CD LAB FINAL REPORT

Kumar Sashank

AP20110010229

1. Identify the tokens

Observation:
Here we taken input code from a file, and print the tokens for the given code in source text file. Reading line by line and saving in a string and finding out the tokens.

Source Code:

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

bool isValidDelimiter(char ch)
{
    if (ch == ' ' || ch == '+' || ch == '-' || ch == '*' ||
        ch == '/' || ch == ',' || ch == ';' || ch == '>' ||
        ch == '<' || ch == '=' || ch == '(' || ch == ')' ||
        ch == '[' || ch == ']' || ch == '{' || ch == '}')
        return (true);
    return (false);
}
bool isValidOperator(char ch)
{
    if (ch == '+' || ch == '-' || ch == '*' ||
        ch == '/' || ch == '>' || ch == '<' ||
        ch == '=')
        return (true);
    return (false);
}
// Returns 'true' if the string is a VALID IDENTIFIER.
bool isvalidIdentifier(char *str)
{
    if (str[0] == '0' || str[0] == '1' || str[0] == '2' ||
        str[0] == '3' || str[0] == '4' || str[0] == '5' ||
        str[0] == '6' || str[0] == '7' || str[0] == '8' ||
        str[0] == '9' || isValidDelimiter(str[0]) == true)
        return (false);
    return (true);
}
bool isValidKeyword(char *str)
{
    if (!strcmp(str, "if") || !strcmp(str, "else") || !strcmp(str,
"while") || !strcmp(str, "do") || !strcmp(str, "break") || !strcmp(str,
```

```
"continue") || !strcmp(str, "int") || !strcmp(str, "double") ||
!strcmp(str, "float") || !strcmp(str, "return") || !strcmp(str, "char")
|| !strcmp(str, "case") || !strcmp(str, "char") || !strcmp(str,
"sizeof") || !strcmp(str, "long") || !strcmp(str, "short") ||
!strcmp(str, "typedef") || !strcmp(str, "switch") || !strcmp(str,
"unsigned") || !strcmp(str, "void") || !strcmp(str, "static") ||
!strcmp(str, "struct") || !strcmp(str, "goto"))
        return (true);
    return (false);
}
bool isValidInteger(char *str)
{
    int i, len = strlen(str);
    if (len == 0)
        return (false);
    for (i = 0; i < len; i++)
    {
        if (str[i] != '0' && str[i] != '1' && str[i] != '2' && str[i]
!= '3' && str[i] != '4' && str[i] != '5' && str[i] != '6' && str[i] !=
'7' && str[i] != '8' && str[i] != '9' || (str[i] == '-' && i > 0))
            return (false);
    }
    return (true);
}
bool isRealNumber(char *str)
{
    int i, len = strlen(str);
    bool hasDecimal = false;
    if (len == 0)
        return (false);
    for (i = 0; i < len; i++)
    {
        if (str[i] != '0' && str[i] != '1' && str[i] != '2' && str[i]
!= '3' && str[i] != '4' && str[i] != '5' && str[i] != '6' && str[i] !=
'7' && str[i] != '8' && str[i] != '9' && str[i] != '.' || (str[i] == '-
' && i > 0))
            return (false);
        if (str[i] == '.')
            hasDecimal = true;
    }
    return (hasDecimal);
}
char *subString(char *str, int left, int right)
{
```

```
    int i;
    char *subStr = (char *)malloc(sizeof(char) * (right - left + 2));
    for (i = left; i <= right; i++)
        subStr[i - left] = str[i];
    subStr[right - left + 1] = '\0';
    return (subStr);
}
void detectTokens(char *str)
{
    int left = 0, right = 0;
    int length = strlen(str);
    while (right <= length && left <= right)
    {
        if (isValidDelimiter(str[right]) == false)
        {
            // printf("%c", str[right]);
            right++;
        }
        if (isValidDelimiter(str[right]) == true && left == right)
        {
            if (isValidOperator(str[right]) == true)
                printf("Valid operator : '%c'\n", str[right]);
            right++;
            left = right;
        }
        else if (isValidDelimiter(str[right]) == true && left != right
|| (right == length && left != right))
        {
            char *subStr = subString(str, left, right - 1);
            if (isValidKeyword(subStr) == true)
                printf("Valid keyword : '%s'\n", subStr);
            else if (isValidInteger(subStr) == true)
                printf("Valid Integer : '%s'\n", subStr);
            else if (isRealNumber(subStr) == true)
                printf("Real Number : '%s'\n", subStr);
            else if (isvalidIdentifier(subStr) == true &&
isValidDelimiter(str[right - 1]) == false)
                printf("Valid Identifier : '%s'\n", subStr);
            else if (isvalidIdentifier(subStr) == false &&
isValidDelimiter(str[right - 1]) == false)
                printf("Invalid Identifier : '%s'\n", subStr);
            left = right;
        }
    }
```

