NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

FINAL ASSESSMENT FOR
Semester 2 AY2015/16

CS1010E - PROGRAMMING METHODOLOGY

April 2016 Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. This paper contains **TWO (2)** parts and comprises **TWELVE (12)** 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 assessment.
5. Please write your Student Number below **CLEARLY**.

STUDENT 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 Total		
17(a)		
17(b)		
17(c)		
17 Total		
18(a)		
18(b)		
18(c)		
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*, if any, of the code fragment in question.
Only *one line* of answer is required.
Assume that all appropriate preprocessor directives and symbolic constants have already been defined.

1. (3 marks)

```
int x = 5, y = 8, z = 13, ans;
ans = x + y < z ? ++x < y ? x++ : y++ : x < z ? x++ : z++;
printf("%d %d %d %d", x, y, z, ans);
```

Solution:

2. (3 marks)

```
#define twice(x) x + x

int main(void) {
    int y = 3;
    printf("%d", twice(y) * twice(y));
}
```

Solution:

3. (3 marks)

```
int i, j = 0;
for (i = 0; i < 789; i++) {
    switch (i) {
        case 0:
        case 1: if (i < 1) break; i = 123;
        case 456: j = 666; break;
        default: j = 777;
    }
    if (j == 666) break;
}
printf("%d %d", i, j);
```

Solution:

4. (3 marks)

```
#define N 5

int a[][N] = {
  {31421,1,    1,      1,    93439},
  {1,    54541,1,      32419,1},
  {1,    1,    124349,1,    1},
  {1,    63299,1,      43541,1},
  {48659,1,    1,      1,    56451}
};
int i, j, sum = 0;

for (i = 0; i < N; i++)
    for (j = 0; j < N; j++) {
      if (N - j == i + 1) continue;
      if (i != j) sum += a[i][j];
    }
printf("%d", sum);
```

Solution:

5. (3 marks)

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

Solution:

6. (3 marks)

```
int i, j, a[10]  = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
for (i = 0; i < 10; i++) {
    k = rand() % 10;
    for (j = 0; j < i; j++) if (a[j] == k) {
        i--;
        break;
    }
    if (i == j) a[i] = k;
}
for (i = 0, j = 0; i < 10; i++) j += a[i];
printf("%d ", j);
```

Solution:

7. (3 marks)

```
void foo(int a[]);

int main(void) {
    int i, size = 8, a[] = {1, 13, 6, 7, 9, 8, 4, -1};
    foo(&a[4]);
}

void foo(int b[]) {
    int i = 0;
    while (b[i] >= 0) {
         b[i] = 10 - b[i];
         printf("%d ", b[i]);
         i++;
    }
}
```

Solution:

8. (3 marks)

```
char s[] = "abc";
int i;
for(i = 0; s[i]; i++)
    printf("%c%c%c ", s[i], *(s+i), *(i+s));
```

Solution:

9. (3 marks)

```
int eightyone(int);

int main() {
    printf("%d", eightyone(100));
}

int eightyone(int x) {
    if (x > 100) return x - 10;
    else return eightyone(eightyone(x + 11));
}
```

Solution:

10. (3 marks)

```
void foo(int, int);

int main(void) {
    foo(3, 3);
}

int foo(int n, int m) {
    if (n && m) {
        printf("%d ", n);
        if (n % 2) foo(n - 1, m); else foo(n, m - 1);
    }
}
```

Solution:

11. (4 marks)

```
int foo(int []);

int main(void) {
    int a[] = {4, 9, 1, 8, 7, 6, 5, 2, 0};
    printf("%d ", foo(a));
}

int foo(int b[]) {
    int i;
    if (!b[0]) return 0;
    else return b[0] > (i = foo(b + 1)) ? b[0] : i;
}
```

Solution:

12. (4 marks)

```
int a[] = {9,8,7,6,5}, b = 5, *x, *y;
x = a;
x += 2;
*x = b;
y = x;
y++;
*y = 1;
printf("%d %d %d %d %d", a[0], a[1], a[2], a[3], a[4]);
```

Solution:

13. (4 marks)

```
char *foo(char *);

int main(void) {
  char s[100];
    strcpy(s, "how ?now ");
    strcat(s, "brown? cow");
    printf("%s", foo(s));
}

char *foo(char *s) {
    char *s2;
    while(*s != ’?’) s++;
    s2 = ++s;
    while(*s != ’?’) s++;
    *s = ’\0’;
    return s2;
}
```

Solution:

14. (4 marks)

```
struct {
    int a[3];
    char b[3];
} s[3] ={{{1,2,3}, ’z’}, {{4,5,6}, ’y’}, {{7,8,9}, ’w’}}, *p;
int i;

p = &s[1];
p++;
i = *((p->a) + 2);
printf("%d", i);
```

Solution:

15. (4 marks)

```
struct student {
    char firstname[10], *lastname;
    int matric_no;
};

void foo(struct student, char *, char *, int matric_no);

int main(void) {
    char name[10] = "Wong";
    struct student s = {"David", name, 100001};
    foo(s, "Peter", "Lee", 200002);
    printf("%s %s %d", s.firstname, s.lastname, s.matric_no);
}

void foo(struct student s, char *name1, char *name2, int matric_no) {
    strcpy(s.firstname, name1);
    strcpy(s.lastname, name2);
    s.matric_no = matric_no;
}
```

Solution:

PART II: Programming Questions (*50 marks*)

For all the questions below, you may assume that all mentioned preprocessor directives, function prototypes and symbolic constants have already been appropriately pre-defined.

