

```

package org.example; /*
    Program to implement Recursive Descent Parser in Java
    Versione originale di Manav Sanghavi modificata da Gennaro Costagliola

    Grammar:
    E -> x + T
    T -> (E)
    T -> x
*/

// Anche se questo programma lo gestisce, la grammatica sopra non
// potrà mai portare ad un backtrack (dove per backtrack si intende visita
// di uno stesso token più di una volta) PERCHE'?

import java.util.*;

class RecDesParser {
    static int ptr;
    static char[] input;

    public static void main(String args[]) {
        // ATTENZIONE: qui non esiste analizzatore lessicale
        // In questo esempio non esistono token ma solo caratteri input.

        // ATTENZIONE: Inserire i simboli input in un array (siano token o altro)
        // prima di parsarli è accettabile solo in questo esempio (di solito
        // i simboli input sono ottenuti on demand dal parser)

        System.out.println("Enter the input string:");
        String s = new Scanner(System.in).nextLine();
        input = s.toCharArray();
        ptr = 0;
        boolean isValid = E();
        if((isValid) & (ptr == input.length)) {
            System.out.println("The input string is valid.");
        } else {
            System.out.println("The input string is invalid.");
        }
    }

    static boolean E() { // produzione E -> x + T

        int fallback = ptr;
        if(isEndOfFile() || input[ptr++] != 'x') {
            ptr = fallback;
            return false;
        }
        if(isEndOfFile() || input[ptr++] != '+') {
            ptr = fallback;
            return false;
        }
        if(T() == false) {
            ptr = fallback;
            return false;
        }
        return true;
    }

    static boolean T() {
        int fallback = ptr;
        if(!isEndOfFile() && input[ptr] == 'x') { // produzione T -> x
            ptr++;
            return true;
        }
        else {
            if(isEndOfFile() || input[ptr++] != '(') { // produzione T -> (E)
                ptr = fallback;
                return false;
            }
            if(E() == false) {
                ptr = fallback;
                return false;
            }
            if(isEndOfFile() || input[ptr++] != ')') {

```

```
        ptr = fallback;
        return false;
    }
    return true;
}
}
static boolean isEndOfFile(){
    return (ptr >= input.length);
}
}
```