```
    return;
}

int main()
{
    FILE *fp;
    char *line = NULL;
    size_t len = 0;
    ssize_t read;
    fp = fopen("token.txt", "r");
    if (fp == NULL)
        exit(EXIT_FAILURE);
    while ((read = getline(&line, &len, fp)) != -1)
    {
        // printf("%s", line);
        detectTokens(line);
    }
    fclose(fp);
    if (line)
        free(line);
    return 0;
}
```

Text file:

CPP > compiler_lab > token.txt

You, 2 months ago | 1 author

```
1  int a=24;
2  float b=24;
3  x=(5);
4  x+++2
5  string str=Hola;
```

Output:

```
Valid keyword : 'int'
Valid Identifier : 'a'
Valid operator : '='
Valid Integer : '24'
Valid Identifier : '
'
Valid keyword : 'float'
Valid Identifier : 'b'
Valid operator : '='
Valid Integer : '24'
Valid Identifier : '
'
Valid Identifier : 'x'
Valid operator : '='
Valid Integer : '5'
Valid Identifier : '
'
Valid Identifier : 'x'
Valid operator : '+'
Valid operator : '+'
Valid operator : '+'
Invalid Identifier : '2
'
Valid Identifier : 'string'
Valid Identifier : 'str'
Valid operator : '='
Valid Identifier : 'Hola'
```

Lex Programs:

1. Lex program to check whether the given number is even or odd**.**

Observation:
Here we written a lex program to check the whether the given number is even or odd.

Code:

```
%{
#include<stdio.h> int
i;
%}

%%

[0-9]+   {i=atoi(yytext);
if(i%2==0)
printf("Even");          else
printf("Odd");}
%%
int yywrap(){}

/* Driver code */ int
main()
{
yylex();
return 0;
}
```

Output:

```
❯ ./a.out
45
Odd
66
Even
^C
```

2. Lex Program to check valid Mobile Number.

Observation:
Here we written a lex program to check the whether the given string is valid mobile number or not using string count rules.

Code:

```
%{
    #include<stdio.h>
#include<stdlib.h>
int count = 0;      char
str[100];
%}
%%
[0-9] { count++;
          str[count-1] = yytext[0];
        }
\n {return 0;}
%%
int yywrap(){}; int
main(){
yylex();

    if(str[0] != '0' && count==10){
printf("Valid Mobile Number\n");          exit(0);
    }     else{          printf("Invalid
Mobile Number\n");
    }
    return 0;
}
```

Output:

```
❯ ./a.out
9948734564
Valid Mobile Number
❯ ./a.out
822883
Invalid Mobile Number
```

3. Lex code to find the length of the longest word.

Observation:
Here we written a lex program to take few strings as input divided with spaces. We will print the length of longest string in given strings.

Code:

```
%{
int max = 0;
%}

%%
[a-zA-Z]+ {if(yyleng>max) max = yyleng;}
\n {return 0;}
%%
int yywrap(){}; int main()
{ yylex(); printf("largest:
%d", max); printf("\n");
}
```

Output:

```
❯ ./a.out
hola kumar sashank
  largest: 7
```

4. Write LEX programs for to check well formedness of the parenthesis or balanced parenthesis.

Observation:

Here we written a lex program to check well formedness of the parenthesis or balanced parenthesis.

Code:

```
%{
    #include<stdio.h>
int flag=0,ln=1;
%}
%%
"(" {flag++;}
")" {flag--;}

[\n] {  if(flag==0)
    {
        printf("\nNo missing at line : %d ",ln);
}
    else
{
        printf("\nMissing at line no. : %d",ln);
    }
    if(flag>0 || flag<0)
    {   printf("\n missing");
    }
    flag=0;
ln++;
} %%
int yywrap(){} int
main()
{
    char fname[100];     printf("\nEnter
the name of file\n");
scanf("%s",fname);
yyin=fopen(fname,"r+");     yylex();
return 0;
}
```

Output:

