NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

PLACEMENT EXAMINATION FOR CS1010/CS1010E
Semester 1 AY2014/2015

July 2014 Time allowed: 2 hours

INSTRUCTIONS TO CANDIDATES

1. This assessment paper contains **SIX (6)** questions and comprises **ELEVEN (11)** printed pages, including this page.
2. This is a **CLOSED BOOK** examination. The maximum possible score is **100 marks**.
3. Calculators are allowed, but not laptops, PDAs, or other electronic devices.
4. Write all your answers in the **ANSWER SHEETS** provided.
5. Submit both the **QUESTION PAPER** and the **ANSWER SHEETS** at the end of the exam.
6. Do **NOT** look at the questions until you are told to do so.

Q1. Tracing **[Total: 15 marks]**

For each of the following program fragments, write down its output. Each question is worth 3 marks. You may assume that necessary header files have been included.

1.1

```
int i, key = 7, index,
    arr[10] = {1, 3, 5, 7, 9, 2, 4, 0, 7, 9};

for (i = 0; i < 10; i++)
    if (arr[i] == key)
        index = i;

printf("%d\n", index);
```

1.2

```
char str[] = {'A', 'B', '\0', 'D', 'E', '\0', 'G'};

printf("%s\n", str);
printf("%s\n", &str[3]);
printf("%s\n", str+6);
```

1.3

```
char str1[] = "Hello", *str2 = "Hello";

if ( str1 != str2 )
    printf("A");
else
    printf("B");

if ( !strcmp(str1, str2) )
    printf("C\n");
else
    printf("D\n");
```

1.4

```
int num = 3746521, val = 0;

while (num > 0) {
    val = val*10 + num%10;
    num /= 10;
}

printf("%d\n", val);
```

1.5

```
int num[10][10], newnum[10*10];

for (i = 0; i < 10; i++)
    for (j = 0; j < 10; j++)
        num[i][j] = i*10 + j;

for (i = 0; i < 10; i++) {
    for (j = 0; j < 10; j++) {
        newnum[100-count-1] = num[i][j];
        count++;
    }
}

printf("%d %d\n", newnum[33], newnum[76]);
```

Q2. Tracing **[Total: 15 marks]**

For each of the following programs, write down its output. Each question is worth 3 marks.

2.1

```
#include <stdio.h>

int foo(int arr[], int size);

int main(void) {

    int a[] = {1, 2, 3, -5, 6};

    printf("%d %d\n", foo(a, 5), foo(a, 2));

    return 0;
}

int foo(int arr[], int size) {

    int i, sum = 0;

    for (i = size-1; i>=0 && i>size-4; i--)
        sum += arr[i];

    return sum;
}
```

2.2

```
#include <stdio.h>
#include <string.h>

typedef struct {
    int a;
    char b[10];
} mystruct;

void weird(int a, char b[]);

int main(void) {

    mystruct mine = {5, "abc"};

    weird(mine.a, mine.b);

    printf("%d %s\n", mine.a, mine.b);

    return 0;
}

void weird(int a, char b[]) {
    a = 10;
    strcpy(b, "xy");
}
```

2.3

```
#include <stdio.h>

int a(int n);

int main(void) {

    printf("%d\n", a(4));

    return 0;
}

int a(int n) {
    if (n == 1 || n == 2)
        return 1;
    else
        return a( a(n-1) ) + a( n-a(n-1) );
}
```

2.4

```
#include <stdio.h>

int f(int *p, int q);
void g(int *m, int n);

int main(void) {

    int x = 10, y = 20, z = f(&x, y);

    printf("%d %d %d\n", x, y, z);

    return 0;
}

int f(int *p, int q) {
    g(p, q);
    return 2 * (*p) + q;
}

void g(int *m, int n) {
    (*m)++;
    n--;
}
```

