# WEEK 1 HOMEWORK

### Exercises on Conditional (Decision)

Test with values from the <<Controller>>. Do not have interactive input.

**Exercise (if-else):** Write a program which outputs "PASS" if the input is more than or equal to 50; or outputs "FAIL" otherwise. Use (a) just an ‘if’ (b) an ‘if’ and an ‘else’. In both cases only have one return statement.

**Exercise (if-else):** Write a program which outputs "Odd Number" if the input integer number is odd, or “Even Number” otherwise. Use (a) just an ‘if’ (b) an ‘if’ and an ‘else’. In both cases only have one return statement.

**Exercise (nested-if, switch-case):** Write a program which outputs "ONE", "TWO",... , "NINE", "OTHER" if the input is 1, 2,... , 9 or other, respectively. Use (a) a "nested-if" statement; (b) a "switch-case" statement. (c) table lookup.

### Exercises on Loop (Iteration)

Test with values from the <<Controller>>. Do not have interactive input.

**Exercise (Loop):** Write a program to output the sum of 1, 2, 3, ..., to 100. Also compute and output the average. Use a for loop to calculate the sum.

Modify the program to use a "while-do" loop instead of "for" loop.

Modify the program to use a "do-while" loop.

Modify the program to sum from 111 to 8989, and compute the average. Introduce an int variable called count to count the numbers in the specified range.

Modify the program to sum only the odd numbers from 1 to 100, and compute the average.

Modify the program to sum those numbers from 1 to 100 that is divisible by 7, and compute the average.

Modify the program to find the "sum of the squares" of all the numbers from 1 to 100, i.e. 1\*1 + 2\*2 + 3\*3 + ... + 100\*100.

**END OF WEEK 1 HOMEWORK**

# WEEK 2 HOMEWORK

**Exercise (Loop):** Write a program to compute the product of integers 1 to 10 (i.e., 1×2×3×...×10). Try computing the product from 1 to 11, 1 to 12, 1 to 13 and 1 to 14. Write down the product obtained and explain the results.

**Exercise (Loop):** Write a program to compute the sum of a harmonic series, as shown below, where n=50000. The program shall compute the sum from *left-to-right* as well as from the *right-to-left*. Obtain the difference between these two sums and explain the difference. Which sum is more accurate?

http://www3.ntu.edu.sg/home/ehchua/programming/java/images/ExerciseBasics_HarmonicSum.png

**Exercise (Loop & Condition):** Write a program to compute the value of π, using the following series expansion. You have to decide on the termination criterion used in the computation (such as the number of terms used or the magnitude of an additional term). Is this series suitable for computing π?

http://www3.ntu.edu.sg/home/ehchua/programming/java/images/ExerciseBasics_ComputePI.png

C# maintains the value of π in a double constant called Math.PI. Compare the values obtained, in terms of the ratio between the value computed and the Math.PI, in percents.

Hint: Add to sum if the denominator modulus 4 is 1, and subtract from sum if it is 3.

**Exercise (Loop):** Write a program to display the first 20 Fibonacci numbers F(n), where F(n)=F(n–1)+F(n–2) and F(1)=F(2)=1. Also compute their average.

The first 20 Fibonacci numbers are:

1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765

The average is 885.5

Tribonacci numbers are a sequence of numbers T(n) similar to Fibonacci numbers, except that a number is formed by adding the three previous numbers, i.e., T(n)=T(n-1)+T(n-2)+T(n-3), T(1)=1, T(2)=1 and T(3)=2. Write a program to produce the first twenty Tribonacci numbers.

**END OF WEEK 2 HOMEWORK**

# WEEK 3 HOMEWORK

### Exercises on Nested-Loop

**Exercise (User Input & String Operations):** Write a program, which prompts user for a String, and outputs the *reverse* of the String. The output shall look like:

Enter a String:

**abcdef**

The reverse of String "abcdef" is "fedcba".

