

DAY-4 EXPERIMENTS

25. Write Lex program to identify the keyword or identifier

Program:

```
%{  
%}  
%%  
  
if|else|while|int|switch|for|char|double|float|break|continue {printf("it is a keyword");}  
[a-zA-Z][a-zA-Z0-9]+ {printf("\n%s is identifier",yytext);}  
%%  
  
int yywrap(){}  
  
int main()  
{  
while(yylex());  
}
```

Output:

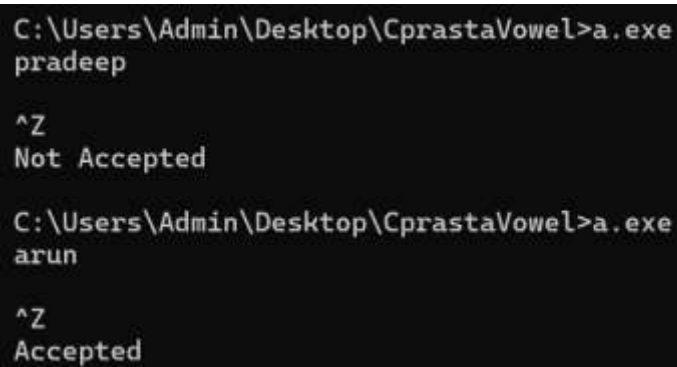
```
C:\Users\Admin\Desktop\Cpraiden>set path=C:\MinGW\bin  
  
C:\Users\Admin\Desktop\Cpraiden>gcc lex.yy.c  
  
C:\Users\Admin\Desktop\Cpraiden>a.exe  
if  
it is a keyword  
swtich  
  
swtich is identifier  
switch  
it is a keyword  
id  
  
id is identifier
```

26. Write a LEX program to accept string starting with vowel

Program:

```
%{  
#include <stdio.h>  
  
int flag = 0;  
%}  
  
%%  
[aeiouAEIOU][a-zA-Z0-9]+ { flag = 1; }  
[a-zA-Z0-9]+  
%%  
  
int main() {  
    yylex();  
    if (flag == 1) {  
        printf("Accepted\n");  
    } else {  
        printf("Not Accepted\n");  
    }  
    return 0;  
}  
  
int yywrap() {  
    return 1;  
}
```

Output:



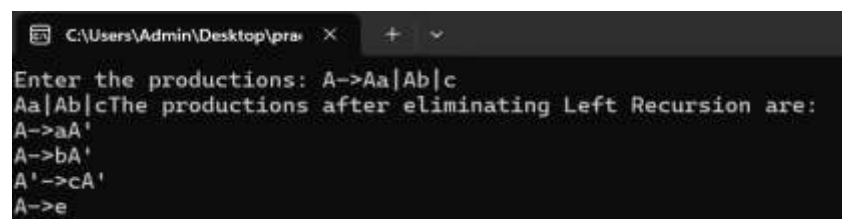
```
C:\Users\Admin\Desktop\CprastaVowel>a.exe  
pradeep  
  
^Z  
Not Accepted  
  
C:\Users\Admin\Desktop\CprastaVowel>a.exe  
arun  
  
^Z  
Accepted
```

27. Implement a C program to eliminate left recursion

Program:

```
#include<stdio.h>
#include<string.h>
int main() {
    char input[100],l[50],r[50],temp[10],tempprod[20],productions[25][50];
    int i=0,j=0,flag=0,consumed=0;
    printf("Enter the productions: ");
    scanf("%1s->%s",l,r);
    printf("%s",r);
    while(sscanf(r+consumed,"%[^]s",temp) == 1 && consumed <= strlen(r)) {
        if(temp[0] == l[0]) {
            flag = 1;
            sprintf(productions[i++],"%s->%s%s'\0",l,temp+1,l);
        }
        else
            sprintf(productions[i++],"%s'->%s%s'\0",l,temp,l);
        consumed += strlen(temp)+1;
    }
    if(flag == 1) {
        sprintf(productions[i++],"%s->e\0",l);
        printf("The productions after eliminating Left Recursion are:\n");
        for(j=0;j<i;j++)
            printf("%s\n",productions[j]);
    }
    else printf("The Given Grammar has no Left Recursion");
}
```

output:



```
C:\Users\Admin\Desktop\pra X + v
Enter the productions: A->Aa|Ab|c
Aa|Ab|cThe productions after eliminating Left Recursion are:
A->aA'
A->bA'
A'->cA'
A->e
```

28. Implement a C program to eliminate left factoring.

Program:

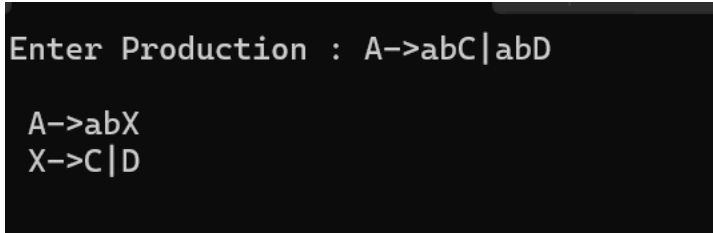
```
#include<stdio.h>
#include<string.h>
int main()
{
    char gram[20],part1[20],part2[20],modifiedGram[20],newGram[20],tempGram[20];
    int i,j=0,k=0,l=0,pos;
    printf("Enter Production : A->");
    gets(gram);
    for(i=0;gram[i]!='|';i++,j++)
        part1[j]=gram[i];
    part1[j]='\0';
    for(j=++i,i=0;gram[j]!='\0';j++,i++)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