2.5

```
#include <stdio.h>

int sort(int arr[], int start, int end);

int main(void) {

    int counter, a[] = {7, 5, 3, 1, 6, 9};

    counter = sort(a, 0, 2);
    printf("%d ", counter);

    printf("%d ", a[2]);

    counter = sort(a, 3, 5);
    printf("%d\n", counter);

    return 0;
}

int sort(int arr[], int start, int end) {

    int counter = 0, done, hold, k;

    do {

        done = 1;
        for (k = start; k < end; k++) {
            if (arr[k] > arr[k+1]) {
                hold = arr[k];
                arr[k] = arr[k+1];
                arr[k+1] = hold;
                done = 0;
            }
        }

        counter++;

    } while (!done);

    return counter;
}
```

Q3. Arrays [Total: 15 marks]

Given an array of **distinct** integers and another integer key, we want to find out the total number of pairs (x, y), such that x + y = key, where x and y are two integers in the given array. Note that (x, y) and (y, x) are considered the same pair.

(a) Assuming that the given array **arr** is **unsorted**, write a function **count_pairs()** to count the total number of pairs. For example, given an array {2, 4, 3, 5} and key 7, there are two pairs (2, 5) and (4, 3) whose sum equals 7.

Function prototype is as follows. Parameter **size** is the number of integers in the array **arr**.

```
int count_pairs(int arr[], int size, int key);
```

[7 marks]

(b) Now consider what if the given array **arr** is **sorted** in ascending order of values? Write a function **count_pairs_sorted()** to count the total number of pairs. For example, given an array {0, 1, 3, 4, 6, 7, 8} and key 7, there are three pairs (0, 7), (1, 6) and (3, 4) whose sum equals 7.

Your solution should be efficient (i.e., takes minimum number of comparisons).

Function prototype is as follows. Parameter **size** is the number of integers in the array **arr**.

```
int count_pairs_sorted(int arr[], int size, int key);
```

[8 marks]

Q4. Recursion [8 marks]

Write a recursive function **reverse_print(int n)** that reads *n* integers from user input and prints them out in the reverse order. For example, if user input is 1 2 3 4 5, this function should print out 5 4 3 2 1.

You are **not allowed to** use any loop structures (*for*, *while* or *do-while* loop) in this recursive function and no helper function is allowed.

A skeleton program has been given in the answer sheet. Note that *n* (the number of integers to read) is passed in as a parameter in the function call.

Two sample runs are given below with user input highlighted in bold.

```
How many values to enter? 5
1 2 3 4 5
5 4 3 2 1
```

```
How many values to enter? 2
-1 2
2 -1
```

Q5. Loops [12 marks]

A positive integer can always be expressed as a product of prime numbers (known as its primary factors). For instance,

2 = 2

4 = 2 * 2

10 = 2 * 5

15 = 3 * 5

125 = 5 * 5 * 5

We are interested in such integers whose primary factors are only 2 or only 5. In the above examples, 2, 4 and 125 satisfy this condition while 10 and 15 not.

Given an integer *n*, write a function **`print(int n)`** to print out in ascending order, all the integers in the range [2, *n*] (both inclusive) whose primary factors are only 2 or only 5.

For example, 2, 4, 5, 8, 16, 25 and 32 will be printed out in sequence if *n* is 32.

You are **not allowed to** use array or recursion in this question.

Two sample runs are given below with user input highlighted in bold.

```
Enter n: 32
2 4 5 8 16 25 32
```

```
Enter n: 24
2 4 5 8 16
```

Q6. Structures **[Total: 35 marks]**

It's the end of the semester. We will calculate the Cumulative Average Point (CAP) for a group of 10 students and identify those students with the highest CAP.

Exam results of all the 10 students are stored in a text file "roster.txt", an example of which is shown below where each row is the record of a student, listing in that sequence, *student matric number (a string of alphabet and digits only), the number of modules taken* and for each module, *its module credit and the grade obtained*.

