**Assignment Sheet #3**

**Exercise GradesAverage (Array):** Write a program called **GradesAverage**, which prompts user for the number of students, reads it from the keyboard, and saves it in an int variable called numStudents. It then prompts user for the grades of each of the students and saves them in an int array called grades.  Your program shall check that the grade is between 0 and 100. A sample session is as follow:

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

**Exercise Hex2Bin (Array for Table Lookup):** Write a program called **Hex2Bin** to convert a hexadecimal string into its equivalent binary string. The output shall look like:

Enter a Hexadecimal string: **1abc**

The equivalent binary for hexadecimal "1abc" is: 0001 1010 1011 1100

**Hints**: Use an array of 16 binary Strings corresponding to hexadecimal number '0' to 'F' (or 'f'), as follows:

String[] hexBits = {"0000", "0001", "0010", "0011",

"0100", "0101", "0110", "0111",

"1000", "1001", "1010", "1011",

"1100", "1101", "1110", "1111"};

**Exercise SumDigits (Command-line arguments):** Write a program called **SumDigits** to sum up the individual digits of a positive integer, given in the command line. The output shall look like:

**java SumDigits 12345**

The sum of digits = 1 + 2 + 3 + 4 + 5 = 15

**Exercise GradesStatistics (Method):** Write a program called **GradesStatistics**, which reads in *n* grades(of int between 0 and 100, inclusive) and displays the *average*, *minimum*, and *maximum*. Display the floating-point values up to 2 decimal places.  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: 52.50

The minimum is: 50

The maximum is: 56

**Exercise *ReverseArrayTest* (Array and Method):** Write a method called reverse(), which takes an array of int and reverse its contents. For example, the reverse of {1,2,3,4} is {4,3,2,1}. The method's signature is as follows:

public static void reverse(int[] array)

Take note that the array passed into the method can be modified by the method (this is called "pass by reference"). On the other hand, primitives passed into a method cannot be modified. This is because a clone is created and passed into the method instead of the original copy (this is called "pass by value").

Also write a test driver to test this method. This would be using the method from the main method.

Hint: You need to use a temp location (an int) to swap the first element with the last element, and so on.

Note: There should NOT be a comma after the last int (Array 1, 2, 3, 4 should print 4, 3, 2, 1)

**Exercise PerfectNumberTest** A positive integer is called a perfect number if the sum of all its 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. Other perfect numbers are 28, 496, ...

The following algorithm can be used to test for perfect number:

// To test whether int x is a perfect number

int sum = 0;

for (int i = 1; i < x; ++i) {

if (x is divisible by i) {

i is a proper divisor;

add i into the sum;

}

}

if (sum == x)

x is a perfect number

else

x is not a perfect number

**translate the above pseudocode into a Java program called PerfectNumberTest.**

**Exercise GCD** The Euclidean algorithm is as follows:

GCD(a, b) // assume that a >= b

while (b != 0) {

// Change the value of a and b: a ← b, b ← a mod b, and repeat until b is 0

temp ← b

b ← a mod b

a ← temp

}

// after the loop completes, i.e., b is 0, we have GCD(a, 0)

GCD is a

Let us look into the Euclidean algorithm, GCD(a, b) = a, if b is 0. Otherwise, we replace a by b; b by (a mod b), and compute GCD(b, a mod b). Repeat the process until the second term is 0. Try this out on pencil-and-paper to convince yourself that it works.

**Write a program called GCD, based on the above algorithm.**

**Exercise (Primes):** A positive integer is a prime if it is divisible by 1 and itself only. Write a method called isPrime(int posInt) that takes a positive integer and returns true if the number is a prime. Write a program called PrimeList 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 prime (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%)]

**Hints**: To check if a number n is a prime, the simplest way is try dividing n by 2 to √n.

**Exercise FibonacciInt (Overflow) :** Write a program called **FibonacciInt** to list all the Fibonacci numbers, which can be expressed as an int (i.e., 32-bit signed integer in the range of [-2147483648, 2147483647]). The output shall look like:

F(0) = 1

F(1) = 1

F(2) = 2

...

F(45) = 1836311903

F(46) is out of the range of int

Hints: The maximum and minimum values of a 32-bit int are kept in constants Integer.MAX\_VALUE and Integer.MIN\_VALUE, respectively. Try these statements:

System.out.println(Integer.MAX\_VALUE);

System.out.println(Integer.MIN\_VALUE);

System.out.println(Integer.MAX\_VALUE + 1);

Take note that in the third statement, Java Runtime does not flag out an overflow error, but silently wraps the number around. Hence, you cannot use F(n-1) + F(n-2) > Integer.MAX\_VALUE to check for overflow. Instead, overflow occurs for F(n) if (Integer.MAX\_VALUE – F(n-1)) < F(n-2) (i.e., no room for the next Fibonacci number).

**Exercise NumberGuess:** Write a program called **NumberGuess** to play the 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:

**java NumberGuess**

Key in your guess:

**50**

Try higher

**70**

Try lower

**65**

Try lower

**61**

You got it in 4 trials!

