NATIONAL UNIVERSITY OF SINGAPORE
SCHOOL OF COMPUTING

Practical Examination 2 (PE2) for Semester 1, AY2011/2
CS1010 — Programming Methodology

29 October 2011 Time Allowed: 2½ hours

INSTRUCTIONS TO CANDIDATES

1. You are only allowed to read this cover page. Do **not** read the question paper until you are told to do so.
2. This paper consists of **2** exercises on **6** pages. Each exercise constitutes 50%.
3. This is an open-book exam. You may bring in any printed material, but **not** electronic devices, including but not limited to laptop, thumb-drive, electronic dictionary and calculator. You are to switch off/silence your mobile phone and keep it out of view.
4. You will be logged into a special NUSNET account at the beginning of the PE. Do not log off until the end of the PE. Do not use your own NUSNET account.
5. A plab account slip will be issued to you at the beginning of the PE. Bring your matriculation card for identification when you collect it. Please leave your matriculation card on the desk in front of you throughout the PE.
6. You are to write your program in the given **plab account**. The host name is **plab0** (not sunfire!). No activity should be done outside this plab account.
7. You do not need to submit your programs to CodeCrunch. We will retrieve your programs and submit them to CodeCrunch after the PE.
8. Skeleton programs are already residing in your plab account. Please leave the programs in the home directory, and use the same program names as specified in the paper. Do **not** create subdirectory to put your programs there or we will not be able to find them!
9. **Only your source codes (.c programs)** from your plab account will be collected after the PE. Hence, how you name your executable files is not important.
10. Please read carefully and follow all instructions in the question. If in doubt, please ask. Raise your hand and the invigilator will attend to you.
11. Any form of communication with other students, or the use of unauthorised materials is considered cheating and you are liable to disciplinary action.
12. Please save your programs regularly during the PE.
13. When you are told to stop, please do so **immediately**, or you will be penalised.
14. At the end of the PE, please **log out from your plab account** and **shut down the PC**.
15. Please check and keep your belongings (esp. matriculation card) before you leave.
16. We will make arrangement for you to retrieve your programs after we have finished grading. Grading may take a week or more.

ALL THE BEST!

CS1010 AY2011/2 Semester 1
Practical Exam 2 (PE2)

Advice on Writing Your Programs – Please read!

- You are advised to spend some time thinking over the tasks to design your algorithms, instead of writing the programs right away.
- Note that for Exercise 1 part (a), you must use recursion.
- If you write a function, you must have a function prototype, and you must put the function definition after the main function.
- Any variable you use must be declared in some function. You are not allowed to use global variables (variables that are declared outside all the functions).
- You may assume that all inputs are valid, that is, you do not need to perform input validity check.
- Manage your time well! Do not spend excessive time on any exercise.
- The rough marking schemes for both exercises are given below.
- Correctness of program is not solely based on the result on CodeCrunch, or based on your own runs on the plab account. If there are mistakes such as "uninitialized variable" or other errors that are not caught by the compiler, even if you managed to run your program with correct result, your program is not considered correct and marks will be deducted.

Rough Marking Scheme For Exercise 1 [50 marks]

1. Part (a): 10 marks all go to correctness of the code.
2. Part (b): 40 marks
   - Style: 5 marks (writing particulars, proper variable names, indentation, appropriate comments, spacing, etc.)
   - Design: 5 marks (deduction on redundant or duplicate statements/variables, unclear logic, etc.)
   - Correctness: 30 marks (10 test data sets; 3 marks each)
3. Special deduction:
   - Use of global variables: Deduct 10 marks

Rough Marking Scheme For Exercise 2 [50 marks]

1. Style: 5 marks (writing particulars, proper variable names, indentation, appropriate comments, spacing, etc.)
2. Design: 5 marks (deduction on redundant or duplicate statements/variables, unclear logic, etc.)
3. Correctness: 40 marks
4. Special deductions:
   - Modification of given code in skeleton program: Deduct 20 marks

Exercise 1: Fast-food or Health-food? (50 marks)

Part (a): 10 marks

You must use recursion for this part. A non-recursive code will not earn you any mark.