```
❯ ./a.out

Enter the name of file
test.txt
a-b
Missing at line no. : 1
 missinga+b%
```

5. Lex program to count number of lines, tabs and spaces used in the input.

Observation:
Here we written a lex program to count the number of lines ,tab spaces and spaces from the given input.

Code:

```
/*of lines other for number of characters */
%{
    #include<stdio.h>
int no_of_spaces = 0; int
no_of_lines = 0; int
no_of_chars = 0; int
no_of_tabs = 0;
%}
%%
([ ])+ no_of_spaces++;
\n ++no_of_lines;
\t no_of_tabs++; .
++no_of_chars; end
{return 0;}
%%
int yywrap(){} int
main()
{ yylex(); printf("number of lines = %d, number of chars =
%d\n,number of tabs =
%d\n,number of spaces= %d\n",
    no_of_lines, no_of_chars, no_of_tabs,no_of_spaces); return
0;
}
```

Output:

```
❯ ./a.out
hi this          is Kumar
Sashank          229
end
number of lines = 2, number of chars = 23
,number of tabs = 4
,number of spaces= 2
```

6. Lex program to count the total number of tokens.

Observation:
Here we written a lex program to count the total number of tokens from the given input using some conditions for keywords, delimiters etc.

Code:

```
%{
int n = 0 ;
%}

%%
"while"|"if"|"else" {n++;printf("keywords : %s\t", yytext);}
"int"|"float" {n++;printf("keywords : %s\t", yytext);}
[a-zA-Z_][a-zA-Z0-9_]* {n++;printf("identifier : %s\t", yytext);}
"<="|"=="|"="|"++"|"-"|"*"|"+" {n++;printf("operator : %s\t", yytext);}
[(){}|, ;] {n++;printf("separator : %s\t", yytext);}
[0-9]*"."[0-9]+ {n++;printf(" float : %s\t", yytext);}
[0-9]+ {n++;printf("integer : %s\t", yytext);}                    .
;

%%
int yywrap(){}

int main()
{
    yylex();     printf("\ntotal no. of token
= %d\n", n); }
```

Output:

```
❯ ./a.out
while(a==0):
keywords : while        separator : (   identifier : a  operator : ==   integer
: 0     separator : )
if(a<=b)
keywords : if   separator : (   identifier : a  operator : <=   identifier : b s
eparator : )
^C
```

7. Lex program to count number of words.

Observation:
Here we written a lex program to count the number of words in a given input.

Code:

```
%{
    #include<stdio.h>
int i=0;
%}

%%
([a-zA-Z])* {i++;}
\n return 0;
%%
int yywrap(){}; int
main(){
yylex();
printf("%d",i);
return 0;
}
```

Output:

```
❯ ./a.out
This is a Compiler Lab Assignment
      6%
```

8. Lex Program to check whether a number is Prime or Not.

Observation:
Here we written a lex program to check the whether the given number is prime or not by dividing numbers from 2 to n/2 to check.

Code:

```
%{
    #include<stdio.h>
int n;
%}

%%
[0-9]+ {n=atoi(yytext);
        for(int i=2; i<=n/2;i++)
        {
            if(n%i==0)
            {
                printf("%d is not a prime number",n);
return 0;
            }
}
        printf("%d is a prime number",n);
return 0;}
 %%
int yywrap(){};

int main()
{
yylex();
return 0;
}
```

Output:

```
❯ ./a.out
56
56 is not a prime number%
❯ ./a.out
73
73 is a prime number%
```

9. Write LEX program which will recognize strings ending with 00

Observation:

Here we written a lex program to check whether the string is ending with 00 or not using IF conditions.

Code:

```
%{
    #include<stdio.h>
int n;
%

%%

([a-zA-z0-9])+ {n=yyleng;
if(yytext[n-1]=='0' && yytext[n-2]=='0')
{
    printf("This string has ended with 00\n");
} else
{
    printf("This string has not ended with 00\n");
}}

%%
int yywrap(){};

int main()
{
yylex();
return 0;
}
```

Output:

