M.V.S.R. ENGINEERING COLLEGE

(Affiliated to Osmania University, Hyderabad)
Nadergul(P.O.), Hyderabad-501510

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Subject : **Compiler Construction Lab- CS 383** Acad. Year : **2014-15**

Class : **BE III/IV Sem – II** Section :

INDEX

9		Evaluation of an Expression using YACC				
10		Intermediate Code Generation using YACC (3 Address Code)				
11		Code Generation:Single and Double Address				
12		Target Code Generation				
13		Code Optimization				

1) Scanner Programs using C

a) Tokenizing an assignment statement

Program

```
//tokenassign.c
#include<stdio.h>
#include<string.h>
main()
{
int i;
char a[20];
printf("Enter an assignment statement:\n");
scanf("%s",a);
for(i=0;i<strlen(a);i++)
{
   if((a[i]>='a'&&a[i]<='z')||(a[i]>='A'&&a[i]<='Z'))
   printf("%c  identifier\n",a[i]);
   else if(a[i]=='=')
   printf("= asignment operator\n");
   else if(a[i]=='+'||a[i]=='-
'||a[i]=='*'||a[i]=='/'||a[i]=='%')
   printf("%c arithmetic operator\n",a[i]);
}
}
```

Output

```
-bash-4.1$  cc tokenassign.c
```

Output (i)

```
-bash-4.1$  ./a.out
Enter an assignment statement:
p=s+t
p  identifier
= asignment operator
s  identifier
+ arithmetic operator
t  identifier
```

Output (ii)

```
-bash-4.1$  ./a.out
Enter an assignment statement:
a=b*c+d
a  identifier
= asignment operator
b  identifier
* arithmetic operator
c  identifier
+ arithmetic operator
d  identifier
```

1 b)Stand-alone C Scanner

Program

```
//cscanner.c
#include<stdio.h>
#include<ctype.h>
#include<string.h>
int main()
{
int i,j,k,l,m,x=0;
char*
c[10]={"stdio","include","return","void","scanf","printf","else"
,"int","if","main"};
char temp[100]={0},str[20],dir[7];
FILE *fp;
fp=fopen("test.c","r");
printf("\n\n\n");
printf("lexeme\t\ttoken\t\t\tline on \n");
printf("---\t\t---\t\t\t---\n");
for(l=1;!feof(fp);l++)
{
        fgets(temp,100,fp);
        for(i=0;temp[i]!='\0';)
        {
                if(temp[i]==' ')
                i++;
                if(temp[i]=='#')
                {
                        while(x<=6)
                        {
                                dir[x]=temp[i+1+x];
                                x++;
                        }
                        dir[x]='\0';
                        if(strcmp(dir,"include")==0)
                        {

printf("%s\tpreprocessor\t\t%d\n","#include",l);
                            i=i+8;
                        }
                        x=0;
                }
                else if(temp[i]=='\n')
                        temp[i]='\0';
                else if(isalpha(temp[i]))
                {
                        j=0;
                        while(isalnum(temp[i]))
                        {
                                str[j]=temp[i];
                                j++;
                                i++;
                        }
                        str[j]='\0';
                        for(k=0;k<10;k++)
                        {
                                if(!strcmp(c[k],str))
                                break;
                        }
                        if(k==10)

printf("%s\t\tidentifier\t\t%d\n",str,l);
                        else
                           printf("%s\t\treserved 
word\t\t%d\n",str,l);
                }
                else if(isdigit(temp[i]))
                {
                        j=0;
                        while(isdigit(temp[i]))
                        {
                                str[j]=temp[i];
                                i++;
                                j++;
                        }
                        str[j]='\0';
                        printf("%s\t\tkey 
constant\t\t%d\n",str,l);
                }
                else
                {
                        j=0;
                        str[j]=temp[i];
                        j++;
                        switch(temp[i])
                        {
                                case '+':
                                case '-':
                                case '*':
                                case '/':
                                case '=':
                                case '>':
                                case '<':
                                case '%':
```

```
                                {

if((temp[i]=='%')&&(temp[i+1]=='d'||temp[i+1]=='f'||temp[i+1]=='
c'||temp[i+1]=='s'))
                                  {
                                        str[j]=temp[i+1];
                                        i=i+2;
                                        j++;
                                        str[j]='\0';
                                        printf("%s\t\tformat
specifier\t\t%d\n",str,l);
                                        j=0;
                                        break;
                                  }
                                  else
                                  {
                                      str[j]='\0';

printf("%s\t\toperator\t\t%d\n",str,l);
                                      j=0;
                                      i++;
                                      break;
                                  }
                                }
                                case '(':
                                case ')':
                                case '{':
                                case '}':
                                {
                                   str[j]='\0';

printf("%s\t\tparanthesis\t\t%d\n",str,l);
                                   j=0;
                                   i++;
                                   break;
                                }
                                case ';':
                                {
                                   str[j]='\0';
                                   printf("%s\t\tsemi
colon\t\t%d\n",str,l);
                                   j=0;
                                   i++;
                                   break;
                                }
                                default:
                                {
```

```
                                str[j]='\0';
                                j=0;
                                i++;
                                printf("%s\t\tspecial
symbols\t\t%d\n",str,l);
                                }
                        }
                }
        }
}
return 0;
}

//Input file    test.c
#include<stdio.h>
void main()
{
int a=5,b=19;
if(a>b)
printf("a is greater\n");
else
printf("b is greater\n");
}
```

