DATE:

EX NO: 1 **SYMBOL TABLE CREATION**

**AIM**

To Write a C program to create a symbol table.

**ALGORITHM**

1. Start processing.
2. Declare structure for input and output files.
3. Declare File pointers for input and output files.
4. Open Input File(s) in Read mode and Open Output File(s) in write Mode.
5. Read the Intermediate File until EOF occurs.
6. If Symbol is not equal to NULL then
7. Write the Symbol Name and its address into Symbol table.
8. Close all the Files.
9. Print Symbol Table.
10. Stop processing.

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

struct intermediate

{

int addr;

char label[10];

char mnem[10];

char op[10];

}res;

struct symbol

{

char symbol[10];

int addr;

}sy;

void main()

{

FILE \*s1,\*p1;

clrscr();

s1=fopen("inter.txt","r+");

p1=fopen("symbol.txt","w");

while(!feof(s1))

{

fscanf(s1,"%d %s %s %s",&res.addr,res.label,res.mnem,res.op);

if(strcmp(res.label,"NULL")!=0)

{

strcpy(sy.symbol,res.label);

sy.addr=res.addr;

fprintf(p1,"%s\t%d\n",sy.symbol,sy.addr);

}

}

fcloseall();

printf("symbol table created");

getch();

}

**INPUT FILE:**

Inter.txt

0 NULL START 500

500 A DS 100

525 B CD 75

550 C CD 50

575 D CD 25

600 E DC 10

610 FIRST PRINT A

612 NULL READ B

613 NULL END FIRST

**OUTPUT FILE:**

SYMBOL.TXT

A 500

B 525

C 550

D 575

E 600

FIRST 610

**RESULT:**

Thus, the above program was written and the symbol table was created.

DATE: **IMPLEMENTATIONS OF SYMBOL TABLE WITH FEATURES LIKE INSERT,**

Ex.No:2  **MODIFY,SEARCH, AND DISPLAY**

**AIM**

To write a C program to implement symbol table.

**ALGORITHM**

1. Start the program for performing insert, display, search and modify option in symbol table
2. Define the structure of the Symbol Table
3. Enter the choice for performing the operations in the symbol Table.
4. If the entered choice is 1, search the symbol table for the symbol to be inserted.

If the symbol is already present, it displays "Duplicate Symbol". Else, insert the symbol and the corresponding address in the symbol table.

1. If the entered choice is 2, the symbols present in the symbol table are displayed.
2. If the entered choice is 3, the symbol is searched in the symbol table. If it is not found in symbol table, it displays that it is not found.
3. If the entered choice is 4, the symbol to be modified is searched in the symbol table. The address of the label can be modified.
4. Enter choice 5 to exit the program.

**PROGRAM:**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<conio.h>

int cnt=0;

struct symtab

{

char label[20];

int addr;

}sy[50];

void insert();

int search(char \*);

void display();

void modify();

void main()

{

int ch,val;

char lab[10];

do

{

printf("\n 1.Insert\n 2.Display\n 3.Search\n 4.Modify\n 5.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];

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 label");

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:**

1.insert

2.display

3.search

4.modify

5.exit

1

enter the label A

enter the address 2000

1.insert

2.display

3.search

4.modify

5.exit

1

enter the label SUB

enter the address 3000

1.insert

2.display

3.search

4.modify

5.exit

1

enter the label MUL

enter the address 4000

1.insert

2.display

3.search

4.modify

5.exit

2

A 2000

SUB 3000

MUL 4000

1.insert

2.display

3.search

4.modify

5.exit

3

enter the label A

label is found

1.insert

2.display

3.search

4.modify

5.exit

4

enter the label A

label is found

enter the address 5000

1.insert

2.display

3.search

4.modify

5.exit

5

**RESULT:**

Thus, the above program was created and the output was verified.

