K. J. Somaiya College of Engineering, Mumbai-77

Batch: A3 Roll No.: 1911034
Experiment / assignment / tutorial No.___4____
Grade: AA / AB / BB / BC / CC / CD /DD
Signature of the Staff In-charge with date

TITLE: Construction of lexical Analyzer for a program

AIM: To understand the concept of regular expressions in python with their usage

Expected OUTCOME of Experiment:

CO - CO2

Books/ Journals/ Websites referred:

1. Ma'am's ppt
2. Geeksforgeeks.com
3. Tutorialspoint
4. W3Schools

Pre Lab/ Prior Concepts:

Regular Expressions in Python

Regular expressions (called REs, or regexes, or regex patterns) are essentially a tiny, highly specialized programming language embedded inside Python and made available through the `re` module. Using this little language, you specify the rules for the set of possible strings that you want to match; this set might contain English sentences, or e-mail addresses, or TeX commands, or anything you like. You can then ask questions such as "Does this string match the pattern?", or "Is there a match for the pattern anywhere in this string?". You can also use REs to modify a string or to split it apart in various ways.

Regular expression patterns are compiled into a series of bytecodes which are then executed by a matching engine written in C. For advanced use, it may be necessary to pay careful attention to how the engine will execute a given RE, and write the RE in a certain way in order to produce bytecode that runs faster. Optimization isn’t covered in this document, because it requires that you have a good understanding of the matching engine’s internals.

The regular expression language is relatively small and restricted, so not all possible string processing tasks can be done using regular expressions. There are also tasks that *can* be done with regular expressions, but the expressions turn out to be very complicated. In these cases, you may be better off writing Python code to do the processing; while Python code will be slower than an elaborate regular expression, it will also probably be more understandable.

Problem statement:

Implement the Lexical analyser for C language and Construct Symbol Table, Function Table, Operator Table, Keyword Table.

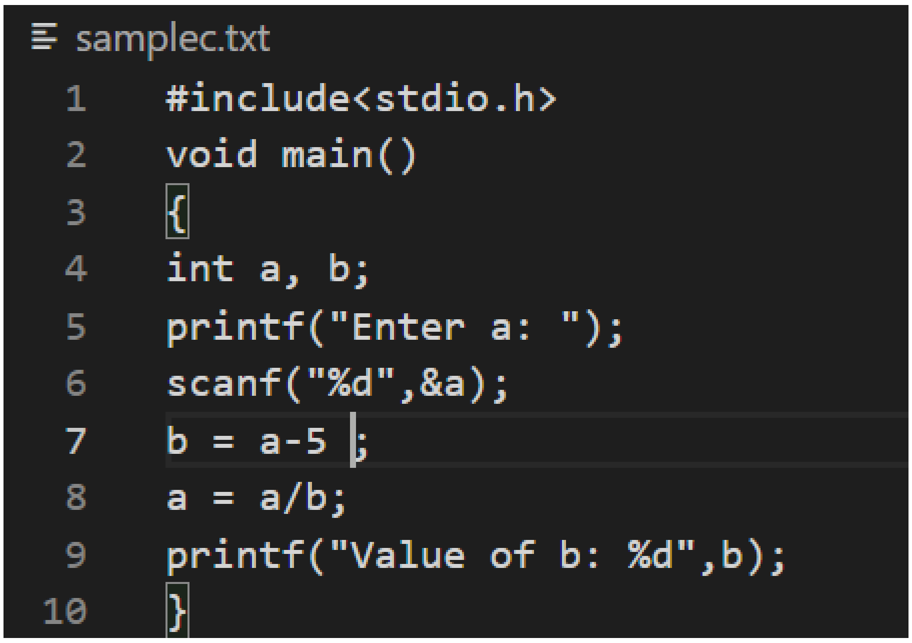
Program (with comments) (Write comments to your program to explain the implementation)