Output

```
-bash-4.1$  cc cscanner.c
-bash-4.1$  ./a.out
lexeme          token                   line on
---             ---                        ---
#include        preprocessor            1
<               operator                1
stdio           reserved word           1
.               special symbols         1
h               identifier              1
>               operator                1
void            reserved word           2
main            reserved word           2
(               paranthesis             2
)               paranthesis             2
{               paranthesis             3
int             reserved word           4
a               identifier              4
=               operator                4
5               key constant            4
,               special symbols         4
b               identifier              4
=               operator                4
19              key constant            4
;               semi colon              4
if              reserved word           5
(               paranthesis             5
a               identifier              5
>               operator                5
b               identifier              5
)               paranthesis             5
printf          reserved word           6
(               paranthesis             6
"               special symbols         6
a               identifier              6
is              identifier              6
greater         identifier              6
\               special symbols         6
n               identifier              6
"               special symbols         6
)               paranthesis             6
;               semi colon              6
else            reserved word           7
printf          reserved word           8
(               paranthesis             8
"               special symbols         8
b               identifier              8
is              identifier              8
greater         identifier              8
\               special symbols         8
n               identifier              8
"               special symbols         8
)               paranthesis             8
;               semi colon              8
}               paranthesis             9
}               paranthesis             10
```

2) Sample programs using LEX

a) Identification of a decimal number

Program

```
//decimal.l
%{
#include<stdio.h>
%}
%%
[\n\t ]+ ;
[+-]?([0-9]+|([0-9]*\.[0-9]+([eE][+-]?[0-9]+)?))
{printf("number\n");}
. {printf("not a number\n");}
%%
main()
{
yylex();
}
```

Output

```
-bash-4.1$  lex decimal.l
-bash-4.1$  cc lex.yy.c -ll
-bash-4.1$  ./a.out
12.34
number
+163.1e12
number
e-
non number
```

2 b) Word count without using files

Program

```
//wordcount.l
%{
#include<stdio.h>
int wc=0,cc=0,lc=0;
%}
word [^ \n\t]+
eol \n
%%
{word} {wc++;cc+=yyleng;}
{eol} {lc++;cc++;}
. {cc++;}
%%
main()
{
yylex();
printf("no. of characters:%d\nno. of words:%d\nno. of
lines:%d\n",cc,wc,lc);
}
```

Output

```
-bash-4.1$  lex wordcount.l
-bash-4.1$  cc lex.yy.c –ll
```

Output (i)

```
-bash-4.1$  ./a.out
Today is the 12th day of
the month February
in the year 2015 AD
no. of characters:65
no. of words:14
no. of lines:3
```

Output (ii)

```
-bash-4.1$  ./a.out
How are you
no. of characters:12
no. of words:3
no. of lines:1
```

2 c) Word count using files

Program

```
//wordcountfiles.l
%{
#include<stdio.h>
int cc=0,wc=0,lc=0;
%}
word [^ \n\t]+
eol \n
%%
{word} {wc++;cc+=yyleng;}
{eol} {cc++;lc++;}
. {cc++;}
%%
main(int argc,char**argv)
{
if(argc>1)
{
FILE *file;
file=fopen(argv[1],"r");
if(!file)
{
fprintf(stderr,"could not open %s\n",argv[1]);
exit(1);
}
yyin=file;
}
yylex();
printf("no. of characters:%d\nno. of lines:%d\nno. of
words:%d\n",cc,lc,wc);
return 0;
}

//Input file test.c
#include<stdio.h>
void main()
{
int a=5,b=19;
if(a>b)
printf("a is greater\n");
else
printf("b is greater\n");
}
```

Output

```
-bash-4.1$ lex wordcountfiles.l
-bash-4.1$  cc lex.yy.c -ll
-bash-4.1$  ./a.out test.c

no. of characters:115
no. of lines:9
no. of words:15
```

