

EXPERIMENT NO 2

Aim : To write Lexical analyzer specification for Pascal programming language.

Code :

-> addition.pas

```
Program Lesson1_Program3;  
Var Num1,Num2,Sum:Integer;  
Begin { no semicolon}  
Write('Input number 1:');  
Readln(Num1);  
Writeln('Input number 2:');  
Readln(Num2);  
Sum := Num1 + Num2 + 2; { addition}  
Writeln(Sum);  
Readln;  
End.
```

->lexical.l

```
% {  
#include<stdio.h>  
#include<string.h>  
char ch_arr[100][100];  
int n=0;  
int check(char c[100])  
{  
    for(int i=0;i<n;i++)
```

```

    {
        if(strcmp(ch_arr[i],c)==0)
            return(0);
    }
    return(1);
}

```

```
int get_id(char c[100])
```

```

{
    for(int i=0;i<n;i++)
    {
        if(strcmp(ch_arr[i],c)==0)
            return(i);
    }

}

% }

```

```
keyword  (Program[^A-Za-z0-9_])|(Var[^A-Za-z0-9_])|(Integer[^A-Za-z0-9_])|(End[^A-Za-z0-9_])|(Begin[^A-Za-z0-9_])
```

```
seperator (\;)
```

```
symbols \+|\-|\*|[/\]\.\|,|(|\)|\:=|\^|\@|\%|\\|(\:)
```

```
comment (\{[A-Za-z0-9_ ]*\})
```

```
string_identifier ('[A-Za-z0-9_ :][A-Za-z0-9_ :]*')
```

```
identifiers ([A-Za-z_][A-Za-z0-9_ ]*)
```

```
constant ([0-9][0-9]*)
```

%%

```
{keyword} { printf(" Keyword [%s] \n",yytext);}
{seperator} {printf("Seperator [;] \n");}
{symbols} {printf(" Symbols [%s] \n",yytext);}
{comment} {printf(" [Comment ignored] \n");}
{string_identifier} {printf(" Character String [%s] \n",yytext);}
{identifiers} {
    if(check(yytext))
    {
        strcpy(ch_arr[n++],yytext);
        printf(" Identifier [%s] ID [%d] \n",yytext,n-1);
    }
    else
    {
        int id=get_id(yytext);
        printf(" Identifier [%s] ID [%d] \n",yytext,id);
    }

}

{constant} {printf(" Constant [%s] \n",yytext);}
```

%%

```

int main(int argc, char *argv[]) {

    FILE *fptr;

    char c;

    printf("-----PROGRAM-----\n");

    fptr = fopen("addition.pas", "r");

    c = fgetc(fptr);

    while (c != EOF)

    {

        printf ("%c", c);

        c = fgetc(fptr);

    }


    fclose(fptr);

    printf("\n");

    printf("-----ANALYSIS-----\n");

    yyin = fopen("addition.pas", "r");

    yylex();

    fclose(yyin);

    printf("\n-----IDENTIFIER-----\n");

    for(int i=0;i<n;i++)

    {

        printf("%s\n",ch_arr[i]);

    }

}

```

```
}
```

```
/*****OUTPUT*****/
```

```
root@coder:/coder/mnt/Rstar/SPCC_PY_CODES# lex lexical.l
```

```
root@coder:/coder/mnt/Rstar/SPCC_PY_CODES# gcc lex.yy.c -ll
```

```
root@coder:/coder/mnt/Rstar/SPCC_PY_CODES# ./a.out
```

```
-----PROGRAM-----
```

```
Program Lesson1_Program3;
```

```
Var Num1,Num2,Sum:Integer;
```

```
Begin {no semicolon}
```

```
Write('Input number 1:');
```

```
Readln(Num1);
```

```
Writeln('Input number 2:');
```

```
Readln(Num2);
```

```
Sum := Num1 + Num2 + 2; {addition}
```

```
Writeln(Sum);
```

```
Readln;
```

```
End.
```

```
-----ANALYSIS-----
```

```
Keyword [Program ]
```

```
Identifier [Lesson1_Program3] ID [0]
```

```
Seperator [;]
```

```
Keyword [Var ]
```

```
Identifier [Num1] ID [1]
```

```
Symbols [,]
```

Identifier [Num2] ID [2]

Symbols [,]

Identifier [Sum] ID [3]

Symbols [:]

Keyword [Integer;]

Keyword [Begin]

