NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

FINAL EXAMINATION FOR
Semester 2 AY2013/2014

CS1010E - PROGRAMMING METHODOLOGY

April 2014 Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. This examination paper contains **TWO (2)** parts and comprises **FIFTEEN (15)** printed pages, including this page.
2. Answer **ALL** questions, using **ONLY** the space indicated.
3. The maximum possible mark is 100.
4. This is an **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		

Question	Marks	Remarks
11		
12		
13		
14		
15		
16		
17		
18		

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 have already been defined.

1. (3 marks)

```
int i = -1, j;
unsigned int u;
double d = 1.9;
char c = 3;
j = d / i;
u = d / c;
printf("%d %d", j, u);
```

2. (3 marks)

```
int i = -2, j = 1, k = 0, l = 5, m;
m = i++||j++&&k++||l++;
printf("%d %d %d %d %d", i, j, k, l, m);
```

3. (3 marks)

```
int x = 7, y = 0; if (x < 5)
if (y > 6)
if (x < 7)
y++;
else
y--;
printf("%d", y);
```

4. (3 marks)

```
int i, j, k = 0;
for (i = 0; i < 999; i+=2) {
    k++;
    for (j = 1; j < 999; j+=2) k--;
}
printf("%d", k);
```

5. (3 marks)

```
int i = 11, j = 3;
while (i != j)
    if (i > j) i -= j; else j -= i;
printf("%d", i);
```

6. (3 marks)

```
int i, j = 0;
for (i = 0; i < 99; i++) {
    switch (i) {
        case 0: j += 1;
        case 1: j += 1; if (i > 1) break; i = 0;
        case 2: if (!i) j += 1; continue;
        default: j += 1;
    }
    if (i) break;
}
printf("%d %d", i, j);
```

7. (3 marks)

```
int f7(int);

main() {
    printf("%d", f7(f7(17)));
}

int f7(int n) {
    do {
        if ((n-- % 5) == 0) break;
    } while (1);
    return n;
}
```

8. (3 marks)

```
#define M 10
int f8(int, int);
int a[M] = {0,1,2,3,4,5,6,7,8,9};

main() {
    printf("%d", f8(0, M - 1));
}

int f8(int i, int j) {
    int x, y, m;
    if (j > i) {
        m = (i + j)/2;
        return ((x = f8(i, m)) > (y = f8(m+1, j)) ? x : y);
    }
    return (a[i]);
}
```

9. (3 marks)

```
void f9a(), f9b();
int i = 4;
main() {
    f9a();
    f9b();
    printf("%d", i);
}
void f9a() {
    if (i <= 0) { i--; return; }
    f9b();
}
void f9b() {
    static int i = 0;
    if (i <= 0) { i--; return; }
    f9a();
}
```

10. (3 marks)

```
char str[10] = {'A', 'B', '\0', 'D', 'E', 'F', '\0', 'H', 'I', '\0'};
printf("%s", str+4);
```

11. (4 marks)

```
char s1[] = "abcabc";
char s2[] = "abccba";
char s3[] = "aabbcc";
char *c, *c2;
int i, j1 = 1, j2 = 1, j3 = 1;

c = s1; c2 = c+strlen(s1) - 1;
for (i = 0; i < strlen(s1)/2; i++) if (*c == *c2) {c++; c2--;} else {j1 = 0; break;}
c = s2; c2 = c+strlen(s2) - 1;
for (i = 0; i < strlen(s2)/2; i++) if (*c == *c2) {c++; c2--;} else {j2 = 0; break;}
c = s3; c2 = c+strlen(s3) - 1;
for (i = 0; i < strlen(s3)/2; i++) if (*c == *c2) {c++; c2--;} else {j3 = 0; break;}
printf("%d %d %d", j1, j2, j3);
```

12. (4 marks)

```
void swap(int *, int *);

int main() {
      int num1 = 33, num2 = 44, *ptr1 = &num1, *ptr2 = &num2;
      swap(ptr2, ptr1);
      printf("%d %d", num1, num2);
}

void swap(int *ptr1, int *ptr2) {
      int *temp = ptr1;
      ptr1 = ptr2;
      ptr2 = temp;
}
```

13. (4 marks)

```
int a[] = {5, 6, 7, 8};
int *p[] = {a, a+1, a+2, a+3};
int **ptr = p;
ptr++;
printf("%d%d%d ", ptr-p, *ptr-a, **ptr); *ptr++;
printf("%d%d%d ", ptr-p, *ptr-a, **ptr); *++ptr;
printf("%d%d%d ", ptr-p, *ptr-a, **ptr);
```

14. (4 marks)

```
struct time {
    int *day, *month, *year;
};
struct time t1, *times;

main() {
    int d = 25, m = 4, y = 2014;
    t1.day = &d;
    t1.month = &m;
    t1.year = &y;
    printf("%d %d %d ", *t1.day, *t1.month, *t1.year);
    times = &t1;
    *(times->day) = 10;
    printf("%d %d %d ", *t1.day, *t1.month, *t1.year);
}
```

15. (4 marks)

```
void update(struct pointer, int, int);

struct pointer{
   int *address;
   int content;
};

main() {
    struct pointer p = {NULL, 20};
    int x = 21;
    update(p, &x, 22);
    printf("%d", p.content);
}

void update(struct pointer p, int *new_address, int new_content)  {
    p.address = new_address;
    p.content = new_content;
}
```

