1 Introduction

In this project we were requested to design and write a compiler for Super-Fortran. It is a very simple imperative language. The project divided in 3 parts and for this first part. we had to produce the lexical analyser of the compiler.

The lexical analyser has been generated using JFlex which take as input a specification (.flex file) with a set of regular expression and corresponding actions.

2 Regular expressions

`Upper = [A-Z]` Here we define a macro of a range that match any upper letter from A to Z.

`Lower = [a-z]` This is the same as the previous one but for lowers letters.

`Alpha = {Upper}|{Lower}` The language of alpha define the union of the languages of the 2 previous macros.

`Numeric = [0-9]` The range of all decimals digits that will be useful for matching numbers.

`AlphaNum = {Alpha}|{Numeric}` A macro that matches all alphanumerics characters

`NZero = [1-9]` We need non zero digit macro because a number literal can't begin with 0.

`Zero = "0"` We need a macro for matching single zero which is the only number that begins with 0.

`VarName = {Lower}({Numeric}|{Lower})*` VarName is the macro that define a variable. That start with a lower case characters and contains only digits and lower case characters.

`EndLine = "\n"` This macro exist because it will be repeated severals time and we want to avoid magic constants.

`Number = {Zero}|({NZero}{Numeric}*)` A number is a zero or a non zero digit followed by an arbitrary number of digits.

`ProgName = {Upper}{AlphaNum}*({Lower}|{Numeric}){AlphaNum}*` A program name start with an upper case characters and is said to be an non upper case word. The choice of the meaning of a non upper case word will be exlpained in the next section.

`BeginComment = "/*"` The begin of a multi line comment.

`Ignore = " " | "\t"` The ignored charachters by the lexer.

3 Explanation of choices and hypotheses

3.1 Macro definitions

We decided to use macros everywhere except when we face one characters or a concatenation of two characters that will not be reused in the flex file. Because macros make the code more readable and the changes in the input langauge easier.

3.2 Unexpected tokens

Unexpected tokens are words that does not match with any regular expression. We consider that they should not be ignored by the lexical analyser that's why an exception is thrown. And for this reason we defined a macro `Ignore` for charachters that should be ignored and are not a part of the grammar of super fortran.

3.3 Non upper case word

For the matching of the Progname we had two choices on how to interpret a non upper case word. First we can define it as word that contains a least one lower case charachters. Or the other option is a word that contains at least one digit or one lower case charachters.

We took the second one because a word that contains a digit can't be confused with a keyword.

4 Bonus question

The difficulty with nested comments problem was to find a regular expression that recognize them. In fact the nested comments problem is similar to the problem of opening/closing parenthese which is not a regular language. For this reason the solution that we found was to use jflex states in which we can put java code.

Nested comment requires to count (or to use a stack), opening and closing strings for comment. With a java integer as a counter that increment when we find the opening sequence and decrement on closing sequence we can track if we are currently inside a comment. It is the case when the counter is positive. When the counter is equal to 0 we are outside comments and when the counter is negative there is more closing sequences that needed and this should be considered as a syntax error.

5 Tests files desciption

6 Conclusion