Aayush Shah
19BCE245
17 August 2022

Compiler Construction

Practical 1

To implement lexical analyse to recognize all distinct token classes

• **Code :**

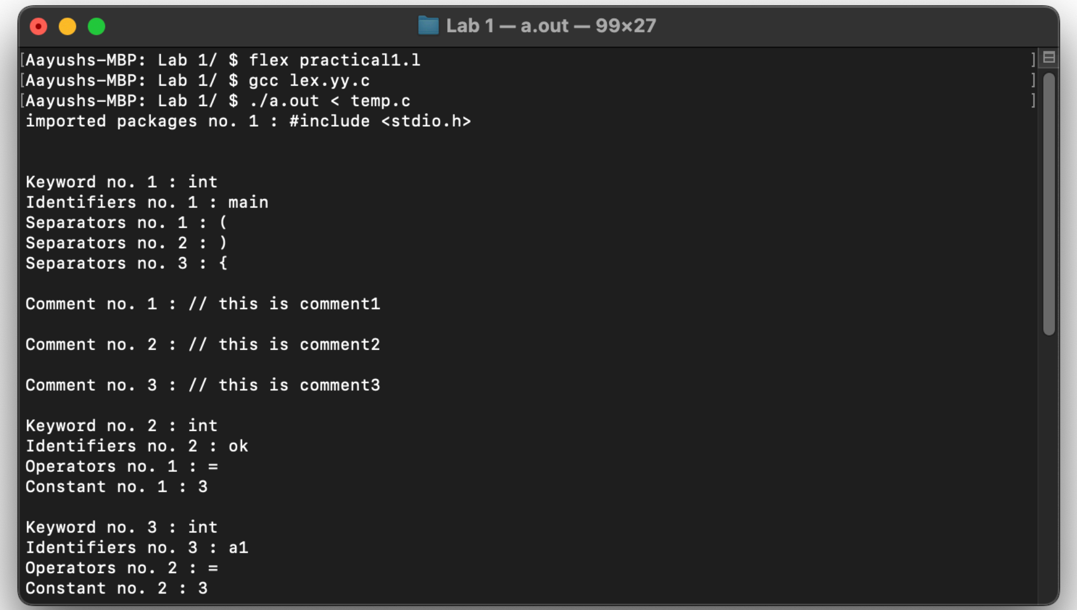
practical1.l

```
/*** 19BCE245 Aayush Shah ***/
/*** To implement lexical analyse to recognize all
distinct token classes ***/

/*** Definition Section ***/
%{
    #include<stdio.h>
    int keywords=0;
    int identifiers=0;
    int separators=0;
    int operators=0;
    int constants=0;
    int comments=0;
    int tokens=0;
    int packages=0;
    int mul_comments=0;
%}

/*** Ruel Section ***/
%%
"#"(.)*   {tokens++;packages++;printf("imported packages
no. %d : %s\n", packages, yytext);}
"auto"|"else"|"long"|"switch"|"break"|"enum"|"register"|"
typedef"|"case"|"extern"|"return"|"union"|"char"|"float"|
"short"|"unsigned"|"const"|"for"|"signed"|"void"|"continu
e"|"goto"|"sizeof"|"volatile"|"default"|"if"|"static"|"wh
ile"|"do"|"int"|"struct"|"_Packed"|"double"    {tokens+
+;keywords++;printf("Keyword no. %d : %s\n",
keywords,yytext);}
([_a-zA-Z][0-9]*)+ {identifiers++;printf("Identifiers no.
%d : %s\n", identifiers, yytext);}
"{"|"("|"}"|")"    {tokens++;separators+
+;printf("Separators no. %d : %s\n", separators,
yytext);}
[+*/><=&^]    {tokens++;operators++;printf("Operators no.
%d : %s\n", operators, yytext);}
[0-9]+    {tokens++;constants++;printf("Constant no. %d :
%s\n", constants, yytext);}
"//"(.)* {tokens++;comments++;printf("Comment no. %d :
%s\n", comments, yytext);}
"/*"(.)"*/" {tokens++;mul_comments++;printf("Multiline
Comment no. %d : %s\n", mul_comments, yytext);}
. ;
%%

/*** Code Section ***/
int yywrap(){
     return 0;
}
int main(){
     yylex();
     printf("\n total no. of token = %d\n", tokens);
     return 0;
}

/*** multiline comment, float constant, character
constant, string constant, symbols***/
```