**DATE: IMPLEMENTATION OF LEXICAL ANALYSER**

**EX.NO:**

**AIM:**

To Write a C program to implement a lexical analyzer

**ALGORITHM:**

1. Start processing
2. Declare file pointers for input and output files
3. Get the name of the file for which lexical analysis to be done
4. Open the source files in read mode and output file in write mode
5. Read the input file until EOF occurs and perform the lexical analysis
6. Write the result to the Output file and display the same
7. Close all the files
8. Stop Processing

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

void main()

{

FILE \*fi,\*fo,\*fop,\*fk;

int flag=0,i=1;

char c,t,a[15],ch[15],file[20];

clrscr();

printf("Enter the file name:");

scanf("%s",file);

fi=fopen(file,"r");

fo=fopen("inter.C","w");

fop=fopen("oper.C ","r");

fk=fopen("key.C","r");

c=getc(fi);

while(!feof(fi))

{

if(isalpha(c)||isdigit(c)||(c=='['||c==']'||c=='.'==1))

fputc(c,fo);

else

{

if(c=='n')

fprintf(fo,"\t$\t");

else

fprintf(fo,"\t%c\t",c);

}

c=getc(fi);

}

fclose(fi);

fclose(fo);

fi=fopen("inter.C","r");

printf("\t\t LEXICAL ANALYSIS \n");

fscanf(fi,"%s",a);

printf("\nLine: %d\n",i++);

while(!feof(fi))

{

if((strcmp(a,"$")==0))

{

printf("\n line:%d\n",i++);

fscanf(fi,"%s",a);

}

fscanf(fop,"%s",ch);

while(!feof(fop))

{

if(strcmp(ch,a)==0)

{

fscanf(fop,"%s",ch);

printf("\t\t%s\t:\t%s\n",a,ch);

flag=1;

}

fscanf(fop,"%s",ch);

}

rewind(fop);

fscanf(fk,"%s",ch);

while(!feof(fk))

{

if(strcmp(ch,a)==0)

{

fscanf(fk,"%s",ch);

printf("\t\t%s\t:\tkeyword\n",a);

flag=1;

}

fscanf(fk,"%s",ch);

}

rewind(fk);

if(flag==0)

{

if(isdigit(a[0]))

printf("\t\t%s\t:\tconstant\n",a);

else

printf("\t\t%s\t:\tidentifier\n",a);

}

flag=0;

fscanf(fi,"%s",a);

}

getch();

}

**OPER.C:**

( openpara

) closepara

{ openbrace

} closebrace

< lesser

> greater

" doublequote

' singleqoute

: colon

; semicolon

# preprocessor

= equal

== assign

% percentage

^ bitwise

& reference

\* star

+ add

- sub

\ backslash

/ slash

**KEY.C**

int

void

main

char

if

while

for

else

printf

scanf

FILE

include

stdio.h

conio.h

iostream.h

**OUTPUT:**

Enter the file name:input.C

LEXICAL ANALYSIS

Line:1

# : preprocessor

Include : keyword

< : lesser

stdio.h : keyword

> : greater

Line:2

# : preprocessor

include : keyword

< : lesser

conio.h : keyword

> : greater

Line:3

void : keyword

main : keyword

( : open paranthesis

) : close paranthesis

Line:4

{ : open braces

Line:5

int : Keyword

a : identifier

b : identifier

c : identifier

; : semicolon

Line:6

a : identifier

= : equal

b : identifier

\* : star

c : identifier

; : semicolon

Line:7

( : open paranthesis

) : close paranthesis

; : semicolon

Line:8

} : close braces

Inter.C:

# include " stdio.h " $

# include " conio.h " $

void main ( ) $

{ $

int a = 10 , b , c ; $

a = b \* c ; $

getch ( ) ; $

} $ $

**RESULT:**

Thus, a lexical analyzer was implemented and the output was verified.

DATE:

EX NO: **IMPLEMENTATION OF LEXICAL ANALYZER USING LEX TOOL**

**AIM:**

To Write a C program to implement lexical analysis using LEX tool.

**ALGORITHM:**

1. Start the program
2. Lex program consists of three parts
   1. Declaration %%
   2. Translation Rules %%
   3. Auxiliary procedure.
3. The declaration section includes declaration of variables, main test, constants and regular definitions.
4. Translation rules of lex program are statements of the form
   1. P1 {action}
   2. P2 {action}
   3. …
   4. …
   5. Pn {action}
5. Compile the lex program to produce output file as lex.yy.c

Eg: $ lex filename.l $ cc lex.yy.c

Step 6. Stop the program.

