Compiler Design Lab

Name: Sreeram Rohith Roll num:CH.EN.U4CSE22044

Exercise 1

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

   Code :

Pgm1.l

```
/* DESCRIPTION/DEFINITION SECTION */
%{
#include<stdio.h>
int lc=0,sc=0,tc=0,ch=0,wc=0;        // GLOBAL VARIABLES
%}

%%
[\n] { lc++; ch+=yyleng;}
[  \t] { sc++; ch+=yyleng;}
[^\t] { tc++; ch+=yyleng;}
[^\t\n ]+ { wc++;  ch+=yyleng;}
%%

int yywrap(){ return 1;    }
/*        After inputting press ctrl+d         */

// MAIN FUNCTION
int main(){
    printf("Enter the Sentence : ");
    yylex();
    printf("Number of lines : %d\n",lc);
    printf("Number of spaces : %d\n",sc);
    printf("Number of tabs, words, charc : %d , %d , %d\n",tc,wc,ch);

    return 0;
}
```

   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 :

```
/*lex program to count number of words*/
%{
#include<stdio.h>
#include<string.h>
int i = 0;
%}

/* Rules Section*/
%%
([a-zA-Z0-9])*    {i++;} /* Rule for counting
                          number of words*/

"\n" {printf("%d\n", i); i = 0;}
%%

int yywrap(void){}

int main()
{
    // The function that starts the analysis
    yylex();

    return 0;
}
```

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 :

```
/*Lex program to take check whether
the given number is even or odd */

%{
#include<stdio.h>
int i;
%}

%%

[0-9]+      {i=atoi(yytext);
            if(i%2==0)
                printf("Even");
            else
           printf("Odd");}
%%

int yywrap(){}

/* Driver code */
int main()
{

    yylex();
    return 0;
}
```

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 :

```
/* Lex program to Count the Positive numbers,
      - Negative numbers and Fractions  */

%{
     /* Definition section */
    int postiveno=0;
    int negtiveno=0;
    int positivefractions=0;
    int negativefractions=0;
%}

/* Rule Section */
DIGIT [0-9]
%%

\+?{DIGIT}+             postiveno++;
-{DIGIT}+               negtiveno++;

\+?{DIGIT}*\.{DIGIT}+   positivefractions++;
-{DIGIT}*\.{DIGIT}+     negativefractions++;
. ;
%%

// driver code
int main()
{
    yylex();
    printf("\nNo. of positive numbers: %d", postiveno);
    printf("\nNo. of Negative numbers: %d", negtiveno);
    printf("\nNo. of Positive numbers in fractions: %d", positivefractions);
    printf("\nNo. of Negative numbers in fractions: %d\n", negativefractions);
    return 0;
}
```

   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 :

```
%{
    int vow_count=0;
    int const_count =0;
%}

%%
[aeiouAEIOU] {vow_count++;}
[a-zA-Z] {const_count++;}
%%
int yywrap(){}
int main()
{
    printf("Enter the string of vowels and consonants:");
    yylex();
    printf("Number of vowels are:  %d\n", vow_count);
    printf("Number of consonants are:  %d\n", const_count);
    return 0;
}
```

   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

```c
#include<stdio.h>
#include<string.h>
int main()
{
        char gram[20],part1[20],part2[20],modifiedGram[20],newGram[20],tempGram[20];
        int i,j=0,k=0,l=0,pos;
        printf("Enter Production : A->");
        gets(gram);
        for(i=0;gram[i]!='|';i++,j++)
                part1[j]=gram[i];
        part1[j]='\0';
        for(j=++i,i=0;gram[j]!='\0';j++,i++)
                part2[i]=gram[j];
        part2[i]='\0';
        for(i=0;i<strlen(part1)||i<strlen(part2);i++)
        {
                if(part1[i]==part2[i])
                {
                        modifiedGram[k]=part1[i];
                        k++;

                        pos=i+1;
                }
        }
        for(i=pos,j=0;part1[i]!='\0';i++,j++){
                newGram[j]=part1[i];
        }
        newGram[j++]='|';
        for(i=pos;part2[i]!='\0';i++,j++){
                newGram[j]=part2[i];
        }
        modifiedGram[k]='X';
        modifiedGram[++k]='\0';
        newGram[j]='\0';
        printf("\n A->%s",modifiedGram);
        printf("\n X->%s\n",newGram);
}
```

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/ccGI1mol.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:

```c
#include<stdio.h>
#include<string.h>
#define SIZE 10
int main () {
char non_terminal;
char beta,alpha;
int num;
char production[10][SIZE];
int index=3; /* starting of the string following "->" */
printf("Enter Number of Production : ");
scanf("%d",&num);
printf("Enter the grammar as E->E-A :\n");
for(int i=0;i<num;i++){
scanf("%s",production[i]);
}
for(int i=0;i<num;i++){
printf("\nGRAMMAR : : : %s",production[i]);
non_terminal=production[i][0];
if(non_terminal==production[i][index]) {
alpha=production[i][index+1];
printf(" is left recursive.\n");
while(production[i][index]!=0 && production[i][index]!='|')
index++;
if(production[i][index]!=0) {
beta=production[i][index+1];
printf("Grammar without left recursion:\n");
printf("%c->%c%c\'",non_terminal,beta,non_terminal);
printf("\n%c\'->%c%c\'|E\n",non_terminal,alpha,non_terminal);
}
else
printf(" can't be reduced\n");
}
else
printf(" is not left recursive.\n");
index=3;
}
}

```

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 ll.c Save

```c
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

char s[20],stack[20];
int main()
{
char m[5][6][3]={"tb","","","tb","","","","","+tb","","","","n","n","fc","","","","fc","","","","","n","*fc","","a","n","n","(","","","(e)","",""};
int size[5][6]={2,0,0,2,0,0,0,3,0,0,1,1,2,0,0,2,0,0,0,1,3,0,1,1,1,1,0,0,3,0,0};
int i,j,k,n,str1,str2;
printf("\n Enter the input string: ");
scanf("%s",s);
strcat(s,"$");
n=strlen(s);
stack[0]='$';
stack[1]='e';
i=1;
j=0;
printf("\nStack\tInput\n");
printf("_____\n");
printf("\n");
while((stack[i]!='$')&&(s[j]!='$'))
{
if(stack[i]==s[j])
{
i--;
j++;
}
switch(stack[i])
{
case 'e': str1=0;
break;
case 'b': str1=1;
break;
case 't': str1=2;
break;
case 'f': str1=4;
break;
case 'c': str1=3;
```

Open ll.c

```c
}
switch(s[j])
{
case 'i': str2=0;
break;
case '+': str2=1;
break;
case '*': str2=2;
break;
case '(': str2=3;
break;
case ')': str2=4;
break;
case '$': str2=5;
break;
}
if(m[str1][str2][0]=='\0')
{
printf("\nERROR");
exit(0);
}
else if(m[str1][str2][0]=='n')
i--;
else if(m[str1][str2][0]=='i')
stack[i]='i';
else
{
for(k=size[str1][str2]-1;k>=0;k--)
{
stack[i]=m[str1][str2][k];
i++;
}
i--;
}
for(k=0;k<=i;k++)
printf("%c",stack[k]);
printf("\t");
for(k=j;k<=n;k++)
printf("%c",s[k]);
printf(" \n ");
}
printf("\n SUCCESS");
return 0;
}
```

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     $
```