3) C Scanner using LEX

Program

```
//cscanner.l
%{
#include<stdio.h>
int lineno=1;
%}
letter [a-zA-Z]
digit [0-9]
id {letter}({letter}|{digit})*
num {digit}+
kw
"int"|"char"|"float"|"printf"|"main"|"if"|"elseif"|"then"|"void"
rop ">"|"<="|">="|"<"
aop "+"|"-"|"*"|"/"
arr ({id}"["{num}"]")
pre "#include"|"#define"
par "["|"]"|"("|")"|"{"|"}"
com ("/*"({id}|"\n")*"*/")
ws [.,;"]
%%
[\n] {lineno++;}
{ws} {printf("\n special symbols=%s \t
lineno=%d",yytext,lineno);}
{rop} {printf("\n relational operator=%s \t
lineno=%d",yytext,lineno);}
{com} {printf("\n comment=%s \t lineno=%d",yytext,lineno);}
"=" {printf("\n assignmnet operator=%s \t
lineno=%d",yytext,lineno);}
{aop} {printf("\n arithmetic operator=%s \t
lineno=%d",yytext,lineno);}
{par} {printf("\n paranthesis=%s \t lineno=%d",yytext,lineno);}
{pre} {printf("\n preprocessor=%s \t lineno=%d",yytext,lineno);}
{kw} {printf("\n keywords=%s \t lineno=%d",yytext,lineno);}
{arr} {printf("\n arrays=%s \t lineno=%d",yytext,lineno);}
{id} {printf("\n identifiers=%s \t lineno=%d",yytext,lineno);}
{num} {printf("\n number=%s \t lineno=%d",yytext,lineno);}
%%
int main(int argc,char* argv[])
{
if(argc>1)
yyin=fopen(argv[1],"r");
else
yyin=stdin;
yylex();
}
yywrap()
```

```
{
exit(0);
}
```

Output

```
-bash-4.1$ lex cscanner.l
-bash-4.1$  cc lex.yy.c -ll
-bash-4.1$  ./a.out test.c
preprocessor=#include   lineno=1
 relational operator=<   lineno=1
 identifiers=stdio       lineno=1
 special symbols=.       lineno=1
 identifiers=h   lineno=1
 relational operator=>   lineno=1
 keywords=void   lineno=2
 keywords=main   lineno=2
 paranthesis=(   lineno=2
 paranthesis=)   lineno=2
 paranthesis={   lineno=3
 keywords=int    lineno=4
 identifiers=a   lineno=4
 assignmnet operator==   lineno=4
 number=5        lineno=4
 special symbols=,       lineno=4
 identifiers=b   lineno=4
 assignmnet operator==   lineno=4
 number=19       lineno=4
 special symbols=;       lineno=4
 keywords=if     lineno=5
 paranthesis=(   lineno=5
 identifiers=a   lineno=5
 relational operator=>   lineno=5
 identifiers=b   lineno=5
 paranthesis=)   lineno=5
 keywords=printf         lineno=6
 paranthesis=(   lineno=6
 special symbols="       lineno=6
 identifiers=a   lineno=6
 identifiers=is          lineno=6
 identifiers=greater     lineno=6\
 identifiers=n   lineno=6
 special symbols="       lineno=6
 paranthesis=)   lineno=6
 special symbols=;       lineno=6
 identifiers=else        lineno=7
 keywords=printf         lineno=8
 paranthesis=(   lineno=8
 special symbols="       lineno=8
 identifiers=b   lineno=8
 identifiers=is          lineno=8
 identifiers=greater     lineno=8\
 identifiers=n   lineno=8
 special symbols="       lineno=8
 paranthesis=)   lineno=8
 special symbols=;       lineno=8
 paranthesis=}   lineno=9
```

4) Elimination of Immediate Left Recursion
Program

```
//leftrecursion.c
#include<stdio.h>
#include<string.h>
char alpha[20]={0};
char beta[20]={0};
char gram[30]={0};
int i=0,j=0,k=0;
void addToBeta(char);
void addToAlpha(char);

void ELR()
{
for(i=0;gram[i]!='\0';i++)
if(gram[i]=='>')
     break;
for(i=i+1;gram[i]!='\0';i++)
{
     if(gram[i]==gram[0])
     {
          for(i=i+1;gram[i]!='\0'&&gram[i]!='|';i++)
               addToAlpha(gram[i]);
          addToAlpha(';');
     }
     else
     {
          for(;gram[i]!='\0'&&gram[i]!='|';i++)
               addToBeta(gram[i]);
          addToBeta(';');
     }
}
alpha[j]='\0';
beta[k]='\0';
}

void addToAlpha(char ch)
{
alpha[j]=ch;
j++;
}

void addToBeta(char ch)
{
beta[k]=ch;
k++;
}
```

```
int main()
{
printf("\nEnter the Grammar:\n");
scanf("%s",gram);
ELR();
     if(strlen(alpha)==0)
     {
          printf("\nThe grammar is not left recursive");
          return 0;
     }
     else
     {
         printf("The grammar after eliminating the left
recursion is:\n");
         printf("\n%c->",gram[0]);
         for(i=0;beta[i+1]!='\0';i++)
         {
          if(beta[i]==';')
               printf("%c'|",gram[0]);
          else
               printf("%c",beta[i]);
         }
         printf("%c'",gram[0]);
         printf("\n%c'->",gram[0]);
         for(i=0;alpha[i+1]!='\0';i++)
         {
          if(alpha[i]==';')
               printf("%c'|",gram[0]);
          else
               printf("%c",alpha[i]);
         }
            printf("%c'|e\n",gram[0]);
         return 0;
     }
}
```