**PROGRAM:**

%{

/\* program to recognize a c program \*/

int COMMENT=0;

int cnt=0;

%}

identifier [a-zA-Z][a-zA-Z0-9]\*

%%

#.\* { printf("\n%s is a PREPROCESSOR DIRECTIVE",yytext);}

int |

float |

char |

double |

while |

for |

do |

if |

break |

continue |

void |

switch |

case |

long |

struct |

const |

typedef |

return |

else |

goto {printf("\n\t%s is a KEYWORD",yytext);}

"/\*" {COMMENT = 1;}

"\*/" {COMMENT = 0; cnt++;}

{identifier}\( {if(!COMMENT)printf("\n\nFUNCTION\n\t%s",yytext);}

\{ {if(!COMMENT) printf("\n BLOCK BEGINS");}

\} {if(!COMMENT) printf("\n BLOCK ENDS");}

{identifier}(\[[0-9]\*\])? {if(!COMMENT) printf("\n %s IDENTIFIER",yytext);}

\".\*\" {if(!COMMENT) printf("\n\t%s is a STRING",yytext);}

[0-9]+ {if(!COMMENT) printf("\n\t%s is a NUMBER",yytext);}

\)(\;)? {if(!COMMENT) printf("\n\t");ECHO;printf("\n");}

\( ECHO;

= {if(!COMMENT)printf("\n\t%s is an ASSIGNMENT OPERATOR",yytext);}

\<= |

\>= |

\< |

== |

\> {if(!COMMENT) printf("\n\t%s is a RELATIONAL OPERATOR",yytext);}

%%

int main(int argc,char \*\*argv)

{

if(argc > 1)

{

FILE \*file;

file = fopen(argv[1],"r");

if(!file)

{

printf("could not open %s \n",argv[1]);

exit(0);

}

yyin = file;

}

yylex();

printf("\n\n Total No.Of comments are %d",cnt);

return 0;

}

int yywrap()

{

return 1;

}

**OUTPUT:**

#include<stdio.h>

#include<stdio.h> is a PREPROCESSOR DIRECTIVE

main()

FUNCTION

main(

)

{

BLOCK BEGINS

int a,b;

int is a KEYWORD

a IDENTIFIER,

b IDENTIFIER;

}

BLOCK ENDS

/\*program is over\*/

/\*The program identifies tokens in every input line\*/

^Z

Total No. Of comments are 2

**RESULT:**

Thus, lexical analysis for a C program using lex tool was implemented and verified.

DATE:

EX NO: **GENERATE YACC SPECIFICATION FOR A FEW SYNTACTIC CATEGORIES**

1. **Program to recognize a valid arithmetic expression that uses operators +,-,\* and /**

**AIM:**

To generate YACC specification to recognize a valid arithmetic expression.

**ALGORITHM:**

**PSEUDO CODE:**

**Lex:**

1. {Declaration and regular definition] Define header files to include first section

2. [translation rule] Tokens generated are used in yacc files [a-z A-Z] alphabets are returned 0-9 one or more combinations of integers

**Yacc:**

1. Accept token generated in lex part as input

2. Specify the order of procedure

3. Define rules with end points

4. Parse input string from standard input by calling yyparse() main function.

5. Print the result of any rules defined matches as arithmetic expression as valid

6. If none of the rule defined matches print arithmetic expression is invalid.

**PROGRAM:**

**arith.l**

%{

#include"y.tab.h"

#include<math.h>

extern yylval;

%}

%%

[0-9]+ {yylval=atoi(yytext);return NUM;}

[+] {return '+';}

[-] {return '-';}

[\*] {return '\*';}

[/] {return '/';}

[\t]+;

[\n] {return 0;}

{return yytext[0];

}

%%

**arith.y**

%{

#include<stdio.h>

%}

%token NUM

%left '-''+'

%right '\*''/'

%%

start: exp {printf("%d\n",$$);}

exp:exp'+'exp {$$=$1+$3;}

|exp'-'exp {$$=$1+$3;}

|exp'\*'exp {$$=$1\*$3;}

|exp'/'exp

{

if($3==0)

yyerror("error");

else

{

$$=$1/$3;

}

}

|'('exp')'{$$=$2;}

|NUM {$$=$1;}

;

%%

main()

{

printf("Enter the Expr. in terms of integers\n");

if(yyparse()==0)

printf("Valid Expression\n");

}

yywrap(){}

yyerror()

{

printf("Invalid Expression\n");

}

**OUTPUT:**

Enter the expr. In terms of integers

5+10

Valid Expression

Enter the expr. In terms of integers

6\*

Invalid Expression

**RESULT:**

Thus, a Program to recognize a valid arithmetic expression that uses operators +,-,\* and / was created and implemented.

