CS1010 Programming Methodology
Week 6: Repetition Statements

It is not hard to learn more. What is hard is to unlearn when you discover yourself wrong. ~ *Martin H. Fischer*

To students:

Three weeks of discussion sessions have passed and so we have removed the usual preamble, since you should already know what is expected of you by now.

Some programs for this session are available on the module website:

http://www.comp.nus.edu.sg/~cs1010/3_ca/discussion.html

Please be reminded that **your Practical Exam 1 (PE1) is this Saturday!** Please look up the CS1010 website, "PE" page for more information.

Have a nice one-week recess, but don't forget to practise, practise, practise!

I. Discussion on Lab #2

Your DL will discuss lab #2 exercises with you. Please get your doubts cleared.

II. Loops

1. A number of students wrote such a function below for their Lab 2 Ex 1 (**collatz.c**). When they run the program on their UNIX account, it looks perfect, but when they submit to CodeCrunch, it fails miserably. The code has a very big mistake. What is it? (So, don't make this mistake again next time!)

```
// Compute the number of iterations ...
// Precond: n > 0
int count_iterations(int n) {
   int count;

   while (n > 1) {
      if (n%2 == 1)
         n = 3*n + 1;
      else
         n /= 2;
      count++;
   }

   return count;
}
```

2. Consider the following program.

```
#include <stdio.h>
int main(void) {
    int n, i, count;

    printf("Enter n: ");
    scanf("%d", &n);
    count = 0;
    i = 1;
    while (i < n) {
        if (!(n%i))
            count++;
        i++;
    }
    printf("count = %d\n", count);
}
```

(a) Study the ‘if’ condition **(!(n%i))** in this program. Sometimes you would see such code in books. However, such code might not be readable for some. How would you change it to something more readable, yet retaining its meaning? (A similar example is in a Week 4 lecture slide!)

(b) Assuming that the user enters 100, what is the final value of **count**?

(c) In general, assuming that the user enters a positive number, describe the purpose of this program.

(d) If the final value of **count** is 1, what can be said about the input value **n**?

(e) Is it possible to remove the **i++;** statement, and put the increment of **i** in one of the existing statements? (However, this may make the program more cryptic and we don’t encourage it, as readability is more important at this point. We bring this up here because you may see such code elsewhere.)

(f) Knowing what this program does, could you change the **while** condition to make the program works a little more efficiently (i.e. it takes fewer iterations to compute the answer)?

3. We discussed the **Greatest Common Divisor (GCD)** problem before. Adam and Brusco each wrote the following functions. Study their functions.

   Adam's code:

```
// Returns the GCD of a and b
// Precond: a>=0, b>=0 and not both = 0
int adam_gcd(int a, int b) {
    int divisor;

    if (a == 0)
        return b;
    else if (b == 0)
        return a;

    divisor = (a < b) ? a : b;

    while (1) {
        if ((a%divisor == 0) && (b%divisor == 0))
            return divisor;
        divisor--;
    }
    return 1;
}
```

   Brusco drew the idea from the Euclid's algorithm and wrote this code:

```
// Returns the GCD of a and b
// Precond: a>=0, b>=0 and not both = 0
int brusco_gcd(int a, int b) {
    int temp, remainder;

    // Swap a with b if a < b
    if (a < b) {
        temp = a; a = b; b = temp;
    }

    while (b != 0) {
        remainder = a % b;
        a = b;
        b = remainder;
    }

    return a;
}
```

   (a) Compare the two programs. Which is better and why? Trace them with a number of test cases.

(b) Log into your UNIX account and copy **q3.c** from the cs1010 account as follows:

```
cp ~cs1010/discussion/prog/week6/q3.c .
```

(You may also download q3.c from the CS1010 website, "CA" → "Discussion" page.) Compile **q3.c** and test it on this pair of data: **987654321** and **987654322**. What do you observe?

(c) Brusco's function can be improved by removing certain lines. How would you do it, and why?

4. In many applications, it is quite common to ask the user whether he/she would want to repeat a task by responding to a y/n (yes/no) question, such as the program below. Here, a variable **resp** of **char** type is used, and the format specifier for character variable is **%c** in the **scanf()** statement. The loop repeats if the user enters '**y**'.

```
char resp;

do {
    printf("Hello again!\n");
    printf("Do you want to continue (y/n)? ");
    scanf("%c", &resp);
} while (resp == 'y');
```

Like what you did for question 3, copy **q4.c** from the cs1010 account as follows:

```
cp ~cs1010/discussion/prog/week6/q4.c .
```

Test out the program and what do you observe? Does it work correctly? Do you know how to make the code work? Discuss your discovery in class. (Hint: There could be a number of solutions, some of which involve the use of certain format specifier. Managing input/output can be quite a complicated thing.)