Output

```
-bash-4.1$  cc leftrecursion.c
```

Output (i)

```
-bash-4.1$  ./a.out
Enter the Grammar:
S->aS|bS|e

The grammar is not left recursive
```

Output (ii)

```
-bash-4.1$  ./a.out
Enter the Grammar:
E->E+T|E-T|T
The grammar after eliminating the left recursion is:

E->TE'
E'->+TE'|-TE'|e
```

5) Left factoring a given grammar
Program

```
//leftfactor.c
#include<stdio.h>
char a[30];
char b[3];
char g[30];
char gr[30];
char p1[30];
char p2[30];
void left()
{
int i=0,j=0,k=0;
while(gr[i]!='>')
i++;
j=0;
for(i=i+1;gr[i]!='|';i++)
p1[j++]=gr[i];
p1[j]='\0';
printf("first production is:%s\n",p1);
j=0;
for(i=i+1;gr[i]!='\0';i++)
p2[j++]=gr[i];
p2[j]='\0';
printf("second production is:%s ",p2);
while(p1[k]==p2[k])
k++;
j=0;
for(i=0;i<k;i++)
a[j++]=p2[i];
a[j]='\0';
printf("\nalpha is %s\n",a);
j=0;
for(i=k;p1[i]!='\0';i++)
b[j++]=p1[i];
b[j]='\0';
printf("\nbeta is %s\n",b);
j=0;
for(i=k;p2[i]!='\0';i++)
g[j++]=p2[i];
g[j]='\0';
printf("\ngamma is %s\n",g);
}

int main()
{
printf("\nenter the grammar:");
scanf("%s",gr);
left();
printf("\nleft factored grammar is:\n");
printf("%c->%sX\n",gr[0],a);
printf("X->%s|%s\n",b,g);
return 0;
}
```

Output

```
-bash-4.1$ cc leftfactor.c
```

Output (i)

```
-bash-4.1$  ./a.out
enter the grammar:S->aA|aB
first production is:aA
second production is:aB
alpha is a

beta is A

gamma is B
left factored grammar is:
S->aX
X->A|B
```

Output (ii)

```
-bash-4.1$  ./a.out
enter the grammar:S->ieltAb|ieltBa
first production is:ieltAb
second production is:ieltBa
alpha is ielt

beta is Ab

gamma is Ba
left factored grammar is:
S->ieltX
X->Ab|Ba
```

6)Top-down Parsers

a) Recursive Descent Parser (Grammar-1)

Program

```
//recdescentparserg1.c
#include<stdio.h>
void E();
void E1();
void T();
void T1();
void F();
void match(char);
int flag=1;
char ch,t;

int main()
{
printf("The grammar is\n");
printf("E->E+T|T\n");
printf("T->T*F|F\n");
printf("F->i\n");
printf("The elimination of left recursion is needed for
implementing recursive descent parser\n");
printf("The grammar after elimination of left recursion is\n");
printf("E->TE'  \n");
printf("E'->+TE'|e \n");
printf("T->FT'\n");
printf("T'->*FT'\n");
printf("F->i\n");
printf("enter input string and end the string with $\n");
scanf("%c",&ch);
E();
if((ch=='$')&&(flag!=0))
     printf("successful\n");
else
     printf("unsuccessful\n");
}

void match(char t)
{
  if(ch==t)
     scanf("%c",&ch);
  else
     flag=0;
}

void E()
{
```

```
T();
E1();
}

void E1()
{
  if(ch=='+')
  {
     match('+');
     T();
     E1();
  }
  else
     return;
}

void T()
{
F();
T1();
}

void T1()
{
  if(ch=='*')
  {
     match('*');
     F();
     T1();
  }
  else return;
}

void F()
{
match('i');
}
```

Output

```
-bash-4.1$ cc recdescentparserg1.c
```

Output (i)

```
-bash-4.1$  ./a.out
The grammar is
E->E+T|T
T->T*F|F
F->i
```

```
The elimination of left recursion is needed for implementing
recursive descent parser
The grammar after elimination of left recursion is
E->TE'
E'->+TE'|e
T->FT'
T'->*FT'
F->i
enter input string and end the string with $
i+i*i$
successful
```

Output (ii)

```
-bash-4.1$  ./a.out
The grammar is
E->E+T|T
T->T*F|F
F->i
The elimination of left recursion is needed for implementing
recursive descent parser
The grammar after elimination of left recursion is
E->TE'
E'->+TE'|e
T->FT'
T'->*FT'
F->i
enter input string and end the string with $
i*i-i/i$
unsuccessful
```

6 b) Recursive Descent Parser (Grammar-2)