1. **Program to recognize a valid variable which starts with a letter followed by any number of letters or digits**

**AIM:**

To generate a YACC specification to recognize a valid variable.

**ALGORITHM:**

1. {Declaration and regular definition] Define header files to include first section
2. Include header file y.tab.h
3. Define tokens
4. Declare [a-z][0-9] as s
5. Call function yyerror()
6. If error exist then print “its not a identifier” and exit
7. In main() call yyparse(), if yyparse() print “It’s a identifier”

**PROGRAM:**

Letteryacc.y

%{

#include<stdio.h>

int valid=1;

%}

%token digit letter

%%

start : letter s

s : letter s

| digit s

|

;

%%

int yyerror()

{

printf("\nIts not an identifier!\n");

valid=0;

return 0;

}

int main()

{

printf("\nEnter a name to be tested for identifier");

yyparse();

if(valid)

{

printf("\n It is an identifier!\n");

}

}

int yywrap (void )

{

return 1;

}

Letterlex.l

%{

#include "y.tab.h"

%}

%%

[a-zA-Z] return letter;

[0-9] return digit;

. return yytext[0];

\n return 0;

%%

**OUTPUT:**

Enter the name to be tested for identifier :abc

It is an Identifier!

Enter the name to be tested for identifier :6x

It is not an Identifier!

**RESULT:**

Thus a YACC specification was created to recognize a valid variable.

1. **Implementation of Calculator using LEX and YACC**

**AIM:**

To implement a calculator using LEX and YACC specification.

**ALGORITHM:**

1. {Declaration and regular definition] Define header files to include first section
2. Include header file y.tab.h
3. Declare the various functions.
4. Define the functions and implement it with function calls and define yyerror().
5. If error occurs during parsing, notify that it is a parsing error.
6. Call the functions based on the input and print the desired output.

**PROGRAM:**

**Calc.y**

%{

#include "y.tab.h"

#include <stdio.h>

#include <math.h>

#define PI 3.141592

int i,j;

%}

%union

{

double dval;

}

%token <dval> NUMBER

%token SIN COS TAN SQRT SQUARE RECI EXP MOD CUBE FACT

%left LN LOG

%left '+''-''\*''/'

%right '^'

%nonassoc NEG

%type <dval> E

%%

SL : S '\n'

| SL S '\n'

;

S : E {printf("=%g\n",$1);}

;

E : E '+' E {$$=$1+$3;

printf("Addition is");}

|E '-' E {$$=$1-$3;

printf("subtraction is");

}

|E '\*' E {$$=$1\*$3;

printf("Multiplication is");

}

|E '/' E {

if($3==0)

printf("Error! Divide by zero!!");

else

$$=$1/$3;

printf("Division is");

}

|E '^' E {$$=pow($1,$3);

printf("Power is=");

}

|SIN '(' E ')' {$$=sin($3/180\*PI);

printf("SIN is=");

}

|COS '(' E ')' {$$=cos($3/180\*PI);

printf("COS is");

}

|TAN '(' E ')' { if($3==90)

printf("Undefined (Infinity)");

else

$$=tan($3/180\*PI);

printf("TAN is");

}

|SQRT '(' E ')' {$$=sqrt($3);

printf("Square Root is");

}

|SQUARE '(' E ')' {$$=$3\*$3;

printf("square is=");

}

|EXP '(' E ')' {$$=exp($3);

printf("EXPONENTIAL is");

}

|RECI '(' E ')' {$$=1/($3);

printf("Reciprocal is");}

|CUBE '(' E ')' {$$=$3\*$3\*$3;

printf("Cube is");}

|FACT '(' E ')' {

$$=1;

for(j=1;j<=$3;j++)

$$=$$\*j;

printf("Factorial is");

}

|'(' E ')' {$$=$2;

printf("The simple number is");}

|'-' E %prec NEG {$$=-$2;

printf("The number with Negative sign is ");

