注意：考試開始鈴響前，不得翻閱試題，並不得書寫、畫記、作答。

國立清華大學 108 學年度碩士班考試入學試題

系所班組別：**服務科學研究所 乙組**

考試科目(代碼)：**計算機概論(5302)**

—作答注意事項—

1. 請核對答案卷（卡）上之准考證號、科目名稱是否正確。
2. 作答中如有發現試題印刷不清，得舉手請監試人員處理，但不得要求解釋題意。
3. 考生限在答案卷上標記「由此開始作答」區內作答，且不可書寫姓名、准考證號或與作答無關之其他文字或符號。
4. 答案卷用盡不得要求加頁。
5. 答案卷可用任何書寫工具作答，惟為方便閱卷辨識，請儘量使用藍色或黑色書寫；答案卡限用 2B 鉛筆畫記；如畫記不清（含未依範例畫記）致光學閱讀機無法辨識答案者，其後果一律由考生自行負責。
6. 其他應考規則、違規處理及扣分方式，請自行詳閱准考證明上「**國立清華大學試場規則及違規處理辦法**」，無法因本試題封面作答注意事項中未列明而稱未知悉。

Please note that all questions have ONLY ONE CORRECT answer.

1. **(5%)** You find the following code (syntax inspired by C / Java) in a program:

```
for (int k = 0; k < 20; k = k + 2) {
  if (k % 3 == 1) {
    print(k + " ");
  }
}
```

What should be the result of running the code?

(a) `4 16`

(b) `4 10 16`

(c) `0 6 12 18`

(d) `1 4 7 10 13 16 19`

(e) `0 2 4 6 8 10 12 14 16 18`

2. **(5%)** A taxi company needs to model each type of car they use: *maximum riders* it can take, whether it has an *entertainment system*, and average number of *kilometers per liter* (kml). Which of the following is the **most appropriate object-oriented design**?

(a) Use one *class*, `Taxi`, with three instance *variables*: `max_riders`, `entertainment`, and `kml`

(b) Use four unrelated *classes*: `Taxi`, `Riders`, `Entertainment`, and `Kml`

(c) Use a *class* `Taxi` with three *subclasses* of itself: `Riders`, `Entertainment`, and `Kml`

(d) Use a *class* `Taxi`, with *subclasses* `Riders` and `Kml`, and another *class* `Entertainment`

(e) Use three *classes*: `Riders`, `Entertainment`, and `Kml`, each with a subclass `Taxi`

3. **(5%)** A university needs to implement a system to model which *courses* are offered by *teachers*, which *students* are taking a course, and which teachers are also *advisors* for students. Not all teachers are advisors, and not all advisors are teaching. Courses, teachers, advisors, and students have identity numbers and names. Consider some functions we must implement:

- `salary`: returns integer amount someone needs to be paid for their total teaching and/or advising.
- `courses_offered(...)`: returns array of courses a teacher is currently offering.
- `courses(...)`: returns array of courses a student is currently taking.
- `advisees(...)`: returns array of students who are being advised by and advisor.

Which of the following object-oriented elements is **not essential** to modeling this system?

(a) Separate classes for each of: **Course**, **Teacher**, **Advisor**, **Student**

(b) Superclass **Person** with subclasses: Teacher, Advisor, and Student

(c) Superclass **Faculty** with subclasses: Teacher and Advisor

(d) All of the above are essential

(e) More than one of the above (a – c) are not essential

The questions on <u>this page</u> are based on the following information.

Your colleague has written some code in a new language (inspired by Java / Python). It defines a function called `longestSequence` that is supposed to find the *longest continuous sequence of repeating* values in an array of numbers. For example:

```
arr = [4, 6, 8, 8, 2, 9, 8, 8, 8, 7]
result = longestSequence(arr, 8)
```

If it works, `result` should be `3` – the longest sequence of value `8` in `arr`.

```
01:  int longestSequence(Array<int> numbers, int value) {
02:    int length = 0
03:    int longest = 0
04:
05:    for (num in numbers) {
06:      if (num == value) {
07:        length++
08:      }
09:      else {
10:        if (length > longest) {
11:          longest = length
12:        }
13:      }
14:    }
15:
16:    if (length > longest) {
17:      longest = length
18:    }
19:    return longest
20:  }
```

4. **(10%)** During tests, you find that the function doesn't actually work as it is supposed to!

What number is the function returning, as it is written right now?

(a) The length of the shortest sequence of the value in the array of numbers

(b) The length of the array of numbers

(c) The total number of times a value occurs in the array of numbers

(d) The length of the first sequence of the value in the array of numbers

(e) The length of the last sequence of the value in the array of numbers

5. **(10%)** Your embarrassed colleague suddenly remembers he accidentally deleted the following line of code:

```
length = 0
```

Where will you add this line of code to make the function work as supposed?

(a) Between lines 5 and 6

(b) Between lines 9 and 10

(c) Between lines 10 and 11

(d) Between lines 12 and 13

(e) Between lines 13 and 14

6. (10%) While reviewing a colleague's code, you find two classes (written in a language inspired by Javascript) that uses the "<" operator for inheritance:

```
class Actor {
  function act() {
    print("acting ")
    rest()
  }

  function rest() {
    print("sleeping ")
  }
}

class BadActor < Actor {
  function act() {
    super.act()
    print("crying ")
  }

  function rest() {
    print("smoking ")
    super.rest()
  }
}
```

If you have an object called somebody defined by:

```
Actor somebody = BadActor.new()
```

What output should we expect to see if we run somebody.act() ?