Program

```
//recdescentparserg2.c
#include<stdio.h>
void E();
void T();
void match(char);
int flag=1;
char ch,t;

int main()
{
printf("The grammar is\n");
printf("E->x+T \n");
printf("T->(E)|x \n");
printf("Enter input string and end with dollar\n");
scanf("%c",&ch);
E();
if((ch=='$')&&(flag!=0))
     printf("successful\n");
else
     printf("unsuccessful\n");
}

void match(char t)
{
     if(ch==t)
          scanf("%c",&ch);
     else flag=0;
}

void E()
{
     if(ch=='x')
          match('x');
     match('+');
     T();
}

void T()
{
     if(ch=='(')
     {
        match('(');
        E();
        match(')');
     }
     else
          match('x');
}
```

Output

```
-bash-4.1$ cc recdescentparserg2.c
```

Output (i)

```
-bash-4.1$  ./a.out
The grammar is
E->x+T
T->(E)|x
Enter input string and end with dollar
x$
unsuccessful
```

Output (ii)

```
-bash-4.1$  ./a.out
The grammar is
E->x+T
T->(E)|x
Enter input string and end with dollar
x+(x+x)$
successful
```

7) First and Follow functions

Program

```
//firstandfollow.c
#include<stdio.h>
#include<string.h>
char S[4][4]={"Aad","Bdb","Z"};
char A[3][3]={"aAb","BBa","Z"};
char B[3][3]={"c","b","Z"};
int len=3;
int p,q,r,s;
char St[5],Sf[5];
char At[5],Af[5];
char Bt[5],Bf[5];
int z;

void disp()
{
printf("\nS->Aad|Bdb\n");
printf("A->aAb|BBa\n");
printf("B->c|b\n");
}

int term_not(char p)
{
if(p=='a'||p=='b'||p=='c'||p=='d')
     return 1;
else
     return 0;
}

void first(char v,char* arr)
{
int ans;
char ch;
int y;
int flag=0;
static int i,j,k;
ans=term_not(v);
  if(ans==1)
  {
     for(y=0;y<4;y++)
     {
          if(arr[y]==v)
          {
               flag=1;
               break;
          }
     }
     if(flag!=1)
     {
          arr[z]=v;
          z++;
     }
  }
  else
  {
     ch=v;
     switch(ch)
     {
          case 'S':
                i=0;
                while(strcmp(S[i],"Z")!=0)
                {
                    first(S[i][0],arr);
                       i++;
                }
                i=0;
                break;
          case 'A':
                j=0;
                while(strcmp(A[j],"Z")!=0)
                {
                    first(A[j][0],arr);
                    j++;
                }
                break;
          case 'B':
                k=0;
                if(v=='B')
                    while(strcmp(B[k],"Z")!=0)
                    {
                         first(B[k][0],arr);
                         k++;
                    }
                break;
     }
   }
}

void follow(char syn,char *arr)
{
    if(syn=='S')
     printf("$");
    p=0;
    while(strcmp(S[p],"Z")!=0)
    {
     for(q=0;q<len;q++)
     {
          if(S[p][q]==syn)
               first(S[p][q+1],arr);
     }
     p++;
    }
    s=0;
    while(strcmp(A[s],"Z")!=0)
    {
     for(q=0;q<len;q++)
          if(A[s][q]==syn)
               first(A[s][q+1],arr);
     s++;
    }
    r=0;
    while(strcmp(B[r],"Z")!=0)
    {
     for(q=0;q<len;q++)
          if(B[r][q]==syn)
               first(B[p][q+1],arr);
     r++;
    }
}

void first_call()
{
printf("Firsts are:\n\n");
int p;
z=0;
printf("\n");
printf("First(S):");
first('S',St);
for(p=0;p<4;p++)
     printf("%c",St[p]);
printf("\n");
printf("First(A):");
z=0;
first('A',At);
for(p=0;p<4;p++)
     printf("%c",At[p]);
printf("\n");
printf("First(B):");
z=0;
first('B',Bt);
```

```
for(p=0;p<4;p++)
     printf("%c",Bt[p]);
printf("\n\n");
}

void follow_call()
{
printf("Follows are:\n\n");
z=0;
printf("Follow(S):");
follow('S',Sf);
for(p=0;p<4;p++)
     printf("%c",Sf[p]);
printf("\n");
z=0;
printf("Follow(A):");
follow('A',Af);
for(p=0;p<4;p++)
        printf("%c",Af[p]);
printf("\n");
z=0;
printf("Follow(B):");
follow('B',Bf);
for(p=0;p<4;p++)
        printf("%c",Bf[p]);
printf("\n\n");
}

int main()
{
int choice;
disp();
printf("select an option:\n1.Firsts 2.Follows 3.exit\n");
scanf("%d",&choice);
   while(choice<3)
   {
     if(choice==1)
          first_call();
     else
          follow_call();
     printf("Select an option:\n1.Firsts 2.Follows 3.exit\n");
     scanf("%d",&choice);
   }
}
```

