請勿攜去
Not to be taken away

香 港 中 文 大 學
The Chinese University of Hong Kong

二0一八至一九年度下學期科目考試

Course Examination 2nd Term, 2018-19

科目編號及名稱
Course Code & Title : CSCI 3100 Software Engineering

時間 小時 分鐘
Time allowed : 3 hours 0 minutes

學號 座號
Student I.D. No. : Seat No. :

Instructions

Write your Student ID above and in the Answer Book first.

Part I multiple choice questions

All **Part I** questions are multiple choice questions. Each multiple choice question has **exactly one** correct answer. Put down your choice ("A"-"E") in the box at the left margin of each question. *Answers elsewhere will not be marked.* Each question is worth 2 points. An incorrect answer receives 0 points (no negative points), and an unanswered question receives 0 points. The maximum score of **Part I** is 40 points.

Part II open questions

All **Part II** questions (totally 6 of them) are essay questions, each worthy of various marks. Answer all 6 questions. Write all your Part II answers on the Answer Book, and clearly identify the corresponding question numbers. *Answers elsewhere or not clearly identified will not be marked.* The maximum score of **Part II** is 70 points.

Part I Multiple choice questions (2 points each. Write the answer in the left margin box. 40 points in total.)

MCQ not to be provided

P.2-8

Part II

Note:

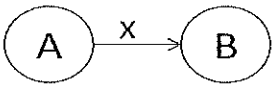
(1) All **Part II** questions (totally 6 questions) are essay questions, each worthy various points as indicated. Answer all 6 questions. The maximum score of **Part II** is 70 points.
(2) Write **all** your Part II answers on the Answer Book.

Question 1: Specification Technique – Finite State Machine (10 points)

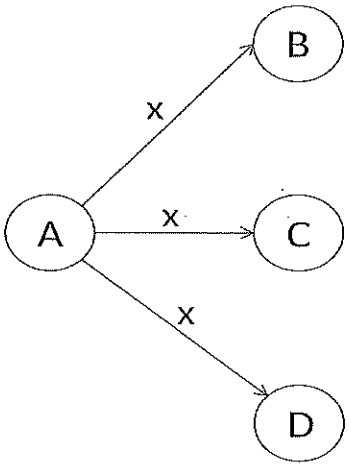
Finite State Machines (FSM) are often used to recognized patterns in the text. Generally, there are two types of finite state machine, deterministic and non-deterministic. In deterministic finite state machine, there's exactly one state transition for every input-state pair.

In non-deterministic finite state machine, there can be 0 or more state transitions for every input-state pair. When there's no state transition for a given input-state pair, then we say that the finite state machine had crashed, meaning that it can't proceed processing the input, and therefore it doesn't accept the input.

An example is provided in the following illustration. In a deterministic finite state machine, from ONE state, the machine can go to another ONE state given the input. In non-deterministic finite state machine, from ONE state, the machine can go to MANY states given one input.



(a). Deterministic



(b). Non-Deterministic

(1) Design a non-deterministic FSM that recognizes any sequence of '0's and '1's which contains the pattern '0100'. (4 points)

(2) Based on your solution as an example, can you guarantee to design a non-deterministic FSM to accept any one specifically expressed pattern, such as '0100' above? If so, how? If not, why? You need to concretely explain your solution. (2 points)

(3) Finally, design a deterministic FSM that recognizes any sequence of '0's and '1's which contains the pattern '0100'. (4 points)

Question 2: Specification Technique – Petri Nets (15 points)

Petri Nets are often designed to model the workflow of process activities. In this question, let us consider an integrated system devoted to ***Healthcare at Home*** management at the operational level. Suppose that we have a smart home, i.e., a home equipped with sensors, actuators and communication networks connecting the electrical devices and allowing them to be remotely monitored and controlled. The system is developed to monitor the daily living of an inhabitant into his/her dwelling, detect the possible troubles and accidents, and communicate with family, doctors and emergency services.

(1) First of all, we take a look at the module that manages the fall accident of an inhabitant in this system. Initially, the module is working quietly until the detection component detects a fall in the apartment, which enables a tolerance period of 10 seconds. The inhabitant should push the "ok_btn" button to indicate that he/she is fine. Otherwise, after the 10 seconds, the system will initialize the emergency procedure determined by a fall and call the doctor and family. However, during the emergency procedure, there are two possibilities: (a) if the doctor push the "doc_ok_btn" after checking the inhabitant; (b) or the inhabitant pushes the "ok_btn". In these two cases, the system will send a message of "no problem" to family or to the doctor and the family, respectively. Please draw a **Petri Net** to describe the workflow above, and provide *detailed explanations* on *all* the places and the transitions. (5 points)