```
❯ ./a.out
wefwerwe63
This string has not ended with 00

refef00
This string has ended with 00

123400998
This string has not ended with 00
```

10. Implement lexical analyzer using LEX for recognizing the following tokens:

- A minimum of 10 keywords of your choice
- Identifiers with the regular expression: letter(letter | digit)*
- Integers with the regular expression: digit+
- Relational operators: <, >, <=, >=, ==, !=
- Ignores everything between multi line comments (/* …. */)

Observation:
Here we written a lex program to recognize the identifiers, some keywords, regular expressions, operators .
Code:

```
%{
    #include<stdio.h>
int key=0;
%}

%%
"int"|"float"|"char"|"while"|"if"|"else"|"do"|"signed"|"unsigned"|"brea
k" {printf("%s is a keyword", yytext);}
[0-9]+ {printf("Integers with regular expression digt+ ");}
([a-z]([a-z]|[0-9]))* {printf("indetifier recognized\n");}
[<|>|<=|>=|==|!=]+ {printf("relational operators");}
"/"[^/]"/" {printf("text under coments recognized");}
"//".* {printf("");}
%%
int yywrap(){}

int main(){     printf("Enter
the code\n");      return
yylex();}
```

Output:

```
❯ ./a.out
Enter the code
int a=10;
int is a keyword arelational operatorsIntegers with regular expression digt+ ;
a+b=10;
a+brelational operatorsIntegers with regular expression digt+ ;
if(a<=b)
if is a keyword(arelational operatorsb)
784
Integers with regular expression digt+
^C
```

YACC Programs:

1. YACC program to recognize strings of { a^nb | n≥0 }

Observation:
Here we written a yacc program to recognize the strings of { a^nb|n≥0 } .

Lexcode:

```
%{
/* Definition section */
#include "y.tab.h"
%}

/* Rule Section */
%%
[aA] {return A;} [bB] {return B;}
\n {return NL;}
. {return yytext[0];}
%%

int yywrap()
{
return 1;
}
```

YACC code:

```
%{
/* Definition section */
#include<stdio.h>
#include<stdlib.h>
%}

%token A B NL

/* Rule Section */
%%
stmt: A S B NL {printf("valid string\n");
exit(0);}
;
S: S A
|
;
%%

int yyerror(char *msg)
{
```

```
printf("invalid string\n"); exit(0);
}


//driver code main()
{
printf("enter the string\n"); yyparse();
}
```

Output:

Valid String:

```
enter the string
aaab
valid string
```

Invalid String:

```
enter the string
aabb
invalid string
```

2. YACC program to recognize strings of { anb | n≥5 }

Observation:
Here we written a yacc program to recognize the strings of { anb | n≥5 }

Lexcode:

```
%{
#include "y.tab.h"
%}

%%
[aA] {return A;} [bB] {return B;}
\n {return NL;}
. {return yytext[0];}
%%

int yywrap()
{
return 1;
}
```

YACC code:

```
%{
```

```
/* Definition section */
#include<stdio.h>
#include<stdlib.h>
%}

%token A B NL

/* Rule Section */
%%
stmt: A A A A A S B NL {printf("valid string\n");
            exit(0);}
;
S: S A
|
;
%%

int yyerror(char *msg)
{
printf("invalid string\n"); exit(0);
}
//driver code main()
{
printf("enter the string\n"); yyparse();
}
```

Output:

Valid String:

```
enter the string
aaaaab
valid string
```

Invalid String:

```
enter the string
aaaaabb
invalid string
```

3. YACC program to recognize strings of { anbn | n≥0 }

Observation:
Here we written a yacc program to recognize the strings of { anbn | n≥0 }

LEXcode:

```
%{
/* Definition section */
#include "y.tab.h"
%}

/* Rule Section */
%%
[aA] {return A;} [bB] {return B;}
\n {return NL;}
. {return yytext[0];}
%%

int yywrap()
{
return 1;
}
```

YACC code:

```
%{
/* Definition section */
#include<stdio.h>
#include<stdlib.h>
%}

%token A B NL

/* Rule Section */
%%
stmt: S NL { printf("valid string\n");              exit(0); }
;
S: A S B |
;
%%

int yyerror(char *msg)
{
```

```
printf("invalid string\n"); exit(0);
}

//driver code main()
{
printf("enter the string\n"); yyparse();
}
```

Output:

Valid String:

```
enter the string
aaabbb
valid string
```

Invalid String:

