SRM UNIVERSITY AP
Amaravati

Department of Computer Science and Engineering

Compiler Design Lab (CS 306)

Chaitanya Sai Nutakki

AP21110010253

Section-D

List of Lab Exercises

Week 1: Language Recognizer

1. Write a program in C that recognizes the following languages.
   a. Set of all strings over binary alphabet containing even number of 0's and even number of 1's.
   b. **Lab Assignment:** Set of all strings ending with two symbols of the same type.

```
#include<stdio.h>
#include<stdlib.h>
#define max 100
int main(){
      char str[max];
      int c1=0,c0=0;
      FILE *fp1 = fopen("q.txt", "r");

      fgets(str, max,fp1);

      for (int i = 0; str[i] != '\0'; i++)
      {
      if(str[i]=='1'){
            c1++;
      }
      if(str[i]=='0'){
            c0++;
      }
      }
      printf("%d %d",c0,c1);
      FILE *fp2 = fopen("w.txt", "w");
      if (c1%2==0 && c0%2==0)
```

```
      fprintf(fp2, "String is accepted, it has even number of 0's and even
number of 1's");
      else
      fprintf(fp2, "String is not accepted, it does not have even number of
0's and even number of 1's");

      fclose(fp1);
      fclose(fp2);
      return 0;
}
```

Input	1010
Output	String is not accepted, it does not have even number of 0's and even number of 1's

Week 2: Implementation of Lexical analyzer using C

2. Implement a lexical analyzer using C 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: <, >, <=, >=, ==, !=
   - Storing identifiers in symbol table.
   - Using files for input and output.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_KEYWORDS 100
#define MAX_INDENT_LEN 100

char keywords[MAX_KEYWORDS][MAX_INDENT_LEN]={
      "int",
      "float",
      "char",
      "double",
      "if",
      "else",
```

```
      "for",
      "while",
      "do",
      "switch",
      "main()"
};

int isKeyword(char *str){
      for(int i=0;i<MAX_KEYWORDS;i++){
      if(strcmp(str,keywords[i])==0){
            return 1;
      }
      }
      return 0;
}

int isIdentifier(char *str){
      if(!((str[0]>='a' && str[0]<='z') || (str[0]>='A' && str[0]<='Z'))){
      return 0;
      }
      for(int i=1;i<strlen(str);i++){
      if(!((str[i]>='a' && str[i]<='z') || (str[i]>='A' && str[i]<='Z') ||
(str[i]>='0' && str[i]<='9'))){
            return 0;
      }
      }
      return 1;
}

int isInteger(char *str){
      for(int i=0;i<strlen(str);i++){
      if(!(str[i]>='0' && str[i]<='9')){
            return 0;
      }
      }
      return 1;
}

int isRelationalOperator(char *str){
      if(strcmp(str,"<")==0 || strcmp(str,">")==0 || strcmp(str,"<=")==0 ||
strcmp(str,">=")==0 || strcmp(str,"==")==0 || strcmp(str,"!=")==0){
      return 1;
      }
      return 0;
}
```

```
int isParenthesis(char *str){
      if(strcmp(str,"(")==0 || strcmp(str,")")==0 || strcmp(str,"{")==0 ||
strcmp(str,"}")==0){
      return 1;
      }
      return 0;
}

int main(){
      FILE *fp;
    fp=fopen("input.txt","r");
      if(fp==NULL){
      printf("Error opening file\n");
      exit(0);
      }
      char ch;
      char str[MAX_INDENT_LEN];
      int i=0;
      FILE *fp2=fopen("output.txt","w");
      while((ch=fgetc(fp))!=EOF){
      if(ch==' ' || ch=='\n' || ch=='\t'){
            if(i>0){
                  str[i]='\0';
                  if(isKeyword(str)){
                  // printf("%s is a keyword\n",str);
                    fprintf(fp2,"Keyword             : %s\n",str);
                  }
                  else if(isIdentifier(str)){
                  // printf("%s is an identifier\n",str);
                    fprintf(fp2,"Identifier          : %s\n",str);
                  }
                  else if(isInteger(str)){
                  // printf("%s is an integer\n",str);
                    fprintf(fp2,"Interger            : %s\n",str);
                  }
                  else if(isRelationalOperator(str)){
                  // printf("%s is a relational operator\n",str);
                    fprintf(fp2,"Relational Operator : %s\n",str);
                  }
                  else if(isParenthesis(str)){
                  // printf("%s is a parenthesis\n",str);
                    fprintf(fp2,"Parenthesis         : %s\n",str);
                  }
                  else{
                  // printf("%s is an invalid token\n",str);
                  fprintf(fp2,"Invalid Token    : %s\n",str);
```

