KG College of Engineering & Technology
(Approved by AICTE, New Delhi, Affiliated to JNTUH, Hyderabad)
Chilkur (Village), Moinabad (Mandal), R. R Dist, TS-501504

DEPARTMENT OF COMPUTER SCIENCE ENGINEERING

COMPILER DESIGN LAB MANUAL

Subject Code : CS605PC
Regulation : R18/JNTUH
Academic Year : 2020-2021

III B. TECH II SEMESTER

COMPUTER SCIENCE AND ENGINEERING

KG REDDY COLLEGE OF ENGINEERING AND TECHNOLOGY

Affiliated o JNTUH, Chilkur,(V), Moinabad(M) R. R Dist, TS-501504

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION AND MISSION OF THE INSTITUTION

VISION

To become self-sustainable institution this is recognized for its new age engineering through innovative teaching and learning culture, inculcating research and entrepreneurial ecosystem, and sustainable social impact in the community.

MISSION

- To offer undergraduate and post-graduate programs that is supported through industry relevant curriculum and innovative teaching and learning processes that would help students succeed in their professional careers.
- To provide necessary support structures for students, this will contribute to their personal and professional growth and enable them to become leaders in their respective fields.
- To provide faculty and students with an ecosystem that fosters research and development through strategic partnerships with government organisations and collaboration with industries.
- To contribute to the development of the region by using our technological expertise to work with nearby communities and support them in their social and economic growth.

VISION AND MISSION OF CSE DEPARTMENT

VISION

To be recognized as a department of excellence by stimulating a learning environment in which students and faculty will thrive and grow to achieve their professional, institutional and societal goals.

MISSION

- To provide high quality technical education to students that will enable life-long learning and build expertise in advanced technologies in Computer Science and Engineering.
- To promote research and development by providing opportunities to solve complex engineering problems in collaboration with industry and government agencies.
- To encourage professional development of students that will inculcate ethical values and leadership skills while working with the community to address societal issues.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PROGRAM EDUCATIONAL OBJECTIVES (PEOS):

A graduate of the Computer Science and Engineering Program should:

PEO1	**Program Educational Objective1: (PEO1)** The Graduates will provide solutions to difficult and challenging issues in their profession by applying computer science and engineering theory and principles.
PEO2	**Program Educational Objective2 :(PEO2)** **The** Graduates have successful careers in computer science and engineering fields or will be able to successfully pursue advanced degrees.
PEO3	**Program Educational Objective3: (PEO3)** The Graduates will communicate effectively, work collaboratively and exhibit high levels of Professionalism, moral and ethical responsibility.
PEO4	**Program Educational Objective4 :(PEO4)** **The** Graduates will develop the ability to understand and analyse Engineering issues in a broader perspective with ethical responsibility towards sustainable development.

PROGRAM OUTCOMES (POS):

PO1	**Engineeringknowledge**:Applytheknowledgeofmathematics,science,engineering Fundamentals andanengineeringspecializationtothesolutionofcomplexengineeringproblems.
PO2	**Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3	**Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4	**Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5	**Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO6	**The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7	**Environment and sustainability**: Understand the impact of the professional engineering Solutions in societal and environmental contexts, and demonstrate the knowledge of, and needfor sustainable development.
PO8	**Ethics**: Apply ethical principles and commit to professional ethics and responsibilities andnorms of the engineering practice.
PO9	**Individual and team work**: Function effectively as an individual, and as a member or leader In diverse teams, and in multi-disciplinary settings.
PO10	**Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11	**Project management and finance**: Demonstrate knowledge and understanding of the Engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12	**Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES(PSOS):

PSO1	**Problem Solving Skills** – Graduate will be able to apply computational techniques and software principles to solve complex engineering problems pertaining to software engineering.
PSO2	**Professional Skills** – Graduate will be able to think critically, communicate effectively, and collaborate in teams through participation in co and extra-curricular activities.
PSO3	**Successful Career** – Graduates will possess a solid foundation in computer science and engineering that will enable them to grow in their profession and pursue lifelong learning through post-graduation and professional development.