```
enter the string
aab
invalid string
```

CPP CODES

1. First and Follow

Observation:
Here we written a CPP program to find the first and follow for given productions.

Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{
    char fin[10][20], st[10][20], ft[20][20], fol[20][20];
    int a = 0, e, i, t, b, c, n, k, l = 0, j, s, m, p;
    printf("enter the no. of productions\n");
    scanf("%d", &n);
    printf("enter the productions in a grammar\n");
    for (i = 0; i < n; i++)
        scanf("%s", st[i]);
    for (i = 0; i < n; i++)
        fol[i][0] = '\0';
    for (s = 0; s < n; s++)
    {
        for (i = 0; i < n; i++)
        {
            j = 3;
            l = 0;
            a = 0;
        l1:
            if (!((st[i][j] > 64) && (st[i][j] < 91)))
            {
                for (m = 0; m < l; m++)
                {
                    if (ft[i][m] == st[i][j])
                        goto s1;
                }
                ft[i][l] = st[i][j];
                l = l + 1;
            s1:
                j = j + 1;
            }
            else
            {
                if (s > 0)
                {
                    while (st[i][j] != st[a][0])
```

```
                {
                    a++;
                }
                b = 0;
                while (ft[a][b] != '\0')
                {
                    for (m = 0; m < l; m++)
                    {
                        if (ft[i][m] == ft[a][b])
                            goto s2;
                    }
                    ft[i][l] = ft[a][b];
                    l = l + 1;
                s2:
                    b = b + 1;
                }
            }
        }
        while (st[i][j] != '\0')
        {
            if (st[i][j] == '|')
            {
                j = j + 1;
                goto l1;
            }
            j = j + 1;
        }
        ft[i][l] = '\0';
    }
}
printf("first pos\n");
for (i = 0; i < n; i++)
    printf("FIRS[%c]=%s\n", st[i][0], ft[i]);
fol[0][0] = '$';
for (i = 0; i < n; i++)
{
    k = 0;
    j = 3;
    if (i == 0)
        l = 1;
    else
        l = 0;
k1:
    while ((st[i][0] != st[k][j]) && (k < n))
```

```
        {
            if (st[k][j] == '\0')
            {
                k++;
                j = 2;
            }
            j++;
        }
        j = j + 1;
        if (st[i][0] == st[k][j - 1])
        {
            if ((st[k][j] != '|') && (st[k][j] != '\0'))
            {
                a = 0;
                if (!((st[k][j] > 64) && (st[k][j] < 91)))
                {
                    for (m = 0; m < l; m++)
                    {
                        if (fol[i][m] == st[k][j])
                            goto q3;
                    }
                    fol[i][l] = st[k][j];
                    l++;
                q3:
                    p++;
                }
                else
                {
                    while (st[k][j] != st[a][0])
                    {
                        a++;
                    }
                    p = 0;
                    while (ft[a][p] != '\0')
                    {
                        if (ft[a][p] != 'e')
                        {
                            for (m = 0; m < l; m++)
                            {
                                if (fol[i][m] == ft[a][p])
                                    goto q2;
                            }
                            fol[i][l] = ft[a][p];
                            l = l + 1;
```

```
                    }
                    else
                        e = 1;
                q2:
                    p++;
                }
                if (e == 1)
                {
                    e = 0;
                    goto a1;
                }
            }
        }
        else
        {
        a1:
            c = 0;
            a = 0;
            while (st[k][0] != st[a][0])
            {
                a++;
            }
            while ((fol[a][c] != '\0') && (st[a][0] != st[i][0]))
            {
                for (m = 0; m < l; m++)
                {
                    if (fol[i][m] == fol[a][c])
                        goto q1;
                }
                fol[i][l] = fol[a][c];
                l++;
            q1:
                c++;
            }
        }
        goto k1;
    }
    fol[i][l] = '\0';
}
printf("follow pos\n");
for (i = 0; i < n; i++)
    printf("FOLLOW[%c]=%s\n", st[i][0], fol[i]);
printf("\n");
return 0;
```

Output:

```
Enter the no. of productions: 3
Enter the productions:
S->A|a
A->a|c|B
B->d
First Positions
First[S]=acd
First[A]=acd
First[B]=d
Follow Positions
Follow[S]=$
Follow[A]=$
Follow[B]=$
```

2. Predictive Parser

Observation:

Here we written a CPP program to find the predictive parsing table for the given productions.