}

|LOG E {$$=log($2)/log(10);

printf("LOG of base 10 is");

}

|LN E { $$=log($2);

printf("LOG with base 2 is");

}

|MOD '(' E','E')' {$$=fmod($3,$5);

}

|NUMBER

;

%%

extern FILE\*yyin;

int main()

{

do{

yyparse();

}while(!feof(yyin));

}

yyerror(char\*a)

{

fprintf(stderr,"parse error!!!");

}

int yywrap()

{

return 1;

}

**Calc.l**

%{

#include "y.tab.h"

#include<math.h>

%}

NUMBER [0-9]+|([0-9]\*"."[0-9]+)

%%

{NUMBER} {yylval.dval=atof(yytext);

return NUMBER;}

sin|SIN {return SIN;}

cos|COS {return COS;}

tan|TAN {return TAN;}

sqrt|SQRT {return SQRT;}

square|SQUARE {return SQUARE;}

exp|EXP {return EXP;}

reci|RECI {return RECI;}

cube|CUBE {return CUBE;}

fact|FACT {return FACT;}

LOG { return LOG;}

ln|LN {return LN;}

MOD|mod {return MOD;}

[\t]+ ;

\n|. return yytext[0];

%%

**OUTPUT:**

C:\FLEX WINDOWS\BISON\BIN\gcc lex.yy.c y.tab.c

C:\FLEX WINDOWS\BISON\BIN\calc.exe

MOD(4,2)

=0

sin(90)

SIN is==1

tan(90)

Undefined (Infinity)TAN is=90

cube(3)

Cube is=27

square(2)

square is==4

sqrt(25)

Square Root is=5

reci(5)

Reciprocal is=0.2

exp(10)

EXPONENTIAL is=22026.5

LOG10

LOG of base 10 is=1

ln10

LOG with base 2 is=2.30259

=5

(9)

The simple number is=9

3^2

Power is==9

3\*3

Multiplication is=9

3+4

Addition is=7

4-2

subtraction is=2

3/0

Error! Divide by zero!!Division is=3

32/2

Division is=16

fact(5)

Factorial is=120

**RESULT:**

Thus, the calculator was implemented with YACC and Lex specification and the output was verified.

DATE:

EX NO: **CONSTRUCTION OF DAG**

**AIM:**

To write a C program to construct of DAG(Directed Acyclic Graph)

**ALGORITHM:**

1. Start the program
2. Check for postfix expression and construct a syntax tree.
3. Construct the equivalent DAG of the input from the syntax tree.
4. Obtain output by giving the input between parentheses.
5. Stop the program.

**PROGRAM:**

#include<stdio.h>

void main()

{

struct da

{

int ptr,left,right;

char label;

}dag[25];

int ptr,j,change,n=0,i=0,x,y,k;

char store,\*input1,input[25],var;

clrscr();

for(i=0;i<25;i++)

{

dag[i].ptr=NULL;

dag[i].left=NULL;

dag[i].right=NULL;

dag[i].label=NULL;

}

printf("\n\nEnter the Expression\n\n");

scanf("%s",input1);

for(i=0;i<25;i++)

input[i]=NULL;

a:

for(i=0;input1[i]!=')';i++);

for(j=i;input1[j]!='(';j--);

for(x=j+1;x<i;x++)

if(isalpha(input1[x]))

input[n++]=input1[x];

else

if(input1[x]!='0')

store=input1[x];

input[n++]=store;

for(x=j;x<=i;x++)

input1[x]='0';

if(input1[0]!='0')goto a;

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

{

dag[i].label=input[i];

dag[i].ptr=i;

if(!isalpha(input[i])&&!isdigit(input[i]))

{

dag[i].right=i-1;

ptr=i;

var=input[i-1];

if(isalpha(var))

ptr=ptr-2;

else

{

ptr=i-1;

b:

if(!isalpha(var)&&!isdigit(var))

{

ptr=dag[ptr].left;

var=input[ptr];

goto b;

}

else

ptr=ptr-1;

}

dag[i].left=ptr;}}

printf("\n Syntax tree and the equivalent DAG for the given expression is \n \n");

printf("\n\n PTR \t\t LEFT PTR \t\t RIGHT PTR \t\t LABEL \n\n");

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

printf("\n%d\t%d\t%d\t%c\n",dag[i].ptr,dag[i].left,dag[i].right,dag[i].label);

getch();

}

**OUTPUT:**

Enter the Expression

((a+b)\*c)

Syntax tree for the given expression is

PTR LEFT PTR RIGHT PTR LABEL

0 0 0 a

1 0 0 b

2 0 1 +

3 0 0 c

4 2 3 \*

**RESULT:**

The DAG of the given input is generated and the output is verified.