[Comment ignored]

Identifier [Write] ID [4]

Symbols [(]

Character String ['Input number 1:']

Symbols [)]

Seperator [;]

Identifier [Readln] ID [5]

Symbols [(]

Identifier [Num1] ID [1]

Symbols [)]

Seperator [;]

Identifier [Writeln] ID [6]

Symbols [(]

Character String ['Input number 2:']

Symbols [)]

Seperator [;]

Identifier [Readln] ID [5]

Symbols [(]

Identifier [Num2] ID [2]

Symbols [)]

Seperator [;]

Identifier [Sum] ID [3]

Symbols [:=]

Identifier [Num1] ID [1]

Symbols [+]

Identifier [Num2] ID [2]

Symbols [+]

Constant [2]

Seperator [;]

[Comment ignored]

Identifier [Writeln] ID [6]

Symbols [(]

Identifier [Sum] ID [3]

Symbols [)]

Seperator [;]

Identifier [Readln] ID [5]

Seperator [;]

Keyword [End.]

-----IDENTIFIER-----

Lesson1_Program3

Num1

Num2

Sum

Write

Readln

Writeln

root@coder:/coder/mnt/Rstar/SPCC_PY_CODES#

*/

PostLab Assignment :

Write a Lex program that converts a file to "Pig Latin". Specifically, assume the file is a sequence of words(groups of letters) separated by white space. Every time you encounter a word:

1. If the first letter is a consonant, move it to the end of the word and then add "ay" as a suffix.
2. If the first letter is a vowel, just add "ay" to the end of the vowel.

% {

#include<stdio.h>

#include<string.h>

FILE *fptrw;

% }

ws ([]|[\n])


```
word ([a-zA-Z][a-zA-Z]*)
```

```
% %
```

```
{ws} {
```

```
printf("%s",yytext);
```

```
fprintf(fpw,"%s",yytext);
```

```
}
```

```
{word} {
```

```
if(yytext[0]=='a' || yytext[0]=='e' || yytext[0]=='i' || yytext[0]=='o' || yytext[0]=='u'
```

```
|| yytext[0]=='A' || yytext[0]=='E' || yytext[0]=='I' || yytext[0]=='O' || yytext[0]=='U'
```

```
)
```

```
{
```

```
char final[100];
```

```
final[0]=yytext[0];
```

```
final[1]='a';
```

```
final[2]='y';
```

```
int i=3,j=1;
```

```
while(yytext[j]!='\0')
```

```
{
```

```
final[i++]=yytext[j++];
```

```
}
```

```
final[i]='\0';
```

```
    printf("%s",final);
    fprintf(fpw,"%s ",final);
}
else
{
    char temp[200];
    char first;
    first=yytext[0];
    int j=1,i=0;
    while(yytext[j]!='\0')
    {
        temp[i++]=yytext[j++];
    }
    temp[i++]=first;
    temp[i++]='a';
    temp[i++]='y';
    temp[i]='\0';

    printf("%s",temp);
    fprintf(fpw,"%s ",temp);
}
}
```

%%

```
int main(int argc, char *argv[]) {  
    FILE *fptr;  
    char c;  
    printf("-----TEXT-----\n");  
    fptr = fopen("postlab.txt", "r");  
    fptrw = fopen("program.txt", "w");  
    c = fgetc(fptr);  
    while (c != EOF)  
    {  
        printf ("%c", c);  
        c = fgetc(fptr);  
    }  
  
    fclose(fptr);  
    printf("\n");  
    printf("-----ANALYSIS-----\n");  
    yyin = fopen("postlab.txt", "r");  
    yylex();  
    fclose(yyin);  
    printf("\n");  
}
```

Output:

```
// root@coder:/coder/mnt/Rstar/SPCC_PY_CODES# lex postlab.l
// root@coder:/coder/mnt/Rstar/SPCC_PY_CODES# gcc lex.yy.c -ll
// root@coder:/coder/mnt/Rstar/SPCC_PY_CODES# ./a.out
// -----TEXT-----
// Hi    Answer me
// Rstar Here
// -----ANALYSIS-----
// iHay    Aaynswer emay
// starRay ereHay
// root@coder:/coder/mnt/Rstar/SPCC_PY_CODES#
```

Conclusion : From this experiment I learn about lex programming and how to design a lexical analyzer using C language and lex. This knowledge will be helpful in designing compiler.