Source Code:

```
// predictive parser program note: epsilon is denoted by 'e' and sample
production is E->+TE|e
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{
    char fin[10][20], st[10][20], ft[20][20], fol[20][20];
    int a = 0, e, i, t, b, c, n, k, l = 0, j, s, m, p;
    printf("enter the no. of productions\n");
    scanf("%d", &n);
    printf("enter the productions in a grammar\n");
    for (i = 0; i < n; i++)
        scanf("%s", st[i]);
    for (i = 0; i < n; i++)
        fol[i][0] = '\0';
    for (s = 0; s < n; s++)
    {
        for (i = 0; i < n; i++)
        {
            j = 3;
            l = 0;
            a = 0;
        l1:
            if (!((st[i][j] > 64) && (st[i][j] < 91)))
            {
                for (m = 0; m < l; m++)
                {
                    if (ft[i][m] == st[i][j])
                        goto s1;
                }
                ft[i][l] = st[i][j];
                l = l + 1;
            s1:
                j = j + 1;
            }
            else
            {
                if (s > 0)
                {
```

```
                while (st[i][j] != st[a][0])
                {
                    a++;
                }
                b = 0;
                while (ft[a][b] != '\0')
                {
                    for (m = 0; m < l; m++)
                    {
                        if (ft[i][m] == ft[a][b])
                            goto s2;
                    }
                    ft[i][l] = ft[a][b];
                    l = l + 1;
                s2:
                    b = b + 1;
                }
            }
        }
        while (st[i][j] != '\0')
        {
            if (st[i][j] == '|')
            {
                j = j + 1;
                goto l1;
            }
            j = j + 1;
        }
        ft[i][l] = '\0';
    }
}
printf("first pos\n");
for (i = 0; i < n; i++)
    printf("FIRS[%c]=%s\n", st[i][0], ft[i]);
fol[0][0] = '$';
for (i = 0; i < n; i++)
{
    k = 0;
    j = 3;
    if (i == 0)
        l = 1;
    else
        l = 0;
k1:
```

```
        while ((st[i][0] != st[k][j]) && (k < n))
        {
            if (st[k][j] == '\0')
            {
                k++;
                j = 2;
            }
            j++;
        }
        j = j + 1;
        if (st[i][0] == st[k][j - 1])
        {
            if ((st[k][j] != '|') && (st[k][j] != '\0'))
            {
                a = 0;
                if (!((st[k][j] > 64) && (st[k][j] < 91)))
                {
                    for (m = 0; m < l; m++)
                    {
                        if (fol[i][m] == st[k][j])
                            goto q3;
                    }
                    fol[i][l] = st[k][j];
                    l++;
                q3:
                    p++;
                }
                else
                {
                    while (st[k][j] != st[a][0])
                    {
                        a++;
                    }
                    p = 0;
                    while (ft[a][p] != '\0')
                    {
                        if (ft[a][p] != 'e')
                        {
                            for (m = 0; m < l; m++)
                            {
                                if (fol[i][m] == ft[a][p])
                                    goto q2;
                            }
                            fol[i][l] = ft[a][p];
```

```
                        l = l + 1;
                    }
                    else
                        e = 1;
                q2:
                    p++;
                }
                if (e == 1)
                {
                    e = 0;
                    goto a1;
                }
            }
        }
        else
        {
        a1:
            c = 0;
            a = 0;
            while (st[k][0] != st[a][0])
            {
                a++;
            }
            while ((fol[a][c] != '\0') && (st[a][0] != st[i][0]))
            {
                for (m = 0; m < l; m++)
                {
                    if (fol[i][m] == fol[a][c])
                        goto q1;
                }
                fol[i][l] = fol[a][c];
                l++;
            q1:
                c++;
            }
        }
        goto k1;
    }
    fol[i][l] = '\0';
}
printf("follow pos\n");
for (i = 0; i < n; i++)
    printf("FOLLOW[%c]=%s\n", st[i][0], fol[i]);
printf("\n");
```

```
    s = 0;
    for (i = 0; i < n; i++)
    {
        j = 3;
        while (st[i][j] != '\0')
        {
            if ((st[i][j - 1] == '|') || (j == 3))
            {
                for (p = 0; p <= 2; p++)
                {
                    fin[s][p] = st[i][p];
                }
                t = j;
                for (p = 3; ((st[i][j] != '|') && (st[i][j] != '\0'));
p++)
                {
                    fin[s][p] = st[i][j];
                    j++;
                }
                fin[s][p] = '\0';
                if (st[i][t] == 'e')
                {
                    b = 0;
                    a = 0;
                    while (st[a][0] != st[i][0])
                    {
                        a++;
                    }
                    while (fol[a][b] != '\0')
                    {
                        printf("M[%c,%c]=%s\n", st[i][0], fol[a][b],
fin[s]);
                        b++;
                    }
                }
                else if (!((st[i][t] > 64) && (st[i][t] < 91)))
                    printf("M[%c,%c]=%s\n", st[i][0], st[i][t],
fin[s]);
                else
                {
                    b = 0;
                    a = 0;
                    while (st[a][0] != st[i][3])
                    {
```

```
                        a++;
                    }
                    while (ft[a][b] != '\0')
                    {
                        printf("M[%c,%c]=%s\n", st[i][0], ft[a][b],
fin[s]);
                        b++;
                    }
                }
                s++;
            }
            if (st[i][j] == '|')
                j++;
        }
    }
    return 0;
}
```

