Ceng#

Programming Languages Project 1
Lexical Analyzer for Ceng#

<u>Project Definition</u>: The Program should accept a source file called code_file.ceng and produce a text file named as code.lex that contains all the tokens of the code.lex listed one after the other.

Lexical rules for the programming language Ceng# are as follows:

~~1~~ - Identifiers:

- Maximum identifier size is 25 characters. If you use an identifier larger than that, the lexical analyzer issues an error message.
- Ceng# language is case sensitive
- Identifiers start with an alphabetic character (a letter) and are composed of one or more letters, digits or_ (underscore)
- Example Token: Identifier(my_var_1)

2- String constants

- String constants of Ceng# are delimited by double quotes (ASCII code 34)as in “this is a string”
- String constants have unlimited size
- String constants cannot contain the double quote character. when you reach one, the string terminates.
- If a string constant cannot terminate before the file end, there should be a lexical error issued.

3- Integer constants

- Maximum integer size is 10 digits. If you use an integer value longer than that, the lexical analyzer issues an error message.
- Negative values are supported.
- Example Token: IntConst(352)

4- Keywords:

- Keywords are: break,case,char,const,do,else,enum,float,for,if,int ,double long, struct, return, static, while
- Ceng# language is case sensitive, and all the keywords are standardized as lower case. You cannot write the same word as “while” OR “While” OR “WHILE”.
- Example Token: Keyword(while)

5- Operators

- Valid operators of the language are +,-,*,/,++,--,==, <, >, <=,>=, =
- Example Token: Operator(++)

6- Brackets

- LeftPar: (RightPar:)
- LeftCurlyBracket: { RightCurlyBracket: }
- Example Token: LeftCurlyBracket

7- End of line: ;

- Example Token: EndOfLine

8- Comments: Anything between /* and */ is a comment. Single line comment is //

- If a comment cannot terminate before the file end, there should be a lexical error issued.
- Comments are just like blank space, and they provide <u>no tokens</u>.

Example:

if code_file.ceng contains:

```
number=number_1+25; /*addition*/

number_1++; /*increment*/
```

code.lex would be:

```
Identifier: number

Operator : =

Identifier :number_1

Operator: +

IntConst: 25

EndOfLine

Comment

Identifier : number_1

Operator: ++

EndOfLine

Comment
```

Time: 24.05.2022

Team: 2 / 3 people