Course Objectives

- To provide hands-on experience on web technologies.
- To develop client-server application using web technologies
- To introduce server-side programming with Java servlets and JSP
- To understand the various phases in the design of a compiler
- To understand the design of top-down and bottom-up parsers.
- To understand syntax directed translation schemes.
- To introduce lex and yacc tools.

Course Outcomes

- CO 1: Design and develop interactive and dynamic web applications using HTML, CSS, JavaScript and XML.
- CO 2:Analyze thelex and yacc tools for developing a scanner and a parser.
- CO 3:Design and implement LL and LR parsers.
- CO 4: Apply client-server principles to develop scalable and enterprise web applications

CO-PO & PSO Mapping:

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO 1	PSO 2	PSO 3
CO1	2	-	-	1	-	-	-	-	-	-	-	-	-	-	-
CO2	2	-	-	1	-	-	-	-	-	-	-	-	1	-	-
CO3	2	1	-	1	-	-	-	-	-	-	-	-	1	1	-
CO4	2	2	2	1	-	-	-	-	-	-	-	-	2	2	1

B.TECH CSE- 3-2
COMPILER DESIGN LAB

S. No	List of Experiments
1.	Write a LEX Program to scan reserved word & Identifiers of C Language
2.	Implement Predictive Parsing algorithm
3.	Write a C program to generate three address code.
4.	Implement SLR(1) Parsing algorithm
5.	Design LALR bottom up parser for the given language