Output

```
-bash-4.1$ cc firstandfollow.c
-bash-4.1$  ./a.out
S->Aad|Bdb
A->aAb|BBa
B->c|b
select an option:
1.Firsts 2.Follows 3.exit
1
Firsts are:


First(S):acb
First(A):acb
First(B):cb

Select an option:
1.Firsts 2.Follows 3.exit
2
Follows are:

Follow(S):$
Follow(A):ab
Follow(B):dcba

Select an option:
1.Firsts 2.Follows 3.exit
3
```

8)Bottom-Up Parser- Shift Reduce Parser

Program

```
//shiftreduceparser.c
#include<stdio.h>
#include<string.h>
#include<ctype.h>

struct stack
{
char a[50];
int top;
}s;
char instring[50];

int isoperator(char c)
{
if(c=='+'||c=='*'||c=='/'||c=='%')
     return 1;
else
     return 0;
}

void shift_red()
{
char str[30];
char *ptr=instring;
int i,j;
while(*ptr!='$')
{
s.top++;
s.a[s.top]=*ptr++;
for(i=0;i<=s.top;i++)
str[i]=s.a[i];
str[i]='\0';
printf("%s\t\t%s\t\t%s\n",str,ptr,"shift");

if(isalpha(s.a[s.top]))
{
s.a[s.top]='E';
for(i=0;i<s.top;i++)
str[i]=s.a[i];
str[i]='\0';
printf("%s\t\t%s\t\t%s\n",str,ptr,"reduce");
}
}

if(s.top==1&&s.a[s.top]=='E')
```

```
{
printf("%s\t\t%s\t\t%s\n","$E","$","accept");
printf("\nsuccessful!!!");
return;
}

if(s.top<3||(s.top==1&&s.a[s.top]!='E'))
goto e1;
for(i=s.top;i>=3;i--)
{
if(i==s.top&&s.a[i]=='E')
{
f2:if(isoperator(s.a[i-1]))
{
i=i-2;
f1:switch(s.a[i])
{
case ')':if(s.a[i-1]==')')
{
i=i-1;
goto f1;
}
else if(s.a[i-1]=='E')
{
i=i-1;
if(s.a[i-1]=='(')
{
s.a[i-1]='E';
for(j=1;j<=s.top-2;j++)
s.a[j]=s.a[j+2];
s.top=s.top-2;
for(j=0;j<=s.top;j++)
str[j]=s.a[j];
str[j]='\0';

if(s.a[s.top]=='E'&&s.top==1)
{
printf("%s\t\t%s\t\t%s\n",str,ptr,"accept");
printf("\nsuccessful!!!");
return;
}
else
printf("%s\t\t%s\t\t%s\n",str,ptr,"reduce");
}
else goto f2;
}
else
{
e1:
for(j=0;j<=s.top;j++)
str[j]=s.a[j];
str[j]='\0';
printf("%s\t\t%s\t\t%s\n",str,ptr,"error");
printf("\nunsuccessful\n");
return;
}
i=s.top+1;
break;
case 'E':s.a[i]='E';
for(j=i+1;j<=s.top-2;j++)
s.a[j]=s.a[j+2];
s.top=s.top-2;
for(j=0;j<=s.top;j++)
str[j]=s.a[j];
str[j]='\0';
if(s.a[s.top]=='E'&&s.top==1)
{
printf("%s\t\t%s\t\t%s\n",str,ptr,"accept");
printf("\nsuccessful");
return;
}

else
printf("%s\t\t%s\t\t%s\n",str,ptr,"reduce");
i=s.top+1;
break;
}

if(i==3)
goto e1;
}

else
goto e1;
}

else if(i==s.top&&s.a[i]==')')
goto f1;
}
}

int main()
{
```

```
s.top=0;
s.a[s.top]='$';
printf("\n\n\tSHIFT REDUCE PARSING\n");
printf("\n");
puts("\nE->E+E\nE->E-E\nE->E*E\nE->(E)\nE->E/E\nE->i\n");
printf("\nEnter the string to be parsed:");
scanf("%s",instring);
strcat(instring,"$");
printf("stack\t\tbuffer\t\toperation\n");
shift_red();
return 0;
}
```

Output

```
-bash-4.1$ cc shiftreduceparser.c
```

Output (i)

```
-bash-4.1$  ./a.out
       SHIFT REDUCE PARSING

E->E+E
E->E-E
E->E*E
E->(E)
E->E/E
E->i

Enter the string to be parsed:E+E*E
stack           buffer          operation
$E              +E*E$           shift
$               +E*E$           reduce
$E+             E*E$            shift
$E+E            *E$             shift
$E+             *E$             reduce
$E+E*           E$              shift
$E+E*E          $               shift
$E+E*           $               reduce
$E+E            $               reduce
$E              $               accept

successful
```

