NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

PLACEMENT EXAMINATION FOR CG1101/CS1010/CS1010E
Semester 1 AY2012/2013

PAPER 2

17 July 2012 Time allowed: 2 hours

INSTRUCTIONS TO CANDIDATES

1. This examination paper consists of **FIVE (5)** questions and comprises **SIXTEEN (16)** printed pages.
2. This is a **closed book** examination.
3. Answer all questions in this question paper.
4. You may use **2B pencil** to write your programs.
5. Fill in your NUS Student Number, or if you have not obtained your NUS Student Number, your NUS Application Number below:

6. Fill in your Name below:

FOR INTERNAL USE ONLY

Qn	*Maximum*	*Mark*
1-20	20	
21	8	
22	8	
23	12	
24	40	
25	12	
Total	100	

Important notes:

- Your programs should exhibit signs of good programming practice. Departure from it may result in some penalty to be incurred.
- Use 2B pencil to write your programs to minimise messy correction.
- You should try to write within the boxes provided, which should have sufficient space to contain your code. Marks may be deducted if your code is longer than necessary. The last two blank pages are to be used only if you have no choice.

21. **[8 marks]**
Write a C program to read in a person's height (in metres) and mass (in kilograms), and compute the person's body mass index (BMI) and health risk.

The BMI is computed as the ratio of mass and square of height, that is,

$$\text{BMI} = \text{mass} / (\text{height})^2$$

The health risk is based on the computed BMI and it is categorized into 4 types:

Risk type	BMI	Description
2	27.5 and above	High risk of developing heart disease, high blood pressure, stroke, diabetes
1	23.0 to 27.4	Moderate risk of developing heart disease, high blood pressure, stroke, diabetes
0	18.5 to 22.9	Low risk (healthy range)
-1	18.4 and below	Risk of developing problems such as nutritional deficiency and osteoporosis

The following conditions must be satisfied:

- Your program must use the structure type as given on the next page.
- Your program should use only variable(s) of the given structure type. No variable of any other types should be used.
- The BMI computed should be corrected to the nearest one decimal place. For example, if the height and mass entered by the user are 1.76 and 67 respectively, the computed BMI is 21.629648 and hence it should be corrected to 21.6. If the height and mass are 1.76 and 69.3 respectively, the computed BMI is 22.372160 and it should be corrected to 22.4.

21. (continued) Write your program below.

```
#include <stdio.h>

typedef struct {
   float height;
   float mass;
   float bmi;
   int   riskType;
} record;
```

Total marks for Q21:

22. **[8 marks]**
Write a recursive function **sumDigits(int n)** to compute the sum of the digits in a positive integer *n* plus the number of even digits in *n*. You may assume there are no leading zeros in *n*.

For example, sumDigits(1234) returns 12, while sumDigits(12034) returns 13.

(Note that no mark will be given if your function is non-recursive.)

```
int sumDigits(int n)
```

Total marks for Q22:

23. **[12 marks]**
Write a C program to randomly generate even integers in the range [2, 10] and assign them into a 100-element integer array **arr**. The program then determines and outputs the value that is generated most frequently. If there is more than one such value, you may output any one of them. You should use #define directives for constants.

(For example, if twenty one 2s, eighteen 4s, twenty five 6s, seventeen 8s, and nineteen 10s are generated, then your program should output 6.)

Total marks for Q23:

Total marks for Q23:

24. **[40 marks]**

A C program may contain other functions besides the main() function. When a function *x* calls a function *y* directly, we denote this by (*x* → *y*). For this question, we shall assume that if a program contains *n* functions, the functions are identified by integers 0 through *n* – 1, with function 0 representing the main() function.

A text file **calls.in** contains a list of such direct calls. An example is given below. The first line contains an integer indicating the number of functions (up to 10); the second line contains an integer indicating the number of lines of data that follow it; and the subsequent lines show the direct calls (0 → 2), (3 → 1), (4 → 3), (4 → 1), (0 → 3), (5 → 3), (2 → 2) (0 → 4), and (4 → 5).

```
6
9
0 2
3 1
4 3
4 1
0 3
5 3
2 2
0 4
4 5
```

The text file **calls.in** above is just an example; its content may be changed. You may assume that a C program has at most 10 functions (including the main() function), and data in the text file are all valid.

