Parser Implementation

Octavian Dogar

In my parser implementation I had the following constraints:

* Identifiers must not be longer than 8 chars;
* A dual symbol table for constants and identifiers
* A plain hashing table for the implementation of the symbol table

Because the insertion in a native java hashing table is asynchronous and does not have a incrementing order, I used the Collections static method for synchronizing a linked hashed map.

For the representation of the PIF, I used a plain ArrayList containing generic Pair instances for tokentype and symTblPosition with an autoincrementing index at the moment of addition.

In parsing the tokens, I use java’s StringTokeniser, with a file-uploaded String as the input.

Regexes were used in differentiating among the tokens. The token codes are taken from a final static hardcoded map containing all token types coupled with their corresponding type.

Regexes List:

Identifiers: **[a-zA-Z]+[a-zA-Z0-9]\***

**Translation: any string that start with a letter and has one or more letters or numbers after the first char**

Separators: **[,.'\\[\\]\\{\\}\\(\\)\\?;:]+**

**Translation: any char of the above list (excepting the escape slashes)**

String Constants: **'\\p{Alnum}+'**

Translation: **any string surrounded by 2 ‘ chars**

**Numeric Constants: [-]?[\\d]+**

**Translation: zero or one – followed by any number**

**Operators: [\\+\\-\\\*/<=>!]+**

Translation: any char of the above

In the case of the reserved words, I checked whether the token is found in the hardcoded reserved words list:

(**"char"**,**"const"**,  
 **"do"**,**"if"**,**"in"**,**"bool"**,**"print"**,  
 **"while"**,**"for"**,**"int"**,**"break"**,**"&&"**,**"||"**,  
 **"real"**,**"String"**,**"struct"**)

The SymbolTable uses 2 native HashMaps with LinkedHashMap implementaitons (separation between constants and identifiers) :

**constsMap** = Collections.*synchronizedMap*(**new** LinkedHashMap<>());  
**identsMap** = Collections.*synchronizedMap*(**new** LinkedHashMap<>());

In the parsing of the input, I counted the lines and incremented a counter each time the newline char is found. Thus, at the moment of encountering a new unidentified token, an exception is thrown, execution is halted and the last parsed line is given in the exception message.

An error could be under the form of any char outside the alphabet.

Ex: \*&#^%@\*&#111

Or a larger than 9 chars identifier:

Ex: askdnalsdnlnwjqn