COMPUTER SCIENCE AND ENGINEERING
REGULATION – R - 13
III YEAR – II SEMISTER

COMPILER DESIGN LAB MANUAL

PREPARED BY

ANIL KUMAR PRATHIPATI
NARASARAOPETA ENGINEERING COLLEGE
NARASARAOPETA.

Lexical Analyzer implimentation by using C program

```
#include<string.h>
#include<ctype.h>
#include<stdio.h>

void keyword(char[]);

void main()
{
FILE *f1,*f2,*f3;
char c,str[10],st1[10];
int num[100],lineno=0,tokenvalue=0,i=0,j=0,k=0;
clrscr();
printf("\nEnter the c program\n");
f1=fopen("input.txt","w");
while((c=getchar())!=EOF)
putc(c,f1);
fclose(f1);
f1=fopen("input.txt","r");
f2=fopen("identifier.txt","w");
f3=fopen("specialchar.txt","w");
while((c=getc(f1))!=EOF)
{
        if(isdigit(c))
        {
        tokenvalue=c-48;
        c=getc(f1);
        while(isdigit(c))
          {
                tokenvalue=tokenvalue*10+(c-48);
                c=getc(f1);
          }
        num[i++]=tokenvalue;
        ungetc(c,f1);
        }
        else if(isalpha(c))
        {
        putc(c,f2);
        c=getc(f1);
        while(isdigit(c)||isalpha(c)||c=='_'||c=='$')
          {
                putc(c,f2);
                c=getc(f1);
          }
        putc(' ',f2);
        ungetc(c,f1);
        }
        else if(c==' '||c=='\t')
        printf(" ");
        else if(c=='\n')
        lineno++;
        else putc(c,f3);
}
fclose(f2);
fclose(f3);
fclose(f1);
printf("\nThe numbers in the program are:");
for(j=0;j<i;j++)
printf(" %d ",num[j]);
printf("\n");
f2=fopen("identifier.txt","r");
k=0;
printf("The keywords and identifiersare:");
while((c=getc(f2))!=EOF)
{
        if(c!=' ')
        str[k++]=c;
        else
          {
                str[k]='\0';
                keyword(str);
                k=0;
          }
}
fclose(f2);
f3=fopen("specialchar.txt","r");
printf("\nSpecial characters are");
while((c=getc(f3))!=EOF)
printf(" %c ",c);
printf("\n");
fclose(f3);
printf("Total no. of lines are:%d",lineno);
getch();
}

void keyword(char str[10])
{
if(strcmp("for",str)==0||strcmp("while",str)==0||strcmp("do",str)==0||
  strcmp("int",str)==0||strcmp("float",str)==0||strcmp("char",str)==0||
  strcmp("double",str)==0||strcmp("static",str)==0||
  strcmp("switch",str)==0||strcmp("case",str)==0) // TYPE 32 KEYWORDS
  printf("\n%s is a keyword",str);
else
  printf("\n%s is an identifier",str);
}
```

<u>OUTPUT</u>

```
Enter the c program
int main()
{
int a=10,20;
char ch;
float f;
}^Z

The numbers in the program are: 10  20
The keywords and identifiersare:
int is a keyword
main is an identifier
int is a keyword
a is an identifier
char is a keyword
ch is an identifier
float is a keyword
f is an identifier
Special characters are (   )   {   =   ,   ;   ;   ;   }
Total no. of lines are:5
```

<u>Lexical Analyzer implimentation by using LEX tool</u>

```
// Program name as "lexicalfile.l"
%{
#include<stdio.h>
%}

delim [\t]
ws {delim}+
letter [A-Za-z]
digit [0-9]
id {letter}({letter}|{digit})*
num {digit}+(\.{digit}+)?(E[+/-]?{digit}+)?

%%
ws {printf("no action");}
if|else|then {{printf("%s is a keyword",yytext);} // TYPE 32 KEYWORDS
{id} {printf("%s is a identifier",yytext);}
```

```
{num} {printf(" it is a number");}
"<" {printf("it is a relational operator less than");}
"<=" {printf("it is a relational operator less than or equal");}
">" {printf("it is a relational operator greater than");}
">=" {printf("it is a relational operator greater than");}
"==" {printf("it is a relational operator equal");}
"<>" {printf("it is a relational operator not equal");}
%%

main()
{
 yylex();
}
```