16. (15 marks)

You are given a money account to invest and you are to compute the value in the account after some number M of months. For each of the M months, there is *multiplier* N which will increase the sum of money. (Think of this as an interest payment.) This for example, if N were 5 for the first month, and 2 for the second month, then the value of the account after these two months would be $1.02 * (1.05 * A_0)$ where A_0 is the original amount in the account.

(a) (6 marks)

Given a starting capital of A_0, determine the profit after M months assuming that the multiplier for each month is a *random number* between 1 and 10. For this purpose, write code for a function `final_amount(a0, m)` below where the first argument `a0` is the original amount in the account and `m` denotes the number of months. The function returns the final amount in the account.

```
double final_amount(int a0, int m) {











}
```

(b) (9 marks)

Now consider the same problem as (a) except that now that it is TWICE as likely that the multiplier N is between 6 and 10 (rather than between 1 and 5).

```
double final_amount_twicelikely_noloss(int a0, int m) {











}
```

17. (20 marks)

Assume a matrix of dimension $N \times M$ where $N > 0$ and $M > 0$ are defined separately. The elements of this matrix are *all* the numbers from 0 through $N * M - 1$, ie. all the elements are distinct. We call such a matrix *standard*. A *move* is defined to the be the new standard matrix obtained by swapping the element containing 0 with one of its *neighbor* elements, residing in the cell to the right or bottom of the 0 element. (Note that the cell to the left and top are not considered neighbors.)

In a standard matrix, we say that its *distance* is the minimum number of moves needed to place the 0 element in the *bottom-right* corner, ie. the last cell.

For example, using N = M = 4, consider the first matrix A below. There is a move, by exchanging A[1][1] with A[2][1], to produce the second matrix. The distance of the first matrix is 4 (and the distance of the second matrix is 3).

3	11	5	13
1	0	12	14
2	7	8	4
15	6	9	10

3	11	5	13
1	7	12	14
2	0	8	4
15	6	9	10

(a) (5 marks)

Write code for a function `get_closer_by_one(a[N][M])` below which when given a standard matrix `a`, makes one move such that the new matrix has a shorter distance than the original.

```
void get_closer_by_one(int a[N][M]) {









}
```

(b) (5 marks)

Write code for a function `move_to_bottom_right(a[N][M])` below which when given a standard matrix `a`, makes *zero or more* moves move such that the final matrix has 0 in the bottom-right corner. In the code, you should not call any function.

```
void move_to_bottom_right(int a[N][M]) {









}
```

(c) (10 marks)

In this question, assume that the standard matrix in question contains 0 in the top left hand corner, ie: 0 is in the first row and column.

Now suppose that a move is associated with a *cost*, and this is simply the number that the 0 element is being swapped with. The cost of a sequence of moves is thus the sum of the cost of each individual move in the sequence.

Write code for a function `min_cost(a[N][M])` when given a standard matrix a, determines the *minimal cost* of a sequence of moves move such that the final matrix has 0 in the bottom-right corner. This means that no other sequence has a smaller cost.

HINT: Consider a recursive solution where the general problem is to compute *mincost(i, j)* which is the minimal cost of of solving the *submatrix* of the original matrix A defined as the cells starting from row index i and column index i. The value of *mincost(i, j)* depends on having computed the values of *mincost(i + 1, j)* and *mincost(i, j + 1)*. For example, suppose N = 4 and M = 4 were the numbers of rows and columns respectively for the original matrix. Then to compute *mincost(0, 0)*, we would have recursive calls to first compute the values of *mincost(1, 0)* and *mincost(0, 1)*, and then, compute the result based on these two sub-results. Pictorially, suppose the original matrix was the first matrix below. Then *mincost(1, 0)* and *mincost(0, 1)* are obtained from the second and third matrices below. (The answer for *mincost(0, 0)* in this example is 42, from traversing the path 0, 1, 2, 7, 8, 4, 10.)

0	11	5	13
1	3	12	14
2	7	8	4
15	6	9	10

0	3	12	14
2	7	8	4
15	6	9	10

0	5	13
3	12	14
7	8	4
6	9	10

```
#define N ...
#define M ...

int min_cost(int a[N][M]) {





}

int recursive_min_cost(int a[N][M], int i, int j) {
















}
```

18. (15 marks)

We wish to build a database of employees, each of which is represented by a *structure*. It should contain the following *attributes*:

- *id*, an identifying number for that employee,
- *salary*
- *age*, and
- *two managers*, identified by the two identifiers of the two managers.

Each employee may have one or two managers, and if employee A is a manager of employees B and C, then it is not the case that B is a manager of A or C, and similarly, it is not the case that C is a manager of A or B. Furthermore, there is only one employee who who has no managers. In what follows, we say that A is a manager of B to mean that A is in the *management chain* of B.

(a) (5 marks)

Define the structure *employee* below in accordance with the description above. Note that in this question, we have assembled all the employees in an array `all_employees`.

```
struct employee {









} all_employees[1000];
```

(b) (5 marks)

Assume that the array `all_employees` has already been populated with data, and that the total number of employees in this array is given by the symbolic constant N.

Write code for a function `is_employee(i)` below which when given a number i, determines if *id* is indeed an identifying number of an employee. You are allowed to write additional (helper) functions.

```
int is_employee(int id) {









}
```

(c) (5 marks)

We define an employee as a *good employee* if his/her salary is higher than one of his/her managers, OR if his/her salary is higher than some other *older* employee. Write code for a function `good_employee(id)` which when given an employee identifier i determines if that employee is good. You are allowed to write additional (helper) functions.

```
int good_employee(int id) {













}
```

END OF PAPER