

Compiler Design Lab

Name: A.V.K Sai Surya

Roll num:CH.EN.U4CSE22008

Exercise 1

1. Aim : Write a Lex program to count number of lines, spaces and etc.

Code :

```
Pgm1.l      x      Pgm2.l      x
1 /* DESCRIPTION/DEFINITION SECTION */
2 %{
3 #include<stdio.h>
4 int lc=0,sc=0,tc=0,ch=0,wc=0;          // GLOBAL VARIABLES
5 %}
6
7
8 %%
9 [\n] { lc++; ch+=yyleng;}
10 [ \t] { sc++; ch+=yyleng;}
11 [^\t] { tc++; ch+=yyleng;}
12 [^\t\n ]+ { wc++; ch+=yyleng;}
13 %%
14
15 int yywrap(){ return 1; }
16 /*      After inputting press ctrl+d      */
17
18 // MAIN FUNCTION
19 int main(){
20     printf("Enter the Sentence : ");
21     yylex();
22     printf("Number of lines : %d\n",lc);
23     printf("Number of spaces : %d\n",sc);
24     printf("Number of tabs, words, charc : %d , %d , %d\n",tc,wc,ch);
25
26     return 0;
27 }
```

Output :

```
asecomputerlab@ase-computer-lab:~$ lex Pgm1.l
asecomputerlab@ase-computer-lab:~$ cc lex.yy.c -lfl
asecomputerlab@ase-computer-lab:~$ ./a.out
Enter the Sentence : hello everyone i am rohan
new things are meant to be learnt
jjshlkdjh as
jsdkjhas

Number of lines : 6
Number of spaces : 12
Number of tabs, words, charc : 1 , 14 , 85
```

2. Aim : Write a Lex program to count number of words in given sentence.

Code :

```
1 /*lex program to count number of words*/
2 %{
3 #include<stdio.h>
4 #include<string.h>
5 int i = 0;
6 %}
7
8 /* Rules Section*/
9 %%
10 ([a-zA-Z0-9])*    {i++;} /* Rule for counting
11                      number of words*/
12
13 "\n" {printf("%d\n", i); i = 0;}
14 %%
15
16 int yywrap(void){}
17
18 int main()
19 {
20     // The function that starts the analysis
21     yylex();
22
23     return 0;
24 }
```

Output :

```
asecomputerlab@ase-computer-lab:~$ lex Pgm2.l
asecomputerlab@ase-computer-lab:~$ cc lex.yy.c -lfl
asecomputerlab@ase-computer-lab:~$ ./a.out
hi rohan
2
hello
1
```

3. Aim : Write a Lex program to check whether the given number is even or odd

Code :

```
1 /*Lex program to take check whether
2 the given number is even or odd */
3
4 %{
5 #include<stdio.h>
6 int i;
7 %}
8
9 %%
10
11 [0-9]+      {i=atol(yytext);
12              if(i%2==0)
13                  printf("Even");
14              else
15                  printf("Odd");}
16 %%
17
18 int yywrap(){}
19
20 /* Driver code */
21 int main()
22 {
23
24     yylex();
25     return 0;
26 }
```

Output :

```
asecomputerlab@ase-computer-lab:~$ lex Pgm3.l
asecomputerlab@ase-computer-lab:~$ cc lex.yy.c -lfl
asecomputerlab@ase-computer-lab:~$ ./a.out
44
Even
49
Odd
```

4. Aim : Write a Lex program to count the positive numbers, negative numbers and fractions.

Code :

```
1 /* Lex program to Count the Positive numbers,
2    - Negative numbers and Fractions */
3
4 %{
5     /* Definition section */
6     int postiveno=0;
7     int negtiveno=0;
8     int positivefractions=0;
9     int negativefractions=0;
10 %}
11
12 /* Rule Section */
13 DIGIT [0-9]
14 %%
15
16 \+?[DIGIT]+          postiveno++;
17 -[DIGIT]+           negtiveno++;
18
19 \+?[DIGIT]*\.[DIGIT]+ positivefractions++;
20 -[DIGIT]*\.[DIGIT]+  negativefractions++;
21 . ;
22 %%
23
24 // driver code
25 int main()
26 {
27     yylex();
28     printf("\nNo. of positive numbers: %d", postiveno);
29     printf("\nNo. of Negative numbers: %d", negtiveno);
30     printf("\nNo. of Positive numbers in fractions: %d", positivefractions);
31     printf("\nNo. of Negative numbers in fractions: %d\n", negativefractions);
32     return 0;
33 }
```