Fabulous Hong wants to find out how many ways he can take *n* meals (*n* > 0), if he is only given the choice of fast-food meal or health-food meal. Being a very health-conscious person, he does not want to take two consecutive fast-food meals.

For example, if he has to take 4 meals, he can have 8 different ways of doing it, which are shown below, where 'F' stands for a fast-food meal and 'H' a health-food meal:

FHFH, FHHF, FHHH, HFHF, HFHH, HHFH, HHHF, HHHH

You are to help Fabulous. Write a program **food.c** that asks for *n* (the number of meals, a positive integer), and calculates how many ways Fabulous can take these *n* meals. You must write a recursive function:

int enumerate(int n)

to compute the number of ways to take the *n* meals.

The sample run below (input in bold) shows an example:

```
Enter total number of meals: 4
Number of combinations = 8
```

Hints:

- For recursion, you need to think of the base cases first. There are 2 base cases here, one is *n* = 1, and the other is *n* = 2. When *n* = 1, the function should return 2 since there are 2 ways (F or H) to take one meal.
- For the general case, you need to find out how many ways to take *n* meals. You can take this approach. If the first meal is a fast-food meal, how many ways are there to take the rest of the meals? On the other hand, if the first meal is a health-food meal, how many ways are there to take the rest of the meals? You then combine the answers.

Skeleton Program:

- The skeleton program **food.c** is given in your plab account.
- Do NOT modify the given main() function. There is no need to.
- In the skeleton program, the function **enumerate()** for part (a) returns a dummy value -100. (This is so that if you are unable to do part (a), you can still work on part (b) without getting a compilation error.) You are to replace it with your own code.

Part (b): 40 marks

This part is <u>*unrelated*</u> *to part (a). Hence, if you are unable to do part (a), you can still attempt this part. You may choose to use iteration or recursion for this part.*

Fabulous Hong's parents are away for a holiday. He is given a *budget* and he only has two choices: fast-food meal at a cost of *ff_cost* per meal, or health-food meal at a cost of *hf_cost* per meal.

Note that *budget*, *ff_cost* and *hf_cost* are all positive integers in dollars.

Fabulous wants to spend as much as is allowed by his budget. If there is more than one plan that makes this possible, then being a health-conscious person, he will choose the plan with more health-food meals. There is no constraint on the number of meals he must take.

You are to help Fabulous plan his meals. After doing part (a), the program **food.c** asks for 3 inputs: *budget*, *ff_cost* and *hf_cost*, and determines how many fast-food meals and health-food meals Fabulous can take while spending as much as possible.

The sample run below (input in bold) shows an example where Fabulous will take 8 fast-food meals and 1 health-food meal, spending a total of $122. No other plan allows him to spend more than that without exceeding his budget. For example, if he takes 3 fast-food meals ($39) and 4 health-food meals ($72), he will spend only $111.

```
Enter total number of meals: 4                    } For part (a)
Number of combinations = 8

Enter budget: $123                                } For part (b)
Enter fast-food cost per meal: $13
Enter health-food cost per meal: $18
Number of fast-food meals = 8
Number of health-food meals = 1
```

The second sample run below (input in bold) shows an example where Fabulous will take 2 fast-food meals and 12 health-food meals, spending a total of $50. There is another plan that allows him to spend a total of $50, by taking 5 fast-food meals and 5 health-food meals. Since he prefers to take more health-food meals, the former is better.

```
Enter total number of meals: 1                    } For part (a)
Number of combinations = 2

Enter budget: $50                                 } For part (b)
Enter fast-food cost per meal: $7
Enter health-food cost per meal: $3
Number of fast-food meals = 2
Number of health-food meals = 12
```

Skeleton Program:

- The skeleton program **food.c** is given in your plab account.
- Do <u>NOT</u> modify the given main() function. There is no need to.
- In the skeleton program, the function **enumerate()** for part (a) returns a dummy value -100. (This is so that if you are unable to do part (a), you can still work on part (b) without getting a compilation error.)