```
                }
                i=0;
            }
        }
        else{
            str[i++]=ch;
        }
        }
        fclose(fp);
        fclose(fp2);
        return 0;
}
```

Input	`#include <stdio.h>`
	`int main()`
	`{`
	`    printf("Hello, World!");`
	`}`

Output	Invalid Token : #include Invalid Token : <stdio.h> Keyword : int Keyword : main() Parenthesis : { Invalid Token : printf("Hello, Invalid Token : World!");

Week 3: Introduction to LEX tool

3. Implement the following programs using Lex tool
    a. Identification of Vowels and Consonants
    b. count number of vowels and consonants
    c. Count the number of Lines in given input
    d. Recognize strings ending with 00
    e. Recognize a string with three consecutive 0's

A

```
%{
#include <stdio.h>

int vow_count = 0;
int const_count = 0;
int line_count = 0;
%}
```

B

```
%%
[aeiouAEIOU] {printf("%s is a Vowel\n", yytext); vow_count++;}
[a-zA-Z] {printf("%s is a Consonant\n", yytext); const_count++;}
\n {line_count++;}
"00" {printf("String ends with 00\n");}
"000" {printf("String has three consecutive 0's\n");}
.
%%
```

C

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

int main() {
            printf("Enter String\n");
      yylex();
      printf("Number of vowels are: %d\n", vow_count);
      printf("Number of consonants are: %d\n", const_count);
      printf("Number of lines are: %d\n", line_count);
      return 0;
}
```

D

```
week3-c.l
%{
```

```
int num_lines=0;
%}
%%
\n ++num_lines;
. {}
%%
int main()
{
yylex();
printf("no of lines=%d",num_lines);
}
int yywrap()
{
return 1;
}
```

E

```
%{
/* Definition section */
#include<stdio.h>
#include <stdlib.h>
void yyerror(const char *str)
{
printf("\nSequence Rejected\n");
}

%}

%token ZERO ONE

/* Rule Section */
%%

r : s {printf("\nSequence Accepted\n\n");}
;

s : n
| ZERO a
| ONE b
;

a : n a
| ZERO
;

b : n b
```

```
| ONE
;

n : ZERO
| ONE
;

%%

#include"lex.yy.c"
//driver code
int main()
{
      printf("\nEnter Sequence of Zeros and Ones : ");
      yyparse();
      printf("\n");
      return 0;
}
```

Input	qwerty000
Output	q is a Consonant w is a Consonant e is a Vowel r is a Consonant t is a Consonant y is a Consonant String has three consecutive 0's

Week 4: Implementation of lexical analyzer using LEX

4. 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 (/* …. */)
- Storing identifiers in symbol table
- Using files for input and output.

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

#define MAX_IDENTIFIER_LENGTH 50
#define MAX_CONSTANT_LENGTH 11 // Maximum integer constant length (10 digits
+ '\0')

int line_number = 1; // For tracking line numbers
%}

/* Regular expressions for tokens */
letter            [a-zA-Z]
digit             [0-9]

/* Token definitions */
%%
int               { printf("INT\n"); }
"{"               { printf("LBRACE\n"); }
"}"               { printf("RBRACE\n"); }
";"               { printf("SEMICOLON\n"); }
"="               { printf("ASSIGN\n"); }
"("               { printf("LPAREN\n"); }
")"               { printf("RPAREN\n"); }
"["               { printf("LBRACKET\n"); }
"]"               { printf("RBRACKET\n"); }
"if"              { printf("IF\n"); }
"then"            { printf("THEN\n"); }
"else"            { printf("ELSE\n"); }
"endif"           { printf("ENDIF\n"); }
"while"           { printf("WHILE\n"); }
"do"              { printf("DO\n"); }
"enddo"           { printf("ENDDO\n"); }
"print"           { printf("PRINT\n"); }
{letter}{letter}* {
      if (strlen(yytext) > MAX_IDENTIFIER_LENGTH) {
      fprintf(stderr, "Error: Identifier too long at line %d\n",
```

