Github source code: https://github.com/nicolagutanu/flcd/tree/main/carina/lab3

Delivered: the documentation (about the lexical analyzer and the symbols table), input (my small programs from lab1a and the tokens of my language), PIF.out (pif output of the program for each of the analyzed programs), ST.out (symbols table for each of the analyzed programs)

For the lexical analyzer:

function read_keywords(key_file):

in params: key_file: string -> name of the file containing the keywords

out params: none, the list of keywords is a global variable

Reads from the file, one line at a time, all the specified keywords of my language.

function read_from_file():

in params: none

out params: prog: string -> a string containing the program to be analyzed

input: name of the file

Reads from the file a string containing the program.

function write_to_file():

in params: none

out params: none

input: names of the files in which we will write the pif, the id symbols table and the constants symbol table

Writes to a specified file the pif of the analyzed program, the symbols table for the ids and the symbols table for the constants.

function get_location(err):

in params: err: string -> error found by the analyzer

out params: the line and starting position of the error

Searches the initial program for the specific line and starting position where the error occurred and returns them.

function separate(prog):

in params: prog: string -> given program to analyze

out params: separated: list of strings -> list of tokens

Parses through the given program and separates the tokens into a list using the given keywords of the language.

function check_identifier(identifier):

in params: identifier: string

out params: True or False

Checks, using a regex, if the given identifier is a valid identifier in my language.

`regex = '^[a-z][A-Za-z0-9_]*' ->` the identifier needs to start with a lowercase letter and continue with 0 or more characters that can be lowercase letters, uppercase letters, digits or underscore.

function check_constant(const):
    in params: const: string
    out params: True or False
Checks, using multiple regexes, if the given constant is either a number, a character or a string defined in my language.
`regex_nr = '^[-+]?[0-9]+$' ->` the number constant can start with a sign (-, +) and then continue with at least one digit
`regex_char = '^[a-zA-Z0-9]$' ->` the character constant can be one and only one lowercase letter, uppercase letter or digit and I also check that it is between quotes later in the code
`regex_string = '^[a-zA-Z0-9]+$' ->` the string constant needs to be at least one lowercase letter, uppercase letter or digit and I also check if it's between quotes later in the code

function analyze(program):
    in params: program: list of strings -> list of program's tokens
    out params: none
Goes through the list of tokens and categorizes them as identifiers, constants or anything else (reserved words, operators, separators). After being categorized, they are added to the pif with their positions (0 - reserved words/operators/separators, position in specific tree for ids and constants). If they are neither of the above mentioned, it specifies that the program is lexically incorrect and prints the location of the error.

For the symbols table:

function insert(self, identifier, pos):
    in params: identifier: string -> identifier found in code
        Pos: integer -> its index in the code (n th identifier)
    out params: none
We traverse the branches of the tree alphabetically (left or right compared to the root) to find where we could fit the new identifier, saving the name of the identifier and it's position in the code. If the identifier already exists in the tree, we return (don't make any changes, the initial position of the identifier in the code is the only one saved).

function search(self, identifier):
    in params: identifier: string -> identifier we want to search
    out params: False or (True and position of identifier)

We traverse the tree alphabetically once again while comparing the given identifier to the existing ones in the tree on the left/right branches until we either get to the end of the branch (in which case, the identifier doesn't exist) or until we've found it and returning it's position.

```
function print(self):
        in params: none
        out params: none
```
We traverse the branches in inorder and print the nodes.

```
function size(self):
        in params: none
        out params: size of tree: int
```
Traverses the tree and counts how many nodes are in it.

```
function write_to_file(self, file):
        in params: file: string -> name of file in which we write our results
        out params: none
```
We traverse the tree just as if we would print it, but instead we write the identifier and its position to the given file.