Exercise 2: Sets of Integers (50 marks)

Mathematically, a set is collection of distinct objects. In this exercise, you are given two sets of integers and expected to complete the following tasks:

1. Find the **maximum value** of two sets. For example, the maximum value of {5, 1, 2} and {2, 4, 3} is 5.
2. Find the **union** of two sets. The union of set A and B, denoted by A ∪ B, is a set of all things which are members of either A or B. For instance, the union of {1, 2, 5} and {2, 4, 3} is {1, 2, 5, 4, 3}.
3. Find the **symmetric difference** of two sets. The symmetric difference of set A and B is a set of all things that appear either in A or B, but not both. For example, the difference between {2, 1, 5} and {2, 4, 3} is {1, 4, 5, 3}.

You are given a skeleton file which contains the following functions:

```
// Part A: code to be filled in by you ~ 10%
int main (void);

// read a set ~ completed
int read_set (int set[]);

// Part B: code to be filled in by you ~ 5%
int set_max (int setA[], int sizeA, int setB[], int sizeB);

// Part C: code to be filled in by you ~ 15%
int set_union (int setA[], int sizeA, int setB[], int sizeB);

// Part D: code to be filled in by you ~ 20%
int set_difference (int setA[], int sizeA, int setB[], int sizeB);

// print out a set in ascending order of elements ~ completed
void ordered_print_set (int set[], int size);
```

Two sets (both not empty and contain no more than 10 distinct integers) will be read in via `read_set` function and stored `setA` and `setB` arrays in `main` function. You are to complete Part A through Part D with the details as follows:

1. **Part A**: complete the `main` function to read in a user command (a capital character) and dispatch computation to respective function. A user command would be either 'M' (meaning find the maximum value of `setA` and `setB`), or 'D' (meaning find the difference of `setA` and `setB`), or 'U' (meaning find the union of `setA` and `setB`).
2. **Part B**: complete the `set_max` function which takes two sets and their sizes (actual number of integers contained) as parameters and returns the maximum integer of such two sets.
3. **Part C**: complete the `set_union` function which updates `setA` to be the union of `setA` and `setB`. Suppose `setA` is {1, 3, 5} and `setB` is {2, 5, 3}. After function invocation, `setA` would be updated to {1, 3, 5, 2} (order of values is not relevant). This function returns the updated size of `setA` (4 in this example).

4. **Part D**: complete the `set_difference` function which updates `setA` to be the difference of `setA` and `setB`. Suppose `setA` is {3, 1, 5} and `setB` is {2, 3, 5}. After function invocation, `setA` would be updated to {1, 2} (order of values is not relevant). This function returns the updated size of `setA` (2 in this example).

Restrictions on Coding

1. You are **not** allowed to define any additional array in Part A to Part D (i.e., just work on the given arrays `setA` and `setB`).
2. You are **not** allowed to define any additional functions (i.e., just work on the given functions in Part A to Part D).
3. You are **not** allowed to use string, or any sorting mechanism in your functions.
4. You are **not** allowed to modify any other parts of the given program (i.e., complete Part A to part D only).

Heavy penalty will be imposed for breaking the above restrictions.

Sample runs

Sample run #1

```
Enter the number of elements in set 1: 1
Enter the 1 elements: 9
Enter the number of elements in set 2: 1
Enter the 1 elements: -9
Enter command (M for max value; U for union; D for difference): M
Max value is: 9
```

Sample run #2

```
Enter the number of elements in set 1: 3
Enter the 3 elements: 5 7 9
Enter the number of elements in set 2: 2
Enter the 2 elements: 7 28
Enter command (M for max value; U for union; D for difference): U
The union of two sets is (in ascending order):  5 7 9 28
```

Sample run #3

```
Enter the number of elements in set 1: 2
Enter the 2 elements: 9 7
Enter the number of elements in set 2: 2
Enter the 2 elements: 7 9
Enter command (M for max value; U for union; D for difference): D
The difference of two sets is (in ascending order):  null
```

END OF PAPER