(2) Now we consider another module that manages the events of heart attack or respiration troubles. Initially, the module is working quietly until the detection component detects a heart attack of the inhabitant. The heart attack can happen when the inhabitant is in the bed or in the bathtub. For the heart attack in bed, it can be detected immediately or after some respiration troubles. In the latter condition, if the heart attack is not detected and the inhabitant does not push the "ok" button to indicate that he/she is fine, then the doctor is advised. Three cases are possible then: (a) the heart attack is detected; (b) the inhabitant pushes the "ok" button; (c) the call center tries to wake up the inhabitant. Particularly, in case (b) the doctor is advised that the emergency is terminated. After the heart attack in bed is detected, the system will call an emergency. As for the heart attack in bathtub, the system will immediately call an emergency since it is detected in the inhabitant's bathtub. Subsequently, the doctor and the family are alerted. Please draw a **Petri Net** to describe the workflow above, and provide *detailed explanation* on *all* the places and the transitions. (10 points)

Question 3: Software Design with TDN, GDN and Stepwise Refinement (10 points)

Suppose we are going to design a distributed word counting system, which is to count the frequency of each word in the documents. The system should connect a database and fetch the documents. The system then **splits** the documents into N partitions and **allocates** them to N computing nodes. Upon finishing the allocation, the system uses the MapReduce method for parallel counting defined as the following: each node **maps** the document text into the format of (word, count) pairs. After the mapping, the pairs of each node are **shuffled** across each other to ensure that pairs with the same word are allocated on the same computing node; Finally, each node **reduces** the shuffled (word, count) pairs by summing up the count for each word as its frequency. The results of word frequency should be returned to users.

(1) Using TDN, describe design of the distributed word counting system. Concentrate your design on the *module interface*. The *implementation* section could be made short by using comments. (3 points)

(2) Continuing the above TDN, please use GDN to describe the distributed word counting system. (3 points)

(3) Now use stepwise refinement to describe how the word pairs are reduced (in C/C++ or pseudo codes). You may assume the pairs are already shuffled before the reduction. You only need to do it in two iterations, i.e., first provide a high level (but non-trivial) design, and then refine the design in more detail. (4 points)

Question 4: Software Design and UML (10 points)

In this problem, you are required to draw UML diagrams to describe the functionalities of a microblog platform.

(1) Please draw a UML class diagram to illustrate the relationship between different entities according to the following description (You don't need to draw the attributes and operations of each class). (4 points)

In the microblog platform, there are two types of blogs that Bloggers can post and share with others, which are **Micro blog** and **Long article**. These two blogs are both inherited from a class named **Blog**. Besides **Blogger**, this platform has another kind of **User** named **Admin** who is in charge of the management of all blogs. Each blogger has no limit on the number of micro blog they would like to post, but for long article, the maximum number is 50.

(2) Please draw a UML sequence diagram to depict the process of a blogger posting a long article according to the following description. (6 points)

After the blogger finishes writing a long article, he/she sends a posting long article request to the **client side**, which will then be forwarded to the **platform server**. Since each blogger has limited quota for long article, the server will inquire from **user database** about the feasibility of the blogger posting this long article. If yes, server will update blogger's quota in user database, save the long article to **blog database**, and send a successful posting message back to client side; otherwise, an unsuccessful posting message will be sent to the client side.

Question 5: Software Design and Implementation (15 points)

Given a binary tree, determine if it is a valid binary search tree (BST).

A BST is defined as follows:

- The left subtree of a node contains only nodes with keys **less than or equal to** the node's key.
- The right subtree of a node contains only nodes with keys **greater than or equal to** the node's key.
- Both the left and right subtrees must also be binary search trees.

For example,
```
   2          3
  / \        / \
 1   3      1   2
```
is a valid BST but the second tree is not, because the root node's value is 3 but its right child's value is 2.

Let's consider the binary tree defined as follows:

```
struct TreeNode {
  int val;
  struct TreeNode *left;
  struct TreeNode *right;
};
```

(1) The following function is a recursive solution. Fill out the 6 numbered blanks in the following function and put them in the Answer Book with the corresponding numbers. Also given the number of tree nodes is N, **what is the time complexity** (in big O notation) of this solution? (7 points)

```
#include <limits.h>
#include <stdbool.h>
#include <stdlib.h>
bool traverse(struct TreeNode *root, int upper_bound, int lower_bound) {
  if (root == NULL) return true;
  if (root->val >= _____①_____ || root->val <= _____②_____)
    return false;
  return traverse(root->left, _____③_____, _____④_____) &&
         traverse(root->right, _____⑤_____, _____⑥_____);
}
```

```
bool isValidBST_recursive(struct TreeNode *root) {
  return traverse(root, INT_MAX, INT_MIN);
}
```

