertion of there are matching records found for both *ID* and *transactionID*. Retrieve the rest of the information by accessing the matched records and apply appropriate cleaning and extraction methods to be able to properly insert those fields in the student payment structure. Once the node is created, go to the matching transaction record and append the address of that node in the *paymentNodes* array and also update the *nodesCount* field to track its usage. Finally, insertions should be done at the appropriate chain which can be determined by the program and year extracted. Do update the paymentCounts array as well. Verify the contents by calling the appropriate display function.	
`TransactionSet getInvalidTransactions( TransactionSet *allTransactions)`	Removes the invalid transaction records from *allTransactions* and put them into a new set to be returned to the calling function. An invalid transaction record is one whose amount is less than the product of P750.00 times the number of nodes referenced (i.e. 2→P1.500.00, 3→P2,250.00, …), signifying overusage of the same transaction ID while populating the payments linked lists. Verify the contents by calling the appropriate display function.	20
`StudentPaymentList getInvalidPayments( TransactionSet dupTransactions)`	For each item in the given list, access the referenced nodes and **remove** them from the original linked list. **Gather** those removed nodes and place them altogether into a newly created linked list (still sorted) and return to the calling function. Verify the contents by calling the appropriate display function.	25
`void displayValidPaymentPercentage( StudentPaymentTempArrayList all, PaymentLinkedLists validPayments)`	Display the count and percentage of valid payments with respect to total payments. See display format below. `Valid payments: 75/100 (75.00%)`	5
`void generateEmailLists( PaymentLinkedLists validPayments, StudentPaymentList invalidPayments)`	Generate two files (`validEmails.txt` and `invalidEmails.txt`) that will contain the emails with valid and invalid payments based from the *validPayments* and *invalidPayments* lists respectively.	15
TOTAL POINTS (TASK 5)		**95**

DATA STRUCTURE DEFINITIONS:

```
#define PIA_MAGIC_NUMBER 72453

/***** STUDENT INFO - ARRAY *****/

typedef struct{
	char email[30]; // "<IDNumber>@usc.edu.ph"
	char name[100]; // "FirstName MI. LastName" or "LastName, FirstName MI."
	char progYear[100]; // "BS Computer Science - 3" or "BSCS-3" (varied spacings)
}StudentType;  // Information of a student

typedef struct{
	StudentType *students;
	int studentCount;  // Stores the number of active students
}*StudentList; // Student list using array implementation version 4, sorted by email.


/***** STUDENT PAYMENTS - ARRAY *****/

typedef struct{
	char ID[10];
	char transactionNumber[40];
}StudentPaymentPartialType;

typedef struct{
	StudentPaymentPartialType list[PIA_MAGIC_NUMBER];
	int studentCount;	// Stores the number of active elements
}StudentPaymentTempArrayList; // Temporary array implementation of payments which
   // will be transformed into a filtered linked list shown below. No sorting needed.


/***** STUDENT PAYMENTS - LINKED LIST *****/

typedef struct{
	char firstName[30], MI, lastName[30];
}NameType;

typedef struct{
	char email[30];
	NameType name;
	char transactionNumber[40];
	char timestamp[100]; // Format: "YYYY-MM-DD HH:MM:SS"
}StudentPaymentType;

typedef struct SPNode{
	StudentPaymentType studPayment;
	struct SPNode* prev; // Address of previous node
	struct SPNode* next; // Address of next node
}StudentPaymentNode, *StudentPaymentList; // Note: Doubly linked list

typedef enum{
  // Constants - Format in new CSV file
	BSCS,   // BS Computer Science
	BSIT,   // BS Information Technology
	BSIS    // BS Information Systems
}program; // Recall the equivalent integer values
```

```c
typedef struct{
	StudentPaymentList lists[4][3];
	int paymentCounts[4][3];
}PaymentLinkedLists;

/*
   Definition of PaymentLinkedLists:
 - lists is a 2D array of pointers to the head of the linked list of student nodes,
   grouped by year level and then by program.
 - paymentCounts is a 2D array of integers storing the count of payment nodes for
   each linked list in the 2D pointer array.
 - First index represents the year level (0:first year, 1:second year, 2:third
   year, 3:fourth year), and second index represents the integer value of the
   program enum type defined above.
 - Example: Payments from BSCS-1 students are stored in a linked list whose head
   pointer is in lists[0][0]. The # of BSCS-1 payments is in paymentCounts[0][0].
 - Each list will be sorted in ascending alphabetical order of last name followed by
   the first name.
*/

/***** TRANSACTONS - CURSOR-BASED *****/

typedef struct{
	char transactionNumber[40];
	float amount;     // P750.00 per person/node
	int timestamp[6]; // 0:year, 1:month, 2:day, 3:hour, 4:minute, 5:second
	StudentPaymentList *paymentNodes[10]; // Addresses of the payment nodes
	                                      // having the given transaction number
	int nodesCount;       // Number of payment nodes currently referenced.
}TransactionType; // Info for each transaction.
  // If amount is less than P750 times number of nodes, means the transaction ID is
  // overused. Invalidate or remove all the payment nodes that used this transaction
  // number by accessing appropriately from the nodes field.

typedef struct{
	TransactionType data;
	int next; // index of the next transaction node
}TransactionVHNode;

typedef struct{
	TransactionVHNode transactions[PIA_MAGIC_NUMBER];
	int avail; // index of the first free transaction node
}VirtualHeapForTransactions;

typedef struct{
	int head;  // -1 is a sentinel value for NULL
	VirtualHeapForTransactions *VHptr;
}TransactionSet; // index of the node in virtual heap storing the first item of the
	// transaction set using cursor based implementation, sorted by timestamp.
```

Bonus! Draw a representation of a variable of each of the following data types:

1. StudentList (With 2 items)
2. PaymentLinkedlists (With 2 items)
3. TransactionSet (With 2 items)