OUTPUT

```
 lex  lexicalfile.l
cc  lex.yy.c  -ll
if
if is a keyword
number
number is a identifier
254
It is a number
<>
it is a relational operator not equal
^Z
```

SHIFT REDUCE PARSER

```
#include"stdio.h"
#include"conio.h"
#include"string.h"
char ip_sym[15],stack[15];
int ip_ptr=0,st_ptr=0,len,i;
char temp[2],temp2[2];
char act[15];
void check();

void main()
{
clrscr();
printf("\n\t\t SHIFT REDUCE PARSER\n");
printf("\n GRAMMER\n");
printf("\n E->E+E\n E->E/E");
printf("\n E->E*E\n E->E-E\n E->id");
printf("\n enter the input symbol:\t");
```

```
gets(ip_sym);
printf("\n\t stack implementation table");
printf("\n stack\t\t input symbol\t\t action");
printf("\n______\t\t ____________\t\t ______\n");
printf("\n $\t\t%s$\t\t\t--",ip_sym);
strcpy(act,"shift ");
temp[0]=ip_sym[ip_ptr];
temp[1]='\0';
strcat(act,temp);
len=strlen(ip_sym);
for(i=0;i<=len-1;i++)
{
stack[st_ptr]=ip_sym[ip_ptr];
stack[st_ptr+1]='\0';
ip_sym[ip_ptr]=' ';
ip_ptr++;
printf("\n $%s\t\t%s$\t\t\t%s",stack,ip_sym,act);
strcpy(act,"shift ");
temp[0]=ip_sym[ip_ptr];
temp[1]='\0';
strcat(act,temp);
check();
st_ptr++;
}
st_ptr++;
check();
}
void check()
{
int flag=0;
temp2[0]=stack[st_ptr];
temp2[1]='\0';
if(islower(temp2[0]))
{
 stack[st_ptr]='E';
flag=1;
}
if((!strcmp(temp2,"+"))||(!strcmp(temp2,"*"))
   ||(!strcmp(temp2,"/"))||(!strcmp(temp2,"-")))
{
 flag=1;
}
if((!strcmp(stack,"E+E"))||(!strcmp(stack,"E/E"))
   ||(!strcmp(stack,"E*E"))||(!strcmp(stack,"E-E")))
{
if(!strcmp(stack,"E+E"))
{
strcpy(stack,"E");
printf("\n $%s\t\t%s$\t\t\tE->E+E",stack,ip_sym);
}
else
if(!strcmp(stack,"E/E"))
{
strcpy(stack,"E");
printf("\n $%s\t\t %s$\t\t\tE->E/E",stack,ip_sym);
}
else
if(!strcmp(stack,"E-E"))
{
strcpy(stack,"E");
printf("\n $%s\t\t %s$\t\t\tE->E-E",stack,ip_sym);
}

else
{
strcpy(stack,"E");
printf("\n $%s\t\t%s$\t\t\tE->E*E",stack,ip_sym);
}
flag=1;
st_ptr=0;
}
if(!strcmp(stack,"E")&&ip_ptr==len)
{
printf("\n $%s\t\t%s$\t\t\tACCEPT",stack,ip_sym);
getch();
exit(0);
}
if(flag==0)
{
printf("\n $%s\t\t\t%s\t\t reject",stack,ip_sym);
exit(0);
}
return;
}
```

<u>OUTPUT:</u>

<u>1)</u>

```
                    SHIFT REDUCE PARSER

 GRAMMER

 E->E+E
 E->E/E
 E->E*E
 E->E-E
 E->id
 enter the input symbol:      a+b*c

         stack implementation table
 stack          input symbol         action
_______       ______________       _______

 $            a+b*c$              --
 $a            +b*c$              shift a
 $E            +b*c$               E->a
 $E+            b*c$               shift +
 $E+b            *c$               shift b
 $E+E            *c$                E->b
 $E             *c$              E->E+E
 $E*             c$              shift *
 $E*c             $              shift c
 $E*E             $               E->c
 $E              $             E->E*E
 $E              $             ACCEPT
```

