Truth Tables

Your ultimate goal is to reproduce the output of the website trutabgen.com. The input to your program is a string representing a boolean expression and the output is a full truth table.

Example

```
Input:
!A || !B && (C || D)

Output:
[T, T, T, T] F
[T, T, T, F] F
[T, T, F, T] F
[T, T, F, F] F
[T, F, T, T] T
[T, F, T, F] T
[T, F, F, T] T
[T, F, F, F] F
[F, T, T, T] T
[F, T, T, F] T
[F, T, F, T] T
[F, T, F, F] T
[F, F, T, T] T
[F, F, T, F] T
[F, F, F, T] T
[F, F, F, F] T
```

The *input* will consist of the following: up to 5 variables from the set "A B C D E", negation "!", and java symbols for and and or "||", parenthesis and whitespace.

The output will be a truth table with the appropriate number of rows and values for the variables in alphabetical order. (You can assume that no variable are missing, for example "A && D" is not valid input). The final column of the output will be the truth value for the input expressions given the values found in the row.

You will be given code that can evaluate expressions such as "true && !(false || true)". You will provide the remainder of the code for the project, which is divided into several parts.

Part 1 : Decimal to Binary

Write the function below

```
public static String dec2binString(int dec, int minLength) {
   // return a string representation of the binary
         // value of "dec". If the string length is less
         // than minLength, padd with leading zeros
         // Example input: 17, 7. output: 0010001
}
```

Part 2: Binary to Decimal

```
public static int bin2dec(String bin) {
         // input is a binary string
         // output is a base 10 integer
         // example: input 0010001, output 17
  }
```

Part 3: Testing

Test your code by selecting several strings of varying lengths and converting them from binary to decimal and back.

Then convert several integers from decimal to binary and back.

In all cases, the inversion should produce the original input.

Part 4: Truth Table Input

Look at the truth tables generated by trutabgen.com. The rows are in a certain logical order. You will write a function that produces the rows (the final column will come later). Hint: Think about parts 1-2.

```
public static void truthTableRows(int numVars) {
    // print the input rows of a truth table
    // that uses numVars number of boolean variables
}
```

Part 5: Finishing the Project

This part is more open ended, but now you have all the pieces. Your goal is to produce a function as described above. Take as input a *boolean expression with exactly four variables ABCD* and produce a truth table output. You will need to

- generate the input rows in the appropriate order
- for each row,
  - replace the variables with the string “true” or “false”
  - evaluate the resulting expression using a provided function
  - print the row to stdout

Test your code on the following expressions, and make up some of your own

- `A && B && C && D`
- `A && (!B || !C) || D`
- `D && (!D || C || B) && A || C`

Structuring hint it might be nice to have a method that takes input as the input expression (string) and an integer, and output T or F depending on whether the row corresponding to that integer is true or false.

Calling my code

Download and save my parser in the same folder as your code. You do *not* need to edit my file. But in your java file you can say

```
boolean result = BooleanExpressionParser.parseBooleanExpression(expression);
```

to get the value of a string expression.

Java trick The ternary operator is very handy in this lab

```
int y = 10;
String x;

x = (y > 0) ? "T" : "F";
// x is T since 10 > 0

// this replaces the following code
if (y > 0) {
    x = "T";
} else {
    x = "F";
}
```

Test Cases

- `!(A && B) || (A && !B) || (C && !D && !A)`
- `FFFFTTTTTTTTTTTT (from the top down)`

- `!(A || !B || C) || (!A && B) || (D && !D && !A)`
- `FFFFFFFFTTTTFFFF`

- `(A && B || !C) || (!D)`
- `TTTTFTTTFTTTFTTT`

Bonus Extension: 3-SAT

A 3-SAT boolean expression is an expression like

```
(x || y || !z) && (!x || !w || z) && (!y || !a || c)
```

which is a series of size-3 *disjunctions* connected by boolean `and`. A disjunction is a series of variables, possibly negated, connected by boolean `or`. The **3-SAT problem** asks whether it is possible to set each variable to either `true` or `false` so that the entire expression evaluates to `true`. If such a setting exists, the expression is said to be *satisfiable*.

The purpose of this bonus lab is to approximate the fraction of 3-SAT expressions, of a given size, that are satisfiable. The input will be two parameters: the number of variables and the number of disjunctions. You will generate some number of expressions randomly and solve each one for satisfiability, if possible. Then report the fraction of the test cases that were satisfiable.

```
Input: 2 integers, num_vars and num_disjunctions
       3 <= num_vars <= 26
       1 <= num_disjunctions <= 100

Output: A real number 0<r<1.
```