```
line_number);
      exit(1);
      }
      printf("IDENTIFIER: %s\n", yytext);
}

{digit}+ {
      if (strlen(yytext) > MAX_CONSTANT_LENGTH) {
      fprintf(stderr, "Error: Integer constant too long at line %d\n",
line_number);
      exit(1);
      }
      printf("CONSTANT: %s\n", yytext);
}

"<"               { printf("RELOP: LT\n"); }
"<="              { printf("RELOP: LE\n"); }
"=="              { printf("RELOP: EQ\n"); }
">="              { printf("RELOP: GE\n"); }
">"               { printf("RELOP: GT\n"); }
"!="              { printf("RELOP: NE\n"); }
"-"               { printf("ADDOP: SUB\n"); }
"+"               { printf("ADDOP: ADD\n"); }
"*"               { printf("MULOP: MUL\n"); }
"/"               { printf("MULOP: DIV\n"); }
%%
int main() {
      yylex();
      return 0;
}
int yywrap(){
return 1;
}
```

Input	

```
#include <stdio.h>
int main()
{
      int a=10+20;
      if(a>10){
        printf("qq");
      }
    printf("Hello, World!");
}
```


Output	#IDENTIFIER: include
	
	RELOP: LT
	
	IDENTIFIER: stdio
	
	.IDENTIFIER: h
	
	RELOP: GT
	
	int main()
	
	INT
	
	IDENTIFIER: main
	
	LPAREN
	
	RPAREN
	
	{
	
	LBRACE
	
	int a=10+20;
	
	INT
	
	IDENTIFIER: a
	
	ASSIGN

	CONSTANT: 10
	
	ADDOP: ADD
	
	CONSTANT: 20
	
	SEMICOLON
	
	if(a>10){
	
	IF
	
	LPAREN
	
	IDENTIFIER: a
	
	RELOP: GT
	
	CONSTANT: 10
	
	RPAREN
	
	LBRACE
	
	printf("qq");
	
	IDENTIFIER: printf
	
	LPAREN
	
	"IDENTIFIER: qq
	
	"RPAREN

	SEMICOLON
	
	}
	
	RBRACE
	
	printf("Hello, World!");
	
	IDENTIFIER: printf
	
	LPAREN
	
	"IDENTIFIER: Hello
	
	, IDENTIFIER: World
	
	!"RPAREN
	
	SEMICOLON
	
	}

Week 5: Lexical Analyzer

5. Lab Assignment:

Consider the following mini Language, a simple procedural high-level language, only operating on integer data, with a syntax looking vaguely like a simple C crossed with Pascal. The syntax of the language is definedby the following BNF grammar:

<program> ::= <block>

<block> ::= { <variabledefinition> <slist> } | { <slist> }

<variabledefinition> ::= int<vardeflist>;

<vardeflist> ::= <vardec> | <vardec>, <vardeflist>

<vardec> ::= <identifier> | <identifier> [<constant>]

<slist> ::= <statement> | <statement>; <slist>

<statement> ::= <assignment> | <ifstatement> | <whilestatement> | <block> | <printstatement> | <empty>

<assignment> ::= <identifier> = <expression> | <identifier> [<expression>] = <expression>

<ifstatement> ::= <bexpression> then <slist> else <slist> endif | if <bexpression> then <slist> endif

<whilestatement> ::= while <bexpression> do <slist> enddo

<printstatement> ::= print (<expression>)

<expression> ::= <expression> <additionop> <term> | <term> | addingop> <term>

<bexpression> ::= <expression> <relop> <expression>

<relop> ::= < | <= | == | >= | > | !=

<addingop> ::= + | -

<term> ::= <term><mulitop> <factor> | <factor>

<multop> ::= * | /

<factor> ::= <constant> | <identifier> | <identifier> [<expression>] | (<expression>)

<constant> ::= <digit> | <digit> <constant>

<identifier> ::= <identifier> <letterordigit> | <letter>

<letterordigit> ::= <letter> | <digit>

<letter> ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

<digit> ::= 0|1|2|3|4|5|6|7|8|9

<empty> has the obvious meaning

Comments (zero or more characters enclosed between the standard C / Java style comment brackets /*...*/) can be inserted. The language has rudimentary support for 1-dimensional arrays. The declaration int a[3] declares an array of three elements, referenced as a[0], a[1] and a[2]. Note also that you should worry about the scoping of names.

A simple program written in this language is:

