National Exams December 2018

17-Comp-A4

Program Design and Data Structures

3 Hours Duration

Notes:

1. If doubt exists as to the interpretation of a question, the candidate is urged to submit with the answer paper a clear statement of any assumptions made.

2. No calculator permitted. This is a Closed book exam.

3. Answer any six of the nine questions.

4. Any six questions constitute a complete paper. Only the first six questions as they appear in your answer book will be marked.

5. For questions that ask the candidate to write a program, **pseudocode** or any high-level language (e.g. **C** or **C++**) is acceptable unless otherwise specified. In all cases, marking will emphasize the operation of the program and not syntactic details.

6. All questions have equal weight. The total mark is out of 120.

Question 1:	(a) 10 marks; (b) 10 marks.
Question 2:	20 marks.
Question 3:	20 marks.
Question 4:	20 marks.
Question 5:	20 marks.
Question 6:	20 marks.
Question 7:	20 marks.
Question 8:	20 marks.
Question 9:	20 marks.

Question 1. *Programming.*

(a) *Buoyancy* is the ability of an object to float. Archimedes' principle states that the buoyant force is equal to the weight of the fluid that is displaced by the submerged object. The buoyant force can be computed by: $F_b = V \times \gamma$

Where F_b is the buoyant force, V is the volume of the submerged object, and γ is the specific weight of the fluid. If F_b is greater than or equal to the weight of the object, then it will float, otherwise it will sink.

Write a program that inputs the weight (in pounds) and radius (in feet) of a sphere and outputs whether the sphere will sink or float in water. Use γ =62.4lb/ft^3 as the specific weight of water. The volume of a sphere is computed by $(4/3)\pi r^3$.

(b) Consider the scene in a closed-room secret design session for the 2005 model of a big-name automobile. All around the conference table are engineers and executives trying to strike a balance between greed (which leads them to build the cheapest car possible) and fear (which pulls them in the other direction, towards more reliable cars).

The relation between the mean time between failure (MTBF) of a system and its subsystems can be described as:

$$\frac{1}{MTBF_{total}} = \frac{1}{MTBF_{sub1}} + \frac{1}{MTBF_{sub2}} + \cdots + \frac{1}{MTBF_{subn}}$$

and is correct if the failure of any subsystem dooms the entire car.

Assume that the costs and MTBF's of various subsystems are:

Subsystem		MTBF	Manufacturing Cost
brakes	disc	4 years	\$15.00
	drum	5 years	\$25.00
engine	wankle	3 years	\$1,067.00
	conventional	6 years	\$1,850.00
suspension	air	9 years	\$430.00
	oil	7 years	320.00
electrical	computer	2 years	130.00
	standard	4 years	\$40.00

Write a program that models each combination of subsystems and answers the following:

1. Which system is the cheapest?
2. Which has the longest MTBF?
3. Which system has the lowest cost per failure-free year?

Hint: use nested loops.

Question 2. *Programming.*

An integer is said to be "self-describing" if the following holds. When digit positions are labeled 0 to n-1 from left to right, the digit in each position is equal to the number of times the digit position appears in the number. Thus, the integer 2020 is a 4-digit self-describing integer. Position 0 has value 2 and there are two 0's in the number; position 1 has value 0 and there are no 1's in the number; position 2 has value 2 and there are two 2's; and position 3 has value 0 since there are zero 3's. These are also self-describing integers: 1210 and 3211000.

Write a program that prompts the user for a positive integer and responds by printing a message that indicates if the entered number is self-describing or not.

Hint: you may want to read the digits into an array.

Question 3. *Programming.*

A **magic square** is a square array of integers such that the sum of every row, the sum of every column and the sum of each of the two diagonals are all equal. This is an example of a 4x4 magic square.

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

Write a program that reads the size of the square *n* and allocate an *n*x*n* two-dimensional array of integers. The program then reads n^2 integers that represent the rows of the square from top to bottom. The program finally determines whether or not the square is a magic square.

Question 4. *File I/O.*

Write a program that will compute the average word length (i.e., the average number of letters in a word) for a file that contains text. A word is defined as any string of symbols that is preceded and followed by one of the following: a blank, a comma, a period, the beginning of a line, or the end of a line.

Question 5. *File I/O.*

A plain text file may contain 3 types of parentheses: round brackets: ( ), square brackets: [ ], and curly brackets: { }. Parentheses are *balanced* if every opening parenthesis is closed in the reverse order opened. Thus '([ ])' is balanced but '([ )]' is not. Write a program to prompt the user for a file name, open and read plain text from the file and determine if parentheses that appear in the text are balanced or not. The program should print a simple message that indicates if all parentheses are balanced or not.

Hint: You may want to use a stack. If you decide to use one, you need not show the code that implements it, just use it.

Question 6. *Object-Oriented Design.*

Sets of numbers are used in many applications. However, some languages, like **C++**, do not have "Set" as a data type, nor do they directly support set operations.

Design and write a **C++** class (call it **Set**) for supporting sets and their operations. Your class should allow for the declaration of sets, both empty and initialized with elements. It should allow for the following set operations: addition of an element, deletion of an element, and checking if an element is a member of the set. Your implementation should allow for sets of various number types (i.e., sets of integers, sets of floats, etc).

You have freedom to select the syntax of the above operations. State any assumption you make clearly. Separate your class into a **Set.h** header file and a **Set.cc** implementation file.

Question 7. *Pointer-based Data Structures.*

A node of a linked list can be defined as follows, expressed in C:

```
typedef struct node {
    int data;
    struct node *next;
} NODE;
```

Write a function `dupl()` that duplicates each node in a linked list. More specifically, for each node *A* on the list, the function creates a new node that has the same `data` value as *A* and inserts this new node immediately following *A*.

The header of the function is shown below. The function must work correctly for empty lists.

```
/* duplicate nodes in the list pointed to by head */
void dupl(NODE *head);
```

Question 8. *Pointer-based Data Structures.*

A node in a binary tree can be defined as follows, expressed in C:

```
typedef struct treenode {
    int data;
    struct element *left;
    struct element *right;
} TreeNode;
```

Write a function `inorder()` that traverses the tree using inorder traversal. The header of the function is shown below. The function must work correctly for an empty tree. Recall that inorder traversal visits the left side of the tree, followed by visiting the node and finally visiting the right side of the tree. Assume that visiting a node simply prints its `data` to the standard output.

```
/* Inorder traversal
*/
void inorder (TreeNode *root);
```

Question 9. *Algorithm Design.*

Consider an array of length `n` holds two different values: `0`'s and `1`'s. Write a program that puts all the `0`'s the left of the array and the `1`'s to the right. This is an example for a 15-element array:

```
Array at input:   0 1 0 1 0 0 1 1 0 1 0 1 0 0 0
Array at output:  0 0 0 0 0 0 0 0 0 1 1 1 1 1 1
```

You may not use a sorting program, nor may you go through the array to count the number of `0`'s and `1`'s. You must solve the problem in one pass or traversal of the array.