2)

SHIFT REDUCE PARSER

```
 GRAMMER

 E->E+E
 E->E/E
 E->E*E
 E->E-E
 E->id
 enter the input symbol:      a+b*+c

         stack implementation table
 stack          input symbol         action
_______       ______________       _______

 $            a+b*+c$              --
 $a            +b*+c$              shift a
 $E            +b*+c$               E->a
 $E+            b*+c$               shift +
 $E+b            *+c$               shift b
 $E+E            *+c$                E->b
 $E            *+c$            E->E+E
 $E*            +c$            shift *
 $E*+            c$            shift +
 $E*+c            $            shift c
 $E*+E            $             E->c
 $E*+E                          reject
```

Recursive Descent Parser of a given grammer

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

char input[10];
int i=0,error=0;
void E();
void T();
void Eprime();
void Tprime();
void F();

void main()
{
        clrscr();
        printf("Enter an arithmetic expression   :\n");
        gets(input);
        E();
        if(strlen(input)==i&&error==0)
                printf("\nAccepted..!!!");
        else
                printf("\nRejected..!!!");
        getch();
}
/*if(input[i]=='i')
        {
                i++;
                if(input[i]=='d')
                i++;

        }
        else if(input[i]=='(')
        {
                i++;
                E();
                if(input[i]==')')
                i++;
        }
        else
        error=1;*/



void E()
{
    T();
    Eprime();
}

void Eprime()
{
    if(input[i]=='+')
    {
    i++;
    T();
    Eprime();
    }
}

void T()
{
    F();
    Tprime();
}

void Tprime()
{
    if(input[i]=='*')
    {
                    i++;
                    F();
                    Tprime();
    }
}

void F()
{
        if(input[i]=='(')
        {
                i++;
                E();
                if(input[i]==')')
                i++;
        }
        else if(isalpha(input[i]))
        {
                i++;
                while(isalnum(input[i])||input[i]=='_')
                i++;
        }
        else
                error=1;
}
```

<u>OUTPUT</u>

```
1)
Enter an arithmetic expression   :
sum+month*interest

Accepted..!!!

2)
Enter an arithmetic expression   :
sum+avg*+interest

Rejected..!!!
```

<u>Find the FIRST of a given grammer</u>

```
#include<stdio.h>
#include<ctype.h>

void FIRST(char[],char );
void result(char[],char);
int nop;
char prod[10][10];

void main()
{
int i;
char choice;
   char c;
   char res1[20];
   clrscr();
   printf("How many number of productions ? :");
   scanf(" %d",&nop);
   printf("enter the production string like E=E+T\n");
   for(i=0;i<nop;i++)
   {
        printf("Enter productions Number %d : ",i+1);
        scanf(" %s",prod[i]);
   }
   do
   {
        printf("\n Find the FIRST of  :");
        scanf(" %c",&c);
        FIRST(res1,c);
        printf("\n FIRST(%c)= { ",c);
        for(i=0;res1[i]!='\0';i++)
        printf(" %c ",res1[i]);
        printf("}\n");
        printf("press 'y' to continue : ");
        scanf(" %c",&choice);
   }
   while(choice=='y'||choice =='Y');
}

void FIRST(char res[],char c)
{
   int i,j,k;
   char subres[5];
   int eps;
   subres[0]='\0';
   res[0]='\0';

   if(!(isupper(c)))
   {
        result(res,c);
              return ;
   }
   for(i=0;i<nop;i++)
   {
        if(prod[i][0]==c)
        {
        if(prod[i][2]=='$')
        result(res,'$');
        else
           {
              j=2;
              while(prod[i][j]!='\0')
              {
              eps=0;
              FIRST(subres,prod[i][j]);
              for(k=0;subres[k]!='\0';k++)
                       result(res,subres[k]);
              for(k=0;subres[k]!='\0';k++)
                  if(subres[k]=='$')
```

