ECOLE
POLYTECHNIQUE
DE BRUXELLES

MA1 - IRIF

COMPILING PROJECT

INTRODUCTION TO LANGUAGE THEORY AND COMPILING

Authors

Eliot COSYN

Quentin ROELS

Course

INFO-F403

Professor

Gilles GEERAERT

Academic Year

2022-2023

Table of contents

1 Lexical Analyser

1.1 Implementation

1.1.1 Regular Expressions

To match the tokens, we first had to determine the regular expressions of the keywords, numbers, variable names and program names allowed by the Fortress language

Keywords

The number of keywords in Fortress is pretty low. Therefore, to match those tokens, we can simply ask the lexical analyser to search for them specifically instead of searching uppercase word. To do that we can use the following regular expression : "TOKEN" (where TOKEN is a known token of Fortress such as BEGIN, IF, END,...). That way we don't return any symbol for an all uppercase word thinking that it's a Fortress keyword when it's not.

Numbers

"A [Number] represents a numerical constant, and is made up of a string of digits only, without leading zeroes", we then have to match the number 0 or any string of numbers starting with a non zero symbol [1-9] and followed by any numerical symbol [0-9]*. Since the minus sign is a token on its own, we expect him to be match from the regular expression "-" of previous section and we do not need to match it with the [Number] regular expression.

$$Number = ([1-9][0-9]*|0)$$

Then, to prevent any leading zeroes, we applied a filter with the [WrongNumber] regular expression. I.e. the numbers starting with at least one zero followed by any other digit.

$$WrongNumber = 0 + [0-9] +$$

Variables name

"A [VarName] identifies a variable, which is a string of digits and lowercase letters, starting with a letter". We thus have to match any string of symbol starting with a lowercase letter [a-z] and followed by any alphanumerical (non uppercase) symbol ([a-z]|[0-9])*.

$$VarName = [a-z]([a-z]|[0-9])*$$

Program name

"A [ProgName] identifies the program name, which is a string of digits and letters, starting with an uppercase letter but not entirely uppercase (e.g. FaCTORIAL, although not very pretty, is accepted, FactorialPrgm also, but FACTORIAL is not, so that it is not confused with a keyword)". To do that we need to put an uppercase letter at the start of the word [A-Z], and force the existence of a lowercase letter [a-z] somewhere in the word. Since it could be anywhere from thes start to the end of the word, we use the Kleene closure of alphanumerical symbols ([A-Z]|[a-z]|[0-9])* before and after the mandatory lowercase letter.

$$ProgramName = [A-Z]([A-Z]|[a-z]|[0-9])*[a-z]([A-Z]|[a-z]|[0-9])*$$

Due to the program name specifications, our lexer was not able to make the difference between [ProgName] and "TOKEN"[VarName], i.e. between a program name and a Fortress token linked to a variable with no white space separating them.

To avoid that problem, we added a PROGNAME state that we enter when matching the token "BEGIN". That way, our lexer already forces a program name directly after the keyword "BEGIN", which will not have a negative impact on the compilation of a valid Fortress script.

1.1.2 Detecting comments

As mentioned earlier, the job of the Lexical Analyser is to return the tokens of the language we want to compile. Since the content of the comments is useless for the compiler, we need to drop it.

Fortress comments

There are two ways for commenting a Fortress code :

- Starting with the string "::", short comments end when the end of line "$\backslash n$" symbol is reached
- Starting and ending with "%%", long comments allow to write on multiple line until the end symbol is reached

For both type of comments we created a new jflex state that will do nothing when matching symbols and will wait for the end of comment token. When the end of comment is matched, it returns to the main state and restart searching for Fortress symbols.

Nested comments

The problem we faced with nested comments in Fortress is due to the symbol used, "%%" is both the starting and ending comment symbol. This causes the program iterating on each symbol to not be able to tell if the matched token is used to start or end a comment. Which makes it impossible to use the same solution as nested parenthesis, using a stack to count how many parenthesis are currently open, and thus, how many are yet to close.

For nested comments to be implemented in our Fortress compiler, we should be able to know if "%% A %% B %% C %%" means that A and C are two different long comments **OR** that B is nested in the primary comment.

One possible way to implement the nested comments in our compiler would be to first consider them as two separated comments and then check if B is syntactically correct in Fortress. If so, we let it slide to the next phase of the compiling process, else, we drop the tokens like we would have done with comments. The limitation is then "no valid Fortress language in nested comments" instead of "no nested comments".

1.2 Testing

To test our lexical analyser, we edited the Fortress example file in different ways to see how the lexer would react in those situations.

1.2.1 Wrong tokens

First, we wrote all sorts of bad tokenized Fortress scripts to verify that the program exits with the appropriate error code. Every kind of token was reviewed by misspelling it or writting it poorly :

1. numbers (ex: 0012)
2. variable name (ex: reSult, 7result)
3. keywords (ex: READ -> REED)
4. program name (ex: FACTORIAL)

The correct use of long comments was also tested mainly to check if the comment was closed before the end of the file.

1.2.2 Valid files

Finally, we wrote valid Fortress scripts with no spaces or new line, or even more white spaces than necessary to check if our lexer could tokenize them properly and thus met the specifications of the project.