DATE:

EX NO: **CONSTRUCTION OF PARSE TREE FROM A REGULAR EXPRESSION**

**AIM:**

To write a yacc program that accepts a regular expression as input and produce its parse tree as output.

**DESCRIPTION:**

Productions used in this program are:

S -> re

re -> re\* | re+ | (re) | re|re | re.re | ALPHABET

Some of the valid regular expressions are:

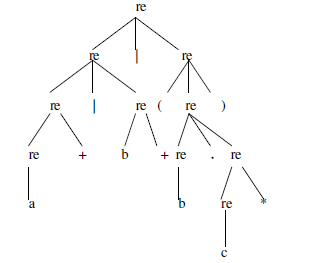
a+ | b\* | (b.c\*)

(a.b)\* | a+

(a | b\*) | a+

(a) | b+

Parse tree for regular expression: a+ | b\* | (b.c\*) is



Some of the invalid regular expressions are:

(ab\*) | a+

a\* + b\*

(a+b)\*

**PROGRAM:**

%{

#include <stdio.h>

#include <ctype.h>

#include <stdlib.h>

#include <string.h>

/\* To store the productions \*/

#define MAX 100

int getREindex ( const char\* );

signed char productions[MAX][MAX];

int count = 0 , i , j;

char temp[MAX + MAX] , temp2[MAX + MAX];

%}

%token ALPHABET

%left '|'

%left '.'

%nonassoc '\*' '+'

%%

S : re '\n' {

printf ( "This is the rightmost derivation--\n" );

for ( i = count - 1 ; i >= 0 ; --i ) {

if ( i == count - 1 ) {

printf ( "\nre => " );

strcpy ( temp , productions[i] );

printf ( "%s" , productions[i] );

}

else {

printf ( "\n => " );

j = getREindex ( temp );

temp[j] = '\0';

sprintf ( temp2 , "%s%s%s" , temp ,

productions[i] , (temp + j + 2) );

printf ( "%s" , temp2 );

strcpy ( temp , temp2 );

}

}

printf ( "\n" );

exit ( 0 );

}

re : ALPHABET {

temp[0] = yylval; temp[1] = '\0';

strcpy ( productions[count++] , temp );

}

| '(' re ')'

{ strcpy ( productions[count++] , "(re)" ); }

| re '\*'

{ strcpy ( productions[count++] , "re\*" ); }

| re '+'

{ strcpy ( productions[count++] , "re+" ); }

| re '|' re

{strcpy ( productions[count++] , "re | re" );}

| re '.' re

{strcpy ( productions[count++] , "re . re" );}

;

%%

int main ( int argc , char \*\*argv )

{

/\*

Parse and output the rightmost derivation,

from which we can get the parse tree

\*/

yyparse();

return 0;

}

yylex()

{

signed char ch = getchar();

yylval = ch;

if ( isalpha ( ch ) )

return ALPHABET;

return ch;

}

yyerror()

{

fprintf(stderr , "Invalid Regular Expression!!\n");

exit ( 1 );

}

int getREindex ( const char \*str )

{

int i = strlen ( str ) - 1;

for ( ; i >= 0 ; --i ) {

if ( str[i] == 'e' && str[i-1] == 'r' )

return i-1;

}

}

**OUTPUT:**

C:\Users\student.IT>cd C:\Flex Windows\Bison\bin

C:\Flex Windows\Bison\bin>yacc parse.y

C:\Flex Windows\Bison\bin>gcc y.tab.c

C:\Flex Windows\Bison\bin>a.exe

a+|(a)

This is the rightmost derivation--

re => re | re

=> re | (re)

=> re | (a)

=> re+ | (a)

=> a+ | (a)

**RESULT:**

Thus, the YACC program was executed and the output was verified.