Output (ii)

```
-bash-4.1$  ./a.out
        SHIFT REDUCE PARSING


E->E+E
E->E-E
E->E*E
E->(E)
E->E/E
E->i


Enter the string to be parsed:i*i
stack           buffer          operation
$i              *i$             shift
$               *i$             reduce
$E*             i$              shift
$E*i            $               shift
$E*             $               reduce
$E              $               accept

Successful
```

9) Evaluation of Expression using YACC

Program

```
//expeval.y
%{
#include<stdio.h>
#include<ctype.h>
#define YYSTYPE double
%}
%token NUM
%left '+''-'
%left '*''/'
%right UMINUS
%%
lines:lines expr '\n'{printf("%g\n",$2);}
|lines'\n'
|/*empty*/
;
expr:expr'+'expr{$$=$1+$3;}
|expr'-'expr{$$=$1-$3;}
|expr'*'expr{$$=$1*$3;}
|expr'/'expr{$$=$1/$3;}
|'('expr')'{$$=$2;}
|'-'expr %prec UMINUS{$$=-$2;}
|NUM
;
%%

main()
{
yyparse();
}

void yyerror(char *s)
{
printf("\nerror\n");
}

yylex()
{
char c;
while((c=getchar())==' ');
if((c=='.')||(isdigit(c)))
{
     ungetc(c,stdin);
     scanf("%lf",&yylval);
     return NUM;
}
return c;
}
```

Output

```
-bash-4.1$ yacc expeval.y
-bash-4.1$  cc y.tab.c
-bash-4.1$  ./a.out
16-525/25*45/9
-89
(3-2)*(125/5)
25
```

10) Intermediate Code Generation using YACC(3 Address Code)

Program

```
//lexinput.l
opr [*+-/=]
%%
[\t]. {}
[a-zA-Z]* {sscanf(yytext,"%c",&yylval);return id;}
[\n] {return(yytext[0]);}
{opr} {return(yytext[0]);}
%%
yywrap()
{
return 1;
}

//intercodegen.y
%{
#include<stdio.h>
#include<ctype.h>
int i;
%}

%token id
%%
L:id'='E'\n' {printf("%c=%c\n",$1,$3);}
|
;

E:E'+'T{$$='p';printf("%c=%c+%c\n",$$,$1,$3);}
|E'-'T{$$='r';printf("%c=%c-%c\n",$$,$1,$3);}
|T
;

T:T'*'F{$$='q';printf("%c=%c*%c\n",$$,$1,$3);}
|T'/'F{$$='s';printf("%c=%c/%c\n",$$,$1,$3);}
|F
;

F:'-'G{$$='t';printf("%c=-%c\n",$$,$2);}
|G
;

G:id{$$=$1;}
;
%%
#include"lex.yy.c"
main()
```

```
{
printf("Three Address Code is:\n");
yyparse();
}

void yyerror(char *s)
{
printf("\n error!!\n");
}
```

Output

```
-bash-4.1$ yacc intercodegen.y
-bash-4.1$ lex lexinput.l
-bash-4.1$  cc y.tab.c
```

Output (i)

```
-bash-4.1$  ./a.out
Three Address Code is:
a=b+c*d
q=c*d
p=b+q
a=p
```

Output (ii)

```
-bash-4.1$  ./a.out
Three Address Code is:
v=t/e+n-a*s
s=t/e
p=s+n
q=a*s
r=p-q
v=r
```

11) Code Generation: Single and Double Address

Program

```
//codegeneration.c
#include<stdio.h>

int main()
{
int i=0,ch,k=3;
char a[100];
printf("\n enter a string: ");
scanf("%s",a);
printf("\n 1.Single Address 2.Double Address\n");
printf("Enter choice:");
scanf("%d",&ch);
switch(ch)
{
case 1:
printf("\nLOAD %c\n",a[2]);
while(a[i]!='\0')
{
if(a[i+3]=='+')
     printf("\nADD %c\n",a[i+4]);
else if(a[i+3]=='-')
     printf("\nSUB %c\n",a[i+4]);
i=i+2;
}
printf("\nSTORE %c\n",a[0]);
break;
case 2:
while(a[k]!='\0')
{
if(a[k]=='+')
     printf("\nADD %c %c\n",a[2],a[k+1]);
else if(a[k]=='-')
     printf("\nSUB %c %c\n",a[2],a[k+1]);
k=k+2;
}
if(a[1]=='=')
     printf("\nMOV %c %c\n",a[0],a[2]);
break;
}
}
```

Output

```
-bash-4.1$  cc codegeneration.c
```

Output (i)

```
-bash-4.1$  ./a.out
```

```
enter a string: a=b-c+d-e

 1.Single Address 2.Double Address
Enter choice:1

LOAD b

SUB c

ADD d

SUB e

STORE a
```

Output (ii)

```
-bash-4.1$  ./a.out
enter a string: x=y+z-a+b

 1.Single Address 2.Double Address
Enter choice:2

ADD y z

SUB y a

ADD y b

MOV x y
```

