NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

FINAL EXAMINATION FOR
Semester 2 AY2012/13

CS1010E - PROGRAMMING METHODOLOGY

May 2013 Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. This examination paper contains **TWO (2)** parts and comprises **SIXTEEN (16)** printed pages, including this page.
2. Answer **ALL** questions, using **ONLY** the space indicated.
3. The maximum possible mark is 100.
4. This is a **OPEN BOOK** examination.
5. Please write your Matriculation Number below.

MATRICULATION NUMBER: ____________________

EXAMINER'S USE ONLY

Question	Marks	Remarks
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		

Question	Marks	Remarks
16(a)		
16(b)		
16(c)		
16 Total		
17(a)		
17(b)		
17(c)		
17 Total		
18(a)		
18(b)		
18(c)		
18(d)		
18(e)		
18 Total		

PART I: Short Questions (*50 marks*)

In this part, there are 15 questions.
Each question contains a code fragment.
Using the space indicated, write the *output* of the code fragment in question.
Only one or two lines are output for each question.

1. (3 marks)

```
int main() {
    int x = 3, y = 4, z = 7, ans;
    ans = x + y < z ? x < y ? x++ : y++ : x < z ? x++ : z++;
    printf("%d %d %d %d\n", x, y, z, ans);
}
```

2. (3 marks)

```
#define ADD(x,y) x + y
int main() {
    int i;
    i = ADD(5,6) * ADD(5,6);
    printf("%i\n", i);
}
```

3. (3 marks)

```
int main() {
    int x = 2;
    switch(x) {
         case 1:  x++;
         case 2:  x++;
         case 3:  x++; break;
         case 4:  x++;
         default: x++;
    }
    printf("%d\n", x);
}
```

4. (3 marks)

```
int main() {
    int i;
    for(i = 0; i <= 5; i++);
    do  {
        printf("%d ", i);
    } while((i -= 2) > 0);
    printf("\n");
}
```

5. (3 marks)

```
int main() {
    int i, j, temp,  size = 6, arr[] = {7,9,3,4,1,13};
    for(i = 0; i < size; i++)
        for(j = i+1; j < size; j++)
            if(arr[j] > arr[i] ) {
                temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }

    for(i = 0; i < size; i++) printf("%d ", arr[i]);
    printf("\n");
}
```

6. (3 marks)

```
int main() {
    char s[] = {'a','b','A', 'B', '\0'};
    char *p, *q;
    p = &s[2];
    q = &s[3];
    printf("%d\n", (*p - *(p - 2)) - (*q - *(q - 2)));
}
```

7. (3 marks)

```
int main() {
  int i, j;
    srand(1);
    i = rand();
    j = rand();
    while (i < j) j--;
    printf("%d\n", (i < j));
}
```

8. (3 marks)

```
int main() {
    int i, a[3][3] = {{1,2,3}, {4,5,6}, {7,8,9}};
    for(i = 0; i < 3; i++) printf("%d ",  *(*(a+i)+i));
    printf("\n");
}
```

9. (3 marks)

```
int main() {
    char a[4][15]= { "Rain in Spain","Falling Mainly","in the", "plains"}, *p;
    p = &a[0][0];
    printf("%c", *p);
    p = p + 1;
    printf("%c", *p);
    p = &a[2][0];
    printf("%c", *p);
    p = &a[3][4];
    printf("%c\n", *p);
}
```

10. (3 marks)

```
void foo(char *s1, char *s2);
int main() {
    char s1[100], s2[100];
    strcpy(s1, "rain + in + spain"); strcpy(s2, "mainly + in plain");
    foo(s1, s2);
    printf("%s ", s1); printf("%s\n", s2);
}
void foo(char *s1, char *s2) {
    while(*s1 != ’+’) s1++;
    while(*s2 != ’+’) {
        *s1 = *s2;
        s1++;
        s2++;
    }
    *s2 = ’\0’;
}
```

11. (4 marks)

```
void t(int), u(int), v(int);

int main() {
    t(5);
}
void t(int x) {
    printf("%d ", x--);
    v(x--);
    u(x);
}
void u(int x) {
    printf("%d ", x--);
    v(x--);
}
void v(int x) {
    printf("%d ", x--);
}
```

