NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

FINAL ASSESSMENT FOR
Semester 2 AY2014/15

CS1010E - PROGRAMMING METHODOLOGY

April 2015 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 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 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 i = 1;
double d = 2, d2;
char c = '4', c2, c3;
i = d / 3;
d2 = (c - '0') / 2;
c2 = 'a' + d - 0.0001; c3 = 'a' + d + 0.0001;
printf("%d %.1f %c %c\n", i, d2, c2, c3);
```

Solution:

2. (3 marks)

```
#define double(x) x + x
int main () {  printf("%d\n", 23 / double(8)); }
```

Solution:

3. (3 marks)

```
int x = 1, y = -1, z = 9;
x = (++y && x++) || z++ > y ? x - y : x + y;
printf("%d\n", x);
```

Solution:

4. (3 marks)

```
int i, j = 0;
for (i = 0; i < 999; i++) {
  switch (i % 4) {
    case 0: case 1: if (i < 100) break;
    case 2: if (i == 402) j = 77; break;
    default: j = 66;
  }
  if (j > 69) break;
}
printf("%d %d\n", i, j);
```

Solution:

5. (3 marks)

```
int f05(int);
int g1 = 1, g2 = 2;

int main() {
  printf("%d\n", f05(f05(0)));
}
int f05(int n) {
  int i = 3; static int g1 = 4; extern int g2;
  return (n + i++ + g1++ + g2++);
}
```

Solution:

6. (3 marks)

```
#define N 5
int a06[][N] = {
  {1,1,1,1,99},
  {1,1,1,99,1},
  {1,1,99,1,1},
  {1,99,1,1,1},
  {99,1,1,1,1}
};
int main() {
  int i, j, sum = 0;
  for (i = 0; i < N; i++)
    for (j = 0; j < N; j++) {
      if (N - j == i + 1) continue;
      sum += a06[i][j];
    }
  printf("%d\n", sum);
}
```

Solution:

7. (3 marks)

```
void f07(int, int [], int *);
int main() {
  int a = 0, b[5] = {1};
  f07(a, b, &b[1]);
  printf("%d %d %d %d\n", a, b[0], b[1], b[2]);
}
void f07(int a, int b[], int *c) {
  a = 1;  b[1] = 2;  *(c + 1) = 3;
}
```

Solution:

8. (3 marks)

```
int sum(int *, int);
int a8[] = {1,1,1,1,1,1,1,1,1,1,-1}; // 10 copies of 1

int main() {
  int i;
  for (i = 1; a8[i] != -1; i++) a8[i] += sum(a8, i - 1);
  printf("%d\n", a8[i-1]);
}
int sum(int a[], int n) {
  int s;
  for (s = 0; n >= 0; n--) s += a[n];
  return s;
}
```

Solution:

9. (3 marks)

```
int x = 84, y = 7, count = 0;
while (count++ < 9999) {
   if (rand() > 2222) break;
   x = x + 84;
   y = y + 7;
}
printf("%d\n", x / y);
```

Solution:

10. (3 marks)

```
char s1[] = "aaaa"; char s2[] = "bbbb"; char s3[] = "cccc";

int main() {
  char s[100], *t, *u;
  strcpy(s, s1); strcat(s, s2); strcat(s, s3);
  for (t = s; *t != '\0'; ) {
    u = t;
    if (*u == 'b')
        while (*u != '\0') {
          *u = *(u + 1);
          u++;
        }
    else t++;
  }
  printf("%s\n", s);
}
```

Solution:

11. (4 marks)

```
int count = 0;

int main() {
  int i; i = fib(4); printf("%d %d\n", i, count);
}
int fib(int n) {
  count++;
  if (n <= 1) return n;
  return fib(n - 1) + fib(n - 2);
}
```

Solution:

12. (4 marks)