(a) `acting sleeping`

(b) `acting sleeping crying`

(c) `acting sleeping smoking crying`

(d) `acting smoking sleeping crying`

(e) `acting smoking crying sleeping`

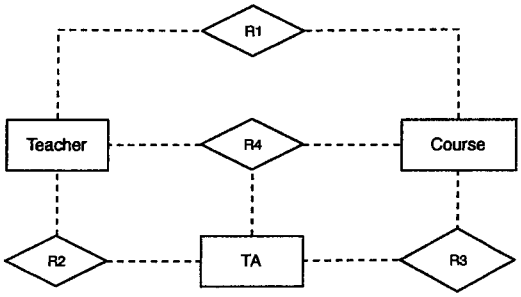
The questions on the <u>next page</u> are based on the information on this page.

Your company is hired by a high school to redesign a database system that models how teachers and teaching assistants (TAs) are assigned to courses. You must design a system that helps to follow their policies and constrains unwanted relationships.

Here are some definitions for you to know:

- There are two **Semesters** (fall/spring) in a **Year** (e.g., 2019).
- A **Course** is a semester-long activity on a topic (e.g., CS 321 – Algorithms)
- A **Teacher** is a person qualified to teach courses.
- A **TA** is a student who cannot teach courses but can assist qualified teachers.

The model must include all **three entities** (Teacher, TA, and Course) used by their prior system so that all data is kept.



<u>Your company is trying to decide</u> on which **binary relationships** (R1, R2, R3) **and/or ternary relationship** (R4) to use, and what those relationships might be used for.

Entities will be implemented as tables, and *relationships* (R1, R2, R3, R4) might be implemented either foreign keys (1:1) or join tables (1:n, m:n). Attributes (such as id, name, semester, year, etc.) can be assigned to entities or to join tables of 1:n and m:n relationships.

If you are allowed to implement a relationship, you are welcome to assign it any meaning to it you wish, and implement any unique **constraints** on table attributes for it that you need. You cannot place constraints on relationships you cannot model.

There are **three basic policies** your system must help enforce:

- A course can only be held at most once every semester (i.e., only one section).
- Only one teacher is responsible to teach a given course in a given semester, but the same course can be taught by different teachers in different semesters.
- One or more TAs can be optionally assigned to a course in a given semester.

7. (10%) If your team is only allowed to implement a **ternary relationship** (R4) but no binary relationships, where would you record the attributes of *semester* (spring/fall) and *year* (e.g., 2019) for a course?

(a) Teacher entity

(b) TA entity

(c) Course entity

(d) Ternary relationship R4

(e) None of the above are appropriate

The school is also considering adding one or more **new policies**, mentioned below.

8. (10%) If your team is only allowed to implement the **ternary relationship** (R4) but no binary relationships, which policies (basic or new) CANNOT be enforced by the database? (only consider one new policy at a time)

(a) One or more of the three basic policies listed earlier

(b) (new) A teacher cannot use someone they are currently advising as a TA for a course

(c) (new) One teacher cannot use the same TA for two different courses in the same semester

(d) (new) A TA cannot work on more than one course in a given semester

(e) More than one of the above (a – d) cannot be implemented

9. (10%) If your team can only implement **binary relationships** (R1, R2, R3) but no ternary relationships, which policies (basic or new) CANNOT be enforced by the database? (only consider one new policy at a time)

(a) One or more of the three basic policies listed earlier

(b) (new) A teacher cannot use a student they are currently advising, as a TA for a course

(c) (new) One teacher cannot use the same TA for two different courses in the same semester

(d) (new) A TA cannot work on more than one course in a given semester

(e) More than one of the above (a – d) cannot be implemented

10. (10%) Considering the three basic AND the three new policies mentioned so far, what is the **smallest set of relationships** you need to implement ALL of them?

(a) Only binary R1 and ternary R4

(b) Only binary R2 and ternary R4

(c) Only binary R3 and ternary R4

(d) All of the binary relationships R1, R2, R3, and also ternary R4

(e) Some other combination not shown above

The questions on this page are NOT related to any earlier questions

11. (5%) You are building a system to receive photo images throughout the day, and analyze all the images at the end of each day, in one of these a predefined orders: (1) by order received, or (2) by reverse order received, or (3) by size of image in bytes. Which data structure is **least suitable** for any of the three scenarios?

(a) Queue

(b) Stack

(c) Hash Table

(d) Binary Tree

(e) Doubly Linked List

12. (5%) Your systems is going to receive tasks that need to be grouped by type of urgency (now, soon, later, never). *However, we might get other types of urgency.* Which data structure, by itself, might be **most suitable** for storing these tasks so that our system can retrieve all tasks of a given type upon request?

(a) Array

(b) Hash Table

(c) Stack

(d) Queue

(e) Doubly Linked List

13. (5%) Which single data structure is **most suitable** to create an *expert system*, that asks 'yes/no' questions to find the answer to a problem. For example:

```
Do you like making your own applications (y/n)? "Y"
Do you prefer to avoid in a project (y/n)?  "N"
Do you prefer to work alone (y/n)?  "N"
Would you like to have your own startup one day (y/n)?  "Y"
Based on your choices, we recommend the following major: Service Science
```

(a) Hash Table

(b) Stack

(c) Queue

(d) Doubly Linked List

(e) Binary Tree