Original Program:

```
import  re
f = open("samplec.txt","r")
program = f.read()
print(program)
keyword=['auto','break','case','char','const','continue','default','do','
double','else','enum','extern','float','for','goto','if','int','long','re
gister','return','short','signed','sizeof','static','struct’,’switch','ty
pedef','union','unsigned','void','volatile','while']
keyarray=[]
for x in keyword:
    if re.findall(x,program):
        keyarray.append(x)
litarray = re.findall("[0-
9]{1,5}",program) #all the numerical characters present are already store
d in the litarray
```

```
operatorarray = re.findall("[|][|]|[&][&]|[=][=]|[<][=]|[>][=]|[<][<]|[>]
[>]|[!][=]|[*][=]|[+][=]|[-][=]|[/][=]|[+][+]|[-][-]|[+,-
,*,/,%,=,&,|,!]", program)

funarray =re.findall("\s*\w*\(.*\)",program)

print("------------------------")
print("Keyword Table")
print("----------------------")
i=1
for x in keyarray:
    print(i,x)
    i=i+1

print("----------------")
print("Literal Table")
print("----------------")
i=1
for x in litarray:
    print(i,x)
    i=i+1

print("----------------")
print("Operators Table")
print("----------------")
i=1
for x in operatorarray:
    print(i,x)
    i=i+1

print("----------------")
print("Functions Table")
print("----------------")
i=1
testarray = list()
for text in funarray:
    x=text.split('(')

    if x[0] not in testarray:
        testarray.append(x[0])

i=1
for text in testarray:
    print(i,text)
    i=i+1

result=list()
for line in f:
    line=line.strip("\n")
    x=re.findall("[@_!#$%^&*()<>?/\|}{~:]",line)
    result.extend(x)
    result = list(set(result))
print("Special symbols Table:-")
index=1
for i in result:
    print(index,"\t",i)
    index=index+1
```