Output:

```
enter the no. of productions
3
enter the productions in a grammar
S->A|a
A->a|e
B->b|e
first pos
FIRS[S]=ae
FIRS[A]=ae
FIRS[B]=be
follow pos
FOLLOW[S]=$
FOLLOW[A]=$
FOLLOW[B]=

M[S,a]=S->A
M[S,e]=S->A
M[S,a]=S->a
M[A,a]=A->a
M[A,$]=A->e
M[B,b]=B->b
```

3. Shift Reduce Parser

Observation:
Here we written a CPP program for shift reduced parser for the given productions.

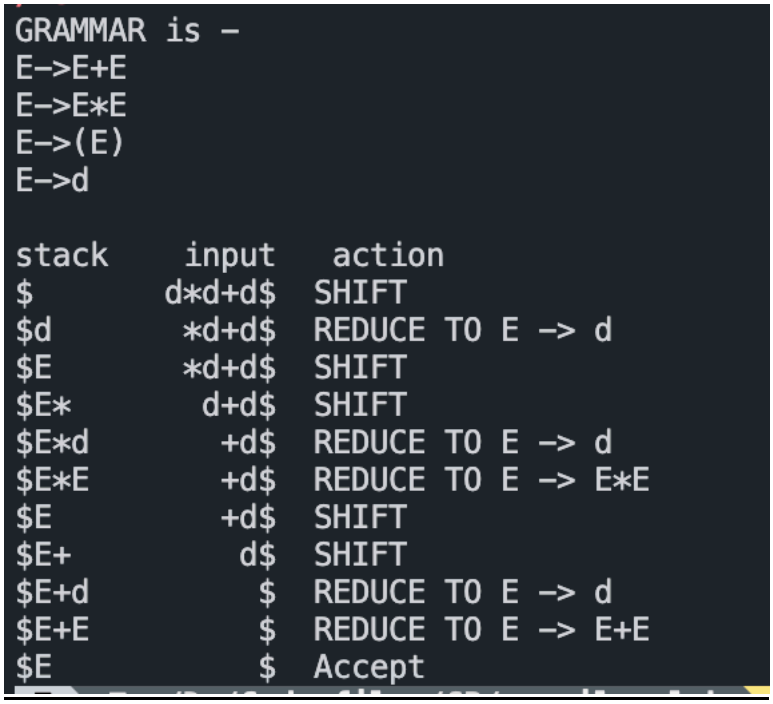
Source Code:

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

int z = 0, i = 0, j = 0, c = 0;
char a[16], ac[20], stk[15], act[10];
void check()
{
    strcpy(ac, "REDUCE TO E -> ");
    for (z = 0; z < c; z++)
    {
        if (stk[z] == 'd')
        {
            printf("%sd", ac);
            stk[z] = 'E';
            stk[z + 1] = '\0';
            printf("\n$%s\t%s$\t", stk, a);
        }
    }
    for (z = 0; z < c - 2; z++)
    {
        if (stk[z] == '(' && stk[z + 1] == 'E' && stk[z + 2] == ')')
        {
            printf("%s(E)", ac);
            stk[z] = 'E';
            stk[z + 1] = '\0';
            stk[z + 2] = '\0';
            printf("\n$%s\t%s$\t", stk, a);
            i = i - 2;
        }
    }
    for (z = 0; z < c - 2; z++)
    {
        if (stk[z] == 'E' && stk[z + 1] == '*' && stk[z + 2] == 'E')
        {
            printf("%sE*E", ac);
            stk[z] = 'E';
```