12) Target Code Generation

Program

```
//targetcodegeneration.c
#include<stdio.h>
#include<ctype.h>
#define max 20
static int r=0;
int top=-1;
char st[max];

int main()
{
int h;
char ip[10];
printf("\nEnter the postfix expression:\n");
scanf("%s",ip);
h=0;
    while(ip[h]!='\0')
    {
     while(islower(ip[h])!=0)
     {
          push(ip[h]);
          h++;
     }
     getregister();
     printf("\nload %c R%d\n",st[top-1],r);
     switch(ip[h])
     {
          case '+':
               {
                   printf("\nadd %c R%d\n",st[top],r);
                   empty();
                   h++;
                   }
               break;
          case '-':
               {
                  printf("\nsub %c R%d\n",st[top],r);
                           empty();
                  h++;
                         }
               break;
           case '*':
               {
                           printf("\nmul %c R%d\n",st[top],r);
                           empty();
                           h++;
```

```
                            }
                        break;
             case '/':
                 {
                               printf("\ndiv %c R%d\n",st[top],r);
                               empty();
                               h++;
                            }
                        break;
            default:
                   printf("Invalid choice");
       }
      }
return 0;
}

empty()
{
char t1;
pop();
pop();
printf("\n st R%d t\n",r);
t1='t';
push(t1);
}
getregister()
{
r++;
if(r>2)
       r=1;
}

push(int x)
{
if(top==max-1)
     printf("\n stack full\n");
else
{
     top++;
     st[top]=x;
}
}
pop()
{
if(top==-1)
     printf("\n stack empty\n");
else
```

```
     top--;
}
```

Output

```
-bash-4.1$  cc targetcodegeneration.c
```

Output (i)

```
-bash-4.1$  ./a.out
Enter the postfix expression:
abc*+
load b R1
mul c R1
 st R1 t
load a R2
add t R2
 st R2 t
```

Output (ii)

```
-bash-4.1$  ./a.out
Enter the postfix expression:
xy+

load x R1

add y R1

 st R1 t
```

13) Code Optimization

Program

```
//codeoptimization.c
#include<stdio.h>
#include<string.h>

void main()
{
char
str[25][50],op[25][50],forloopparam[90],righthandparam[10][40],l
efthandparam[90];
int i=0,k=0,j=0,m=0,n=0,q=0,s=0,l=0;
int flag[10]={0},count[10]={0};
printf("\nInput the loop to be optimized:\n");
gets(str[0]);
while(str[k][i++]!=';');
while(str[k][i++]!=';');
while(str[k][i]!=')')
{
     if(isalpha(str[k][i]))
          forloopparam[j++]=str[k][i];
     i++;
}
i=0;
strcpy(op[l++],str[0]);
gets(str[0]);
while(str[0][i++]!='{');
strcpy(op[l++],str[0]);
k=0;
while(gets(str[k])&&str[k][0]!='}')
{
     while(str[k][i++]!='=');
          lefthandparam[n++]=str[k][i-2];
     k++;
     i=0;
}
for(m=0,i=0;m<k;m++)
{
     while(str[m][i++]!='=');
     while(str[m][i]!=';')
     {
          if(isalpha(str[m][i]))
               righthandparam[m][count[m]++]=str[m][i];
          i++;
     }
i=0;
}
```

```
for(m=0;m<k;m++)
for(s=0;s<count[m];s++)
for(i=0;i<n;i++)
{
     if(righthandparam[m][s]==lefthandparam[i])
     flag[m]=1;
}
for(i=0;i<k;i++)
for(q=0;q<j;q++)
{
     if(lefthandparam[i]==forloopparam[q])
          flag[i]=1;
}
for(m=1;m<k;m++)
for(s=0;s<count[m];s++)
for(i=0;i<j;i++)
{
        if(righthandparam[m][s]==forloopparam[i])
        flag[m]=1;
}
printf("\n\nOptimized loop is\n");
for(i=0;i<l;i++)
puts(op[i]);
for(i=0;i<k;i++)
if(flag[i]==1)
     puts(str[i]);
puts(str[k]);
for(i=0;i<k;i++)
if(flag[i]==0)
     printf("%s\t\t",str[i]);
}
```

Output

```
-bash-4.1$  cc codeoptimization.c
```

Output (i)

```
-bash-4.1$  ./a.out
Input the loop to be optimized:
for(c=0,i=1;c<=4;c++)
{
c=i+1;
d=a+b;
}
```

```
Optimized loop is
for(c=0,i=1;c<=4;c++)
{
c=i+1;
}
d=a+b;
```

Output (ii)

```
-bash-4.1$  ./a.out
Input the loop to be optimized:
for(int i=0;i<6;i++)
{
i=i*2;
a++;
}


Optimized loop is
for(int i=0;i<6;i++)
{
i=i*2;
}
a++;
```