**Hints**: Use Math.random() to produce a random number in double between 0.0 and (less than) 1.0. To produce an int between 0 and 99, use:

int secretNumber = (int)(Math.random()\*100);

**Extra Programs if you want a challenge**

**Exercise WordGuess:** Write a program called WordGuess 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:

**java WordGuess testing**

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

Trial 1: t\_\_t\_\_\_

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

Trial 2: t\_\_t\_\_g

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

Trial 3: te\_t\_\_g

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

Congratulation!

You got in 4 trials

Hints:

* Set up a boolean array to indicate the positions of the word that have been guessed correctly.
* Check the length of the input String to determine whether the player enters a single character or a guessed word. If the player enters a single character, check it against the word to be guessed, and update the boolean array that keeping the result so far.
* Try retrieving the word to be guessed from a text file (or a dictionary) randomly.

**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 method called isPerfect(int posInt) that takes a positive integer, and return true if the number is perfect. Similarly, write a method called isDeficient(int posInt) to check for deficient numbers.

Using the methods, write a program called PerfectNumberList 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 DateUtil:** Complete the following methods in a class called DateUtil:

* boolean isLeapYear(int year): returns true if the given year is a leap year. A year is a leap year if it is divisible by 4 but not by 100, or it is divisible by 400.
* boolean isValidDate(int year, int month, int day): returns true if the given year, month and day constitute a given date. Assume that year is between 1 and 9999, month is between 1 (Jan) to 12 (Dec) and day shall be between 1 and 28|29|30|31 depending on the month and whether it is a leap year.
* int getDayOfWeek(int year, int month, int day): returns the day of the week, where 0 for SUN, 1 for MON, ..., 6 for SAT, for the given date. Assume that the date is valid.
* String toString(int year, int month, int day): prints the given date in the format "xxxday d mmm yyyy", e.g., "Tuesday 14 Feb 2012". Assume that the given date is valid.

To find the day of the week (Reference: Wiki "Determination of the day of the week"):

1. Based on the first two digit of the year, get the number from the following "century" table.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **1700-** | **1800-** | **1900-** | **2000-** | **2100-** | **2200-** | **2300-** | **2400-** |
| 4 | 2 | 0 | 6 | 4 | 2 | 0 | 6 |

1. Take note that the entries 4, 2, 0, 6 repeat.
2. Add to the last two digit of the year.
3. Add to "the last two digit of the year divide by 4, truncate the fractional part".
4. Add to the number obtained from the following month table:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Jan** | **Feb** | **Mar** | **Apr** | **May** | **Jun** | **Jul** | **Aug** | **Sep** | **Oct** | **Nov** | **Dec** |
| Non-Leap Year | 0 | 3 | 3 | 6 | 1 | 4 | 6 | 2 | 5 | 0 | 3 | 5 |
| Leap Year | 6 | 2 | same as above | | | | | | | | | |

1. Add to the day.
2. The sum modulus 7 gives the day of the week, where 0 for SUN, 1 for MON, ..., 6 for SAT.

For example: 2012, Feb, 17

(6 + 12 + 12/4 + 2 + 17) % 7 = 5 (Fri)

You can compare the day obtained with the Java's Calendar class as follows:

// Construct a Calendar instance with the given year, month and day

Calendar cal = new GregorianCalendar(year, month - 1, day); // month is 0-based

// Get the day of the week number: 1 (Sunday) to 7 (Saturday)

int dayNumber = cal.get(Calendar.DAY\_OF\_WEEK);

String[] calendarDays = { "Sunday", "Monday", "Tuesday", "Wednesday",

"Thursday", "Friday", "Saturday" };

// Print result

System.out.println("It is " + calendarDays[dayNumber - 1]);

**Exercise (Prime Factors):** Write a method isProductOfPrimeFactors(int posInt) that takes 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). You may need to use the isPrime() method in the previous exercise.

Write a program called PerfectPrimeFactorList 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%)]

**Exercise GradesHistogram (Method):** Write a program called **GradesHistogram**, which reads in *n* grades (as in the previous exercise), and displays the horizontal and vertical histograms. For example:

0 - 9: \*\*\*

10 - 19: \*\*\*

20 - 29:

30 - 39:

40 - 49: \*

50 - 59: \*\*\*\*\*

60 - 69:

70 - 79:

80 - 89: \*

90 -100: \*\*

\*

\*

\* \* \*

\* \* \* \*

\* \* \* \* \* \*

0-9 10-19 20-29 30-39 40-49 50-59 60-69 70-79 80-89 90-100

**Hints**:

* Declare a 10-element int arrays called bins, to maintain the counts for [0,9], [10,19], ..., [90,100]. Take note that the bins's index is mark/10, except mark of 100.
* Write the codes to print the star first. Test it. Then print the labels.
* To print the horizontal histogram, use a nested loop:
* To print the vertical histogram, you need to find the maximum bin count (called maxBinCount) and use a nested loop: