J.N.T.U.H UNIVERSITY COLLEGE OF ENGINEERING SCIENCE AND TECHNOLOGY HYDERABAD, KUKATPALLY, HYDERABAD – 500085

CERTIFICATE

This is to certify that **KOLLURU ANUDEEPIKA** of CSE(Regular) IV year, I Semester bearing with Hall-Ticket number **22011A0538** has fulfilled her **COMPILER DESIGN LAB** record for the academic year 2025-2026.

Signature of the HOD ***Signature of the Staff***

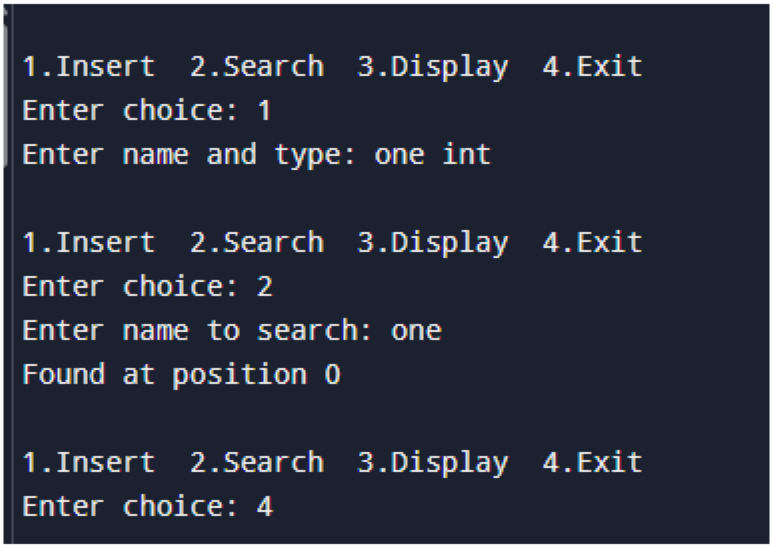
Internal Examiner ***External Examiner***

INDEX

14	Write a C program to construct Recursive Descentparser for the following grammar	18
15	write recursive descent parser for the grammar S->(L) S->a L->L,S L->S	20
16	. Write a C program to calculate first function for the grammar E->E+T E->T T->T*F T->F F->(E)/id	21

1. Implementation of symbol table.

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

#define MAX 50

struct Symbol {
    char name[30];
    char type[10];
} table[MAX];

int count = 0;

/* Insert a new symbol */
void insert(char name[], char type[]) {
    strcpy(table[count].name, name);
    strcpy(table[count].type, type);
    count++;
}

/* Search for a symbol */
int search(char name[]) {
    for (int i = 0; i < count; i++) {
        if (strcmp(table[i].name, name) == 0)
            return i;
    }
    return -1;
}

/* Display table */
void display() {
    printf("\nSymbol Table:\n");
    for (int i = 0; i < count; i++)
        printf("%s\t%s\n", table[i].name, table[i].type);
}

int main() {
    int choice;
    char name[30], type[10];

    while (1) {
        printf("\n1.Insert  2.Search  3.Display  4.Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        if (choice == 1) {
            printf("Enter name and type: ");
            scanf("%s %s", name, type);
            insert(name, type);
        }
        else if (choice == 2) {
            printf("Enter name to search: ");
            scanf("%s", name);
            int pos = search(name);
            if (pos == -1)
                printf("Not found\n");
            else
                printf("Found at position %d\n", pos);
        }
        else if (choice == 3) {
            display();
        }
        else
            break;
    }
    return 0;
}
```



```
1.Insert  2.Search  3.Display  4.Exit
Enter choice: 1
Enter name and type: one int

1.Insert  2.Search  3.Display  4.Exit
Enter choice: 2
Enter name to search: one
Found at position 0

1.Insert  2.Search  3.Display  4.Exit
Enter choice: 4
```

2. Write a C program to design a lexical analyzer that recognizes identifiers and keywords of flow control statements of C language

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

int main() {
  int i, flag = 0;
  char str[50];

  printf("Enter string: ");
  scanf("%s", str);

  // Check for control-flow keywords
  if( strcmp(str,"if")==0 || strcmp(str,"else")==0 || strcmp(str,"do")==0 ||
    strcmp(str,"for")==0 || strcmp(str,"break")==0 || strcmp(str,"while")==0 ||
    strcmp(str,"switch")==0 || strcmp(str,"case")==0 || strcmp(str,"default")==0 )
  {
    printf("Keyword of control flow statements\n");
    return 0;
  }
  // Check for identifier
  if (isalpha(str[0]) || str[0] == '_') {
    flag = 0;
    for (i = 1; i < strlen(str); i++) {
      if (!(isalnum(str[i]) || str[i] == '_')) {
        flag = 1;   // invalid character found
```

```
            break;
          }
        }

        if (flag == 0)
          printf("Identifier\n");
        else
          printf("Not a keyword or identifier\n");
    }
    else {
        printf("Not a keyword or identifier\n");
    }

    return 0;
}
```

Output 1:

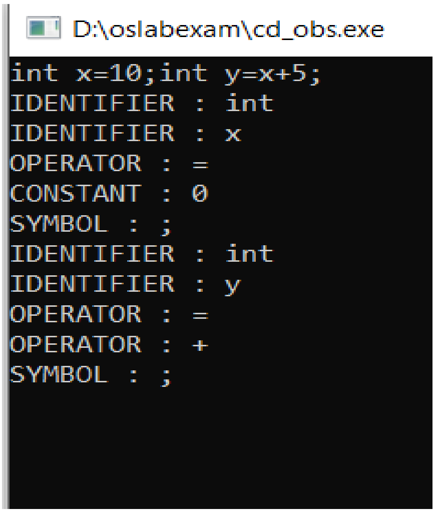
```
enter string
abcd
identifier
```

3) Write a C program to design a lexical analyzer that recognizes identifiers,constants,comments,operatorsetc

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

void printToken(char *type, char *value){
printf("%s : %s\n", type, value);
}

int main(){
  int c;
char buf[100];
int i;

while((c = getchar()) != EOF){
    if(isspace(c))
      continue;
```

```
if(isalpha(c)){
i = 0;
buf[i++] = c;
while(isalnum(c = getchar()))
buf[i++] = c;
buf[i] = '\0';
ungetc(c, stdin);

printToken("IDENTIFIER", buf);
      continue;
    }
if(isdigit(c)){
i = 0;
buf[i++] = c;
while(isdigit(c = getchar()))
buf[i++] = c;
buf[i] = '\0';
ungetc(c, stdin);

printToken("CONSTANT", buf);
      continue;
    }
if(c == '/'){
int d = getchar();

      /* Single-line comment // */
if(d == '/'){
while((c = getchar()) != '\n' && c != EOF);
printToken("COMMENT", "single-line");
      }
else if(d == '*'){
int prev = 0;
while((c = getchar()) != EOF){
if(prev == '*' && c == '/') break;
prev = c;
        }
printToken("COMMENT", "multi-line");
      }
      else {
ungetc(d, stdin);
printf("OPERATOR : /\n");
      }
      continue;
    }
int d = getchar();
```

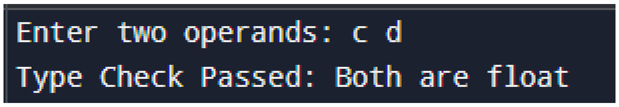
```
if(c=='+' || c=='-' || c=='*' || c=='=' || c=='<' || c=='>' ){
printf("OPERATOR : %c\n", c);
          continue;
     }
else{
ungetc(d, stdin);
      }
printf("SYMBOL : %c\n", c);
   }

   return 0;
}
```



```
int x=10;int y=x+5;
IDENTIFIER : int
IDENTIFIER : x
OPERATOR : =
CONSTANT : 0
SYMBOL : ;
IDENTIFIER : int
IDENTIFIER : y
OPERATOR : =
OPERATOR : +
SYMBOL : ;
```

4. Implement type checking

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

struct Symbol {
   char name[20];
   char type[10];
} table[] = {
   {"a", "int"},
   {"b", "int"},
   {"c", "float"},
   {"d", "float"}
};

char* getType(char var[]) {
   for (int i = 0; i < 4; i++)
     if (strcmp(table[i].name, var) == 0)
       return table[i].type;
   return "unknown";
}

int main() {
   char op1[20], op2[20];

   printf("Enter two operands: ");
   scanf("%s %s", op1, op2);

   char *t1 = getType(op1);
   char *t2 = getType(op2);

   if (strcmp(t1, t2) == 0)
     printf("Type Check Passed: Both are %s\n", t1);
   else
     printf("Type Error: %s is %s, but %s is %s\n", op1, t1, op2, t2);

   return 0;
}
```

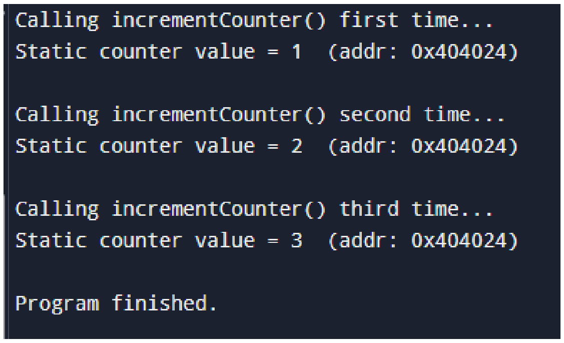


```
Enter two operands: c d
Type Check Passed: Both are float
```

5. Implement any one storage allocation strategies (heap, stack, static)

Static

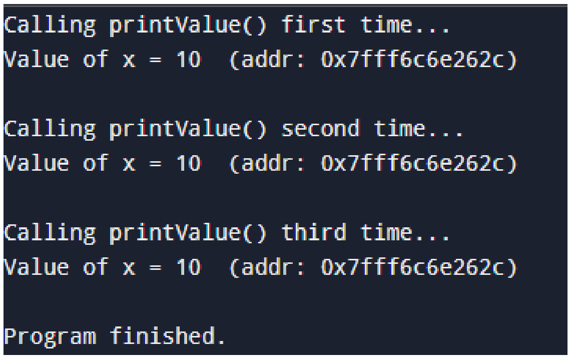
```
#include <stdio.h>

void incrementCounter() {
static int counter = 0;
counter++;
printf("Static counter value = %d  (addr: %p)\n", counter, (void *)&counter);
}

int main(void) {

printf("Calling incrementCounter() first time...\n");
incrementCounter();

printf("\nCallingincrementCounter() second time...\n");
incrementCounter();

printf("\nCallingincrementCounter() third time...\n");
incrementCounter();

printf("\nProgram finished.\n");

return 0;
}
```



```
Calling incrementCounter() first time...
Static counter value = 1  (addr: 0x404024)

Calling incrementCounter() second time...
Static counter value = 2  (addr: 0x404024)

Calling incrementCounter() third time...
Static counter value = 3  (addr: 0x404024)

Program finished.
```

Stack

```
#include <stdio.h>
voidprintValue() {
int x = 10;   // STACK STORAGE (automatic variable)
              // lives only inside this function
printf("Value of x = %d  (addr: %p)\n", x, (void *)&x);
}

int main(void) {

printf("Calling printValue() first time...\n");
printValue();

printf("\nCallingprintValue() second time...\n");
printValue();

printf("\nCallingprintValue() third time...\n");
printValue();

printf("\nProgram finished.\n");

return 0;
}
```

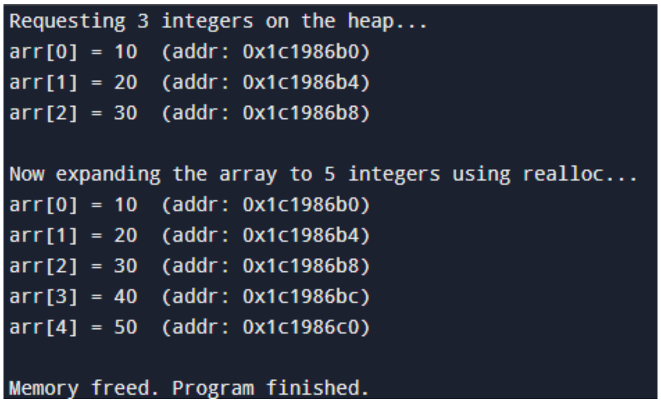


```
Calling printValue() first time...
Value of x = 10  (addr: 0x7fff6c6e262c)

Calling printValue() second time...
Value of x = 10  (addr: 0x7fff6c6e262c)

Calling printValue() third time...
Value of x = 10  (addr: 0x7fff6c6e262c)

Program finished.
```

Heap

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

int main(void) {
intinitial_size = 3;
intnew_size = 5;

printf("Requesting %d integers on the heap...\n", initial_size);

int *arr = (int *)malloc(initial_size * sizeof(int));
if (arr == NULL) {
perror("malloc failed");
return 1;
  }

  // Fill with values and print addresses
for (inti = 0; i<initial_size; ++i) {
arr[i] = (i + 1) * 10;
printf("arr[%d] = %d  (addr: %p)\n", i, arr[i], (void *)&arr[i]);
  }

printf("\nNow expanding the array to %d integers using realloc...\n", new_size);

int *tmp = (int *)realloc(arr, new_size * sizeof(int));
if (tmp == NULL) {
    // realloc failed: original block (arr) is still valid, must free it
perror("realloc failed");
free(arr);
return 1;
  }
arr = tmp; // use the (possibly moved) block

  // Initialize new elements
for (inti = initial_size; i<new_size; ++i) {
```

```
arr[i] = (i + 1) * 10;
   }

   // Print all elements and addresses again
for (inti = 0; i<new_size; ++i) {
printf("arr[%d] = %d  (addr: %p)\n", i, arr[i], (void *)&arr[i]);
   }

   // Done: free heap memory
free(arr);
printf("\nMemory freed. Program finished.\n");
return 0;
}
```



```
Requesting 3 integers on the heap...
arr[0] = 10  (addr: 0x1c1986b0)
arr[1] = 20  (addr: 0x1c1986b4)
arr[2] = 30  (addr: 0x1c1986b8)

Now expanding the array to 5 integers using realloc...
arr[0] = 10  (addr: 0x1c1986b0)
arr[1] = 20  (addr: 0x1c1986b4)
arr[2] = 30  (addr: 0x1c1986b8)
arr[3] = 40  (addr: 0x1c1986bc)
arr[4] = 50  (addr: 0x1c1986c0)

Memory freed. Program finished.
```

6.Write a Lex specification to recognize +veintegers,reals and -veintegers,reals.

```
%{
#include <stdio.h>
%}
%%
\+?[0-9]+            { printf(“positive integers\n"); }
-[0-9]+              { printf("negative integers\n"); }
-[0-9]+\.[0-9]+       { printf(“negative real numbers\n" ); }
\+?[0-9]+\.[0-9]+      { printf("positive real numbers\n"); }
%%
int main()
{
yylex();
   return 0;
}
```

```
Compilation: lexnoformat.l
                         cc lex.yy.c -ll
                         ./a.out
                         24
```

```
                positive integer
                +24.12
                positive real number
                -24
                negative integer
                -24.12
                negative real number
```

7.Write a Lex specification for converting real numbers to integers.

```
%{
int i, j;
#include <stdio.h>
#include <stdlib.h>
%}
%%
[0-9]*\.[0-9]+ {
          for (i = 0; i<yyleng; i++)
          {
            if (yytext[i] == '.')
            {
              for (j = 0; j <= i - 1; j++)
                        printf("%c", yytext[j]);
              break;
            }
          }
        }
%%
int main(void)
{
yylex();
  return 0;
}
```

```
Compilation: lexrealtoint.l
                cc lex.yy.c -ll
                ./a.out
                24.12
                24
```

8.Write a Lex specification to print the number of days in a month using a procedure

```
%{
```

```
#include <stdio.h>
int year;
void leap(void);      /* prototype */
%}

%%

jan|mar|may|july|aug|oct|dec{ printf("31 days"); }
april|june|sep|nov{ printf("30 days"); }
feb                { leap(); }
[a-zA-Z]*             { printf("invalid"); }

%%

main()
{
yylex();
}

void leap(void)
{
printf("enter year");
scanf("%d", &year);
   if (year % 4 == 0)
printf("29 days");
   else
printf("28 days");
}
```

```
Compilation: lexdaysinamonth.l
                cc lex.yy.c -ll
                ./a.out
                jan
                31 days
                june
                30 days
                feb
                enter year
                1984
                29 days
```

9. Write a Lex specification to retrieve comments.

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

%%

"//".*          { printf("%s\n", yytext + 2); }

"/*"([^*]|\*+[^/])*"*/" {
                  int i;
                  for(i = 2; yytext[i] && !(yytext[i]=='*' && yytext[i+1]=='/'); i++)
                     putchar(yytext[i]);
                  printf("\n");
                }

.|\n            ;    /* ignore everything else */

%%

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

Compilation: lexcomments.l

```
                      cc lex.yy.c -ll
                      ./a.out
                      Hello //world
                      world
                      hai /*friend*/
                      friend
```

10. Write a Lex specification to design a lexical analyzer that recognizes identifiers and keywords of flow control statements of C language

```
%{
        #include<stdio.h>
%}
%%
If|else|while|do|switch|case|break|for|default{printf("Keyword");}
IF|ELSE|WHILE|DO|SWITCH|CASE|BREAK|FOR|DEFAULT      {printf("Keyword");}
[A-Z a-z]+[a-z A-Z 0-9 _]*      {printf("identifier");}
%%
int main()
{
```

```
yylex();
return 0;
}
```

```
Compilation: lexlexanalysis.l
             cc lex.yy.c -ll
             ./a.out
             If
             Keyword
             FOR
             Keyword
             Abc123_def
             identifier
```

11. Implementation of lexical analyzer using lex tool.

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

%%

[0-9]+                  { printf("NUMBER\t%s\n", yytext); }
[a-zA-Z_][a-zA-Z0-9_]*      { printf("IDENTIFIER\t%s\n", yytext); }
[+\-*/=]                { printf("OPERATOR\t%s\n", yytext); }
[ \t\n]                 ;       /* ignore spaces, tabs, newline */

%%

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

output:

```
a = b + 123;
IDENTIFIER    a
OPERATOR      =
IDENTIFIER    b
```

```
OPERATOR      +
NUMBER  123
OTHER   ;
```

12. Write a lex program to count the number of words and number of lines in a given file or program.

```
%{
int wc=0, lc=0;
%}

%%

\n      { lc++; }
[ \t]+    ;
[^ \t\n]+ { wc++; }

%%

int main() {
   yylex();
   printf("Lines=%d\nWords=%d\n", lc, wc);
   return 0;
}
```

Ouput:

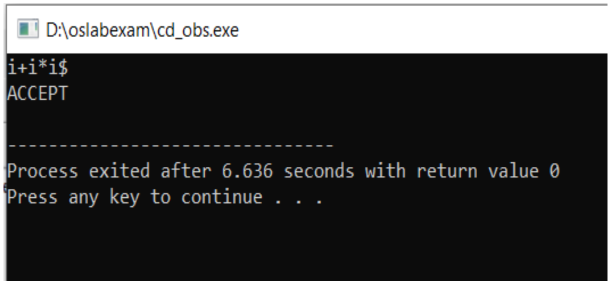
```
hi guys
how are you   (after this press ctrl+D)
Lines=2
Words=5
```

13. Write a C program to construct predictive parser for the following grammar

E → T R
R → + T R | ε
T → F P
P → * F P | ε
F → i | (E)

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

char st[50];
int top=-1;

void push(char c){ st[++top]=c; }
void pop(){ top--; }

int main(){
char in[50];
scanf("%s", in);
   int p=0;
push('$'); push('E');

while(top>=0){
      char X = st[top];
      char a = in[p];

      if(X==a){ pop(); p++; if(a=='$'){ printf("ACCEPT\n"); return 0;} }

else if(X=='E'){ pop(); push('A'); push('T'); }
else if(X=='A'){
if(a=='+'){ pop(); push('A'); push('T'); push('+'); }
else pop();  // epsilon
      }
else if(X=='T'){ pop(); push('B'); push('F'); }
else if(X=='B'){
if(a=='*'){ pop(); push('B'); push('F'); push('*'); }
else pop(); // epsilon
      }
else if(X=='F'){
pop();
```

```
        if(a=='i') push('i');
else if(a=='('){ push(')'); push('E'); push('('); }
else { printf("ERROR\n"); return 0; }
    }
else{printf("ERROR\n"); return 0; }
  }
}
```



```
i+i*i$
ACCEPT

--------------------------------
Process exited after 6.636 seconds with return value 0
Press any key to continue . . .
```

14.Write a C program to construct Recursive Descent parser for the following grammar

E→TR

R→+TR/∈

T→FP

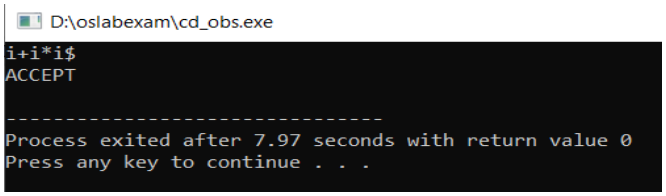
T→*FP/∈

F→a/(E)

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

char in[100];
int p = 0;

char peek(){ return in[p]; }
void advance(){ p++; }
void error(){ printf("ERROR\n"); exit(0); }

void E();
void Ep();
void T();
voidTp();
void F();

void F(){
if(peek() == 'i'){    // id
advance();
  }
else if(peek() == '('){
```

```
advance();
E();
if(peek() == ')') advance();
else error();
  }
else error();
}

voidTp(){
if(peek() == '*'){
advance();
F();
Tp();
  }
  // else epsilon
}

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

void Ep(){
if(peek() == '+'){
advance();
T();
Ep();
  }
  // else epsilon
}

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

int main(){
scanf("%s", in);

E();

if(peek()=='$')
printf("ACCEPT\n");
  else
printf("ERROR\n");
```

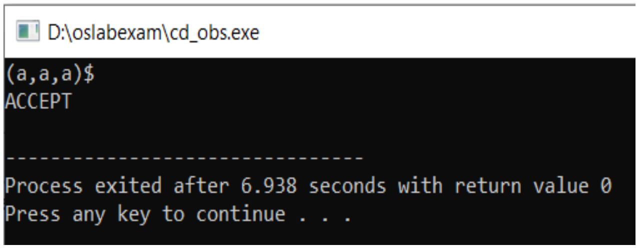
```
}
```



```
i+i*i$
ACCEPT

--------------------------------
Process exited after 7.97 seconds with return value 0
Press any key to continue . . .
```

15. write recursive descent parser for the grammar S->(L) S->a L->L,S L->S

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

char in[100];
int p = 0;

char peek(){ return in[p]; }
void advance(){ p++; }
void error(){ printf("ERROR\n"); exit(0); }

void S();
void L();
voidLp();

void S(){
if(peek()=='a'){
advance();
   }
else if(peek()=='('){
advance();
L();
if(peek()==')') advance();
else error();
   }
else error();
}

voidLp(){
if(peek()==','){
advance();
S();
Lp();
   }
   // else epsilon
}

void L(){
S();
Lp();
}

int main(){
scanf("%s", in);

S();

if(peek()=='$')
printf("ACCEPT\n");
  else
printf("ERROR\n");

  return 0;
}
```



```
(a,a,a)$
ACCEPT

--------------------------------
Process exited after 6.938 seconds with return value 0
Press any key to continue . . .
```

16. Write a C program to calculate first function for the grammar E->E+T E->T T->T*F T->F F->(E)/id

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

char prod[20][20];    // productions
char first[20][20];   // FIRST sets
char nt[20];          // non-terminals list
int n, ntCount = 0;   // number of productions, number of NTs
int done[256] = {0};  // memoization flag

int findNT(char c) {
    for (int i = 0; i < ntCount; i++)
        if (nt[i] == c)
            return i;
    nt[ntCount] = c;
    first[ntCount][0] = '\0';
    return ntCount++;
}

void add(char set[], char c) {
    if (!strchr(set, c)) {
        int l = strlen(set);
        set[l] = c;
        set[l+1] = '\0';
    }
}

// Compute FIRST for non-terminal X
void computeFIRST(char X) {
    int idx = findNT(X);
    if (done[X]) return;   // avoid infinite recursion on left-recursion
    done[X] = 1;

    for (int i = 0; i < n; i++) {
        if (prod[i][0] == X) {
            char *rhs = strchr(prod[i], '>') + 1;

            // Case 1: RHS starts with terminal or symbol like '(' '+'
            if (!isupper(rhs[0]) && rhs[0] != '#') {
                add(first[idx], rhs[0]);
            }
            // Case 2: RHS starts with epsilon
            else if (rhs[0] == '#') {
                add(first[idx], '#');
            }
            // Case 3: RHS starts with non-terminal
            else if (isupper(rhs[0])) {
                computeFIRST(rhs[0]);
                int j = findNT(rhs[0]);
                for (int k = 0; k < strlen(first[j]); k++)
                    add(first[idx], first[j][k]);
            }
        }
    }
}
```

```
int main() {
    printf("Enter number of productions: ");
    scanf("%d", &n);

    printf("Enter productions like E->E+T (use # for epsilon):\n");

    for (int i = 0; i < n; i++) {
        scanf("%s", prod[i]);
        findNT(prod[i][0]);     // record non-terminal
    }

    // Compute FIRST for all non-terminals
    for (int i = 0; i < ntCount; i++)
        computeFIRST(nt[i]);

    printf("\nFIRST sets:\n");
    for (int i = 0; i < ntCount; i++)
        printf("FIRST(%c) = { %s }\n", nt[i], first[i]);

    return 0;
}
```