For example, in the "roster.txt" below, student `A0110793H` takes 3 modules of 4 credits each and obtains grades `"A-"`, `"C"` and `"B+"` in these three modules.

```
A0110793H 3 4 A- 4 C  4 B+
A0094679J 5 4 A+ 4 A+ 4 A+ 5 A  4 A
A0091900M 4 4 B- 4 A- 4 C+ 3 B+
A0091809X 6 3 B  2 B- 4 B  4 B+ 4 C+ 4 D+
A0094752C 5 3 A+ 4 B+ 4 A- 2 B  4 B+
A0116137U 4 4 C+ 4 B- 4 B+ 5 A-
A0112875T 5 4 B  4 B+ 4 C  4 B+ 4 B+
A0112844F 5 5 A  4 B  4 A+ 4 A  4 B+
A0112882M 5 2 A+ 4 B+ 4 B+ 4 A- 4 A-
A0111214K 3 4 A+ 4 A  4 A
```

Figure 1. roster.txt

Assuming that a student may take up to 8 modules in a semester, a **`student_t`** structure is defined as follows:

```
typedef struct {
    char matric[10];    // matric number
    int num_mods;       // number of modules taken
    int credit[8];      // credits of modules taken
                        // credit[0] is the credit
                        // of the first module, etc.
    char grade[8][3];   // grades of modules taken
                        // grade[0] is the grade of
                        // the first module, etc.
} student_t;
```

Your program should print out the student who obtains the highest CAP. In the case there is more than one such student, print out their matric numbers in the order they appear in the "roster.txt". A sample run corresponding to the "roster.txt" given above is shown below; there are two students who achieve a highest CAP of 5.00.

```
5.00 A0094679J
5.00 A0111214K
```

As this question is rather complex, let's apply modular design to break the given task into several modules (functions) and implement them one by one. Note that each module is independent of each other such that even if you cannot do a module, you may still assume it is complete and invoke it as necessary when you do the rest.

(a) Write a function **`read_records()`** that reads 10 student records from "roster.txt" and stores them in an array of **`student_t`** variables such that the first array slot stores information of the first student, etc.

Function prototype is as follows.

```
void read_records(student_t stu[]);
```

[8 marks]

(b) Write a function **`get_grade_point()`** that takes a grade (a string) as parameter, returns corresponding grade point according to the following mapping table.

Grade	A+	A	A-	B+	B	B-	C+	C	D+	D	F
Grade Point	5.0	5.0	4.5	4.0	3.5	3.0	2.5	2.0	1.5	1.0	0

Function prototype is as follows.

```
double get_grade_point(char grade[]);
```

Note that marks will be deducted for long-winded code.

(Hint: long chain of comparison statements can be shortened by using array.)

[8 marks]

(c) Write a function **`cal_cap()`** that takes a student record stored in a **`student_t`** variable, calculates and returns the CAP of this student.

A student's CAP is calculated as:

$$CAP = \frac{\sum (moduleCredit * gradePoint)}{\sum moduleCredit}$$

For example, a student takes 4 modules whose module credits are 4, 4, 5 and 2 and obtains grades "A-", "A", "B+" and "B" respectively, then the CAP is calculated as:

(4 * 4.5 + 4 * 5 + 5 * 4 + 2 * 3.5) / (4 + 4 + 5 + 2) = 4.33.

Function prototype is as follows.

```
double cal_cap(student_t stu);
```

[7 marks]

(d) Write a function **`get_highest_cap()`** that takes an array of 10 **`student_t`** variables and returns the highest CAP among all the students.

Function prototype is as follows.

```
double get_highest_cap(student_t stu[]);
```

[6 marks]

(e) Write a function **`print_top_student()`** that takes an array of 10 **`student_t`** variables and the highest CAP among all these students, prints out the CAP and matric number of the student who obtains the highest CAP. In the case there is more than one such student to print out, output them each on a line in the order they appear in the array.

The CAP of a student should be printed out in two decimal places; check the sample run on page 9 for output format.

Function prototype is as follows.

```
void print_top_student(student_t stu[], double max_cap);
```

[6 marks]

=== END OF PAPER ===