TREE CONTROLLED PARSER CD ATTACHMENT

This is a part of bachelor thesis: Parsing Based on Tree-Controled Grammars, Brno University of Technology 2016.

This CD contains:

```
CD
 |- src             - application source files
     |- tcgp.py
     \- ....
 |- tests           - tests
     |- test.sh
     \- ...
 |- docs            - thesis source files
     |- thesis.tex
     \- ...
 |- thesis.pdf      - thesis in pdf format
 |- MIT_licence.txt
```

Tree controlled grammar parser (TCGP) application

Basic usage is for verification, that text string belongs to specified

grammar. This parser expands common LR parser, so it allows parsing of some non-context free grammars.

Source files are placed in `./src/` folder.

For execution use python3.

Usage:

```
python3 tcgp.py [-h] -g GRAMMAR [-p CHOICE [CHOICE ...]] [-
i INPUT] [-o OUTPUT]

Tree controlled grammar parser

optional arguments:
  -h, --help            show this help message and exit
  -g GRAMMAR, --grammar GRAMMAR
        Input grammar file
  -p CHOICE [CHOICE ...], --print CHOICE [CHOICE ...]
        Decide what to print from these CHOICES:
         - tree:       final derivation tree
         - trees:      derivation tree development
         - stack:      continuously print stack of symbols
         - rules:      continuously print applied rules
         - groups:     lr groups generated from rules
         - table:      lr table
         - eff:        empty, first and follow sets
         - automat:    print final state machine
```

```
          - precedence: print precedence table
          - grammar:    print input grammar
          - scanner:    print input scanner automat
          - all:        print all
  -i INPUT, --input INPUT
          Input string file, <stdin> if not present
  -o OUTPUT, --output OUTPUT
          Output file, <stdout> if not present
```

Grammar file

This file specifies tree controlled grammar.

Mandatory part is controlled grammar, which syntax is following. This part must be defined first:

```
grammar = (
  {<id>, <id>, ..., <id>},          # nonterminals
  {<str>, <str>, ..., <str>},       # terminals
  {                                 # rules
    <id> -> [<id>|<str>]* ;
    <id> -> [<id>|<str>]* ;
    ...
    <id> -> [<id>|<str>]* ;
  },
  <id>                              # start symbol
```

```
)
```

Other optional parts:

- `levels` - define control language by enumeration

```
levels = {
  [<str>]* ;                        # listed symbols
  [<str>]* ;
  ...
  [<str>]* ;
}
```

- `automaton` - define control language by finite automaton

```
automaton = (
  {<id>, <id>, ..., <id>},          # states
  {                                 # rules
    <id> <str> -> <id> ;
    <id> <str> -> <id> ;
    ...
    <id> <str> -> <id> ;
  },
  <id>,                             # start state
  {<id>, <id>, ..., <id>}           # final states
)
```

- `precedence` - define precedence rules for operators priority

```
precedence = (
  <dir>: [<id>|<str>], [<id>|<str>], ..., [<id>|<str>] ;
  <dir>: [<id>|<str>], [<id>|<str>], ..., [<id>|<str>] ;
  ...
  <dir>: [<id>|<str>], [<id>|<str>], ..., [<id>|<str>] ;
)
```

Meaning of shortcuts in syntax of grammar file:

- `<id>` - c-like id
- `<str>` - string bounded by simple quotes (`'`), quote can be escaped by `\`
- `<dir>` - associativity direction in precedence table, values:
    - `left` - left associativity
    - `right` - right associativity

You can also write single line comments starting with `#` symbol.

Input string

Input string is expected to contain grammar terminals. White chars are ignored (used as separators). Terminals don’t have to be separated by white chars, but there can be bad interpretation, if there are multiple ways how to interpret string (e.g. two terminals `a` and `aa`).

Return codes and errors

Depending on input, application returns one of these exit codes:

- 0: Input string belongs to input grammar.
- 1: `NOT_IN_GRAMMAR` - Input string doesn't belong to input grammar.
- 2: `NONDETERM_ERROR` - Nondeterministic step has been applied and then we ran into error. It is not clear, if string belongs to grammar. This problem is described closely in Bc. thesis.
- 3: `GRAMMAR_PARSE_ERROR` - Syntax or logical error in input grammar file.
- 4: `LR_TABLE_ERROR` - Conflict or other problem in LR table.
- 5: `FINITE_AUTOMAT_ERROR` - Logical error in user defined Finite automat.
- 10: `ARGUMENTS_ERROR` - Arguments error.
- 99: `INTERNAL_ERROR` - Other internal error.

Test suite

Test suite is placed in `./tests/` folder and contains many example grammars, including all grammars mentioned in the Bc. thesis.

You can run all tests by bash script by typing `./test.sh` in tests folder.

LaTex source files of thesis

Source files of thesis are placed in `./docs/` folder. For compiling to .pdf format just type `make` in documentation folder. Package `pdflatex` is required.