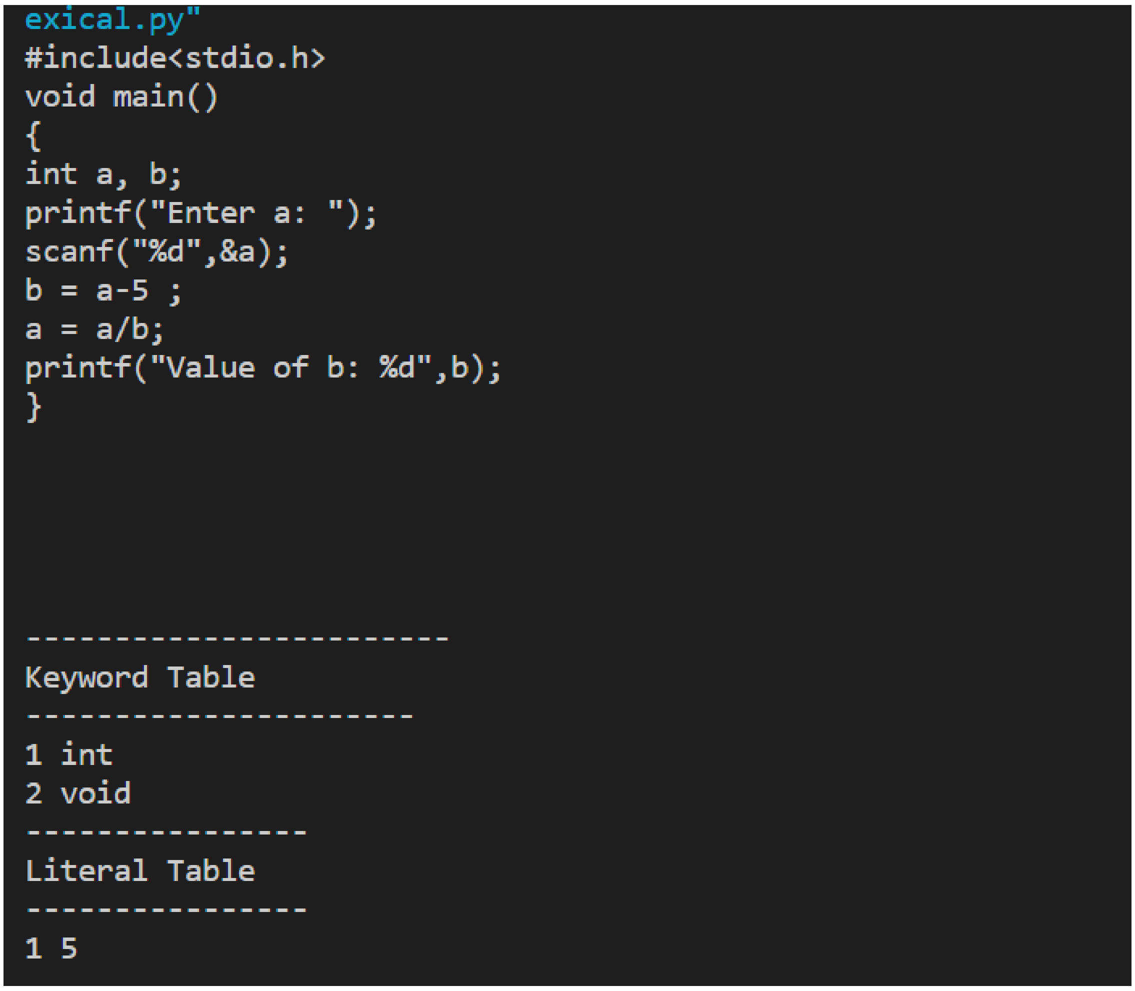
File used for reading the C program



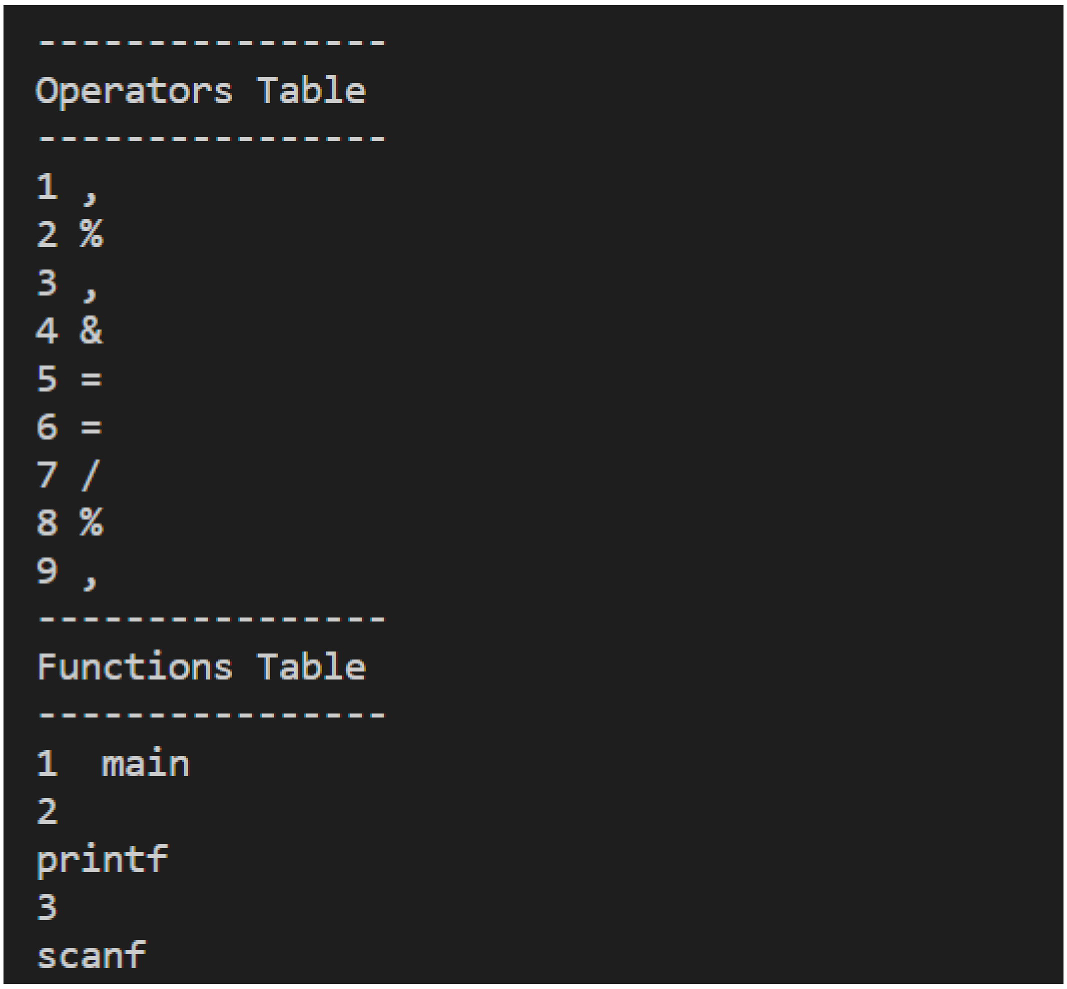
samplec.txt

```
#include<stdio.h>
void main()
{
int a, b;
printf("Enter a: ");
scanf("%d",&a);
b = a-5 ;
a = a/b;
printf("Value of b: %d",b);
}
```

