CSC330 Language Design and Implementation

Assignment 7 (Team Assignment)

**Group members**

**banking.ebnf**

Breaking down the ebnf file line by line :

**digit** ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;

A digit can be any one of the characters from "0" to "9".

**letter** ::= "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z" ;

A letter can be any one of the characters from "a" to "z".

**integer** ::= digit { digit } ;

An integer is defined as a single digit followed by 0 or more additional digits (represented by {})

**program** ::= { statement } ;

This line defines a program as consisting of zero or more statements. The { statement } indicates that statements can repeat zero or more times.

**statement** ::= (create\_account | deposit | withdraw | balance | exit) ";" ;

This line defines a statement as one of the commands (create\_account, deposit, withdraw, balance, or exit) followed by a semicolon.

**identifier** ::= letter { letter | digit } ;

An identifier is defined as a letter followed by zero or more letters or digits. Alphanumeric string starting with a letter.

**first\_name** ::= letter { letter } ;

A first name is defined as a letter followed by zero or more letters. No numbers allowed.

**last\_name** ::= letter { letter } ;

A last name is defined as a letter followed by zero or more letters. No numbers allowed.

**account\_number** ::= identifier ;

An account number is defined as an identifier (alphanumeric string)

**amount** ::= integer ;

An amount is defined as an integer.

**deposit** ::= "deposit" amount "to" account\_number ;

Defines the syntax for depositing an amount to an account. It consists of the keyword "deposit" followed by an amount, the keyword "to", and an account number.

**withdraw** ::= "withdraw" amount "from" account\_number ;

Defines the syntax for withdrawing an amount from an account. It consists of the keyword "withdraw" followed by an amount, the keyword "from", and an account number.

**balance** ::= "balance" "of" account\_number ;

This line defines the syntax for checking the balance of an account. It consists of the keyword "balance" followed by the keyword "of" and an account number.

**create\_account** ::= "create" "account" first\_name last\_name "with" "balance" amount ;

Defines the syntax for creating an account. It consists of the keyword "create" followed by the keyword "account", then a first name, a last name, the keyword "with", the keyword "balance", and finally an amount.

**exit** ::= "exit" | "quit" | "end" ;

This line defines the possible commands for exiting the program. It can be one of the keywords "exit", "quit", or "end".

**Lexer**

We import re library in python to use regular expressions. We then define a Token class to represent tokens produced by the lexer.

\_\_init\_\_ is the constructor method that initializes the token's type, value, line number, and column number.

The \_\_repr\_\_ method provides a way to define how the Token objects are represented as strings.

Coming to the Lexer class made to perform lexical analysis on the code –

\_\_init\_\_ is the constructor method that initializes the lexer with the code to be analyzed and an empty list to store tokens.

tokenize method iterates through the tokens generated by tokenize\_code and adds them to the token list, as long as the token is 'SKIP' or 'NEWLINE'.

tokenize\_code is a generator method that defines patterns for different token types using regular expressions. token\_specification is a list of items where each item contains a token type and a regex pattern to match that type. tok\_regex constructs a single regular expression pattern by joining all individual patterns with the or operator. get\_token compiles this pattern into a regex matcher. We initialise the counter variables to default values.   
line\_num tracks current line number.

line\_start records position at the start of the current line.

pos is the current position in the code.

mo is a match object.

We then use a while loop to iterate as long as there is a match.

If the token type is 'NEWLINE', it updates line\_start and increments line\_num.

If the token type is **not** 'SKIP', it calculates the column number and yields a new Token object.

pos is updated to the end position of the current match.

mo is updated to the next match in the code.

We also add an exception handler so if end of the code is not reached but there are unmatched characters, it raises a RuntimeError.

To test this, we add a sample code and create a Lexer object with this sample code. With a loop, we print the tokens generated in this code.

**Output :**

A screenshot of a computer

Description automatically generated

**References**

<https://bnfplayground.pauliankline.com/>

<https://www.freecodecamp.org/news/what-are-bnf-and-ebnf/>

<https://www.geeksforgeeks.org/python-__repr__-magic-method/>

<https://dev.to/lorenzofelletti/how-to-build-a-regex-engine-in-python-part-2-the-lexer-2b4n>

**Temporary instructions :**

A Word document named Readme.docx. This should thoroughly explain what you have done for the assignment.

Readme.docx (Word Document):

Create a Readme.docx file. This file should be a brief overview of what you did. You will go into details in your video, and your code should be well documented with comments, so you do not need to repeat yourself in this document. But I want you to add anything to this document that you do not feel is well explained in the code or video.

At a minimum, the file should contain the following:

• This file should include all team members’ names, the course, and term information, plus a

title.

• State in this document who wrote which classes, modules, and functions.

• References/citations.

• Assume I know nothing of the topic and nothing of the testing environment, so include

detailed instructions on how I can run your code.

• This should be written as a final report on the assignment, so when I grade, all I really need

to do is look at this and watch your video to gain a full understanding of what you have done

and a full understanding of how to run your code. (However, you will also be uploading all

your files so that I can look at your code and run it if I need further clarification on your

program.)