National University of Singapore

SCHOOL OF COMPUTING

EXAMINATION FOR
Semester 1 AY2012/2013

CS1010E — PROGRAMMING METHODOLOGY

Nov / Dec 2012 Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. This examination paper consists of **FOURTEEN (14)** questions and comprises **TWENTY ONE(21)** printed pages, including this page.
2. Answer **ALL** questions.
3. Answer Section A (Questions 1 to 10) by shading the letter corresponding to the most appropriate answer on the OCR form provided.
4. Answer Section B (Questions 11 to 14) within the space provided in this booklet. You may use pen or pencil to write your answers.
5. This is an **OPEN BOOK** exam. The maximum mark is **80**.
6. Calculators are allowed, but not electronic dictionaries, laptops, tablets, or other computing devices.
7. Do not look at the questions until you are told to do so.
8. Please write your **Student Card number** below.

This portion is for examiner's use only.

Question	**Marks**	**Remarks**
Q11		
Q12		
Q13		
Q14		
Total		

SECTION B (4 Questions : 60 Marks)
Write your answers in the space provided.

11. [**10 marks**] For each of the following functions, write down the purpose of each function **in exactly one sentence** in the corresponding boxes provided in page 13. **DO NOT** give a line-by-line account of what the function does, just state its purpose.

(a)
```
int p(int a, int b) // a and b are positive integers
{
   int c = a;

   while (a%c != 0 || b%c != 0)
      c=c-1;

   return c;
}
```

(b)
```
int q(int a) // a is a positive integer
{
   return (rand()%(a/2 + 1)) * 2;
}
```

(c)
```
int r(int a) // a is a positive integer
{
   int b = 0;

   if (a > 0)
      b = 1 + r(a/10);

   return b;
}
```

(d)
```
int s(int a[], int b, int c) // b is a positive integer
{
   while (b >= 0)
   {
      if (a[b] == c)
         return b;
      b = b - 1;
   }

   return b;
}
```

(e)
```
void t(int a[][N]) // a is N x N square matrix
{
   int i, j, k;

   for (i = 0; i < N/2; i=i+1)
      for (j = i; j < N-1-i; j=j+1)
      {
         k = a[i][j];
         a[i][j] = a[j][N-1-i];
         a[j][N-1-i] = a[N-1-i][N-1-j];
         a[N-1-i][N-1-j] = a[N-1-j][i];
         a[N-1-j][i] = k;
      }

   return;
}
```

Answer:

(a)

(b)

(c)

(d)

(e)

12. Suppose you make an annual deposit of 200 dollars into a bank account which pays 5% interest (or interest rate of 0.05) compounded annually. The following table shows the amount of savings *immediately after* every deposit.

# deposits	Previous balance	Interest	Deposit	Total balance
1	0	0	200	200
2	200	10	200	410
3	410	20.5	200	630.5
4	630.5	31.525	200	862.025
5	862.025	43.10125	200	1105.12625

(a) [**4 marks**] For an annual deposit of d and interest rate r, the final balance immediately after 5 deposits can be expressed as follows:

$$(((((d)(1+r)+d)(1+r))+d)(1+r)+d)(1+r)+d$$

Write a non-recursive `computeBalance` function that takes as argument the annual deposit d, the interest rate r, and the number of deposits n, and returns the total balance immediately after n deposits have been made. For example `computeBalance(200.0,0.05,5)` will return 1105.12625.

Answer:

```
double computeBalance(double d, double r, int n)
{
```

(b) [**2 marks**] For an annual deposit d and interest rate r, express the final balance immediately after n deposits $f_n(d, r)$ as a recurrence relation (or recursive formula).

Answer:

(c) [**4 marks**] Write a recursive `computeBalanceR` function following the recurrence relation defined in question 12b.

Answer:

```
double computeBalanceR(double d, double r, int n)
{
```

13. A *cipher* is a well-defined procedure that encrypts a *plaintext* message into a *ciphertext.* In the following questions, you are to implement different cipher functions that take in the plaintext `str` containing **only lowercase letters**, and replaces `str` with the ciphertext. Assume that the plaintext will not exceed 100 letters in length. You are **not allowed** to use any C string functions.

   (a) [**5 marks**] The *simple cipher* substitutes each letter of the alphabet with another according to a character map. For example, using `"thequickbrownfxjmpsvlazydg"` as the character map, letter `a` is mapped to `t`, `b` is mapped to `h`, etc. As such, the plaintext `"thequickbrownfoxjumpsoverthelazydog"` produces the ciphertext `"vkumlbeohpxzfixyrlnjsxaupvkuwtgdqxc"`.

