NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

EXAMINATION FOR
Semester 1 AY2012/2013

CS1010 – PROGRAMMING METHODOLOGY

November 2012 Time allowed: 2 hours

INSTRUCTIONS TO CANDIDATES

1. This examination paper consists of **FIVE (5)** questions and comprises **ELEVEN (11)** printed pages.
2. This is an **OPEN BOOK** examination.
3. Calculators and electronic dictionaries are not allowed.
4. Answer all questions, and write your answers in the ANSWER SHEETS provided.
5. Fill in your Matriculation Number with a pen, clearly on every page of your ANSWER SHEETS.
6. You may use **2B pencil** to write your codes. Pen is preferred for other questions.
7. Note the penalty will be given for codes that are unclear or unnecessarily long.
8. Note that question 5 is a long question so do set aside enough time for it.
9. You must submit only the ANSWER SHEETS and no other document.

Q1. Multiple Choice Questions (MCQs) [12 marks]

Each MCQ has one correct answer and is worth 2 marks. There is no penalty for wrong answer.

Q1.1 Assuming that the necessary header files have been included, what is the output of the following code fragment?

```
char s1[7] = "ace", *s2 = "bdf";
strcat(s1, s2);
printf("%s\n", s1);
```

A. **acebdf**
B. **bdface**
C. **ace**
D. It will give compile-time error.
E. It will give run-time error.

Q1.2 What is the output of the following program?

```
#include <stdio.h>
int func(int [], int);

int main(void) {
   int a[] = {1, 2, 2, 1, 2, 2, 1}, i;
   for (i=4; i<=7; i++)
      printf("%d ", func(a, i));
   printf("\n");
   return 0;
}

int func(int arr[], int size) {
    int i;
    for (i=0; i<size/2; i++)
        if (arr[i] != arr[size-i-1])
            return 0;
    return 1;
}
```

A. **0 1 0 1**
B. **0 1 1 0**
C. **1 0 0 1**
D. **1 0 1 1**
E. **1 1 0 1**

Q1.3 Given the following function **f()**, what is the return value of **f(4)**?

```
int f(int n) {
   if (n == 1)
      return 3;
   else if (n == 2)
      return 8;
   else
      return 2 * (f(n-1) + f(n-2));
}
```

A. 19
B. 20
C. 30
D. 60
E. 80

Q1.4 What is the output of the following program?

```
#include <stdio.h>
int f(int *);

int main(void) {
   int x = 0;

   if ((--x || f(&x)) && (f(&x) || x++)) {
      x = 3;
   }
   printf("%d\n", x);
   return 0;
}

int f(int *x) {
   *x += 1;
   return *x;
}
```

A. **-1**
B. **0**
C. **1**
D. **2**
E. **3**

Q1.5 What is the output of the following code fragment?

```
int i, sum;
i = sum = 0;
do {
   switch (i%4) {
      case 1: sum += i; break;
      case 3: sum -= i; break;
      default: sum *= 2;
   }
   i++;
} while (i <= 5);

printf("%d %d\n", i, sum);
```

A. **4 3**

B. **5 -2**

C. **6 -4**

D. **6 3**

E. None of the above

Q1.6 Which of the following statements are true?

i) Both __**3**__ and **struct** are invalid variable names.

ii) The following statement would give a run-time error:

```
int x = 1/(int)(2.0/3.0);
```

iii) It is possible for a function to change the value of a variable declared in another function.

iv) A char array of size 3 is needed to store the string "**abc**".

v) The compiler ignores both 5 and 3 in the following function header :

```
void f(int arr[5][3], int row, int col)
```

A. Only (i) and (iii)

B. Only (ii) and (iii)

C. Only (ii) and (v)

D. Only (i) and (iv)

E. None of the above

Q2. [Total: 11 marks]

Q2.1 What is the output of the following program? [3 marks]

```
#include <stdio.h>
#define N 9

int main(void) {
    int a, b;
    for (a=1; a<N/2; a++) {
        for (b=a; b<N/2; b++) {
            if (a*a + b*b < N*N/8) {
                printf("(%d,%d) ", a, b);
            }
        }
    }
    printf("\n");

    return 0;
}
```

Q2.2 What is the output of the following program? [4 marks]

```
#include <stdio.h>

int f(int *, int);
int g(int *);

int main(void) {
    int x = 5, y = 5;
    int sum = f(&x, y);
    printf("%d %d %d\n", sum, x, y);
    return 0;
}

int f(int *x, int y) {
    int sum = 0;

    while (*x > 0) {
        sum += g(&y);
        *x -= 1;
    }
    return sum;
}

int g(int *y) {
    *y *= 2;
    return *y;
}
```

