Lab Assignment 5

Nested Loops

CSE110: Programming Language I

No of Tasks			Points to Score
Classwork	**Evaluation**	**Homework**	**Homework** **7*10 = 70**
3	**2**	**7**	**Assessment** **2*10 = 20**

The students must complete the classwork tasks in the lab class to obtain the lab performance marks. They will also be marked based on the assessment tasks. The lab instructors may show/explain a few of the classwork tasks to the students if necessary. Any plagiarism in classwork or homework will lead to the student getting zero in the entire assignment. A random viva may take place.

Classwork

1. Trace the following code, create a tracing table and write the outputs

```
1  public class T1{
2     public static void main(String args[]){
3       int x = 0, y = 0;
4       int sum = 0;
5       while (x < 10){
6         y = x - 3;
7         while (y < 3){
8            sum = (sum % 3) + x - y * 3 ;
9            System.out.println(sum);
10           y = y + 1;
11        }
12        if (x > 5){
13           x++;
14        }else{
15           x += 2;
16        }
17      }
18    }
19 }
```

2. Write a Java program to take a positive integer ***N*** as user input and print the **first *N* prime numbers starting from 2**. Your code should check all the positive integers starting from 2 and determine whether they are prime or not until ***N*** prime numbers are found.

   Sample Input 1:
   5
   Sample Output 1:
   2
   3
   5
   7
   11
   Sample Input 2:

7

Sample Output 2:

2
3
5
7
11
13
17

3. Write a Java program that will ask for a range (a starting number and an ending number) from the user and print all the Armstrong numbers between that range.

[Armstrong Number: An Armstrong number is a number whose sum of digits raised to the power the number of digits equals to that number.

For example, 371 is an Armstrong number because $3^3 + 7^3 + 1^3 = 371$, here the total number of digits in 371 is 3 **]**

Sample Input 1:

Start: 300
End: 500

Sample Output 1:

Armstrong numbers:
370
371
407

Sample Input 2:

Start: 100
End: 200

Sample Output 2:

Armstrong numbers:
153

Evaluation

1. Write a Java code of a program that reads the value of N from the user and calculates the value of y if the expression of y is as follows:

$$y = -(1) - (1 + 2) - (1 + 2 + 3) - \ldots - (1 + 2 + 3 + \ldots + N)$$

Sample Input:
The value of N: 2
Sample Output:
The value of y: -4

Sample Input:
The value of N: 4
Sample Output:
The value of y: -20

2. Write a Java program that will keep taking even positive integer numbers as inputs from the user and print the number of divisors of those numbers until it gets an odd number and then stops.

Sample Input/Output: (The purple numbers are input.)
Enter Number: 44
44 has 6 divisors
Enter Number: 30
30 has 8 divisors
Enter Number: 8
8 has 4 divisors
Enter Number: 4
4 has 3 divisors
Enter Number: 6
6 has 4 divisors
Enter Number: 20
20 has 6 divisors
Enter Number: 24
24 has 8 divisors
Enter Number: 5

Homework

1. Trace the following code, create a tracing table and write the outputs.

```
1  public class T1 {
2    public static void main(String args[]) {
3        int x = 0, i = 0, sum = 0;
4        i = 1;
5        x = 2;
6        sum = 0;
7        while (i < 20){
8           x = x + i;
9           sum = sum + x + 1;
10          System.out.println(sum);
11          if (x > 5){
12            i += 2;
13          }
14          else {
15            i += 3;
16          }
17       }
18       sum = sum + i;
19       System.out.println(sum);
20   }
21 }
```

2. Trace the following code, create a tracing table and write the outputs.

```
1   public class T3
2   {
3     public static void main(String args[])
4     {
5        int x = 0, y = 0;
6        int sum = 0;
7        while (x < 10){
8          y = x - 3;
9          y = 40;
10         while (y > 22){
11           if ((sum > 30) && (sum < 40)){
12              sum = sum + x * 2 ;
13           }
14           else if ((sum > 40) && (sum < 50)){
15              sum = sum + x * 3;
16           }
17           else {
18              sum = sum + 23;
19           }
20           System.out.println(sum);
21           y = y - 10;
22         }
23         x += 2;
24       }
25    }
26  }
```