5. Printing asterisks again

(a) In Week 5 lecture you were asked to write a function **print_asterisks(int)** to print a certain number of asterisks on a line. Fill the missing part of the function given below. Note that this function does not include printing a newline character at the end of the asterisks. Do <u>not</u> modify any other part of this function.

```
// To print one line of              asterisks.
// Note there is no newline at the end.
void print_asterisks(int k) {
    int i;
    for (i = 1; i <=              ; i++)
        printf("*");
}
```

Fill in these parts

(b) Write a program to read a positive integer **n** and print asterisks according to this pattern. There will be **n** lines of output; the first line consists of 1 asterisk, the second line consists of 3 asterisks, the third line 5 asterisks, and so on. Two examples are shown below. Make use of the function **print_asterisks(int)** in part (a) above.

```
*
***
*****
```

n = 3

```
*
***
*****
*******
*********
***********
```

n = 6

(c) Modify the program such that a slightly different pattern is printed, as shown in the two examples below.

```
  *
 ***
*****
```

n = 3

```
     *
    ***
   *****
  *******
 *********
***********
```

n = 6

III. Exploration – For your one-week recess

Don't you find the Hi-Lo game in the lecture too lame? Firstly, you could only guess the number once. But this can be easily solved, as we have learned repetition statements now. But secondly, the secret number is hard-coded in the program!

Certainly, in playing games, we would want different values to be generated each time we play the game. This involves **random number generation**.

Computers cannot generate true random numbers. The most they could do is to generate **pseudo-random numbers**, numbers that are close enough to being random, but good enough for most practical purposes. We shall explore this and use it to improve our Hi-Lo game.

6. The **rand()** and **srand()** functions.

   (a) Try out the following program **q6a.c** as shown below. What can you deduce about the function **rand()**? Run the program several times. What did you observe about the 10 values printed? (Note that to use the **rand()** function, you need to include **<stdlib.h>**. Google to find out more about **rand()**.)

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

int main(void) {
    int i;
    for (i = 1; i <= 10; i++)
        printf("%d\n", rand());

    return 0;
}
```

(b) What could you do to restrict the numbers generated to be within a range, for example, between 101 and 500 inclusive? Refer to **q6b.c** for the answer after you have thought about it.

(c) The 10 numbers generated in (a) and (b) are always the same, due to the same seed being used. You can use the **srand()** function to 'seed' the pseudo-random number generator. A particular seed (an integer) indicates which pre-determined sequence of pseudo-numbers to use, and a subsequent call to **rand()** will pick up the next number from this sequence. Hence, you only need to call the **srand()** function once, before you call the **rand()** function.

Try out **q6c.c** (please study the code) and enter the same seed on several runs. Then try another seed on several runs. What do you observe.

(d) We cannot imagine asking a player to provide a seed to our pseudo-random number generator each time he/she plays a game, so there should be a better way. How do we choose a seed that happens to be different each time we run the program?

A commonly used trick is to use the **time(NULL)** function, which returns an integer which is the number of seconds since 1st of January, 1970. To use the **time(NULL)** function, you need to include **<time.h>.** Try out **q6d.c** (please study the code) and run it several times.

You are encouraged to post your queries on IVLE forum if you have doubt on this question.

7. **The Hi-Lo game**

Now it's time to write a proper Hi-Lo game called **hilo.c**. Write a Hi-Lo game that generates a secret integer in the range [1, 100]. The player is asked to enter his/her guess, and has up to 5 guesses. Your program should then ask the player if he/she wants to play another game.

You should start with a simple version. For instance, write a function **play_a_game(…)** (add any appropriate parameter(s) yourself) and call it once to play one hi-lo game. Once the program is tested and it runs fine, then add in more code to allow user to play another game. We want to see good modular design in all your programs, and we want you to practise incremental coding.

Your DL will check your program in Week 7.

See the sample run below.

```
The secret number is between 1 and 100.
Enter your guess [1]: 50
Your guess is too high!
Enter your guess [2]: 10
Your guess is too low!
Enter your guess [3]: 20
Congratulations! You did it in 3 steps.
Do you want to play again (y/n)? y

The secret number is between 1 and 100.
Enter your guess [1]: 13
Congratulations! You did it in 1 step.
Do you want to play again (y/n)? y

The secret number is between 1 and 100.
Enter your guess [1]: 20
Your guess is too low!
Enter your guess [2]: 90
Your guess is too high!
Enter your guess [3]: 50
Your guess is too high!
Enter your guess [4]: 30
Your guess is too low!
Enter your guess [5]: 35
Too bad. The number is 37. Better luck next time!

Do you want to play again (y/n)? n

Thanks for playing. Bye!
```

Please be reminded of your **Practical Exam 1 (PE1) this Saturday, 21 September 2013!**
Refer to the CS1010 website, "PE" page for more information. All the best!