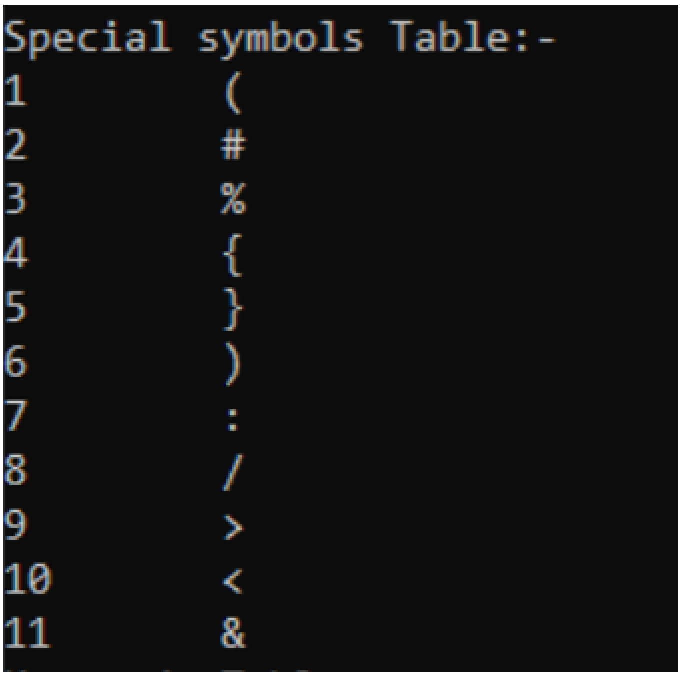
Output:



```
exical.py"
#include<stdio.h>
void main()
{
int a, b;
printf("Enter a: ");
scanf("%d",&a);
b = a-5 ;
a = a/b;
printf("Value of b: %d",b);
}




------------------------
Keyword Table
----------------------
1 int
2 void
----------------
Literal Table
----------------
1 5
```



```
----------------
Operators Table
----------------
1 ,
2 %
3 ,
4 &
5 =
6 =
7 /
8 %
9 ,
----------------
Functions Table
----------------
1  main
2
printf
3
scanf
```



```
Special symbols Table:-
1         (
2         #
3         %
4         {
5         }
6         )
7         :
8         /
9         >
10        <
11        &
```

Conclusion: In this experiment , the concept of regular expressions was well understood and we have implemented Lexical Analyzer using that concept

Date: _____________ **Signature of faculty in-charge**