Q2.3

```
#include <stdio.h>
void unknown(int [], int, int);
void printArray(int [], int);

int main(void) {
   int list[5] = { 2, 0, 1, 4, 3 }, i;

   for (i = 1; i < 5; i++)
      unknown(list, list[0], list[i]);

   printArray(list, 5);
   return 0;
}

void unknown(int arr[], int a, int b) {
   int temp = arr[b];
   arr[b] = arr[a];
   arr[a] = temp;
}

void printArray(int arr[], int size) {
   int i;

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

(a) Describe the function **unknown()** in one sentence. [1 mark]

(b) What is the output of the program? [3 marks]

Q3. [Total: 10 marks]

(a) A ***descending diagonal matrix*** is a square matrix where the diagonal values are in decreasing consecutive numbers and all the non-diagonal values are 0. Matrices *A* and *B* below are examples of descending diagonal matrices, while matrices *C* and *D* below are not (because the diagonal values are not decreasing consecutively for *C* and one of the non-diagonal values is non-zero for *D*).

$$A = \begin{bmatrix} 8 & 0 & 0 & 0 \\ 0 & 7 & 0 & 0 \\ 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 5 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -2 \end{bmatrix}$$

$$C = \begin{bmatrix} 19 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 \\ 0 & 0 & 16 & 0 \\ 0 & 0 & 0 & 15 \end{bmatrix} \qquad D = \begin{bmatrix} 0 & 0 & 0 \\ 1 & -1 & 0 \\ 0 & 0 & -2 \end{bmatrix}$$

Write a function **isDescDiagonal()** that takes in a square matrix and returns 1 if the matrix is a descending diagonal matrix, or returns 0 otherwise. Use the following function header. You may assume that MAX_SIZE is a preprocessor directive defined with an integer ≥ 2 and it denotes the maximum dimension of the square matrix.

[6 marks]

```
int isDescDiagonal(int matrix[][MAX_SIZE], int size)
```

(b) An ***anti-diagonal matrix*** is a square matrix where all the values are zeroes except those on the anti-diagonal, i.e. those running from the lower left corner to the upper right corner. There is no restriction on what values can appear on the anti-diagonal.

Matrices *E* and *F* below are anti-diagonal matrices while matrix *G* is not because one of the non-anti-diagonal values is non-zero.

$$E = \begin{bmatrix} 0 & 0 & 0 & 9 \\ 0 & 0 & 1 & 0 \\ 0 & 8 & 0 & 0 \\ 5 & 0 & 0 & 0 \end{bmatrix} \qquad F = \begin{bmatrix} 0 & 0 & 3 \\ 0 & -1 & 0 \\ -2 & 0 & 0 \end{bmatrix} \qquad G = \begin{bmatrix} 0 & 0 & 0 & 3 \\ 0 & 0 & 6 & 0 \\ 0 & 3 & 0 & 0 \\ 1 & 0 & 0 & 2 \end{bmatrix}$$

Write a function **isAntiDiagonal()** that takes in a square matrix and returns 1 if the matrix is an anti-diagonal matrix, or returns 0 otherwise. Use the following function header. You may assume that MAX_SIZE is a preprocessor directive defined with an integer ≥ 2 and it denotes the maximum dimension of the square matrix. [4 marks]

```
int isAntiDiagonal(int matrix[][MAX_SIZE], int size)
```

Q4. [Total: 12 marks]

A positive integer can always be expressed as a product of prime numbers. For instance,

$60 = 2 \times 2 \times 3 \times 5 = 2^2 \times 3^1 \times 5^1$
$78 = 2 \times 3 \times 13 = 2^1 \times 3^1 \times 13^1$
$13 = 13 = 13^1$

The following function **countPrimes()** takes a positive integer and counts the number of (possibly duplicate) prime numbers required to form the given integer.

`countPrimes(60)` returns 4 (two 2's, one 3 and one 5)
`countPrimes(78)` returns 3 (one 2, one 3 and one 13)
`countPrimes(13)` returns 1 (one 13)

```
int countPrimes(int number) {
   return countPrimesRec(number, 1, 0);
}
```

This function in turn calls a recursive function **countPrimesRec**() that does the job of counting primes. The partial code for **countPrimesRec**() is given. You are to complete it by filling your code and the pre-conditions in the dashed boxes.

Note that function **getPrime(*i*)** is considered given, which returns the *i*-th smallest prime number. For instance:

`getPrime(1)` returns 2; `getPrime(2)` returns 3; and `getPrime(3)` returns 5