```
                              {
                                      eps=1;
                                      break;
                              }
                         if(!eps)
                              break;
                         j++;
                        }
                }
    }
}
   return ;
}
void result(char res[],char val)
{
    int k;
    for(k=0 ;res[k]!='\0';k++)
            if(res[k]==val)
                return;
    res[k]=val;
    res[k+1]='\0';
}
```

<u>OUTPUT</u>

```
How many number of productions ? :8
enter the production string like E=E+T
Enter productions Number 1 : E=TX
Enter productions Number 2 : X=+TX
Enter productions Number 3 : X=$
Enter productions Number 4 : T=FY
Enter productions Number 5 : Y=*FY
Enter productions Number 6 : Y=$
Enter productions Number 7 : F=(E)
Enter productions Number 8 : F=i

 Find the FIRST of  :X

 FIRST(X)= {  +  $ }
press 'y' to continue : Y

 Find the FIRST of  :F

 FIRST(F)= {  (  i }
press 'y' to continue : Y

 Find the FIRST of  :Y
 FIRST(Y)= {  *  $ }
press 'y' to continue : Y

 Find the FIRST of  :E

 FIRST(E)= {  (  i }
press 'y' to continue : Y

 Find the FIRST of  :T

 FIRST(T)= {  (  i }
press 'y' to continue : N
```

<u>Find the FOLLOW of a given grammer</u>

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

int nop,m=0,p,i=0,j=0;
char prod[10][10],res[10];

void FOLLOW(char c);
void first(char c);
void result(char);

void main()
{
 int i;
 int choice;
 char c,ch;
 printf("Enter the no.of productions: ");
 scanf("%d", &nop);
 printf("enter the production string like E=E+T\n");
    for(i=0;i<nop;i++)
    {
           printf("Enter productions Number %d : ",i+1);
           scanf(" %s",prod[i]);
    }
 do
 {
  m=0;
  printf("Find FOLLOW of -->");
  scanf(" %c",&c);
  FOLLOW(c);
  printf("FOLLOW(%c) = { ",c);
  for(i=0;i<m;i++)
```

```
   printf("%c ",res[i]);
  printf(" }\n");
  printf("Do you want to continue(Press 1 to continue....)?");
 scanf("%d%c",&choice,&ch);
 }
 while(choice==1);
}

void FOLLOW(char c)
{
    if(prod[0][0]==c)
            result('$');
 for(i=0;i<nop;i++)
 {
  for(j=2;j<strlen(prod[i]);j++)
  {
   if(prod[i][j]==c)
   {
    if(prod[i][j+1]!='\0')
            first(prod[i][j+1]);
    if(prod[i][j+1]=='\0'&&c!=prod[i][0])
            FOLLOW(prod[i][0]);
   }
  }
 }
}

void first(char c)
{
      int k;
                    if(!(isupper(c)))
                        result(c);
                    for(k=0;k<nop;k++)
                    {
                            if(prod[k][0]==c)
                             {
                             if(prod[k][2]=='$')
                                    FOLLOW(prod[i][0]);
                             else if(islower(prod[k][2]))
                                    result(prod[k][2]);
                             else
                                    first(prod[k][2]);
                             }
                    }
}
```

```
void  result(char c)
{
   int i;
   for( i=0;i<=m;i++)
         if(res[i]==c)
             return;
   res[m++]=c;
}
```

<u>OUTPUT</u>

```
Enter the no.of productions: 8
enter the production string like E=E+T
Enter productions Number 1 : E=TX
Enter productions Number 2 : X=+TX
Enter productions Number 3 : X=$
Enter productions Number 4 : T=FY
Enter productions Number 5 : Y=*FY
Enter productions Number 6 : Y=$
Enter productions Number 7 : F=(E)
Enter productions Number 8 : F=i
Find FOLLOW of -->X
FOLLOW(X) = { $ )  }
Do you want to continue(Press 1 to continue....)?1
Find FOLLOW of -->E
FOLLOW(E) = {$ )  }
Do you want to continue(Press 1 to continue....)?1
Find FOLLOW of -->Y
FOLLOW(Y) = { + $ )  }
Do you want to continue(Press 1 to continue....)?1
Find FOLLOW of -->T
FOLLOW(T) = { +$ )  }
Do you want to continue(Press 1 to continue....)?1
Find FOLLOW of -->F
FOLLOW(F) = { * + $ )  }
Do you want to continue(Press 1 to continue....)?2
```