   Answer:

   ```
   void simple(char *str, char *map)
   {
   ```

   (b) [**5 marks**] The *Caesar's cipher* replaces each letter in the plaintext with another letter that is k positions down the alphabet *with wrap-around.* For example, applying the cipher on the plaintext `"thequickbrownfoxjumpsoverthelazydog"` for $k = 3$ produces the ciphertext `"wkhtxlfneurzqiramxpsvryhuwkhodcbgrj"`.

   Answer:

   ```
   void caesar(char *str, int k)
   {
   ```

(c) [**10 marks**] A *transposition cipher* places letters of the plaintext on a $m \times n$ rectangular grid in column-major order. The ciphertext is read-off from the rectangular grid in row-major order with all empty spaces ignored. For example, applying the transposition cipher on the plaintext `"thequickbrownfoxjumpsoverthelazydog"` over a 8×6 grid,

t	b	j	r	d	
h	r	u	t	o	
e	o	m	h	g	
q	w	p	e		
u	n	s	l		
i	f	o	a		
c	o	v	z		
k	x	e	y		

gives the ciphertext `"tbjrdhrutoeomhgqwpeunslifoacovzkxey"`. You may assume that m and n does not exceed 100 and that the rectangular grid is big enough to store all the letters of the plaintext.

Answer:

```
void transpose(char *str, int m, int n)
{
```

14. An appointment book is used to schedule appointments and keep track of appointment clashes. An appointment consists of a single word description of the appointment, the start time and a duration. Each appointment starts and ends exactly on the hour. The structure definition of `Appointment` is given below.

```
typedef struct
{
   char desc[40] ;    // description not exceeding 40 characters.
   int start;         // start time ranging from hour 0 to hour 23.
   int duration;      // duration in number of hours.
} Appointment;
```

An array `schedule` is declared to store the appointments for one day and is initialized to be "empty", i.e. all data members of structure elements in the array are nullified.

```
Appointment sched[24] = {{"",0,0}};
```

You are advised to study the entire question before attempting to answer each part as all the parts are related. You may write your own auxiliary functions wherever necessary.

(a) [**5 marks**] Write the function `addAppointment` that takes in an existing schedule `sched` and an appointment `appt`. You may assume that appointment `appt` does not clash with existing appointments in the schedule `sched`.

Answer:

```
void addAppointment(Appointment sched[], Appointment appt)
{
```

(b) [**5 marks**] Write the function `isClash` that takes in a schedule `sched` and an appointment `appt`, and determines if there are any existing appointments in `sched` that clashes with `appt`. As an example, suppose there is only one existing appointment for "lecture" at 2pm for a duration of 2 hours in the schedule. Appointments that *clashes* with the above schedule includes

- "Coffee" at 2pm for a duration of 1 hour.
- "Movie" at 12nn for a duration of 3 hours.
- "Nap" at 3pm for a duration of 8 hours.

Answer:

```
#define TRUE 1
#define FALSE 0

int isClash(Appointment sched[], Appointment appt)
{
```

(c) [**5 marks**] Write the function `printSchedule` that takes in a schedule (`sched`) and prints the schedule in chronological order. A sample output is given below.

```
12 1 tutorial
14 2 lecture
```

Each line of output begins with the start time, followed by the duration and the description. The above is in chronological order because the tutorial starts at 12 (noon), which is earlier than the lecture which starts at 14 (or 2pm).

Answer:

```
void printSchedule(Appointment sched[])
{
```

(d) [**5 marks**] Write the `main` function to read appointments from the user and prints out all appointments that do not clash in chronological order. A sample input is given below.

```
14 2 lecture
12 1 tutorial
12 2 lunch
-1
```

Each line consists of an appointment in which the first number denotes the start time, the second number denotes the duration while the rest of the line is the description of the appointment. The last value −1 is a sentinel value. Note that input appointments may possibly clash, so appointments are included into the schedule on a first-come-first-serve basis. In the preceding example, "lecture" and "tutorial" is included, but not "lunch" as it clashes with "tutorial". The program then prints the schedule as follows.

```
12 1 tutorial
14 2 lecture
```

Answer:

```
int main(void)
{
   // Include other declarations if needed.
   Appointment sched[24] = {{"",0,0}};
```

— **END OF PAPER** —