```
#define MAX 100
int count = 0;
int fibarray[MAX];

int main() {
  int i;
  count = 0;
  for (i = 0; i < MAX; i++) fibarray[i] = 0;
  i = fib2(4);
  printf("%d %d\n", i, count);
}
int fib2(int n) { // Assume n < MAX
  count++;
  if (n <= 1) return n;
  if (fibarray[n]) return fibarray[n];
  fibarray[n-1] = fib2(n - 1);
  fibarray[n-2] = fib2(n - 2);
  return fibarray[n-1] + fibarray[n-2];
}
```

Solution:

13. (4 marks)

```
int *hold, x, y, z;
int a[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
int *p[] = {a, a+1, a+2, a+3};
int **ptr = p;
hold = p[2];  p[2] = p[3];  p[3] = hold;
x = **(ptr++); y = **(ptr++); z = **(ptr++);
printf("%d %d %d %d %d\n", **ptr, x, y, z, (ptr + 6) - p);
```

Solution:

14. (4 marks)

```
int p = 1;
struct {
  int next;
  char name[10];
} s[] = {{3, "cow"}, {2, "how"}, {0, "now"}, {-1, "brown"}};

do {
   printf("%s ", s[p].name); p = s[p].next;
} while (p >= 0);
```

Solution:

15. (4 marks)

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

int main() {
  int d = 27, m = 4, y = 2015;
  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\n", *t1.day, *t1.month, *t1.year);
```

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)

An *alpha sequence* is a sequence of alpabetic characters ('a' through 'z') and spaces (' '). A *word* ω of an alpha sequence Σ is a longest sequence of alphebetic characters (not spaces) which appears in Σ. That is, the occurrence of ω in Σ is such that there is not an alphabetic character either to the left or to the right of the occurrence. The *tail* of an alpha sequence is the sequence obtained by removing the first word and following spaces from the sequence (for example, the tail of `" abc def "` is `"def "`).

(a) (5 marks)

Write code for a function `tail` below which, when given a *string* containing an alpha sequence, returns a string which is the tail of the sequence.

```
char *tail(char *s) {




}
```

(b) (5 marks)

Write code for a function `word` below which takes two strings `s` and `w` containing alpha sequences as arguments. The second string `w` contains no spaces. The function is to determine whether the sequence in the first string `s` contains a word that is equal to the string `w`.

```
int word(char *s, char *w) {









}
```

(c) (5 marks)
Write code for a function `repeated` below which, when given a string `s` containing an alpha sequence, determines if it contains any *repeated* word (i.e. contains at least two different occurrences of the same word). In the code, you may call your functions `tail` and `word` from (a) and (b) above, and/or make a recursive call to `repeated`.

```
int repeated(char *s) {




}
```

17. (17 marks)

(a) (5 marks)
In this question, you are only allowed to use at most *one* loop.
Write code for a function `find` below which determines if a given two-dimensional array `a`, which has dimenions of N rows and M columns, contains a particular integer `value`. If so, it returns the row and column position of `a` which contains `value`, in a structure `coordinate`. If not, it returns the row value of -1 and column value of -1, in a structure `coordinate`. (Note that in the function, the number of rows is passed in via the formal parameter `n`.)

```
struct coordinate {
    int row;
    int col;
}
struct coordinate find(int a[][M], int n, int value) {









}
```

(b) (5 marks)

In this question, you are only allowed to use at most *one* loop.
We say that a two dimensional square integer array, say it has N rows and columns, is *diagonal symmetric* if the number in its i^{th} row and j^{th} column is the same as the number in its j^{th} row and i^{th} column, where i and j range from 0 through $N-1$.

Write code for a function `diag_symm` below which determines if a given two-dimensional square array `a` is diagonal symmetric.

```
int diag_symm(int a[N][N]) {




}
```

(c) (7 marks)
Consider a square matrix whose number of rows (which equals its number of columns) is a power of 2. Such a matrix, if its number of rows n is 2 or more, can be naturally partitioned into four submatrices, each with $n/2$ rows, obtained from its top-left, top-right, bottom-left and bottom-right quadrants. For example, the matrix A:

$$\begin{bmatrix} 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 \\ 3 & 3 & 4 & 4 \\ 3 & 3 & 4 & 4 \end{bmatrix} \text{ can be partitioned into } \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} 3 & 3 \\ 3 & 3 \end{bmatrix} \begin{bmatrix} 4 & 4 \\ 4 & 4 \end{bmatrix}$$