**Exercise (User Input & String Operations):** On your phone keypad, the alphabets are mapped to digits as follows: ABC(2), DEF(3), GHI(4), JKL(5), MNO(6), PQRS(7), TUV(8), WXYZ(9).

Write a program, which prompts user for a string (case insensitive), and converts to a sequence of digits. Use a nested-if in this exercise. Modify your program to use an *array* for table look-up later.

**Exercise (Palindrome):** A word that reads the same backward as forward is called a *palindrome*, e.g., "mom", "dad", "racecar", "madam", and "Radar" (case-insensitive). Write a program, that prompts user for a word and outputs ""xxx" is|is not a palindrome".

A phrase that reads the same backward as forward is also called a palindrome, e.g., "Madam, I'm Adam", "A man, a plan, a canal - Panama!" (ignoring punctuation and capitalization). Modify your program to test palindromic phrase.

**Exercises on Array**

**Exercise (Array):** Write a program, which reads in n grades (of int between 0 and 100) and displays the average. Your output shall look like:

Enter the number of students: **3**

Enter the grade for student 1: **55**

Enter the grade for student 2: **108**

Invalid grade, try again...

Enter the grade for student 2: **56**

Enter the grade for student 3: **57**

The average is 56.0

**END OF WEEK 3 HOMEWORK**

# WEEK 4 HOMEWORK

### Exercises on Method

**Exercise (Method):** Write a program, which reads in n grades (between 0 and 100, inclusive) and displays the *average*, *minimum*, *maximum*, and *standard deviation*. Your program shall check for valid input. Your output shall look like:

Enter the number of students: **4**

Enter the grade for student 1: **50**

Enter the grade for student 2: **51**

Enter the grade for student 3: **56**

Enter the grade for student 4: **53**

The average is 52.5

The minimum is 50

The maximum is 56

The standard deviation is 2.29128784747792

Hints: The formula for calculating standard deviation is:

http://www3.ntu.edu.sg/home/ehchua/programming/java/images/ExerciseBasics_GradesAverage.png

**Exercise (Method):**

Write a program accepts an array, and reverses its orders. For example, if the input array is {12, 56, 34, 79, 26}, the reversal is {26, 79, 34, 56, 12}. You MUST NOT use another array in your method (but you need a temporary variable to do the swap). Also write a test class to test this method.

**Exercise (Guess a Number):** Write a program to play a number guessing game. The program shall generate a random number between 0 and 99. The player inputs his/her guess, and the program shall response with "Try higher", "Try lower" or "You got it in n trials" accordingly. For example:

> **NumberGuess**

Key in your guess:

**50**

Try higher

**70**

Try lower

**65**

Try lower

"

You got it in 4 trials!

**Exercise (Guess a Word):** Write a program to guess a word by trying to guess the individual characters. The word to be guessed shall be provided using the command-line argument. Your program shall look like:

**WordGuess testing**

Key in one character or your guess word: **t**

Trail 1: t\_\_t\_\_\_

Key in one character or your guess word: **g**

Trail 2: t\_\_t\_\_g

Key in one character or your guess word: **e**

Trail 3: te\_t\_\_g

Key in one character or your guess word: **testing**

Trail 4: Congratulation!

You got in 4 trials

**END OF WEEK 4 HOMEWORK**

# WEEK 5 HOMEWORK

**Exercise (Day of the Week):** Write a program, which takes a date (in year, month and day), and returns the day of the week.