Output :

```
asecomputerlab@ase-computer-lab:~$ lex Pgm4.l
asecomputerlab@ase-computer-lab:~$ cc lex.yy.c -lfl
asecomputerlab@ase-computer-lab:~$ ./a.out
2 3 -4 5 -8 10

No. of positive numbers: 4
No. of Negative numbers: 2
No. of Positive numbers in fractions: 0
No. of Negative numbers in fractions: 0
```

5. Aim : Write a Lex program to count the vowels and consonants in the given string

Code :

```
1 %{
2     int vow_count=0;
3     int const_count =0;
4 %}
5
6 %%
7 [aeiouAEIOU] {vow_count++;}
8 [a-zA-Z] {const_count++;}
9 %%
10 int yywrap(){}
11 int main()
12 {
13     printf("Enter the string of vowels and consonants:");
14     yylex();
15     printf("Number of vowels are: %d\n", vow_count);
16     printf("Number of consonants are: %d\n", const_count);
17     return 0;
18 }
19
20
21
```

Output :

```
asecomputerlab@ase-computer-lab:~$ lex Pgm5.l
asecomputerlab@ase-computer-lab:~$ cc lex.yy.c -lfl
asecomputerlab@ase-computer-lab:~$ ./a.out
Enter the string of vowels and consonants:i am keerthi rohan

Number of vowels are: 7
Number of consonants are: 8
asecomputerlab@ase-computer-lab:~$ □
```

Exercise 2

1. Aim: To implement eliminate left recursion and left factoring from the given grammar using C program.

Algorithm:

Left Factoring:

- Start the processes by getting the grammar and assigning it to the appropriate variables
- Find the common terminal and non-terminal elements and assign them in a separate grammar
- Display the new and modified grammar.

Code:

```
ex2.c
1 #include<stdio.h>
2 #include<string.h>
3 int main()
4 {
5     char gram[20],part1[20],part2[20],modifiedGram[20],newGram[20],tempGram[20];
6     int i,j=0,k=0,l=0,pos;
7     printf("Enter Production : A->");
8     gets(gram);
9     for(i=0;gram[i]!='|';i++,j++)
10         part1[j]=gram[i];
11     part1[j]='\0';
12     for(j=++i,i=0;gram[j]!='\0';j++,i++)
13         part2[i]=gram[j];
14     part2[i]='\0';
15     for(i=0;i<strlen(part1)||i<strlen(part2);i++)
16     {
17         if(part1[i]==part2[i])
18         {
19             modifiedGram[k]=part1[i];
20             k++;
21             pos=i+1;
22         }
23     }
24     for(i=pos,j=0;part1[i]!='\0';i++,j++){
25         newGram[j]=part1[i];
26     }
27     newGram[j++]='|';
28     for(i=pos;part2[i]!='\0';i++,j++){
29         newGram[j]=part2[i];
30     }
31     modifiedGram[k]='X';
32     modifiedGram[++k]='\0';
33     newGram[j]='\0';
34     printf("\n A->%s",modifiedGram);
35     printf("\n X->%s\n",newGram);
36 }
37
```

Output:

```

asecomputerlab@ase-computer-lab:~/EX2$ gcc ex2.c
ex2.c: In function 'main':
ex2.c:8:2: warning: implicit declaration of function 'gets'; did
you mean 'fgets'? [-Wimplicit-function-declaration]
   8 |     gets(gram);
     |     ^~~~~
     |     fgets
/usr/bin/ld: /tmp/ccGI1m0L.o: in function `main':
ex2.c:(.text+0x5e): warning: the `gets' function is dangerous and
should not be used.
asecomputerlab@ase-computer-lab:~/EX2$ ./a.out
Enter Production : A->aE+bcD|aE+eIT

A->aE+X
X->bcD|eIT

```

Left recursion:

2. Aim: To implement left recursion using C.

Algorithm:

- Start the processes by getting the grammar and assigning it to the appropriate variables.
- Check if the given grammar has left recursion.
- Identify the alpha and beta elements in the production.
- Print the output according to the formula to remove left recursion