Please read through all the parts below before you start writing your answer.

All functions should have appropriate return type and parameters. You are NOT allowed to use global variables.

(a) Write a function **readFile(…)** to read data from a text file "**calls.in**" into a 2-dimensional integer array representing the direct function calls. All data in the text file must be read in this function, and after this function is called, the file should not be read again by the program. [8 marks]

(b) Write a function **countRecursive(…)** to count the number of directly recursive functions. A directly recursive function is one that calls itself. Other types of recursion, such as mutual recursion (eg: (x → y) and (y → x)) or indirect recursion (eg: (x → y), (y → z), and (z → x)) are not considered here. [5 marks]

In our example, there is only one directly recursive function, function 2.

(c) Write a function **countDirectCalls(…)** to return the number of functions a particular function *f* directly calls. The value *f* is a parameter in **countDirectCalls(…)**. [5 marks]

(d) Write a function **canReach(…)** to return 1 (true) if a function *f* calls a function *g* either directly or indirectly. It returns 0 (false) otherwise. The values *f* and *g* are parameters in **canReach(…)**. [8 marks]

24. (continued)

(e) Complete the rest of the program, including the #include and #define directives, declaration of local variables, and the **main()** function which performs the following: [14 marks]

- Call the **readFile(…)** function to read data from the text file **calls.in**.
- Call the **countRecursive(…)** function to count the number of directly recursive functions.
- Read an integer called *target* interactively from the user, then call the function **countDirectCalls(…)** to determine the number of functions this target function directly calls.
  You must check that the value of *target* is valid (or keep reading it until it is valid) before your call **countDirectCalls(…)**.
- Read two integers called *source* and *dest* and call the function **canReach(…)** to check whether function *source* calls function *dest*, directly or indirectly.
  There is no need to perform validity check for *source* and *dest*. You may assume that their values are valid and different.
- Print the outputs in the format as shown in the sample outputs below.

A sample output is shown below.

```
(b) Number of recursive functions = 1
(c) Enter function number (0-5): 5
    Function 5 calls 1 function(s) directly.
(d) Enter 'source' and 'destination' functions: 1 2
    Function 1 does not call function 2, directly or indirectly.
```

Another sample output is shown below.

```
(b) Number of recursive functions = 1
(c) Enter function number (0-5): 4
    Function 4 calls 3 function(s) directly.
(d) Enter 'source' and 'destination' functions: 5 1
    Function 5 calls function 1, directly or indirectly.
```

24. (continued) Write your program in the next few pages, part by part.

24. (continued)

24. (continued)

24. (continued)

Total marks for Q24:

25. **[12 marks]**

The functions in question 24 are identified by numbers for simplicity. In reality, functions are given names.

Write a C program to read a line of text from user, to be assigned into a string which contains at most 20 characters (not counting the null character '\0'). You may assume that the input consists only of letters, spaces, and underscores. Your program should include the following 2 functions:

- A function **convert(…)** to replace all the spaces in the string by underscores.
- A function **search(…)** to search for this modified string in an array of strings, and return its position in the array if it is found, or -1 otherwise.

Part of the program has been completed for you, which you are not allowed to change. Fill in the rest of the program.

```
// Fill in #include directives below



#define MAX_LENGTH 20
#define NUM_FUNCTIONS 5

// Fill in function prototypes below





int main(void) {
   char functions[NUM_FUNCTIONS][MAX_LENGTH+1]
             = { "main", "abc", "d_e_f", "pqr__st",
                 "kkkkkmmmmmrrrrrsssss" };
   char fname[MAX_LENGTH+1];

   printf("Enter function name: ");
   fgets(fname, MAX_LENGTH+1, stdin);
   convert(fname);

   printf("fname found at position %d\n",
          search(functions, NUM_FUNCTIONS, fname));

   return 0;
}
```

25. (continued)

Total marks for Q25:

(This page is used only if you need more space to write your answers.
Please indicate the question number.)

(This page is used only if you need more space to write your answers.
Please indicate the question number.)

END OF PAPER