CS 153 / SE 153
Concepts of Compiler Design

Fall Semester 2015

Department of Computer Science
San Jose State University
Instructor: Ron Mak

Assignment #3

Assigned: Wednesday, September 16
Due: Wednesday, September 30 at 11:59 pm
Team assignment, 100 points max

Pascal set expressions and assignments

Modify the Pascal parser code from Chapter 6 and the interpreter code from Chapter 8 to add the ability to parse and execute **set expressions** and **set assignment statements**. For this assignment, the base type of your sets (the data type of the members of the sets) will be small integers from 0 through 50.

First **modify the Pascal parser** to build parse trees for set expressions. You will need to define new node types and design new tree structures to represent set values and set expressions. Then **modify the executor routines** to interpret set expressions and assignment statements by walking your parse tree.

You choose how to implement the Pascal set values during run time.
Tip: Examine the Java set classes.

Be aware of the following features of Pascal sets:

- **The members of a set are unordered.**
  The set [1, 2, 3, 4] is the same as the set [3, 1, 4, 2].
- **The members of a set are unique.**
  The set union [1, 2, 3, 4] + [3, 4, 5, 6] produces the set [1, 2, 3, 4, 5, 6].
- **The .. token specifies a range of values in a set.**
  [1, 3, 6..9, 2] is the same as the set [1, 2, 3, 6, 7, 8, 9].
- **You can have variables and expressions in a set value.**
  [3, k, 5, 3*m, n] if the value of each expression is 0 through 50.
  (You should flag a set element value that is out of range as a runtime error.)

Recall that the assignment statement executor in the back end sends an **`ASSIGN`** message after executing each assignment statement, and that the message includes the source statement line number, the target variable name, and the expression value. The listener for these messages is the main **`Pascal`** class. You may need to modify the listener code to print set values.

Input file `sets.txt`

Your interpreter must **parse and execute** the following input file. (There should be no invisible control characters in the file other than end-of-line characters.)

```
BEGIN
    low := 15;
    mid := 45;
    high := 50;

    evens  := [0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20];
    odds   := [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21];
    primes := [2, 3, 5, 7, 11, 13, 17, 19, 23, 29];
    teens  := [13..19];

    s1 := evens*odds;
    s2 := evens - teens + [high, mid..47, 2*low];
    s3 := evens*primes + teens;
    s4 := s3 - odds;
    s5 := evens + odds + primes + teens;
    s6 := (primes - teens)*odds;

    b1 := odds - primes = [15, 9, 21, 1];
    b2 := teens <= evens + odds;
    b3 := primes >= teens;
    b4 := odds - teens*primes <> [21, 7, 9, 1, 11, 5, 15, 3];
    b5 := 15 IN teens - primes;

    s7 := [];
    i := 1;

    WHILE i <= 50 DO BEGIN
        s7 := s7 + [i];
        i := 2*i;

        IF 8 IN s7 THEN s7 := s7 + [10]
                   ELSE s7 := s7 - [4]
    END
END.
```

Run your interpreter with the `-i` option to print the parse tree.

For this assignment, none of the variables are declared. Therefore, at run time, a variable can be assigned a value of any data type, integer, real, boolean, or character.

Tip: Write a complete Pascal program including some printing statements that incorporates the above statements. Compile and run it using Free Pascal or Lazarus to serve as a check of the output of your Pascal interpreter.

Input file `seterrors.txt`

Your interpreter must parse the following input file and handle the syntax errors. It should not try to execute any of this file.

```
BEGIN
    error1 := [1, 2  3];           {missing comma}
    error2 := [1,, 2, 3,];         {extra commas}
    error3 := [1, 2, 3;            {missing close square bracket}
    error4 := [2, 1, 3, 1];        {non-unique members}
    error4 := [10..20, 15];        {non-unique members}
    error4 := [15, 10..20];        {non-unique members}
    error4 := [15..25, 10..20];    {non-unique members}
    error5 := [2, 5.., 8];         {invalid range of values}
    error6 := [1, 2, 3] OR [4];    {invalid operator}
    error6 := [1, 2, 3]/[4, 5];    {invalid operator}
    error7 := [1, 2, 3] < [4, 5];  {invalid operator}
    error8 := [1, 2, 3] IN [4];    {invalid operator}
    error9 := 1 IN 2;              {invalid operator}
END.
```

A set constant such as `[2, 1, 3, 1]` is strictly speaking not a syntax error. But since the parser can tell at compile time that the members are non-unique and this is probably not what the programmer intended, it would be helpful to flag it as an error.

A useful tutorial on Pascal sets

http://www.tutorialspoint.com/pascal/pascal_sets.htm

You can ignore the `><` operator and `Include` and `Exclude`. But you must do the `IN` operator.

What to turn in

This is a team assignment. Each team turns in one assignment and each team member will get the same score. Create a zip or gzip compressed file that contains:

- All your Java source files (Just zip the **`src`** directory created by Eclipse, but do not include any **`.class`** or **`.jar`** files.)
- Text files containing the output from running your program.

Name the file after your team, for example, **`SuperCoders.zip`**. Email the file as an attachment in a message to ron.mak@sjsu.edu. Example subject line:

`CS 153 Assignment #3 Super Coders`

In your email message, you can explain any assumptions you made or point out anything especially clever about your program.

Be sure to CC all your team members.