```
{ int a[3], t1, t2;
  t1 = 2; a[0] = 1; a[1] = 2; a[t1] = 3;
  t2 = -(a[2] + t1 * 6)/ a[2] -t1);
  if t2 > 5 then
    print(t2);
```

```
  else  {
    int t3;
    t3 = 99;
    t2 = -25;
    print(-t1 + t2 * t3); /* this is a comment on 2 lines */
  } endif
}
```

Design a Lexical analyser for the above language. The lexical analyser should ignore redundant spaces, tabs, and newlines. It should also ignore comments. Although the syntax specification states that identifiers can be arbitrarily long, you may restrict the length to some reasonable value.

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

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)
      printf("\n%s is a keyword", str);
  else
      printf("\n%s is an identifier", str);
}

int main() {
  FILE *f1, *f2, *f3;
  char c, str[10], st1[10];
  int num[100], lineno = 0, tokenvalue = 0, i = 0, j = 0, k = 0;

  f1 = fopen("q.txt", "r");

  if (f1 == NULL) {
      printf("Failed to open the input file.\n");
      return 1;  // Exit with an error code
  }

  printf("\nThe numbers in the program are");
  while ((c = getc(f1)) != EOF) {
      if (isdigit(c)) {
      tokenvalue = c - '0';
      c = getc(f1);
```

```
    while (isdigit(c)) {
    tokenvalue = tokenvalue * 10 + (c - '0');
    c = getc(f1);
    }
    num[i++] = tokenvalue;
    ungetc(c, f1);
    } else if (isalpha(c)) {
    f2 = fopen("identifier", "a");
    if (f2 == NULL) {
    printf("Failed to open the identifier file.\n");
    return 1; // Exit with an error code
    }
    putc(c, f2);
    c = getc(f1);
    while (isdigit(c) || isalpha(c) || c == '_' || c == '$') {
    putc(c, f2);
    c = getc(f1);
    }
    putc(' ', f2);
    fclose(f2);
    ungetc(c, f1);
    } else if (c == ' ' || c == '\t') {
    printf(" ");
    } else if (c == '\n') {
    lineno++;
    } else {
    f3 = fopen("specialchar", "a");
    if (f3 == NULL) {
    printf("Failed to open the specialchar file.\n");
    return 1;
    }
    putc(c, f3);
    fclose(f3);
    }
}

fclose(f1);

for (j = 0; j < i; j++)
    printf("%d ", num[j]);
printf("\n");

f2 = fopen("identifier", "r");

if (f2 == NULL) {
    printf("Failed to open the identifier file.\n");
```

```
        return 1;
    }

    k = 0;
    printf("The keywords and identifiers are:");
    while ((c = getc(f2)) != EOF) {
        if (c != ' ')
        str[k++] = c;
        else {
        str[k] = '\0';
        keyword(str);
        k = 0;
        }
    }

    fclose(f2);

    f3 = fopen("specialchar", "r");

    if (f3 == NULL) {
        printf("Failed to open the specialchar file.\n");
        return 1;
    }

    printf("\nSpecial characters are: ");
    while ((c = getc(f3)) != EOF)
        printf("%c", c);
    printf("\n");

    fclose(f3);

    printf("Total number of lines are: %d\n", lineno);

    return 0;
}
```