Code:

```

1 #include<stdio.h>
2 #include<string.h>
3 #define SIZE 10
4 int main () {
5     char non_terminal;
6     char beta,alpha;
7     int num;
8     char production[10][SIZE];
9     int index=3; /* starting of the string following "->" */
10 printf("Enter Number of Production : ");
11 scanf("%d",&num);
12 printf("Enter the grammar as E->E-A :\n");
13 for(int i=0;i<num;i++){
14     scanf("%s",production[i]);
15 }
16 for(int i=0;i<num;i++){
17     printf("\nGRAMMAR : : : %s",production[i]);
18     non_terminal=production[i][0];
19     if(non_terminal==production[i][index]) {
20         alpha=production[i][index+1];
21         printf(" is left recursive.\n");
22         while(production[i][index]!=0 && production[i][index]!='|')
23             index++;
24         if(production[i][index]!=0) {
25             beta=production[i][index+1];
26             printf("Grammar without left recursion:\n");
27             printf("%c->%c%c\'',non_terminal,beta,non_terminal);
28             printf("\n%c\''->%c%c\''|E\'',non_terminal,alpha,non_terminal);
29         }
30     else
31         printf(" can't be reduced\n");
32     }
33     else
34         printf(" is not left recursive.\n");
35     index=3;
36 }
37 }
38

```

Output:

```
asecomputerlab@ase-computer-lab:~/EX2$ gcc ex2_left_recursion.c
asecomputerlab@ase-computer-lab:~/EX2$ ./a.out
Enter Number of Production : 2
Enter the grammar as E->E-A :
E->EA|A
A->A|B

GRAMMAR : : : E->EA|A is left recursive.
Grammar without left recursion:
E->AE'
E'->AE'|E

GRAMMAR : : : A->A|B is left recursive.
Grammar without left recursion:
A->BA'
A'->|A'|E
asecomputerlab@ase-computer-lab:~/EX2$ □
```

Result: The program to implement left factoring and left recursion has been successfully executed.

Exercise 3

Aim: To implement LL(1) parsing using C program.

Algorithm:

- 1) Read the input string.
- 2) Using predictive parsing table parse the given input using stack.
- 3) If stack [i] matches with token input string pop the token else shift it repeat the process until it reaches to \$.

Code :

```
Open  llc  Save
1#include<stdio.h>
2#include<string.h>
3#include<stdlib.h>
4
5char s[20],stack[20];
6int main()
7{
8char n[s][5]={{"tb"," "," ","tb"," "," "," ","tb"," "," ","n","n","fc"," "," ","fc"," "," ","n","*fc"," ","n","n","t"," "," ","(e)"," "," "};
9int size[s][5]={0,0,0,2,0,0,0,0,3,0,0,1,1,2,0,0,2,0,0,0,3,0,1,1,1,1,0,0,3,0,0};
10int i,j,k,n,stri,strt;
11printf("\n Enter the input string: ");
12scanf("%s",s);
13strcpy(s,s);
14n=strlen(s);
15stack[0]='$';
16stack[1]='\0';
17i=1;
18j=0;
19printf("\nStack\tinput\n");
20printf("\n\t\t\t\n");
21printf("\n");
22while((stack[i]!='$')&&(s[j]!='$'))
23{
24if(stack[i]==s[j])
25{
26i--;
27j++;
28}
29switch(stack[i])
30{
31case 'e': stri=0;
32break;
33case 'b': stri=1;
34break;
35case 't': stri=2;
36break;
37case 'f': stri=4;
38break;
39case 'c': stri=3;
40}
41switch(s[j])
42{
43case '\t': strt=0;
44break;
45case '+': strt=1;
46break;
47case '*': strt=2;
48break;
49case '(': strt=3;
50break;
51case ')': strt=4;
52break;
53case '$': strt=5;
54break;
55}
56if(n[stri][strt][0]!='\0')
57{
58printf("\nERROR");
59exit(0);
60}
61else if(n[stri][strt][0]=='n')
62i--;
63else if(n[stri][strt][0]=='t')
64stack[i]=s[j];
65else
66{
67for(k=size[stri][strt]-1;k>=0;k--)
68{
69stack[i]=n[stri][strt][k];
70i++;
71}
72i--;
73}
74for(k=0;k<=i;k++)
75printf("%c",stack[k]);
76printf("\t");
77for(k=j;k<=n;k++)
78printf("%c",s[k]);
79printf("\n ");
80}
81printf("\n SUCCESS");
82return 0;
83}
```

Output :

```
asecomputerlab@ase-computer-lab:~$ gedit ll.c
asecomputerlab@ase-computer-lab:~$ gcc ll.c
asecomputerlab@ase-computer-lab:~$ ./a.out
```

Enter the input string: i*i+i

Stack	Input
\$bt	i*i+i\$
\$bcf	i*i+i\$
\$bci	i*i+i\$
\$bcf*	*i+i\$
\$bci	i+i\$
\$b	+i\$
\$bt+	+i\$
\$bcf	i\$
\$bci	i\$
\$b	\$