```

Enter Production : A->abC|abD

A->abX
X->C|D

```

29. Implement a C program to perform symbol table operations.

Program:

```

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int cnt=0;
struct symtab
{
    char label[20];
    int addr;
}
sy[50];
void insert();
int search(char *);
void display();
void modify();
int main()
{
    int ch,val;
    char lab[10];

```

```

do
{
    printf("\n1.insert\n2.display\n3.search\n4.modify\n5.exit\n");
    scanf("%d",&ch);
    switch(ch)
    {
        case 1:
            insert();
            break;
        case 2:
            display();
            break;
        case 3:
            printf("enter the label");
            scanf("%s",lab);
            val=search(lab);
            if(val==1)
                printf("label is found");
            else
                printf("label is not found");
            break;
        case 4:
            modify();
            break;
        case 5:
            exit(0);
            break;
    }
}while(ch<5);
}

void insert()
{

```

```

int val;

    char lab[10];
    int symbol;
    printf("enter the label");
    scanf("%s",lab);
    val=search(lab);
    if(val==1)
    printf("duplicate symbol");
    else
    {
        strcpy(sy[cnt].label,lab);
        printf("enter the address");
        scanf("%d",&sy[cnt].addr);
        cnt++;
    }
}

int search(char *s)
{
    int flag=0,i; for(i=0;i<cnt;i++)
    {
        if(strcmp(sy[i].label,s)==0)
        flag=1;
    }
    return flag;
}

void modify()
{
    int val,ad,i;
    char lab[10];
    printf("enter the labe:");
    scanf("%s",lab);
    val=search(lab);

```

```

    if(val==0)
    printf("no such symbol");
    else
    {
        printf("label is found \n");
        printf("enter the address");
        scanf("%d",&ad);
        for(i=0;i<cnt;i++)
        {
            if(strcmp(sy[i].label,lab)==0)
            sy[i].addr=ad;
        }
    }
}

void display()
{
    int i;
    for(i=0;i<cnt;i++)
        printf("%s\t%d\n",sy[i].label,sy[i].addr);
}

```

Output:


```
3.search
4.modify
5.exit
1
enter the label lable 1
enter the address
1.insert
2.display
3.search
4.modify
5.exit
1
enter the labellable2
enter the address200
```

```
1.insert
2.display
3.search
4.modify
5.exit
2
lable 1
lable2 200
```

```
1.insert
2.display
3.search
4.modify
5.exit
```

30. Develop a lexical Analyzer to test whether a given identifier is valid or not.

Program:

```
%{  
#include <stdio.h>  
%}  
  
%%  
[a-zA-Z][a-zA-Z0-9_]* { printf("%s: Identifier\n", yytext); }  
[^a-zA-Z0-9_ \t\n]+ { printf("%s: Not an Identifier\n", yytext); }  
[ \t\n]+  
. { printf("%s: Not an Identifier\n", yytext); }  
%%  
  
int main(void) {  
    yylex();  
    return 0;  
}  
  
int yywrap() {  
    return 1;  
}
```

Output:

```
C:\Users\Admin\Desktop\Cpraidentifier>flex identifier.l.txt  
C:\Users\Admin\Desktop\Cpraidentifier>set path=C:\MinGW\bin  
C:\Users\Admin\Desktop\Cpraidentifier>gcc lex.yy.c  
C:\Users\Admin\Desktop\Cpraidentifier>a.exe  
id  
id: Identifier  
A  
A: Identifier  
123  
1: Not an Identifier  
2: Not an Identifier  
3: Not an Identifier
```

31.Design a lexical Analyzer to validate operators to recognize the operators +,-,*,/ using regular Arithmetic operators .

Program:

```
%{
#include<stdio.h>

float op1=6,op2=7;
%}

%%

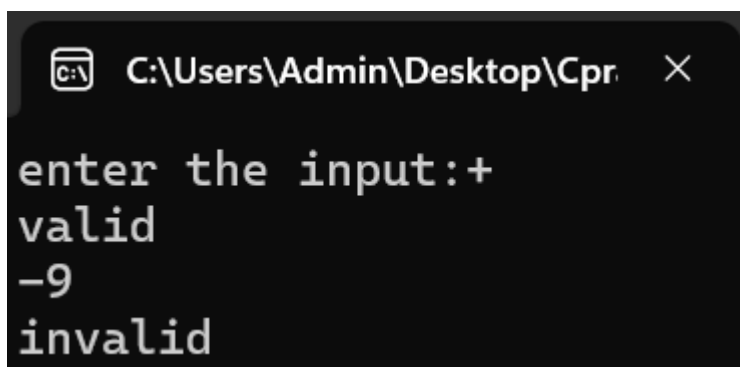
"+" {printf("sum =%lf",op1+op2);}
 "-" {printf("diff=%lf",op1-op2);}
 "*" {printf("mul=%lf",op1*op2);}
 "/" {printf("div=%lf",op1/op2);}
. {printf("enter proper operator.");}

%%

int yywrap(){}

int main()
{
printf("enter number 1");
printf("enter number 2");
printf("Enter the Operator::");
yylex();
}
```

Output:



The screenshot shows a Windows command prompt window with the title bar 'C:\Users\Admin\Desktop\Cpr'. The window contains the following text:

```
enter the input:+
valid
-9
invalid
```