1. Write a LEX Program to scan reserved word & Identifiers of C Language

```
#include&lt;stdio.h&gt;
#include&lt;conio.h&gt;
#include&lt;stdlib.h&gt;
FILE *fp;
FILE *kfp;
char iden[32],num[32];
int scantoken();
int chkkey();
static char ch;
char n[20];
void main()
{
int token=2;
clrscr();
fp=fopen(&quot;source.c&quot;,&quot;r&quot;);
if((kfp=fopen(&quot;key.txt&quot;,&quot;r&quot;))==NULL)
{
printf(&quot;\n Unable to open keyword file&quot;);
exit(0);
}
while(!feof(fp))
{
token=scantoken();
if(token&gt;0)
printf(&quot;\n%d %c %s&quot;,token,ch,n);
else if(token==0)
{

if(chkkey()==1)
printf(&quot;\nKeyword: %s&quot;,iden);
else
printf(&quot;\nIdentifier: %s&quot;,iden);
}
else if(token==1)
printf(&quot;\nNumber: %s&quot;,num);
}
fclose(fp);
fclose(kfp);
}

int scantoken()
{
int token=2,j;
static int lookahead=0;
if(!lookahead)
{
ch=fgetc(fp);
lookahead=0;
}
while(token==2)
{
switch(ch)
```

```
{
case &#39; &#39;:
ch=fgetc(fp);
break;
case &#39;\n&#39;:
ch=fgetc(fp);
break;
case &#39;\t&#39;:
ch=fgetc(fp);
break;
case &#39;/&#39;:
ch=fgetc(fp);
if(ch==&#39;*&#39;)
{
while(1)
{
ch=fgetc(fp);
if(ch==&#39;*&#39;)
{
ch=fgetc(fp);
if(ch==&#39;/&#39;)
break;
}
}
}
break;
case &#39;&lt;&#39;:
ch=fgetc(fp);
if(ch==&#39;=&#39;)
{
token=3;
lookahead=0;
strcpy(n,&quot;less/equal&quot;);
}
else
{
token=2;
lookahead=1;
strcpy(n,&quot;less&quot;);
}
break;
case &#39;&gt;&#39;:
ch=fgetc(fp);
if(ch==&#39;=&#39;)
{
token=5;
lookahead=0;
strcpy(n,&quot;greater/equal&quot;);
}
else
{
token=4;
lookahead=1;
strcpy(n,&quot;greater&quot;);
```

```
}
break;
case &#39;=&#39;:
ch=fgetc(fp);
if(ch==&#39;=&#39;)
{
token=7;
lookahead=0;
strcpy(n,&quot;equalcomp&quot;);
}
else
{
token=2;
lookahead=1;
strcpy(n,&quot;assign&quot;);
}
break;
case &#39;!&#39;:
ch=fgetc(fp);
if(ch==&#39;=&#39;)
{
token=9;
lookahead=0;
strcpy(n,&quot;notequal&quot;);
}
else
{
token=8;
lookahead=1;
}
break;
case &#39;+&#39;:
token=10;
lookahead=0;
strcpy(n,&quot;plus&quot;);
break;
case &#39;-&#39;:
token=11;
lookahead=0;
strcpy(n,&quot;minus&quot;);
break;
case &#39;*&#39;:
token=12;
lookahead=0;
strcpy(n,&quot;multiply&quot;);
break;
case &#39;;&#39;:
token=13;
lookahead=0;
strcpy(n,&quot;semicolon&quot;);
break;
case &#39;{&#39;:
token=14;
lookahead=0;
```

```
strcpy(n,&amp;quot;openbrace&amp;quot;);
break;
case &amp;#39;}&amp;#39;:
token=15;
lookahead=0;
strcpy(n,&amp;quot;closebrace&amp;quot;);
break;
case &amp;#39;(&amp;#39;:
token=16;
lookahead=0;
strcpy(n,&amp;quot;Leftparenthesis&amp;quot;);
break;
case &amp;#39;)&amp;#39;:
token=10;
lookahead=0;
strcpy(n,&amp;quot;Rightparenthesis&amp;quot;);
break;
case &amp;#39;ef&amp;#39;:
token=-3;
strcpy(n,&amp;quot;EOF&amp;quot;);
break;
default:
if((ch&amp;gt;=&amp;#39;a&amp;#39;)&amp;amp;&amp;amp;(ch&amp;lt;=&amp;#39;z&amp;#39;))
{
iden[0]=ch;
iden[1]=&amp;#39;\0&amp;#39;;
j=1;
ch=fgetc(fp);
while((ch&amp;gt;=&amp;#39;a&amp;#39;&amp;amp;&amp;amp;ch&amp;lt;=&amp;#39;z&amp;#39;)||(ch&amp;gt;=&amp;#39;0&amp;#39;&amp;amp;&amp;amp;ch&amp;lt;
=&amp;#39;9&amp;#39;))
{
iden[j]=ch;
j++;
ch=fgetc(fp);
}
iden[j]=&amp;#39;\0&amp;#39;;
lookahead=1;
```



```
token=0;
}
if(ch&amp;gt;=&amp;#39;0&amp;#39;&amp;amp;&amp;amp;ch&amp;lt;=&amp;#39;9&amp;#39;)
{
num[0]=ch;
j=1;
ch=fgetc(fp);
while(ch&amp;gt;=&amp;#39;0&amp;#39;&amp;amp;&amp;amp;ch&amp;lt;=&amp;#39;9&amp;#39;)
{
num[j]=ch;
j++;
ch=fgetc(fp);
}
iden[j]=&amp;#39;\0&amp;#39;;
lookahead=1;
```

```
token=-1;
}
}
}
return token;
}
int chkkey()
{
char *keyw;
int flag=0;
fseek(kfp,0,SEEK_SET);
while(!feof(kfp))
{
fscanf(kfp,&amp;quot;%s&amp;quot;,keyw);
if(strcmp(keyw,iden)==0)
{
flag=1;
break;
}
}
return flag;
}
```

OUTPUT:

create a file with source.c
in that write void main()
{
int a;
}next create another file keyword.txt
in that write void main
int float char
now u will get the output as follows
keyword main
keyword void
16(leftparanthesis
17) right parantehsis
14 { open brace
keyword int
identifier:a
13 ; semicolon
15 } close brace