Now consider the following (mathematical, and not C) function `f` on a square matrix A whose number of rows n is a power of 2:

$$f(A,n) = \begin{cases} \text{the one and only element in } A & \text{if } n = 0 \\ \\ f(A_{tl}, n-1) + & \text{if } n > 0 \text{ and} \\ 2 * f(A_{tr}, n-1) + & A_{tl}, A_{tr}, A_{bl}, A_{br} \\ 3 * f(A_{bl}, n-1) + & \text{respectively denote the top-left, top-right,} \\ 4 * f(A_{br}, n-1) & \text{bottom-left and bottom right quadrants of } A. \end{cases}$$

For example, the value of $f(A, 4)$, where A is the matrix above, can be computed as $(1+2+3+4)+2*(2+4+6+8)+3*(3+6+9+12)+4*(4+8+12+16) = 300$.

Write code for a recursive function f below which takes as input a *one* dimensional integer array a and a nonnegative number n. The array a stores the elements of a square matrix of n rows. Assume that n is a power of 2, and that the array a is large enough to contain the n^2 elements comprising the matrix. The function should return the value of f as defined above.

[**Hint:** While the matrix elements do lie in contiguous locations in the array a, the elements of the *quadrants* of the matrix do not lie in contiguous locations. For each quadrant, copy the approprite elements from a into a new array so that they *are* arranged contiguously. You may now make a recursive call using this new array.]

```
int f(int a[], int n) {









}
```

18. (18 marks)

Consider the following definition of a C structure to represent an *employee* (where the symbolic constants MAXNAME and MAXSUB have been defined elsewhere).

```
struct employee {
    char name[MAXNAME];
    int salary;
    struct employee *subordinates[MAXSUB];
}
```

The number of subordinate employees, stored in the array subordinates, is indicated by a NULL sentinel, that is, if there are n subordinates of this employee, then the first n entries in the subordinates array will contain pointers to (distinct) structures representing employees, while the next extry will be a NULL pointer. The entire collection of employees will be stored in a global array staff[MAXEMP] (where the symbolic constant MAXEMP has been defined elsewhere), defined as follows:

```
struct employee staff[MAXEMP];
```

For example, the following picture of six rows depicts a staff strength of 5 employees, where there are two managers, John and Mary, employees having at least one subordinate. John is in fact a

"senior" manager, an employee with at least one subordinate who is him/herself a manager (Mary). In the last row, the name is the empty string in order to denote there are no further employees.

`staff[]`:

"John"	12345	&staff[1]	&staff[2]	NULL
"Mary"	10045	&staff[3]	&staff[4]	NULL
"Bob"	10044	NULL	NULL	NULL
"Peter"	10043	NULL	NULL	NULL
"Jane"	10042	NULL	NULL	NULL
""	0	NULL	NULL	NULL

(a) (5 marks)
Write code for a function `find` below which, when given the name of an employee, returns the *index position* in the global array `staff` in which (a pointer to) the structure associated with that employee is stored. In case the employee is not represented in `staff`, then a -1 is returned.

```
int find_employee(char *e) {




}
```

(b) (5 marks)
We say that an employee e is in the *subordinate chain* of another employee e_2 if either e is a subordinate of e_2, or, if e is a subordinate of another employee who is in the subordinate chain of e_2. You may assume that an employee has at most one manager, and that no two employees are in the subordinate chain of each other.

Write code for a function `top_manager` below which, when given the name of an employee e, prints the name of the *most senior manager* of e. This is the employee whose subordinate chain includes e, and who is not subordinate to any other employee. (If e has no manager, indicate so.)

```
int top_manager(char *e) {




}
```

(c) (8 marks)
Write code for a function `senority_salary` below which determines if, for each employee contained in the global array `staff`, his/her salary is *more than* that of any employee in his/her subordinate chain.

```
int senority_salary() {




}
```

END OF PAPER