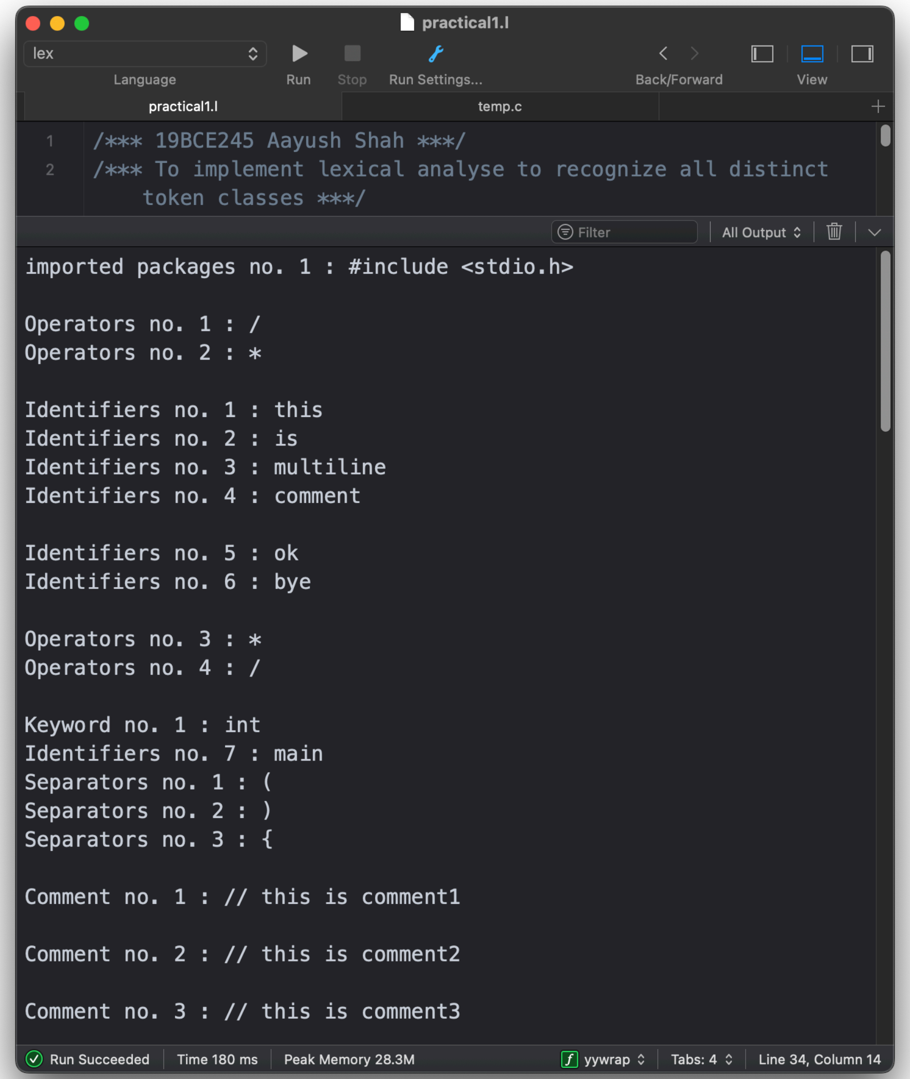
temp.c

```
#include <stdio.h>
/*
this is multiline comment
ok bye
*/
int main() {
    // this is comment1
    // this is comment2
    // this is comment3
    int ok = 3;
    int a1 = 3;
    float bye = 1.0;
    while(ok>=0){
         ok-=1;
    };
    return 0;
}
/*
this is multiline comment no.2
ok bye
*/

/*
this is multiline comment no.3
ok bye
*/


/*
this is multiline comment no.4
ok bye
*/
```