Input	

```
{ int a[3], t1, t2;
t1 = 2; a[0] = 1; a[1] = 2; a[t1] = 3;

t2 = -(a[2] + t1 * 6)/ a[2] -t1);
if t2 > 5 then
print(t2);
else {
int t3;
t3 = 99;
t2 = -25;
print(-t1 + t2 * t3); /* this is a comment on 2 lines */
} endif
}
```

Output	Identifier:
	
	include stdio h int main Write C code here printf Hello world return int a t1 t2 t1 a a a t1 t2 a t1 a t1 if t2 gt then print t2 else int t3 t3 t2 print t1 t2 t3 this is a comment on lines endif int a t1 t2 t1 a a a t1 t2 a t1 a t1 if t2 gt then print t2 else int t3 t3 t2 print t1 t2 t3 this is a comment on lines endif
	
	Special Char:
	
	#<.>(){//("");;}{[],,;
	
	=;[]=;[]=;[]=;
	
	=-([]+*)/[]-);
	
	&;
	
	();
	
	{
	
	;
	
	=;
	
	=-;
	
	(-+*);/**/
	
	}

	} Relational Operators: =======

Week 6: Recursive Descent Parser

6. Implement Recursive Descent Parser for the Expression Grammar given below.

   E □ TE'

   E'□ +TE' | ϵ

   T □ FT'

   T'□ *FT' | ϵ

   F □ (E) | i

7. **Lab Assignment:** Construct Recursive Descent Parser for the grammar G = ({S, L}, {(,), a, ,}, {S □ (L) | a ; L□ L, S | S}, S) and verify the acceptability of the following strings:
   i. (a,(a,a))
   ii. (a,((a,a),(a,a)))

   You can manually eliminate Left Recursion if any in the grammar.

```
#include <iostream>
#include <cctype>

using namespace std;

bool parseE();
bool parseEPrime();
bool parseT();
bool parseTPrime();
bool parseF();

string input;
size_t pos = 0;

void consume() {
```

```
      if (pos < input.length()) {
      pos++;
      }
}

bool isIdentifier(char c) {
      return isalpha(c);
}

bool isOperator(char c) {
      return (c == '+' || c == '*');
}

bool parseF() {
      if (isIdentifier(input[pos])) {
      consume();
      return true;
      } else if (input[pos] == '(') {
      consume();
      if (parseE() && input[pos] == ')') {
            consume();
            return true;
      }
      }
      return false;
}

bool parseTPrime() {
      if (input[pos] == '*') {
      consume();
      if (parseF() && parseTPrime()) {
            return true;
      }
      }
      return true;
}

bool parseT() {
      if (parseF() && parseTPrime()) {
      return true;
      }
      return false;
}

bool parseEPrime() {
      if (input[pos] == '+') {
```

```
        consume();
        if (parseT() && parseEPrime()) {
                return true;
        }
        }
        return true;
}

bool parseE() {
        if (parseT() && parseEPrime()) {
        return true;
        }
        return false;
}

int main() {
        cout << "Enter an expression: ";
        cin >> input;

        if (parseE() && pos == input.length()) {
        cout << "Valid Expression" << endl;
        } else {
        cout << "Invalid Expression" << endl;
        }

        return 0;
}
```

Input	id+id*id
Output	Enter an expression: Invalid Expression

Week 7: Predictive parser

8. Write a C program for the computation of FIRST and FOLLOW for a given CFG

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

void followfirst(char, int, int);
```

```
void follow(char c);
void findfirst(char, int, int);

int count, n = 0;
char calc_first[10][100];
char calc_follow[10][100];
int m = 0;
char production[10][10];
char f[10], first[10];
int k;
char ck;
int e;

int main(int argc, char** argv)
{
            int jm = 0;
            int km = 0;
            int i, choice;
            char c, ch;
            count = 8;

            int nn;scanf("%d",&nn);char cc[100];
            for(int ii=0;i<nn;i++){
                scanf("%s",cc);
                strcpy(production[ii], cc);
                printf("%s\n",production[ii]);
            }

            int kay;
            char done[count];
            int ptr = -1;

            for (k = 0; k < count; k++) {
                        for (kay = 0; kay < 100; kay++) {
                                    calc_first[k][kay] = '!';
                        }
            }
            int point1 = 0, point2, temp;

            for (k = 0; k < count; k++) {
                        c = production[k][0];
                        point2 = 0;
                        temp = 0;