```
            stk[z + 1] = '\0';
            stk[z + 1] = '\0';
            printf("\n$%s\t%s$\t", stk, a);
            i = i - 2;
        }
    }
    for (z = 0; z < c - 2; z++)
    {
        if (stk[z] == 'E' && stk[z + 1] == '+' && stk[z + 2] == 'E')
        {
            printf("%sE+E", ac);
            stk[z] = 'E';
            stk[z + 1] = '\0';
            stk[z + 1] = '\0';
            printf("\n$%s\t%s$\t", stk, a);
            i = i - 2;
        }
    }
    return;
}
int main()
{
    printf("GRAMMAR is -\nE->E+E \nE->E*E \nE->(E)\nE->d\n");
    strcpy(a, "d*d+d");
    c = strlen(a);
    strcpy(act, "SHIFT");
    printf("\nstack \t input \t action");
    printf("\n$\t%s$\t", a);
    for (i = 0; j < c; i++, j++)
    {
        printf("%s", act);
        stk[i] = a[j];
        stk[i + 1] = '\0';
        a[j] = ' ';
        printf("\n$%s\t%s$\t", stk, a);
        check();
    }
    check();
    if (stk[0] == 'E' && stk[1] == '\0')
        printf("Accept\n");
    else
        printf("Reject\n");
}
```

Output:



```
GRAMMAR is -
E->E+E
E->E*E
E->(E)
E->d

stack    input   action
$       d*d+d$  SHIFT
$d       *d+d$  REDUCE TO E -> d
$E       *d+d$  SHIFT
$E*       d+d$  SHIFT
$E*d       +d$  REDUCE TO E -> d
$E*E       +d$  REDUCE TO E -> E*E
$E         +d$  SHIFT
$E+         d$  SHIFT
$E+d          $  REDUCE TO E -> d
$E+E          $  REDUCE TO E -> E+E
$E            $  Accept
```

4. LALR Parser

Observation:
Here we written a CPP program for LALR parser for the given productions.

LEX Code:

```
%{
#include "parser.tab.h" %}
%%
[0-9]+ {yylval = atoi(yytext); return NUMBER;
}
[\t];
\n return 0;
. return yytext[0];
%%
```

YACC Code:

```
%{
#include<stdio.h> %}
%token NUMBER
%%
s: E { printf("The result is =%d\n",$1);} ;
E: E'+'T { $$ = $1 + $3; }
| T {$$ = $1;}
;
T: T'*'F {$$ = $1 * $3; }
| F {$$ = $1;}
;
F: '('E')' { $$ = $2; }
| NUMBER { $$ = $1; } ;
%%
int main(){
yyparse();
}
int yywrap(){
return 1;
}
void yyerror(char *s){ printf("Error %s",s);
}
```

Output:

```
9+8+6*8
The result is =65
```

```
52*57+65
The result is =3029
```

5. Parser2

LEX Code:

```
%{
#include "parser2.tab.h" %}
%%
[0-9]+ {yylval = atoi(yytext); return NUMBER;
}
[\t];
\n return 0;
. return yytext[0]; %%
```

YACC Code:

```
%{
#include<stdio.h> %}
%token NUMBER
%%
S: E { printf("The result is =%d\n",$1);} ;
E: E'+'E { $$ = $1 + $3; }
; E: E'-'E {$$ = $1 - $3;}
; E: E'*'E {$$ = $1 * $3;}
; E: E'/'E {$$ = $1 / $3;}
; E: '-'E {$$ = -$1;}
; E: '('E')' { $$ = $2; }
| NUMBER { $$ = $1; } ;
%%
int main(){
yyparse();
}
int yywrap(){
return 1;
```

```
}
void yyerror(char *s){ printf("Error %s",s);
}
```

Output:

```
8+9*50
The result is =458
```

```
56*84+56
The result is =7840
```