• **Output :**


Lab 1 — a.out — 99×27

```
Aayushs-MBP: Lab 1/ $ flex practical1.l
Aayushs-MBP: Lab 1/ $ gcc lex.yy.c
Aayushs-MBP: Lab 1/ $ ./a.out < temp.c
imported packages no. 1 : #include <stdio.h>


Keyword no. 1 : int
Identifiers no. 1 : main
Separators no. 1 : (
Separators no. 2 : )
Separators no. 3 : {

Comment no. 1 : // this is comment1

Comment no. 2 : // this is comment2

Comment no. 3 : // this is comment3

Keyword no. 2 : int
Identifiers no. 2 : ok
Operators no. 1 : =
Constant no. 1 : 3

Keyword no. 3 : int
Identifiers no. 3 : a1
Operators no. 2 : =
Constant no. 2 : 3
```



```
/*** 19BCE245 Aayush Shah ***/
/*** To implement lexical analyse to recognize all distinct
    token classes ***/
```

```
imported packages no. 1 : #include <stdio.h>

Operators no. 1 : /
Operators no. 2 : *

Identifiers no. 1 : this
Identifiers no. 2 : is
Identifiers no. 3 : multiline
Identifiers no. 4 : comment

Identifiers no. 5 : ok
Identifiers no. 6 : bye

Operators no. 3 : *
Operators no. 4 : /

Keyword no. 1 : int
Identifiers no. 7 : main
Separators no. 1 : (
Separators no. 2 : )
Separators no. 3 : {

Comment no. 1 : // this is comment1

Comment no. 2 : // this is comment2

Comment no. 3 : // this is comment3
```

• **Full Output :**

```
imported packages no. 1 : #include <stdio.h>

Operators no. 1 : /
Operators no. 2 : *

Identifiers no. 1 : this
Identifiers no. 2 : is
Identifiers no. 3 : multiline
Identifiers no. 4 : comment

Identifiers no. 5 : ok
Identifiers no. 6 : bye
```

```
Operators no. 3 : *
Operators no. 4 : /

Keyword no. 1 : int
Identifiers no. 7 : main
Separators no. 1 : (
Separators no. 2 : )
Separators no. 3 : {

Comment no. 1 : // this is comment1

Comment no. 2 : // this is comment2

Comment no. 3 : // this is comment3

Keyword no. 2 : int
Identifiers no. 8 : ok
Operators no. 5 : =
Constant no. 1 : 3

Keyword no. 3 : int
Identifiers no. 9 : a1
Operators no. 6 : =
Constant no. 2 : 3

Keyword no. 4 : float
Identifiers no. 10 : bye
Operators no. 7 : =
Constant no. 3 : 1
Constant no. 4 : 0

Keyword no. 5 : while
Separators no. 4 : (
Identifiers no. 11 : ok
Operators no. 8 : >
Operators no. 9 : =
Constant no. 5 : 0
Separators no. 5 : )
Separators no. 6 : {

Identifiers no. 12 : ok
Operators no. 10 : =
Constant no. 6 : 1

Separators no. 7 : }

Keyword no. 6 : return
Constant no. 7 : 0

Separators no. 8 : }

Operators no. 11 : /
Operators no. 12 : *

Identifiers no. 13 : this
Identifiers no. 14 : is
Identifiers no. 15 : multiline
Identifiers no. 16 : comment
Identifiers no. 17 : no
Constant no. 8 : 2
Operators no. 13 : *
Constant no. 9 : 3

Identifiers no. 18 : ok
Identifiers no. 19 : bye

Operators no. 14 : *
Operators no. 15 : /


Operators no. 16 : /
Operators no. 17 : *

Identifiers no. 20 : this
Identifiers no. 21 : is
Identifiers no. 22 : multiline
Identifiers no. 23 : comment
Identifiers no. 24 : no
Constant no. 10 : 3

Identifiers no. 25 : ok
Identifiers no. 26 : bye

Operators no. 18 : *
Operators no. 19 : /
Operators no. 20 : /
Operators no. 21 : *

Identifiers no. 27 : this
Identifiers no. 28 : is
Identifiers no. 29 : multiline
Identifiers no. 30 : comment
Identifiers no. 31 : no
Constant no. 11 : 4

Identifiers no. 32 : ok
Identifiers no. 33 : bye

Operators no. 22 : *
Operators no. 23 : /


 total no. of token = 52
```