2. Implement Predictive Parsing algorithm

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>
#include<stdlib.h>
#define SIZE 128
#define NONE -1
#define EOS '\0'
#define NUM 257
#define KEYWORD 258
#define ID 259
#define DONE 260
#define MAX 999
charlexemes[MAX];
charbuffer[SIZE];
intlastchar=-1;
intlastentry=0;
inttokenval=DONE;
intlineno=1;
intlookahead;
structentry
{
    char*lexptr;
    inttoken;
}
symtable[100];
structentry
        keywords[]=
{"if",KEYWORD,"else",KEYWORD,"for",KEYWORD,"int",KEYWORD,"float",KEYWORD,
                    "double",KEYWORD,"char",KEYWORD,"struct",KEYWORD,"ret
urn",KEYWORD,0,0
};
voidError_Message(char*m)
{
    fprintf(stderr,"line %d, %s \n",lineno,m);
    exit(1);
```

```
}
intlook_up(chars[ ])
{
    intk;
    for(k=lastentry; k>0; k--)
        if(strcmp(symtable[k].lexptr,s)==0)
            returnk;
    return0;
}
intinsert(chars[ ],inttok)
{
    intlen;
    len=strlen(s);
    if(lastentry+1>=MAX)
        Error_Message("Symbpl table is full");
    if(lastchar+len+1>=MAX)
        Error_Message("Lexemes array is full");
    lastentry=lastentry+1;
    symtable[lastentry].token=tok;
    symtable[lastentry].lexptr=&lexemes[lastchar+1];
    lastchar=lastchar+len+1;
    strcpy(symtable[lastentry].lexptr,s);
    returnlastentry;
}
/*void Initialize()
{
    struct entry *ptr;
    for(ptr=keywords;ptr->token;ptr+1)
        insert(ptr->lexptr,ptr->token);
}*/
intlexer()
{
    intt;
    intval,i=0;
    while(1)
    {
        t=getchar();
```

```
        if(t==' '||t=='\t');
        elseif(t=='\n')
            lineno=lineno+1;
        elseif(isdigit(t))
        {
            ungetc(t,stdin);
            scanf("%d",&tokenval);
            returnNUM;
        }
        elseif(isalpha(t))
        {
            while(isalnum(t))
            {
                buffer[i]=t;
                t=getchar();
                i=i+1;
                if(i>=SIZE)
                    Error_Message("Compiler error");
            }
            buffer[i]=EOS;
            if(t!=EOF)
                ungetc(t,stdin);
            val=look_up(buffer);
            if(val==0)
                val=insert(buffer,ID);
            tokenval=val;
            returnsymtable[val].token;
        }
        elseif(t==EOF)
            returnDONE;
        else
        {
            tokenval=NONE;
            returnt;
        }
    }
}
```

```
voidMatch(intt)
{
    if(lookahead==t)
        lookahead=lexer();
    else
        Error_Message("Syntax error");
}
voiddisplay(intt,inttval)
{
    if(t=='+'||t=='-'||t=='*'||t=='/')
        printf("\nArithmetic Operator: %c",t);
    elseif(t==NUM)
        printf("\n Number: %d",tval);
    elseif(t==ID)
        printf("\n Identifier: %s",symtable[tval].lexptr);
    else
        printf("\n Token %d tokenval %d",t,tokenval);
}
voidF()
{
    //void E();
    switch(lookahead)
    {
    case'(':
        Match('(');
        E();
        Match(')');
        break;
    caseNUM :
        display(NUM,tokenval);
        Match(NUM);
        break;
    caseID :
        display(ID,tokenval);
        Match(ID);
        break;
    default:
```

```
            Error_Message("Syntax error");
        }
    }
    voidT()
    {
        intt;
        F();
        while(1)
        {
            switch(lookahead)
            {
            case'*':
                t=lookahead;
                Match(lookahead);
                F();
                display(t,NONE);
                continue;
            case'/':
                t=lookahead;
                Match(lookahead);
                display(t,NONE);
                continue;
            default:
                return;
            }
        }
    }
    voidE()
    {
        intt;
        T();
        while(1)
        {
            switch(lookahead)
            {
            case'+':
                t=lookahead;
```

```
            Match(lookahead);
            T();
            display(t,NONE);
            continue;
        case'-':
            t=lookahead;
            Match(lookahead);
            T();
            display(t,NONE);
            continue;
        default:
            return;
        }
    }
}
voidparser()
{
    lookahead=lexer();
    while(lookahead!=DONE)
    {
        E();
        Match(';');
    }
}
intmain()
{
    charans[10];
    printf("\n Program for recursive descent parsing ");
    printf("\n Enter the expression ");
    printf("And place ; at the end\n");
    printf("Press Ctrl-Z to terminate\n");
    parser();<br>return0;
}
```

Output-

