CSC 3315

Team report: lexer, part 3

Hajar El Boutahiri <80389>  
Youssef Yousfi <85369>  
Hiba Aqqaoui <94519>

1. **Alternative choice**

The choice of implementing a lexer from scratch using Python and not using lexer constructor tools came from the fact that each teammate wanted to know how a lexical analyzer works in depth. Manually coding the lexer has also been quite convenient when it came to debugging as we could keep track of different iterations throughout the source code file. It is also beneficial for us as we are also going to implement the parser afterwards.

1. **Lexer’s design and functions**

We start by reading from the source code file, we read word by word, and character by character. Then, we use process\_lexeme(), the main lexing function that lexes or in other words, puts together operators, punctuation, reserved words. The lexemes are stored in a list string\_lexeme. Tokens getToken(lexeme) are defined by a function that returns a list for each lexeme (containing a unique numerical identifier and token). For the symbol table, we used a two-dimensional list in which reserved words of our language are pre-stored and to which we will be adding variables. The list is two dimensional in order to hold the ‘symbols’ and their type (reserved word or variable name).

For the literal table, the literals are matched with their name and type. The literal and symbol table both can be output in the terminal.

1. **Lexer User Manual :**

The py file, the source code in flower language (in a text-file named ‘file.txt’) and the output file(‘file2.txt’) should have the same file directories. The input is written in the flower language. The output consists of the lexemes, their respective tokens, and the lines where they occur in the input.

1. **Comments :**

We kept the character literals because we use them in our language. Also, we wrote productions for arithmetic expressions, just not the way we have them in the slides.

1. **Lexer Performance :**

*These code examples are mainly to show that each category works, so some things, such as the statements following an if statement, have been skipped to generate a smaller output.*

|  |  |
| --- | --- |
| **Punctuation** | **Code :**  int main(void){     int a;  int b ;  int c ;  } |
| **Output:** |
| **Unknown Lexemes** | **Code :**  int main(void)^  int a$  if (a>b#  if (a<b&  if (a>=b :  if (a<=b!  if (a!=b|  if (a==b~  a=b 6Rt    }  **Output :** |
| **Symbol Table and Literal Table** | **Code :**  int main(void){  int a;  int c;  int d;  c= 4;  d=5;  a=2;  }  **Output:**  Une image contenant texte  Description générée automatiquement |
| **Operators** | **Code :**  int main(void){     int a,b;     if (a>b)     if (a<b)     if (a>=b)     if (a<=b)     if (a!=b)     if (a==b)     a=b;    } |
| **Output:** |
| **Reserved Words** | **Code:**  int main(void){     int i,b,c;     SET i = 0;     if (b>c)[        GOUP();        GODOWN();     ]     else[        GOLEFT();        GORIGHT();     ]     WHILE i<4     BEGIN        PICKFLOWER();        SET i = i + 1     END     back 0;  } |
| **Output:** |
| **User-Defined IDs** | **Code:**  int main(void){     int a;     bool b;     char c;  } |
| **Output:** |
| **Numeric Literals**  **and**  **String Literals** | **Code:**  int main(void){  int a ;  ctchar b ;     SET a = 214;     SET b = "str"  } |
| **Output:** |
| **Whitespace** | **Code:**  int main(void){     int        a;     bool               b;     b      =        True;     a          = 14;  } |
| **Output:** |