<u>LL(1) Parser of a given grammer</u>

```
#include<string.h>
#include<conio.h>
char a[10];
int top=-1,i;

void error()
{
printf("Syntax Error");
```

```
}

void push(char k[])
{
  for(i=0;k[i]!='\0';i++)
  {
    if(top<9)
    a[++top]=k[i];
  }
}

char TOS()
{
  return a[top];
}

void pop()
{
  if(top>=0)
    a[top--]='\0';
}

void display()
{
  for(i=0;i<=top;i++)
    printf("%c",a[i]);
}

void display1(char p[],int m)
{
  int l;
  printf("\t");
  for(l=m;p[l]!='\0';l++)
    printf("%c",p[l]);
}

char* stack()
{
return a;
}

void main()
{
  char ip[20],r[20],nt,cin;
  int ir,ic,j=0,k;
  char t[5][6][10]={"$","$","TH","$","TH","$",
                           "+TH","$","$","e","$","e",
                           "$","$","FU","$","FU","$",
                           "e","*FU","$","e","$","e",
                           "$","$","(E)","$","i","$"};
  clrscr();
  printf("\nEnter any String(Append with $)");
  gets(ip);
  printf("Stack\tInput\tOutput\n\n");
  push("$E");
  display();
  printf("\t%s\n",ip);
  for(j=0;ip[j]!='\0';)
  {
  if(TOS()==cin)
      {
            pop();
            display();
            display1(ip,j+1);
            printf("\tPOP\n");
            j++;
      }
    cin=ip[j];
    nt=TOS();
      if(nt=='E')ir=0;
      else if(nt=='H')ir=1;
      else if(nt=='T')ir=2;
      else if(nt=='U')ir=3;
      else if(nt=='F')ir=4;
      else {
                error();
                break;
                }
      if(cin=='+')ic=0;
      else if(cin=='*')ic=1;
      else if(cin=='(')ic=2;
      else if(cin==')')ic=3;
      else if(isalpha(cin)){ic=4;cin='i';}
      else if(cin=='$')ic=5;
      strcpy(r,strrev(t[ir][ic]));
      strrev(t[ir][ic]);
      pop();
      push(r);
      if(TOS()=='e')
      {
            pop();
            display();
```

```
            display1(ip,j);
            printf("\t%c->%c\n",nt,238);
      }
      else{
      display();
      display1(ip,j);
      printf("\t%c->%s\n",nt,t[ir][ic]);
      }
      if(TOS()=='$'&&cin=='$')
      break;
      if(TOS()=='$'){
            error();
            break;
            }
      }
      k=strcmp(stack(),"$");
      if(k==0)
      printf("\n Given String is accepted");
    else
    printf("\n Given String is not accepted");
  getch();
}
```

OUTPUT

1)

```
Enter any String(Append with $)i+i*i$
Stack   Input   Output

$E      i+i*i$
$HT     i+i*i$  E->TH
$HUF    i+i*i$  T->FU
$HUi    i+i*i$  F->i
$HU     +i*i$   POP
$H      +i*i$   U->ε
$HT+    +i*i$   H->+TH
$HT     i*i$    POP
$HUF    i*i$    T->FU
$HUi    i*i$    F->i
$HU     *i$     POP
$HUF*   *i$     U->*FU
$HUF    i$      POP
$HUi    i$      F->i
$HU     $       POP
$H      $       U->ε
$       $       H->ε
```

```
 Given String is accepted

2)

Enter any String(Append with $)i+i**i$
Stack   Input   Output

$E      i+i**i$
$HT     i+i**i$ E->TH
$HUF    i+i**i$ T->FU
$HUi    i+i**i$ F->i
$HU     +i**i$  POP
$H      +i**i$  U->ε
$HT+    +i**i$  H->+TH
$HT     i**i$   POP
$HUF    i**i$   T->FU
$HUi    i**i$   F->i
$HU     **i$    POP
$HUF*   **i$    U->*FU
$HUF    *i$     POP
$HU$    *i$     F->$
Syntax Error
 Given String is not accepted
```