```
Program for recursive descent parsing
 Enter the expression And place ; at the end
Press Ctrl-Z to terminate
a*b+c;

 Identifier: a
 Identifier: b
Arithmetic Operator: *
 Identifier: c
Arithmetic Operator: +
5*7;

 Number: 5
 Number: 7
Arithmetic Operator: *
*2;
line 5, Syntax error
```

3. Write a C program to generate three address code.

Aim:-To generate three address codes.

ALGORITHM:

Step1: Begin the program
Step2 : The expression is read from the file using a file pointer
Step3 : Each string is read and the total no. of strings in the file is calculated.
Step4: Each string is compared with an operator; if any operator is seen then the previous string and next string are concatenated and stored in a first temporary value and the three address code expression is printed
Step5 : Suppose if another operand is seen then the first temporary value is concatenated to the next string using the operator and the expression is printed.
Step6 : The final temporary value is replaced to the left operand value.
Step7 : End the program.

PROGRAM: //program for three address code generation

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

struct three
{
char data[10],temp[7];
}s[30];
void main()
{
char d1[7],d2[7]="t";
int i=0,j=1,len=0;
FILE *f1,*f2;
clrscr();
f1=fopen("sum.txt","r");
f2=fopen("out.txt","w");
while(fscanf(f1,"%s",s[len].data)!=EOF)
```

```
len++;
itoa(j,d1,7);
strcat(d2,d1);
strcpy(s[j].temp,d2);
strcpy(d1,"");
strcpy(d2,"t");
if(!strcmp(s[3].data,"+"))
{
fprintf(f2,"%s=%s+%s",s[j].temp,s[i+2].data,s[i+4].data);
j++;
}
else if(!strcmp(s[3].data,"-"))
{
fprintf(f2,"%s=%s-%s",s[j].temp,s[i+2].data,s[i+4].data);
j++;
}
for(i=4;i<len-2;i+=2)
{
itoa(j,d1,7);
strcat(d2,d1);
strcpy(s[j].temp,d2);
if(!strcmp(s[i+1].data,"+"))
fprintf(f2,"\n%s=%s+%s",s[j].temp,s[j-1].temp,s[i+2].data);
else if(!strcmp(s[i+1].data,"-"))
fprintf(f2,"\n%s=%s-%s",s[j].temp,s[j-1].temp,s[i+2].data);
strcpy(d1,"");
strcpy(d2,"t");
j++;
}
fprintf(f2,"\n%s=%s",s[0].data,s[j-1].temp);
fclose(f1);
fclose(f2);
getch();
}
```