                        for (kay = 0; kay <= ptr; kay++)
                                    if (c == done[kay])
```

```
                                        temp = 1;

                        if (temp == 1)
                                continue;

                        findfirst(c, 0, 0);
                        ptr += 1;

                        done[ptr] = c;
                        printf("\n First(%c) = { ", c);
                        calc_first[point1][point2++] = c;

                        for (i = 0 + jm; i < n; i++) {
                                int lark = 0, chk = 0;

                                for (lark = 0; lark < point2; lark++) {

                                        if (first[i] ==
calc_first[point1][lark]) {
                                                chk = 1;
                                                break;
                                        }
                                }
                                if (chk == 0) {
                                        printf("%c, ", first[i]);
                                        calc_first[point1][point2++]
= first[i];
                                }
                        }
                        printf("}\n");
                        jm = n;
                        point1++;
                }
                printf("\n");
                printf("-----------------------------------------------"
                        "\n\n");
                char donee[count];
                ptr = -1;

                for (k = 0; k < count; k++) {
                        for (kay = 0; kay < 100; kay++) {
                                calc_follow[k][kay] = '!';
                        }
                }
                point1 = 0;
                int land = 0;
```

```
            for (e = 0; e < count; e++) {
                        ck = production[e][0];
                        point2 = 0;
                        temp = 0;

                        for (kay = 0; kay <= ptr; kay++)
                                    if (ck == donee[kay])
                                                temp = 1;

                        if (temp == 1)
                                    continue;
                        land += 1;

                        follow(ck);
                        ptr += 1;

                        donee[ptr] = ck;
                        printf(" Follow(%c) = { ", ck);
                        calc_follow[point1][point2++] = ck;

                        for (i = 0 + km; i < m; i++) {
                                    int lark = 0, chk = 0;
                                    for (lark = 0; lark < point2; lark++) {
                                                if (f[i] ==
calc_follow[point1][lark]) {
                                                            chk = 1;
                                                            break;
                                                }
                                    }
                                    if (chk == 0) {
                                                printf("%c, ", f[i]);

calc_follow[point1][point2++] = f[i];
                                    }
                        }
                        printf(" }\n\n");
                        km = m;
                        point1++;
            }
}

void follow(char c)
{
            int i, j;

            if (production[0][0] == c) {
```

```
                        f[m++] = '$';
            }
            for (i = 0; i < 10; i++) {
                        for (j = 2; j < 10; j++) {
                                    if (production[i][j] == c) {
                                                if (production[i][j + 1] !=
'\0') {

followfirst(production[i][j + 1], i,

      (j + 2));
                                                }

                                                if (production[i][j + 1] ==
'\0'
                                                            && c !=
production[i][0]) {

follow(production[i][0]);
                                                }
                                    }
                        }
            }
}

void findfirst(char c, int q1, int q2)
{
            int j;

            if (!(isupper(c))) {
                        first[n++] = c;
            }
            for (j = 0; j < count; j++) {
                        if (production[j][0] == c) {
                                    if (production[j][2] == '@') {
                                                if (production[q1][q2] ==
'\0')
                                                            first[n++]
= '@';
                                                else if (production[q1][q2]
!= '\0'

      && (q1 != 0 || q2 != 0)) {

findfirst(production[q1][q2], q1,
```

```
      (q2 + 1));
                                                }
                                                else
                                                                first[n++]
= '@';
                                    }
                                    else if (!isupper(production[j][2])) {
                                                first[n++] =
production[j][2];
                                    }
                                    else {
                                                findfirst(production[j][2],
j, 3);
                                    }
                        }
            }
}

void followfirst(char c, int c1, int c2)
{
            int k;

            if (!(isupper(c)))
                        f[m++] = c;
            else {
                        int i = 0, j = 1;
                        for (i = 0; i < count; i++) {
                                    if (calc_first[i][0] == c)
                                                break;
                        }