PART II: Programming Questions (*50 marks*)

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

16. (15 marks)

Suppose in the main function we have a two-dimensional integer array declared as

```
int board[N][N];
```

where `N` has been predefined to be some positive integer. Now assume that the entire array is already populated with binary (ie. either 0 or 1) integers. We say a board position which contains the integer 1 is a *queen*, and that two positions in the array are *attacking* if they both hold a queen and they are in the same row, column or diagonal. A (whole) board is said to be attacking if it contains *at least* one pair of attacking position

(a) (5 marks)
Write code for a function `attacking_positions` below so that the function returns *true* if the two positions `board[r1][c1]` and `board[r2][c2]` are attacking.

```
int attacking_positions(int board[][N], int r1, int c1, int r2, int c2) {




}
```

(b) (5 marks)
Write code for a function `attack` below so that the function returns *true* if the array `board[N][N]` is attacking.

```
int attack(board[][N]) {




}
```

(c) (5 marks)

In this question, you are supposed to *assign* binary values into `board`, and not assume that it has already been populated.
Assume `N` is 4.
Write code for a function `four_queens` below computes *one* non-attacking board.
You may assume, for simplicity, that the first queen is in the first row, second column (ie. in `board[0][1]`).

Hint: Have three variables `c2`, `c3` and `c4` in order to store the column positions of the (remaining) three queens in rows 2, 3 and 4 respectively. Write a nested loop where the outermost loop iterates over values of `c2`, second outermost loop iterates over values of `c3`, and finally, the innermost loop iterates over values of `c4`. Test for a solution inside the innermost loop. Once you find one, terminate the entire loop entirely.

```
void four_queens(int board[N][N]) {




}
```

17. (15 marks)

Your investment portfolio (IP) has a balance B of dollars. Every quarter (3 month period), this balance is increased by amount determined by a given quarterly interest rate, or ratio, stated as a fraction. For example, if $B = 100$ initially, and if the prevailing ratio were 0.1, then after one quarter, B would equal \$110. After another quarter, assuming the ratio remains unchanged at 0.1, B will increase to \$121. After yet another quarter at the same ratio, B becomes \$133.10.

(a) (5 marks)

Write code for a *non-recursive* function `invest(b, r, q)` below which returns the balance given: an initial balance `b`, the ratio `r` (which is never changed), and `q` is time period in number of quarters.

```
double invest(double b, double r, int q) {




}
```

(b) (5 marks)

Write code for a function `rec_invest(b, r, q)` below so that the function is a *recursive* version of the one above in part (a).

```
double rec_invest(double b, double r, int q) {




}
```

(c) (5 marks)

Now suppose that the ratio is not fixed, but varies *quarter by quarter*. For example, if the balance is initially 100, and there are two quarters, and the ratio for the first quarter is 0.1, and the ratio for the second quarter is 0.2, then the value returned by your function should be 132.

Write code for a function `invest_ratios` below (recursively, or not) so that it takes an *array* `ratio` as an argument (as opposed to a single integer as before). The i^{th} value in this array (the number values of is the value of quarters) shall define the `ratio` applicable in the i^{th} quarter.

```
double invest_ratios(double b, double r[], int q) {




}
```

18. (20 marks)

Define two *structures* below, one for representing an *exam*, and one for a *student*. The structure for an exam should have attributes for *module code*, *modular credits weightage*, *start time*, and finally, *venue*. The values of each of these attributes is simply a positive integer. The structure for a student should have attributes for *matric number*, and a *collection of exams*, representing the exams this student is enrolled for. This student structure should also have an attribute for storing the *grade* the student in question obtained for *each exam* that he/she is enrolled for.

```
struct one_exam {









}

struct one_student {









}
```

Assume that there are two global databases defined as follows:

```
struct  one_exam  exams[NUM_E];
struct  one_student  students[NUM_S];
```

where the symbolic constants `NUM_E` and `NUM_S` have already been defined. Assume further that the two arrays `exams` and `students` have already been populated with *appropriate* data. In particular, assume that the two global variables `num_exams` and `num_students` (whose values are less than `NUM_E` and `NUM_S` respectively) have been assigned to the number of exams and students in the current databases, respectively.

(a) (5 marks)
Write code for a function `is_enrolled` below that determines if a given student `s`, identified by his/her *matric number*, is enrolled in a given exam `e`, identified by its *module code*.

```
int is_enrolled(int s, int e) {




}
```

(b) (5 marks)
Write code for a function `num_enrolled` below that computes, given a paricular exam `e`, its *total enrolment*.

```
int num_enrolled(int e) {




}
```

(c) (5 marks)
Write code for a function `clash` below that determines if a given student `s` has a *clashing* pair of exams. This means that there are two exams for the student such that

* the start times differ only by at most one, OR
* the start times differ by at most two, AND the venues differ by at least one.

```
int clash(int s) {




}
```

(d) (5 marks)
Write code for a function `cap` below that computes the Cumulative Average Point (CAP) of a given student `s`. This is defined to be the *weighted average* (where the weights are defined the module MC's) of all grades obtained by this student. (You may assume this average is simply the sum of all grades obtained divided by the total number of MC's involved.)

```
int cap(int s) {




}
```

END OF PAPER