Compiler Design(18CSC304J)

Experiment 2

Lexical Analyzer

Harsh Goel
RA1811003010185

Aim: To study and perform lexical analyzer using a programming language.

Language: Python 3.7

Procedure:

1. Create a file or select the file for performing the operations on.
2. For this, created a lexicalAnalyser.py python file
3. Create a C file called input.c in which we will use the lexical analyzer.
4. Write the lexical analyzer code in the python file
5. Run the code and perform the operations required.
6. Note the output and document it.

Algorithm:

- Open the c file using open command in read mode
- Make constants arrays for headers, operators etc. that we have to analyze using the analyzer.
- Loop for all the lines in the c program (file that we imported)
- For all the words , run a loop and check if it is one of constants that we made
- If yes, push to specific token array and increase count
- Print the array and count of the analyzed code.

Code Snippet:

```
import re

f = open('input.c', 'r')

headers = ['<math.h>', '<stdio.h>', '<string.h>', '<conio.h>']
operators = ['=', '+', '-', '/', '*', '++', '--
', '==', '>', '<', '>=', '<=']
keywords = ['void','int', 'float', 'char', 'long', 'return', 'if', 'e
lse', 'include', 'special_symbol_countanf', 'printf', 'main']
identifiers = ['n1', 'n2', 'n3', 'sum']
```

```
literals = ['0', 'is_the_largest_number.', 'Enter_three_different_num
bers: ', ]
symbols = ['{', '}', '[', ']', '(', ')', '#', ';', ',', '"']
header_count = 0
operator_count = 0
keyword_count = 0
identifier_count = 0
literal_count = 0
special_symbol_count = 0

he = []
op = []
ke = []
ide = []
li = []
sy = []

i = f.read()

count = 0
program = i.split('\n')

for line in program:
    tokens = line.split()
    for token in tokens:
        if token in headers:
            header_count += 1
            he.append(token)
        elif token in operators:
            operator_count += 1
            op.append(token)
        elif token in keywords:
            keyword_count += 1
            ke.append(token)
        elif token in identifiers:
            identifier_count += 1
            ide.append(token)
        elif token in literals:
            literal_count += 1
            li.append(token)
        elif token in symbols:
            special_symbol_count += 1
            sy.append(token)
print("\n----------------------------------------------------------
\n")
print("Header Count: {}".format(header_count))
print("Headers", he)
```

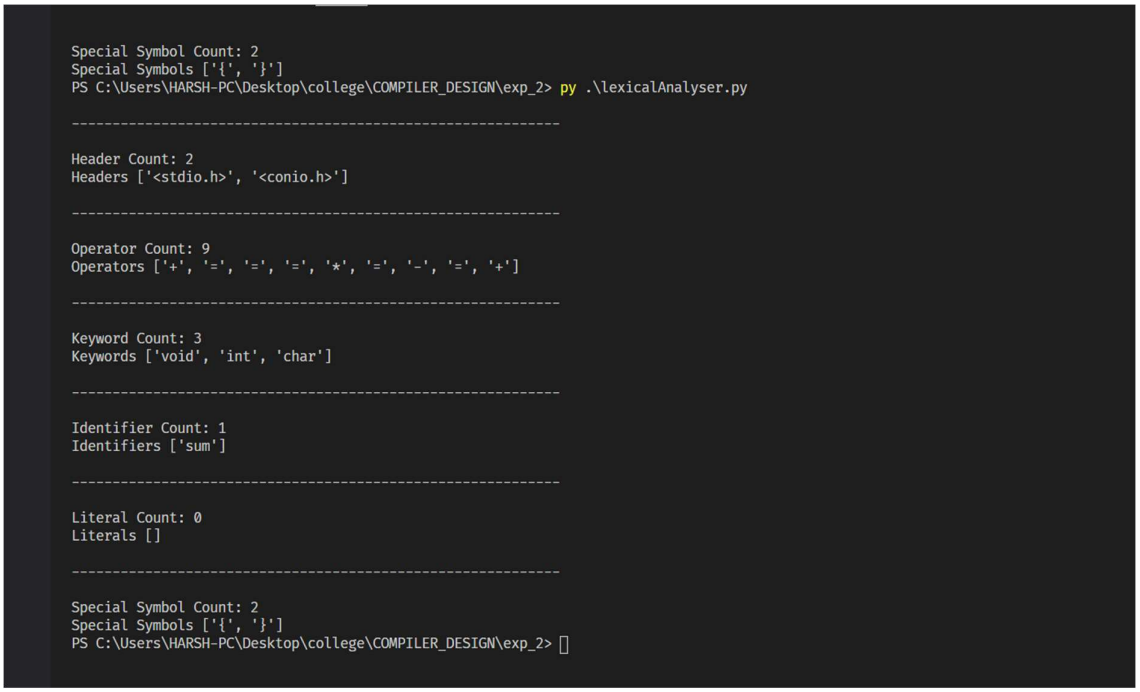
```
print("\n-----------------------------------------------------------------
\n")
print("Operator Count: {}".format(operator_count))
print("Operators",op)
print("\n-----------------------------------------------------------------
\n")
print("Keyword Count: {}".format(keyword_count))
print("Keywords",ke)
print("\n-----------------------------------------------------------------
\n")
print("Identifier Count: {}".format(identifier_count))
print("Identifiers",ide)
print("\n-----------------------------------------------------------------
\n")
print("Literal Count: {}".format(literal_count))
print("Literals",li)
print("\n-----------------------------------------------------------------
\n")
print("Special Symbol Count: {}".format(special_symbol_count))
print("Special Symbols",sy)


'''
for i in he:
    print(i, end=" , ")
print("\n")
print("Operator Count: {}\nOperators:".format(operator_count))
for i in op:
    print(i, end=" , ")
print("\n")
print("Keyword Count: {}\nKeywords:".format(keyword_count))
for i in ke:
    print(i, end=" , ")
print("\n")
print("Identifier Count: {}\nIdentifiers:".format(identifier_count))
for i in ide:
    print(i, end=" , ")
print("\n")
print("Literal Count: {}\nLiterals:".format(literal_count))
for i in li:
    print(i, end=" , ")
print("\n")
print("Special Symbol Count: {}\nSpecial symbols:".format(special_sym
bol_count))
for i in sy:
    print(i, end=" , ")
print("\n")
'''
f.close()
```

C code:

```c
#include <stdio.h>
#include <conio.h>
#define sum a + b;
void main()
{
    int a,b,c;
    char s1[256],s2[256];
    a = 3;
    b = 5;
    c = a * b;
    a = a - c;
    b = c + a;
    printf("%d%d%d",a,b,c);
}
```

Output Screenshots:



Result:

The lexical analyzer was successfully implemented in python and output was recorded.