There is an interesting algorithm for finding the day of week given year, month and day (e.g., 26-9-2010), as follows:

1. Take the last two digit of the year, and add a quarter (divide by 4 and discard the remainder). In our example, 10 + 10/4 = 12
2. Add a value according to the month as follow: Jan: 1, Feb: 4, Mar: 4, Apr: 0, May: 2, Jun: 5, Jul: 0, Aug: 3, Sep: 6, Oct: 1, Nov: 4, Dec: 6. For our example, 12 + 6 (Sep) = 18.
3. Add the day. For our example, 18 + 26 = 44
4. Add a century offset according to century value as follows: 18xx:2, 19xx: 0, 20xx: 6, 21xx: 4. For our example, 44 + 6 (20xx) = 50. For years outside this range, add or subtract 400 to bring the year into this range. This is based on the fact that the calendar repeats every 400 years.
5. For leap year (a leap year is a year that is divisible by 4 and not divisible by 100, or divisible by 400), if month is Jan or Feb, subtract 1. For our example, 2010 is not a leap year.
6. Take modulus 7, and retrieve the day of the week from the array {Sat, Sun, Mon, Tues, Wed, Thurs, Fri}. For our example, 50 % 7 = 1, which is a Sunday.

This above algorithm work for Gregorian dates only. The calendar we used today is known as *Gregorian calendar*, which came into effect in October 15, 1582 in some countries and later in other countries. It replaces the *Julian calendar*. 10 days were removed from the calendar, i.e., October 4, 1582 (Julian) was followed by October 15, 1582 (Gregorian). The only difference between the Gregorian and the Julian calendar is the "leap-year rule". In Julian calendar, every four years is a leap year. In Gregorian calendar, a leap year is a year that is divisible by 4 but not divisible by 100, or it is divisible by 400, i.e., the Gregorian calendar omits century years which are not divisible by 400. Furthermore, Julian calendar considers the first day of the year as march 25th, instead of January 1st.

It is difficult to modify the above algorithm to handle pre-Gregorian dates. A better algorithm is to find the number of days from a known date.

### Exercises on Number Theory

**Exercise (Perfect and Deficient Numbers):** A positive integer is called a *perfect number* if the sum of all its factors (excluding the number itself, i.e., proper divisor) is equal to its value. For example, the number 6 is perfect because its proper divisors are 1, 2, and 3, and 6=1+2+3; but the number 10 is not perfect because its proper divisors are 1, 2, and 5, and 10≠1+2+5.

A positive integer is called a *deficient number* if the sum of all its proper divisors is less than its value. For example, 10 is a deficient number because 1+2+5<10; while 12 is not because 1+2+3+4+6>12.

Write a program that takes a positive integer, and return true if the number is perfect. Similarly, write a method to check for deficient numbers.

Using the methods, write a program that prompts user for an upper bound (a positive integer), and lists all the perfect numbers less than or equal to this upper bound. It shall also list all the numbers that are neither deficient nor perfect. The output shall look like:

Enter the upper bound: **1000**

These numbers are perfect:

6 28 496

[3 perfect numbers found (0.30%)]

These numbers are neither deficient nor perfect:

12 18 20 24 30 36 40 42 48 54 56 60 66 70 72 78 80 ......

[246 numbers found (24.60%)]

**Exercise (Prime):** A positive integer is a *prime* if it is divisible by 1 and itself only. Write a method that takes as input a positive integer and returns true if the number is a prime. Write a program that prompts the user for an upper bound (a positive integer), and lists all the primes less than or equal to it. Also display the percentage of primes (up to 2 decimal places). The output shall look like:

Please enter the upper bound: **10000**

1

2

3

......

......

9967

9973

[1230 primes found (12.30%)]

**Exercise**: Write a method that takes as input a positive integer and return true if the product of all its prime factors (excluding 1 and the number itself) is equal to its value. For example, the method returns true for 30 (30=2×3×5) and false for 20 (20≠2×5). Reuse the method in the previous exercise.

**Exercise**: Write a program that prompts user for an upper bound. The program shall display all the numbers (less than or equal to the upper bound) that meets the above criteria. The output shall look like:

Enter the upper bound: **100**

These numbers are equal to the product of prime factors:

1 6 10 14 15 21 22 26 30 33 34 35 38 39 42 46 51 55 57 58 62 65 66 69 70 74 77 78 82 85 86 87 91 93 94 95

[36 numbers found (36.00%)]