CS1010 Programming Methodology
Week 6: Pointers and Functions II

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

To students:

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

Please be reminded that our **Practical Exam 1 (PE1) is this Saturday 10am!** Please check IVLE workbin **Practical Exam 1** folder for more information.

Have a nice one-week recess!

I. Revision

1. A number of students wrote such a function below for their Lab 2 Ex 1 (**collatz.c**). The code looks correct but has a very fundamental mistake. What is it?

```
// Compute the number of iterations required to bring
// n to the value 1, by applying the collatz operations:
// 3*n+1 if n is odd, or n/2 if n is even.
// Pre-cond: 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;
}
```

Initialize your variables!

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);

    return 0;
}
```

(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?

(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) Knowing what this program does, could you change the 'while' condition to make the program work a little more efficiently (i.e., take fewer iterations to compute the answer)?

II. Problem Solving with Loops

3. In many applications, it is quite common to ask the user whether he/she wants 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'.

```
#include <stdio.h>

int main(void)
{
    char resp;

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

    return 0;
}
```

Copy **Week6_Q3.c** from the cs1010 account as follows:

```
cp ~cs1010/discussion/Week6_Q3.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.

4. Printing asterisks

(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 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. Pointers and Functions with address parameters

5. **Understanding pointers and addresses.**
   You learned pointers (addresses) in Week 6. Do you really understand how they are used? For each of the following, trace the code. Line numbers are added for ease of reference.

   (a) What is the final value of i?

```
int i = 1;              // Line 1
int *p;                 // Line 2

i += 2;                 // Line 3
p = &i;                 // Line 4
(*p)++;                 // Line 5
*p = *p + i;            // Line 6
```

   (b) What is the output?

```
int i = 1, j = 1;       // Line 1
int *p, *q,             // Line 2

i += 2;                 // Line 3
p = &i;                 // Line 4
*p = *p + 3;            // Line 5
q = &i;                 // Line 6
i = *p + *q;            // Line 7
printf("i = %d\n", i);  // Line 8
q = &j;                 // Line 9
i = *p + *q;            // Line 10
printf("i = %d\n", i);  // Line 11
p = &j;                 // Line 12
i = *p + *q;            // Line 13
printf("i = %d\n", i);  // Line 14
```

IV. Design Issues: Programming Methodology: Cohesion

6. We have discussed the **Greatest Common Divisor (GCD)** problem last week. Brusco, being a very inquisitive and adventurous student (and we do love such a student!), tried another new version as shown below.

```
// This program computes the GCD of two non-negative integers,
// not both zeroes.
#include <stdio.h>

void gcd(int, int); // prototype for GCD function

int main(void)
{
   int num1, num2;

   printf("Enter two non-negative integers, not both zeroes: ");
   scanf("%d %d", &num1, &num2);
   GCD(num1, num2);

   return 0;
}

// This function computes the GCD of a and b
// Pre-cond: a and b are both >= 0, and not both = 0
void gcd(int a, int b)
{
   int r;  // r is the remainder of a/b

   while (b > 0)
   {
      r = a%b;
      a = b;
      b = r;
   }

   printf("The GCD is %d\n", a);
}
```

His argument is that: since the answer (variable *a*) is to be returned to the caller and get printed anyway, why can't he just save the returning part (and hence make the function a void function) and print the answer inside the function instead?

However, this move is considered not good, although it works, as it violates some principle of programming. Do you know why?

V. Random Number Generation

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.

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

   (a) Copy **Week6_Q7a.c** from the cs1010 account and try it out. What can you deduce about the function `rand()`? Run the program several times. What do 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=0; i<10; i++)
        printf("Next random number: %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 **Week6_Q7b.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 been 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.

   Complete **Week6_Q7c.c**. Try giving the same seed on several runs first. Then try different seed on several runs. Observe the outputs.

(d) In practice, 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>.** Modify your **Week6_Q7c.c** to use the `time` function rather than asking for a seed from a user.

8. **The Improved Hi-Lo game**

Now it's time to write a proper Hi-lo game called **hilo.c** 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, add in more code to allow the player to play another game (i.e., incremental coding). We want to see good modular design in all your programs. 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!
```