3. Trace the following code, create a tracing table and write the outputs

```
1  public class T4
2  {
3    public static void main(String args[])
4    {
5       int x = 0, p = 0, sum = 0;
6       p = 1;
7       x = 2;
8       double q;
9       sum = 0;
10      while (p < 12){
11         q =  x + p-(sum+7/3)/3.0%2 ;
12         sum = sum + (++x) + (int)q;
13         System.out.println(sum++);
14         if (x > 5){
15            p += 4/2;
16         }
17         else {
18            p += 3%1;
19         }
20      }
21      sum = sum + p;
22      System.out.println(sum);
23   }
24 }
```

4. Trace the following code, create a tracing table and write the outputs

```
1   public class T5{
3      public static void main(String args[]){
5         int x = 0;
6         int sum = 0;
7         while (x < 6){
8           int y = x +1;
9           while (y < 9){
10            if (x%y==0){
11              break;
12            }
13            sum = (sum % 3) + x - y * 3 ;
14            System.out.println(sum);
15            y = y + 1;
16          }
17          if (x > 5){
18            x++;
19          }else{
20             x += 2;
21          }
22           System.out.println(x%3==0);
23       }
24    }
25  }
```

Points to note: How to handle modulus of negative numbers:

Case 1: If the dividend is negative and the divisor is positive

The modulus is performed at first and then the negation is performed.

For example, if we do -5%4 it does 5%4 at first and then negates the output. Which means, (-5)%4 and -5%4 wield the same result, and it is the same for every other case.

Case 2: If the dividend is positive and divisor is negative

The output becomes:

(dividend - nearest smaller multiple of the divisor).

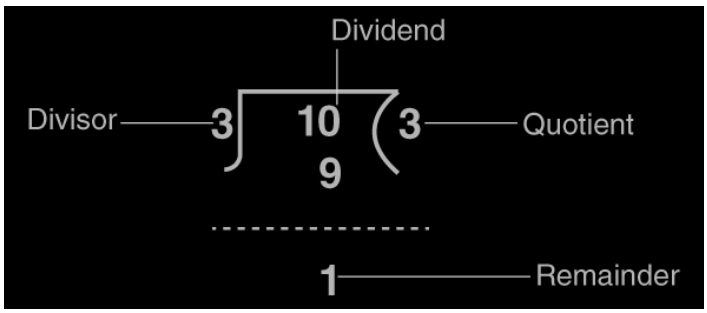
For example, 18%-4 or 18%(-4) is:

18 - 16 (nearest smaller multiple of 4) = 2

Case 3: If both the divisor and the dividend are negative

Case 2 is applied first and a negation occurs because of the minus before the dividend.

For example, -18%-4 or 18%(-4) is:
-(18 - 16) (nearest smaller multiple of 4) = -2



5. Draw a flowchart and write a Java program to take a positive integer ***N*** as user input and print the **first *N* perfect numbers starting from 2**. Your code should check all the positive integers starting from 2 and determine whether they are perfect or not until ***N*** perfect numbers are found.

   Sample Input 1:
   2
   Sample Output 1:
   6
   28

   Sample Input 2:
   3
   Sample Output 2:
   6
   28
   496

6. Write a Java program that will ask the user to specify a range, indicating the starting and ending numbers for generating a times table. The program will generate and display the times table for the specified numbers.

   Sample Input:
   Start: 3
   Stop: 5

   Sample Output:

Times Table of 3:
3 x 1 = 3
3 x 2 = 6
. . . . . .
3 x 10 = 30
Times Table of 4:
4 x 1 = 4
4 x 2 = 8
. . . . . .
4 x 10 = 40
Times Table of 5:
5 x 1 = 5
5 x 2 = 10
. . . . . .
5 x 10 = 50

7. Write a Java program that asks the user for a range, a starting number(inclusive) and an ending number (inclusive). Then, take another input for checking. If the product of all the digits of each number in the range is divisible by the third input, then print the product.

Sample Input 1:
25
30
4

Sample Output 1:
12 16 0

Explanation: The product of all the digits of each number from 25 to 30 are 2 × 5, 2 × 6, 2 × 7, 2 × 8, 2 × 9 and 3 × 0. The final products are 10, 12, 14, 16, 18 and 0 respectively. Out of these numbers only 12, 16 and 0 are divisible by the third input 4 and therefore they are printed.

Sample Input 2:
351
356
9

Sample Output 2:
45 90

Explanation: The product of all the digits of each number from 351 to 356 are 3×5×1, 3×5×2, 3×5×3, 3×5×4, 3×5×5 and 3×5×6. The final products are 15, 30, 45, 60, 75 and 90 respectively. Out of these numbers only 45 and 90 are divisible by 9 and therefore they are printed.