12. (4 marks)

```
int ninetytwo(int);
int main() {
    printf("%d\n", ninetytwo(100));
}
int ninetytwo(int x) {
    if (x > 100) return x - 10;
    else return ninetytwo(ninetytwo(x + 11));
}
```

13. (4 marks)

```
int foo2(int);
int kk = 0;
int main() {
    int i, j, k;
    int kk = 1000;
    printf("%d\n", kk + foo2(foo2(5)));
}
int foo2(int x) {
    int static kk = 10;
    x = x + 2;
    kk = kk + x;
    return x + kk;
}
```

14. (4 marks)

```
struct student {
    char firstname[10], *lastname;
    int matric_no;
};
void foo3(struct student s, char [10], char *, int);
int main() {
    char ln[] = "Smith";
    struct student s = {"David", ln, 12345};
    foo3(s, "Peter", "Lee", 67890);
    printf("%s %s %d\n", s.firstname, s.lastname, s.matric_no);
}
void foo3(struct student s, char name1[10], char *name2, int matric_no) {
    strcpy(s.firstname, name1);
    strcpy(s.lastname, name2);
    s.matric_no = matric_no;
}
```

15. (4 marks)

```
struct mystruct {  int a[5]; };

void f(struct mystruct);
void g(struct mystruct *);
int main() {
    struct mystruct p1, p2;
    p1.a[0] = p2.a[0] = 0;
    f(p1); g(&p2);
    printf("%d %d\n", p1.a[0], p2.a[0]);
}
void f(struct mystruct p) { p.a[0] = 1; }
void g(struct mystruct *p) { p->a[0] = 1; }
```

PART II: Programming Questions (*50 marks*)

16. (15 marks)

The following is a C program which prints N lines of asterisks, in a shape of a triangle:

```
#define N 5
int main() {
    int i, j;
    for (i = 0; i < N; i++) {
        for (j = 0; j <= i; j++) printf("*");
        printf("\n");
    }
}
```

In this example, N = 5, and so the following is printed.

```
*
**
***
****
*****
```

Write a program which prints the following different shapes, where each is drawn assuming N = 7.
Your program must allow for any positive value defining the symbolic constant N.
You may assume that N is an *odd* number.

(a) (3 marks)

```
*******
******
*****
****
***
**
*
```

(b) (5 marks)

```
*
**
***
****
***
**
*
```

(c) (7 marks)

```
   *
  ***
 *****
*******
 *****
  ***
   *
```

17. (15 marks)

(a) (4 marks)

Write a function to sort a given integer array where each element is either 0 or 1.
Writc the function without the user of any *nested* loop.

(b) (4 marks)

Consider sorting an array of floating-point numbers, but where the order is determined only by the *integer portion* of the floating point number. For example, the array comprising the sequence $2.4, 3.5, 2.6, 1.7$ can be sorted into either $1.7, 2.4, 2.6, 3.5$ or into $1.7, 2.6, 2.4, 3.5$. For this example, there are no other possibilities.

We say that two different floating point numbers are *similar* if their integer components are the same. We say that a sorting program is *stable* if the relative positions of similar floating point numbers is maintained. In the example above, a stable sorting algorithm would produce the sequence $1.7, 2.4, 2.6, 3.5$ but not $1.7, 2.6, 2.4, 3.5$.

Consider the following two functions which perform *selection* sort and *bubble* sort respectively.

```
void selection_sort(double x[], int npts) {
    int k, j, m; double hold;
    for (k = npts-1; k > 0; k--) { /*  Move large elements to the right  */
        m = k;
        for (j = 0; j < k; j++) if ((int) x[j] > (int) x[m]) m = j;
        hold = x[m];  x[m] = x[k];  x[k] = hold;
    }
}
void bubble_sort(double x[], int npts) {
    int done, k;
    double hold;
    do {
        done = 1;
        for (k = 0; k <= npts-2; k++)
             if ((int) x[k] > (int) x[k+1]) {
                hold = x[k];  x[k] = x[k+1]; x[k+1] = hold;
                done = 0;
             }
    } while (done == 0);
}
```

Which of the two sorting methods, *selection* sort and *bubble* sort, are stable? Explain briefly.

Solution:

(c) (7 marks)

In this question, we continue to consider the sorting of floating point numbers, and considering all similar floating point numbers to be equal in the sort order. Now, we say that a sorting program is *reverse stable* if the relative positions of floating point numbers is reversed. In the example above, a reverse stable sorting algorithm would produce the sequence 1.7, 2.6, 2.4, 3.5 but not 1.7, 2.4, 2.6, 3.5.

Write a function which implements a sorting method that is reverse stable.
You may assume that for each number n in the array, there is at most one other number which is similar to n.

18. (20 marks)

A *map* comprises a collection of *cities*, each identified by a unique name. Associated with each city is a *location*, represented by a *pair* (x, y) of non-negative integers. (You may think of this pair representing the latitude and longtitude of the city as is used in traditional maps.) Some pairs of cities have a direct flight service betweem them. Finally, the *distance* between two cities is simply the Euclidean distance:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

where (x_1, y_1) and (x_2, y_2) are the two locations in question. For simplicity, we shall henceforth use *integer distance* which is simply the integer portion of this real number expression.

(a) (3 marks)

Design a data structure in order to represent a map. You may need to define an appropriate structure to represent each city, its location, and the cities it has direct flights to/from. You may need an array to store all the cities. Write a function that inputs, from the keyboard, some data that represents a map, and populates your data structure accordingly. The input data is of the following form:

- The first line is an integer stating the number of cities. Say this number is n.
- The next n lines will each contain a string and two positive integers, representing the name and location of each city.
- The next line is an integer stating the number of flight services. Say this number is m.
- The next m lines will each contain a pair of cities.

For example, the following lines of input would describe a map of five cities and six flight services between pairs of these cities.

```
5
singapore 150 10300
mumbai 1914 7294
beijing 4004 11627
tokyo 3596 13974
anchorage 6123 14991
6
singapore mumbai
singapore beijing
beijing tokyo
mumbai tokyo
mumbai beijing
tokyo anchorage
```

(b) (3 marks)

Write a function which, when input with two cities, determines if there is a direct flight service between them. The function returns 0 is there is no direct flight service, and the distance between the two cities otherwise. In the above map, for example, there is no direct service between `singapore` and `tokyo`.

(c) (4 marks)

Write a function which, when input with two cities A and B, determines if there is a *two-step* flight service between them. This service is not direct, but via a third city C. Your function should return 0 if there is no direct or two step service between A and B. If there is a direct service, then it returns the distance between A and B. Otherwise, if there is a city C where there is a direct service between A and C, and a direct service between C and B, the function returns the *sum* of the two inter-city distances involved. In the above map, for example, there is a two-step service between `singapore` and `tokyo`, where the third city is `mumbai`.

(d) (4 marks)

This question is like question (c), except that when there is a *two-step* flight service between A and B, the function chooses the third city C in such a way to return the *shortest* inter-city distance. (Of course, if there is a direct service between A and B, the function returns the distance between A and B.) In the above map, for example, the shortest two-step service between singapore and tokyo is where the third city is beijing.

(e) (6 marks)

Write a function which, when input with two cities A and B, determines if there is a *multi-step* flight service between them. This service may be direct, or via *one or more* other cities. If there is no such service, the function returns 0; otherwise, it returns the sum of all the inter-city distances involved in the service. Note that you need just find one multi-step service, and not necessarily the shortest.

Hint: Write a recursive function `multi(c1, c2)` for this. It returns the desired answer, but using only a selected set of cities. Initially this set contains all the cities. When dealing with original problem `multi(c1, c2)`, determine if some recursive call `multi(c3, c2)`, where there is a direct flight from `c1` to `c3`, can successfully return. It is important that this recursive call `multi(c3, c2)` must *not include* consideration of the city `c1`. Similarly, a recursive call eminating from `multi(c3, c2)` must exclude consideration of `c3`, and so on.

One way to indicate which cities are to be included for consideration is to have an addtional attribute `flag` within each structure representing an individual city.

END OF PAPER