(2) Before writing the non-recursive version, we need to define the Abstract Data Type of a *stack*. A stack could be defined in only four operations, two of them are **init** and **top**. What are the other two operations? Suppose we use array to store the stack, and **init** is defined as `struct TreeNode *stack[MAX_STACK]; int stack_top=0;` Given **top** is defined as `if (stack_top > 0) then stack[stack_top - 1] else NULL`, what are the implementations of the other two operations. (2 points)

(3) The following function is a non-recursive solution. Fill out the 6 numbered blanks in the following function and put them in the Answer Book with the corresponding numbers. (6 points)

```
// Suppose size of stack will not exceed MAX_STACK
#define MAX_STACK 100000

bool isValidBST(struct TreeNode *root) {
    if (root == NULL)
        return true;

    struct TreeNode *stack[MAX_STACK];
    int stack_top = 0; // Top of stack
    int upper_limit_stack[MAX_STACK];
    int upper_limit_stack_top = 0; // Top of stack
    int lower_limit_stack[MAX_STACK];
    int lower_limit_stack_top = 0; // Top of stack

    stack[stack_top++] = root;
    // Suppose all nodes' value are between INT_MAX and INT_MIN
    upper_limit_stack[upper_limit_stack_top++] = INT_MAX;
    lower_limit_stack[lower_limit_stack_top++] = INT_MIN;

    while (stack_top != 0) {
        struct TreeNode *node = stack[--stack_top]; // pop the top element
        int lower_limit = lower_limit_stack[--lower_limit_stack_top];
        int upper_limit = upper_limit_stack[--upper_limit_stack_top];
        if (node->right != NULL) {
            if (node->right->val > node->val) {
                if (upper_limit != INT_MAX && node->right->val >= ____⑦____)
                    return false;
                stack[stack_top++] = node->right;
                lower_limit_stack[lower_limit_stack_top++] = ____⑧____;
                upper_limit_stack[upper_limit_stack_top++] = ____⑨____;
            } else
                return false;
        }

        if (node->left != NULL) {
            if (node->left->val < node->val) {
                if (lower_limit != INT_MIN && node->left->val <= ____⑩____)
                    return false;
                stack[stack_top++] = node->left;
                lower_limit_stack[lower_limit_stack_top++] = ____⑪____;
                upper_limit_stack[upper_limit_stack_top++] = ____⑫____;
            } else
                return false;
        }
    }
    return true;
}
```

Question 6: Software Testing and Verification (10 points)

Given the following C program:

```
// C program to find smallest and second smallest elements
#include <stdio.h>
#include <limits.h> /* For INT_MAX */
1.  void print2Smallest(int arr[], int arr_size)
2.    {
3.     int i, first, second;
4.     /* There should be at least two elements */
5.     if (arr_size < 2)
6.     {
7.         printf(" Invalid Input ");
8.         return;
9.     }
10.    first = second = INT_MAX;
11.    for (i = 0; i < arr_size ; i ++)
12.    {
13.     /* If current element is smaller than first
14.     then update both first and second */
15.       if (arr[i] < first)
16.       {
17.         second = first;
18.         first = arr[i];
19.       }
20.       else if (arr[i] < second && arr[i] != first)
21.         second = arr[i];
22.     }
23.   if (second == INT_MAX)
24.     printf("There is no second smallest element\n");
25.   else
26.     printf("The smallest element is %d and second "
27.                "Smallest element is %d\n", first, second);
28. }
```

(1) Draw the control flow graph of this program for *edge coverage criterion* (i.e., C_1). (You can lump sequential statements into one node. The procedure partition is treated as one node without further analysis.) In this control flow graph, calculate the McCabe's number, or V (G). Show the calculation. (3 points)

(2) Design a test set to achieve 100% *statement coverage* (i.e., C_0). Then design additional tests to achieve 100% *edge coverage* (i.e., C_1). Then design additional tests to achieve 100% *condition coverage* (i.e., C_2). Finally, is it possible to design a test set that achieves 100% *path coverage* (i.e., C_3)? Why? Also, Can 100% *condition coverage* guarantee 100% *edge coverage* and *statement coverage*? (For simplicity, you only need to specify the input value arr[ ] for each test set, without considering arr_size, which is assumed to be automatically obtained as the length of arr[ ].) (5 points)

(3) Design a test set to perform black-box testing for the program. State clearly your testing strategy and corresponding test inputs. (2 points)

End of Exam: Have a good summer break!