EXPERIMENT NO 3

Date of performance :
Date of submission :

Aim: To design a lexical analyzer for a language whose grammar is known.

Software Used: c,c++,java language

Theory:

Lexical analyzer is the first interface between the source program and the compiler. It reads source program character by character from a file or read in bulk from a file and store it in an array and then reading from an array character by character is really an issue. This is to be thought well as lexical anaysis and that too,get_next.char function is the most busy function. So if at all one has to gain in time, he/she has to think on this issue. It expresses the linear lexical structure. It also recognizes the logically cohesive units, so called as lexemes or tokens. The tokens could be keywords, identifiers, operators, labels, punctuations, symbols etc.
Activity of lexical analyzer:
Input: source program in HLL.
Output: Stream of tokens.
Many times you have to look ahead to confirm a token.
Example:- Do 51 =1.25 (Fortran)

This is not a Do statement, however it is valid assignment. In fortran blanks are allowed in identifier and therefore Do51 is valid identifier in fortran, provided 1.25 is there (and not 1, 25). Because 1, 25 would have been then, it would have been a valid Do statement, meaning to say that, keep on doing statement 5, till 1 reaches to 25 from 1. so in such case several characters are to be scanned to take a right decision. An example in c is '+' and '++'.

Algorithm:

1. Start
2. Initialise a character array storing all the key words from c language.
3. open source file in read mode.
4. Read one character at the current position of file pointer.
5. store the character, in a character array if it is not already present.
6. call the display symbol function to display the character if it is a symbol.
7. repeat these steps till end of file.
8. Rewind the file and initialize file pointer to start of file.
9. Read a character at file pointer position.
10. Check its ASCII value, if it falls with in the range of character; add it to the array of character string.
11. If the character is symbol, check the string whether it is a keyword check whether it is already present, add it to temporary array and display it with its priority.
12. If the word is include store the next string as constant in an array and the next value till end of line occurs as the value of constant.
13. If the word is data type store it in an array with flag set to indicate its data type.
14. Repeat from step 9, till end of file.
15. Display the array containing constants and their values as literals and variables and their data types as identifiers.
16. Close the source file.
17. Stop.

Problem definition:-

1) int max(int a,int b)
{

```
                If(a<b)
                Return a;
                Else
                Return b;
        }
    2)
    #include<stdio.h>
    #include<conio.h>
    #define pi=3.142
    main()
    {
    int a,b,c;
    float bc=0;
    char p;
    clrscr();
    printf("enter a number");
    scanf("%d%d",&a,&b);
    c=a+b+bc
    printf("%d",c);
    prinf("enter a character");
    printf("%c",p);
    printf("character is",p);
    getch();
    return 0;
```

PROGRAM TO IMPLEMENT LEXICAL ANALYZER

LEX Program <x1,e>

```
%{
#include"y.tab.h"
extern int yylval;
%}
%%
[0-9]+  {yylval=ato(yytext);
         return NUM;
         }
return yytext[0];
\n return 0;
%%
int yywrap();
{
return 1;
}
```

YACC Program <x1,y>

```
%{
#include<stdio.h>
%}
%token A NUM
%%
state: A'='E
       |E     {print("\n The result=%d\n",$1);}
       ;
E:E'+NUM      {$S=$1+$3;}
   |NUM       {$S=$1;}
   ;
%%
extern FILE *yyin;
main()
```

```
{
 do
 {
   yyparse();
 }while(!feof(yyin));
}
yyerror(char *s)
{
 fprintf(stderr,"%s\n",s);
}
```

OUTPUT:

```
[root@aap root]# lex x1.1
[root@aap root]# yacc -d x1.y
[root@aap root]# cc lex.yy.c.y.tab.c
[root@aap root]# ./a.out
4+6
The result=10
```

LEX Program <anbn.I>

```
%{
#include"y.tab.h"
%}
%%
a  {return A;}
b  {return B;}
.   {return(yytext[0]);}
\n return ('\n');
%%
int yywrap()
{
 return 1;
}
```

YACC Program <anbn.y>

```
%{
%}
%token A B
%%
statement:anbn'\n' {printf("\n Its a valid string!!!");
                    return 0;}
anbn:    A B
   |A anbn B
   ;
%%
main();
{
printf("\n Enter some valid string\n");
yyparse();
}
int yyerror(char *s)
{
printf("\nIt is not in anbn");
}
```

OUTPUT (run 1):

```
[root@aap root]# lex x2.1
[root@aap root]# yacc x2.y
[root@aap root]# cc lex.yy.c.y.tab.c
[root@aap root]# ./a.out

Enter some valid string
aabb
Its a valid string!!!
```

OUTPUT (run 2):

```
[root@aap root]# ./a.out

Enter some valid string
abbb
It is not in anbn
```

OUTPUT (run 3):

```
[root@aap root]# ./a.out

Enter some valid string

It is not in anbn
```

Conclusion: Lexical analyzer has been designed in c language.

SIGN AND REMARK

DATE

R1	R2	R3	R4	R5	Total (15 Marks)	Signature