CS1010E Programming Methodology
Tutorial 4: Repetition Statements

> People learn something every day, and a lot of times it's that what they learned the day before was wrong.
> ~ *Bill Vaughan*

To students:

Three weeks of tutorials 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 the submission deadline for **Problem Set 2** is **19 Feb 2018 6pm!** You just need to submit 10 exercises (inclusive of the compulsory one) before the deadline and can continue working on the rest after that.

I. Manual Tracing

1. Consider the following program.

```
#include <stdio.h>

int main(void) {

  int n, i = 1, count = 0;

  printf("Enter n: ");
  scanf("%d", &n);

  while (i < n) {
    if (!(n%i)) { // part (a)
      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 20, what is the final value of **`count`**?

(c) Assuming that the user always 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)?

2. Manually trace each of the following code fragments and write down its output.

(a)

```
int i, sum;

for (i = 1; i < 1000; i *= 2) {
  sum += i;
}
printf("sum = %d\n", sum);
```

Initialize your variables!

(b)

```
int i, sum = 0;

for (i = 1; i < 1000; i *= 2) {
  sum += i;
}
printf("sum = %d\n", sum);
```

(c)

```
int x, y, count = 0;

for (x = 1; x <= 6; x++) {
  for (y = x + 1; y <= 6; y++) {
    count++;
  }
}
printf("count = %d\n", count);
```

3. **Conversion of loop constructs**

You have learned 3 loop constructs: **for**, **while** and **do-while**. For each of the following parts, a particular loop construct is used. Rewrite the loop construct using the other loop construct (without adding any **if-else** statement).

(a) Turn the following **do-while** loop into **for** loop.

```
int sum = 0, i = -5;

do {
  sum += i;
  i += 5;
} while (i < 100);

printf("sum = %d\n", sum);
```

(b) Turn the following **while** loop into **do-while** loop. Can you describe what the code fragment does?

```
int n;
printf("Enter n: ");
scanf("%d", &n);

while (n < 0) {
  printf("Enter n: ");
  scanf("%d", &n);
}

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

II. Hands-on Session

4. **[Problem Set 2 Exercise #05] Count Positive Integers**

Write a program **count_positive.c** to read 5 integers from user input and count how many of them are positive.

```
Enter 5 integers: 2 5 7 0 3
Count = 4
```

```
Enter 5 integers: 0 -1 0 -3 -2
Count = 0
```

5. Programming with single loops

   (a) Write a function **`int sum_multiples_of_5(int n)`** to sum the first **`n`** positive multiples of 5. For example,

   - if **`n`** is 4, then it computes and returns 5 + 10 + 15 + 20 = 50.
   - if **`n`** is 7, it computes and returns 5 + 10 + 15 + 20 + 25 + 30 + 35 = 140.

Answer:

```
int sum_multiples_of_5(int n) {




}
```

   (b) Write a function **`int sum_n_terms(int n)`** to sum the first **`n`** terms of this series:

   $1 + 3 + 7 + 15 + 31 + \ldots (2^i - 1) + \ldots$

   For example,

   - if **`n`** is 2, it computes and returns 1 + 3 = 4.
   - if **`n`** is 4, it computes and returns 1 + 3 + 7 + 15 = 26..

Answer:

```
int sum_n_terms(int n) {




}
```

(c) Write a function **`int square_sum_digits(int n)`** to compute the square sum of all digits in a positive integer **`n`**:

For example,

- square_sum_digits(23) returns $2^2 + 3^2 = 13$.
- square_sum_digits(803071) returns $8^2 + 0^2 + 3^2 + 0^2 + 7^2 + 1^2 = 123$

Answer:

```
int square_sum_digits(int n) {



}
```

(d) Write a function **`double series(int n)`** to compute the sum of the first **`n`** terms of the following series:

$$\frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \cdots$$

For example,

- series(3) returns 0.87
- series(100) returns 0.79

Answer:

```
double series(int n) {



}
```

6. **[Problem Set 2 Exercise #09] Collatz Problem**

The Collatz problem is named after Lothar Collatz who first proposed it in 1937. It states the following:

Take any natural number *n*. If *n* is even, divide it by 2 to get *n*/2; if *n* is odd, triple it and add 1 to obtain 3**n* + 1 (see mathematical expression below). Repeat the process indefinitely. No matter what number your start with, you will *always* eventually reach 1.

$$f(n) = \begin{cases} \frac{n}{2}, & if\ n\ is\ even \\ 3 * n + 1, & if\ n\ is\ odd \end{cases}$$

You are not required to prove the Collatz conjecture, but to write a program **collatz.c** that reads in a positive integer and determines how many iterations it takes to reach 1.

For example, if *n* is 3, then the answer would be 7 (iterations), as

3 → 10 → 5 → 16 → 8 → 4 → 2 → 1.

You program should contain a function

```
int count_iterations(int n)
```

to count the number of iterations required for the value *n* to reach 1.

```
Enter a natural number: 3
Number of iterations = 7
```

```
Enter a natural number: 1
Number of iterations = 0
```