Input: sum.txt

```
out = in1 + in2 + in3 - in4
```

Output : out.txt

```
t1=in1+in2
t2=t1+in3
t3=t2-in4
out=t3
```

4. Implement SLR(1) Parsing algorithm.

Aim:- To implement SLR() parsing algorithm.
ALGORITHM / PROCEDURE :
SOURCE CODE :

```
#include<stdio.h>
char tab[12][9][2]={"S5","N","N","S4","N","N","1","2","3",
"N","S6","N","N","N","Z","0","0","0",
"N","R2","S7","N","R2","R2","0","0","0",
"N","R4","R4","N","R4","R4","0","0","0",
"S5","N","N","S4","N","N","8","2","3",
"N","R6","R6","N","R6","R6","0","0","0",
"S5","N","N","S4","N","N","0","9","3",
"S5","N","N","N","S4","N","0","0","X",
"N","S6","N","N","SY","N","0","0","0",
"N","R1","S7","N","R1","R1","0","0","0",
"N","R3","R3","N","R3","R3","0","0","0",
"N","R5","R5","N","R5","R5","0","0","0"};

/*Z=accept;
N->error;
X->10;
Y->11;*/
char prod[7][4]={"0","E+T","T","T*F","F","(E)","d"};
 char index[12]={'0','1','2','3','4','5','6','7','8','9','X','Y'};
char term[9]={'d','+','*','(',')','$','E','T','F'};
introw,col,st_pt=0,ip_pt=0;
char input[10], stack[20];
clrscr();
void main()
 {
intj,k;
printf("\n enter input string :");
scanf("%s", input);
strcat(input,"$");
stack[0]='0';
for(j=0;j<7;j++)
strcat(prod[j],"\0");
printf("\n STACK     INPUT \n \n");
while(1)
 {
for(k=0;k<=st_pt; k++)
printf("%c", stack[k]);
printf("          ");
for(k=ip_pt;input[k-1]!='$';k++)
printf("%c", input[k]);
printf("\n");
row=is_index(stack[st_pt]);
col=is_term(input[ip_pt]);
if(tab[row][col][0]=='S')
shift(tab[row][col][1]);
else
if(tab[row][col][0]=='R')
reduce(tab[row][col][1]);
```

```
else
if(tab[row][col][0]=='Z')
 {
printf (" \n success");
getch();
exit(0);
 }
else
if(tab[row][col][0]=='N')
 {
printf("\n error");
getch();
exit(0);
 }
 }
 }
shift(char ch)
   {
st_pt++;
stack[st_pt++]=input[ip_pt++];
stack[st_pt]=ch;
   }
reduce(char ch)
   {
intk,prno,prlen,rowno,colno;
for(k=1;k<7;k++)
if(index[k]==ch)
prno=k;
prlen=strlen(prod[prno]);
for(k=1;k<=2*prlen;k++)
st_pt--;

st_pt++;
if(prno==1||prno==2)
stack[st_pt]='E';
else
if(prno==3||prno==4)
stack[st_pt]='T';
else
if(prno==5||prno==6)
stack[st_pt]='F';
rowno=is_index(stack[st_pt-1]);
colno=is_term(stack[st_pt]);
stack[++st_pt]=tab[rowno][colno][0];
}
is_index(char ch)
{
int k;
for (k=0;k<=9;k++)
if(index[k]==ch)
return(k);
if(ch=='X')
return(10);
else
```

```
if(ch=='Y')
return(11);
}
is_term(char ch)
{
int k;
for(k=0;k<9;k++)
if(term[k]==ch)
return(k);
}
```

OUTPUT:

```
enter input string(d*d)
Stack                                   input
                        (d+d)$
O(4                     d+d)$
0(4d5                   +d)$
0(4f3                    +d)$
0(4T2                    +d)$
0(4E8                    +d)$
0(4E8+6                   d)$
0(4E8+6d5                 )$
0(4E8+6F3                  )$
0(4E8+6T9                 )$
0(4e8)Y                    $
0f3                        $
0T2                                     $
0E1                                     $
success
```

5. Design LALR bottom up parser for the given language

Aim: To Design and implement an LALR bottom up Parser for checking the syntax of the Statements in the given language.

ALGORITHM/PROCEDURE:

Source Code : LALR Bottom Up Parser

```
<parser.l>
%{
#include<stdio.h>
#include "y.tab.h"
%}
%%
[0-9]+ {yylval.dval=atof(yytext);
return DIGIT;
}
\n|. return yytext[0];
%%
<parser.y>
%{
/*This YACC specification file generates the LALR parser for the program
considered in experiment 4.*/
```

```
#include<stdio.h>
%}
%union
{
double dval;
}
%token <dval> DIGIT
%type <dval> expr
%type <dval> term
%type <dval> factor
%%
line: expr '\n' {
printf("%g\n",$1);
}
;
expr: expr '+' term {$$=$1 + $3 ;}
| term
;
term: term '*' factor {$$=$1 * $3 ;}
| factor
;
factor: '(' expr ')' {$$=$2 ;}
| DIGIT
;
%%
int main()
{
yyparse();
}
yyerror(char *s)
{
printf("%s",s);
}
```

Output:

```
$lex parser.l
$yacc –d parser.y
$cc lex.yy.cy.tab.c –ll –lm
$./a.out
2+3
```