Assignment 1
version 1.0

In this assignment, you are required to recognize a program written in Crazy and translate it to a target language.

1. **Error Handling:**

There are 3 kinds of lexical error you are required to check:

- Error Token: when there is a substring that does not match with any description of token, the exception ErrorToken must be raised with the first character of the substring.
- Unclosed string: when a string has the first single quote but no the corresponding single quote in the same line, the exception UnclosedString must be raised with the lexeme of the unclosed string from the first single quote to the last character before the new line or the end-of-file.
- Illegal escape in string: There are only 3 valid escape sequences. If there is another escape sequence appearing in a string, the exception IllegalEscapeInString must be raised with the substring from the first single quote to the illegal escape sequence, inclusively.

The syntax error is processed in the TestUtils.py so you don't need to do anything with this kind of error.

Note that in this assignment, you are NOT required to do type checking

2. **Target language specification:**

If a program is correctly written in Crazy, it will be translated into a list in the target language which is a comma-separated 3-element list enclosed by square brackets followed by a dot and a blank. The first element is the list of variables declared in the variable declaration of the Crazy program. If there is no variable declaration, the corresponding list in the target is empty. The second element represents the function declaration and it is an empty list if there is no user function. The last element is the list of statements translated from those inside the compound statement in the source. For example,

```
program test2;
var a,b:integer;
begin               // no function declared
    a := 1;
    b := 2;
    writeIntLn(a+b);
}
```

is translated to

```
[[var(a,integer),var(b,integer)],[],[assign(a,1),assign(b,2),call(writeIntLn,[add(a,b)])]].
```

(Note that there is a blank after the .)

A variable declaration is translated using the functor var. For example,

```
var a,b,c:real;
var d:integer;
var e:array[5][6] of real;
var f:string;
```

is translated into

```
var(a,real),var(b,real),var(c,real),var(d,integer),var(e,arr([5,6],real)),
var(f,string)
```

A constant declaration:

```
const a = 7;   => const(a,7)
```

A procedure declaration is translated to the following functor:

proc(N, L1,L2) where N is the function name, L1 is the list of parameters, L2 is the list representing a block statement. For example,

```
program test3;
    procedure foo(a:integer) ;
    begin
        var b:integer;
        b = a + 1;
        writeInt(b);
    end;
begin
    foo(1);
end.
```

is translated to

```
[[],[proc(foo,[par(a,integer)],[var(b,integer),assign(b,add(a,1)),call(writeIn
t,[b])])], [call(foo,[1])]].
```

A function is translated similar to a procedure except it uses the functor func(N,L1,T,L2) where T is the return type. For example,

```
program test4;
    function foo(a:integer):integer ;
    begin
        foo := a + 1;
    end;
begin
    foo(1);
end.
```

is translated into

[[],[func(foo,[par(a,integer)],integer,[assign(foo,add(a,1))])],
[call(foo,[1])]].

The target for each statement is as follows:

left := expression => assign(Le,E) where Le is the target of left and E is for expression.

begin…end ⇒ […]

if <expr>then stmt ⇒ if(E,S) where E is the target of <expr> and S is for stmt

if <expr> then stmt1 else stmt2 ⇒ if(E,S1,S2)

while ⇒ while(E,S)

do ⇒ do(L,E) where L is the list of statement targets inside a do statement and E is the target of the condition expression

loop => loop(E,S)

break ⇒ break(null)

continue ⇒ continue(null)

call statement ⇒ call(N,L) where N is the name of the invoked procedure and L is the list of the targets of arguments

The target for each operator is follows:

id[<expr1>]…[<exprn>] ⇒ ele(N,[E1,…,En]) where N is the id, E1,…En are the targets of corresponding <expr1>,…, <exprn>

<expr1> + <expr2> ⇒ add(E1,E2) where Ei is the target of <expri>

<expr1> - <expr2> ⇒ sub(E1,E2)

- <expr> ⇒ sub(E)

not <expr> ⇒ bnot(E)

<expr1> and <expr2> ⇒ band(E1,E2)

<expr1> or <expr2> ⇒ bor(E1,E2)

<expr1> > <expr2> ⇒ greater(E1,E2)

<expr1> < <expr2> ⇒ less(E1,E2)

<expr1> >= <expr2> ⇒ ge(E1,E2)

<expr1> <= <expr2> ⇒ le(E1,E2)

<expr1> <> <expr2> ⇒ ne(E1,E2)

<expr1> = <expr2> ⇒ eql(E1,E2)

<expr1> * <expr2> ⇒ times(E1,E2)

<expr1> / <expr2> ⇒ rdiv(E1,E2)

<expr1> div <expr2> ⇒ idiv(E1,E2)

<expr1> mod <expr2> ⇒ imod(E1,E2)

a function call ⇒ call(N,L) where N is the function name, L is the list of targets of arguments

The target for constants as follows:

integer: 8 ⇒ 8

float: 4.5 ⇒ 4.5

boolean: true ⇒ true

array: $[a_1,a_2,\ldots a_n]$ $\Rightarrow$ array($[a'_1,a'_2,\ldots,a'_n]$), where a'_i is the target of a_i. For example,

[2,3,4] $\Rightarrow$ array([2,3,4])

[[2,3],[3,4]] $\Rightarrow$ array([array([2,3]),array([3,4])])

string: s '…' $\Rightarrow$ str(s'), where s' is conversed from s. The first and the last single-quotes in s are conversed to double quotes. All duplicated single quotes in s are changed to escaped characters (\') in s'. Any double quote in s is changed into escaped character \" in s'. For example,

'abc' $\Rightarrow$ "abc"

'ab''cd' $\Rightarrow$ "ab\'cd"

'ab"cd''ef"g' $\Rightarrow$ "ab\"cd\'ef\"g"

3. Requirements:

You must use ANTLR version 4.9.2 to generate the translator in Python.

You should use Python 3 to compile and run your translator because we will use it to check your result.

All input and output are from and to the console.

You must submit the following files:

- Cra2Pre.g4: the ANTLR file
- PreGenSuite.py: containing at least 100 test-cases

Note that you must NOT compress your files.

The deadline of submission is announced in the website.

You must complete the assignment by yourself, otherwise, you will get 0 mark for the **subject**.