```
// Pre-conditions:
//
//
int countPrimesRec(int number, int index, int count) {
   int prime;

   if (number == 1)
      return count;
   prime = getPrime(index);
   if (number % prime) {
      
   }
   else {
      
   }
}
```

Q5. **[Total: 35 marks]** A startup company decides to provide location-based services. Its customers are a list of stores, and there are at most 100 stores. Each store has a name (at most 12 characters), a location given by (x, y) coordinates (non-negative integers), a radius (of type float) that defines a circle of influence, number of products (a positive integer) sold in the store, and a list of products (at most 20 products in the list).
Each product consists of the name (at most 30 characters), its cost price and retail price (both positive values, and of type float).

You may assume that there is at least one store, and each store has at least one product. You may also assume that store names and product names do not contain space character.

The structure for a store is given as follows:

```
typedef struct {
    char      sname[13];
    int       x, y;
    float     radius;
    int       numProduct;
    product_t products[MAX_PRODUCTS];
} store_t;
```

a) Define the structure **product_t**. [2 marks]

b) Write a function **readStores()** to read from the text file "stores.in" the details of each store and their products into an array of store_t. You may assume the file exists. The format of "stores.in" is shown on the left below, and a sample "stores.in" is shown on the right. Note that the first line of "stores.in" gives the number of stores which is returned by the function. [8 marks]

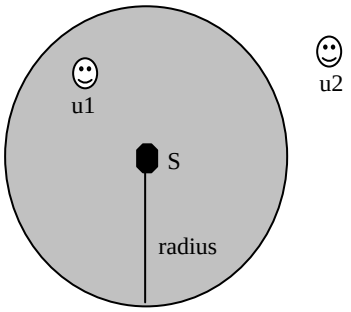
```
numStore
sname1 x1 y1 radius1 numProduct1
pname1 cost1 retail1
pname2 cost2 retail2
pname3 cost3 retail3
...
sname2 x2 y2 radius2 numProduct2
pname1 cost1 retail1
pname2 cost2 retail2
...
```

```
3
ABC_Store 3 4 5.0 3
Pencil_Case 5.00 6.00
Sharpener 1.00 1.02
Pen 2.00 2.70
Cheerful 1 1 3.0 5
Pencil_Case 5.00 5.02
Sharpener 1.00 1.50
Pen 2.00 2.50
Bag 10.00 12.00
Calculator 10.00 10.50
Old_Papa 5 6 10.0 2
Pen 2.00 2.60
Calculator 10.00 12.00
```

```
// Read the number of stores and details of each store
// into array storeList.
// Return the number of stores.
int readStores(store_t storeList[]) {


}
```

c) The startup company would like to send SMS to users who are within the circle of influence of a store. For example, the diagram below shows the circle of influence of a store S. User u1 is within S's circle of influence while u2 is not.



Given a user's current location (x, y) and a particular store, write the function **withinRadius()** that returns 1 if the user's location is within the radius of influence of that store. [2 marks]

```
// Return 1 if user is within store's radius of influence,
// or 0 otherwise.
int withinRadius(int x, int y, store_t store) {



}
```

d) Given a user's current location (x, y), write a function **printNearbyStores()** that prints the names of stores where the user's current location is within their circles of influence. Your function should make use of the **withinRadius()** function in part (c). [3 marks]

```
// Print the names of stores where the user's current
// location is within their circles of influence.
void printNearbyStores(int x, int y, store_t storeList[],
                       int numStore) {



}
```

e) Write a function **labelGoodBuys()** to find good buys. Good buys are products in all the stores whose difference between their retail price and cost price is not more than 5% of the cost price. For these stores, find the corresponding good buys and append "***Must Buy***" to their product names. You may assume that after appending "***Must Buy***" to the product's name, it is still within the 30-character limit. [8 marks]

```
// Find and label the products that are good buys.
// You are to decide on the return type and parameters of
// this function.

______ labelGoodBuys( _____________________ ) {



}
```

f) Given a product name, write a function **findCheapestStore()** to find the store that has the lowest retail price for this product. The function should return the index of this store in the storeList array. If there is more than one store that has the lowest retail price of the product, you may return the index of any one of them.

You may assume that the product concerned is found in at least one of the stores, and the retail price is never more than $1,000.

[12 marks]

```
// Determine which store has the lowest retail price
// for a given product. You are to decide on the
// return type and parameters of this function.

______ findCheapestStore( _____________________ ) {



}
```

=== END OF PAPER ===