                        while (calc_first[i][j] != '!') {
                                    if (calc_first[i][j] != '@') {
                                                f[m++] = calc_first[i][j];
                                    }
                                    else {
                                                if (production[c1][c2] ==
'\0') {

follow(production[c1][0]);
                                                }
                                                else {

followfirst(production[c1][c2], c1,

      c2 + 1);
```

```
                                        }
                                    }
                                    j++;
                                }
                            }
                        }
```

Input	**4** **S=AaAb** **S=BbBa** **A=@** **B=@**
Output	S=AaAb S=BbBa A=@ B=@ First(B) = { @, } First() = { } ---------- Follow(B) = { $, a, } Follow() = { a, $, }

Week 8: Predictive Parser

9. Implement non-recursive Predictive Parser for the grammar

   S -> aBa

   B -> bB | ε

	A	b	$
S	S□aBa		
B	B□ε	B□bB	

10. **Lab Assignment:** Implement Predictive Parser using C for the Expression Grammar

    E □ TE’

    E’□ +TE’ | ε

    T □ FT’

    T’□ *FT’ | ε

    F □ (E) | d

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

char input[100];
int pointer = 0;
int error = 0;

void E();
void Eprime();
void T();
void Tprime();
void F();

void match(char c) {
      if (input[pointer] == c) {
      pointer++;
      } else {
      error = 1;
      }
}
```

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

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

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

void Tprime() {
      if (input[pointer] == '*') {
      match('*');
      F();
      Tprime();
      }
}

void F() {
      if (input[pointer] == '(') {
      match('(');
      E();
      match(')');
      } else if (isalnum(input[pointer])) {
      match(input[pointer]);
      } else {
      error = 1;
      }
}

int main() {
      printf("Enter an expression: ");
      scanf("%s", input);
      E();
      printf("%d\n",pointer);
      if (error == 0 && input[pointer] == '\0') {
      printf("Accepted\n");
      } else {
```

```
        printf("Rejected\n");
        }

        return 0;
}
```

Input	d*d+d
Output	Enter an expression: 5 Accepted

Week 9: Shift Reduce Parser

11. Implementation of Shift Reduce parser using C for the following grammar and illustrate the parser's actions for a valid and an invalid string.

    E□E+E

    E□E*E

    E□(E)

    E□d

12. **Lab Assignment:** Implementation of Shift Reduce parser using C for the following grammar and illustrate the parser's actions for a valid and an invalid string.

    S –> 0S0 | 1S1 | 2

```
#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 S -> ");

        for (z = 0; z < c; z++)
```

```
	{
	if (stk[z] == '2')
	{
		printf("%s2", ac);
		stk[z] = 'S';
		stk[z + 1] = '\0';

		printf("\n$%s\t%s$\t", stk, a);
	}
	}

	for (z = 0; z < c - 2; z++)
	{
	if (stk[z] == '0' && stk[z + 1] == 'S' &&
		stk[z + 2] == '0')
	{
		printf("%s0S0", ac);
		stk[z] = 'S';
		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] == '1' && stk[z + 1] == 'S' &&
		stk[z + 2] == '1')
	{
		printf("%s1S1", ac);
		stk[z] = 'S';
		stk[z + 1] = '\0';
		stk[z + 2] = '\0';
		printf("\n$%s\t%s$\t", stk, a);
		i = i - 2;
	}
	}
	return;
}

int main()
{
	printf("GRAMMAR is -\nS->0S0 \nS->1S1 \nS->2\n");

	strcpy(a, "0102010");
```

```
      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] == 'S' && stk[1] == '\0')
      printf("Accept\n");
      else
      printf("Reject\n");

      return 0;
}
```

Input	01210

Output	GRAMMAR is - S->0S0 S->1S1 S->2 stack input action $ 0102010$ SHIFT $0 102010$ SHIFT $01 02010$ SHIFT $010 2010$ SHIFT $0102 010$ REDUCE TO S -> 2 $010S 010$ SHIFT $010S0 10$ REDUCE TO S -> 0S0 $01S 10$ SHIFT $01S1 0$ REDUCE TO S -> 1S1 $0S 0$ SHIFT $0S0 $ REDUCE TO S -> 0S0 $S $ Accept