LAB 1: *INFIX TO POSTFIX*

AIM: Write a program to convert an infix expression to postfix.

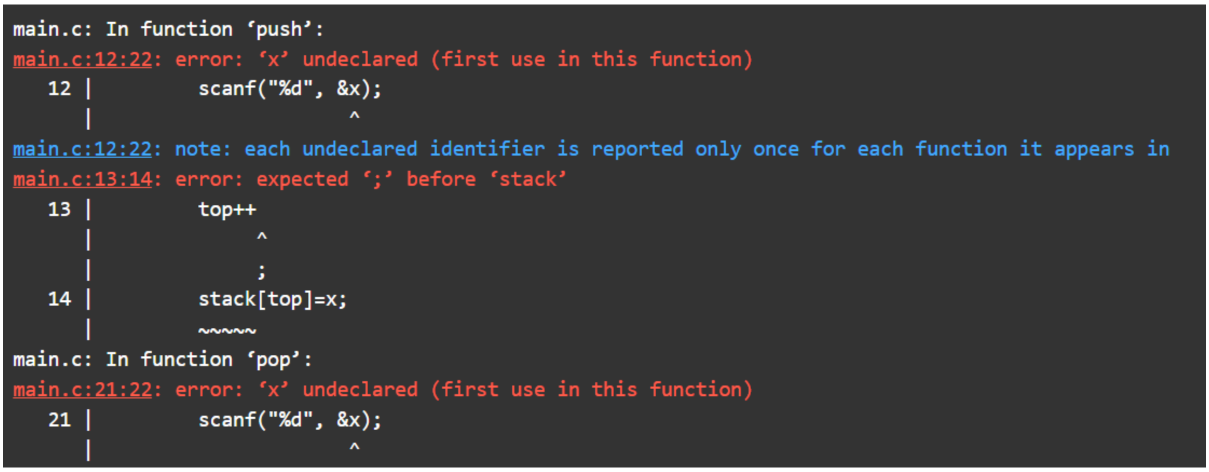
SOFTWARE USED: ONLINE GDB COMPILER

PROGRAM:

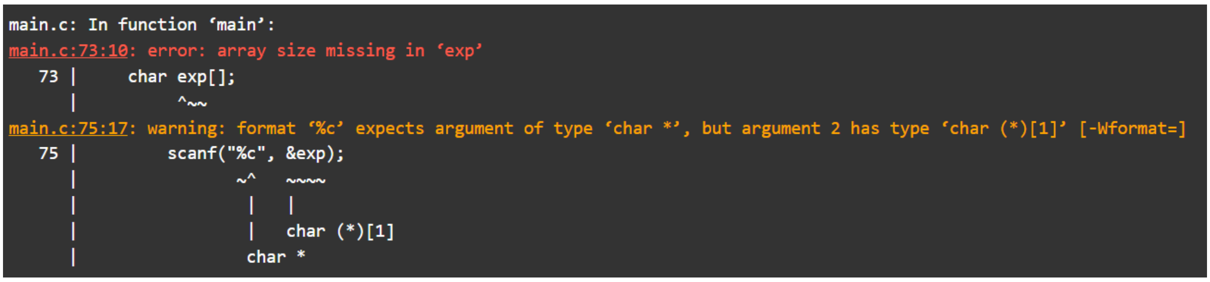
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Function to return precedence of operators
int prec(char c) {
     if (c == '^')
          return 3;
     else if (c == '/' || c == '*')
          return 2;
     else if (c == '+' || c == '-')
          return 1;
     else
          return -1;
}
// Function to return associativity of operators
char associativity(char c) {
     if (c == '^')
          return 'R';
     return 'L'; // Default to left-associative
}

// The main function to convert infix expression to postfix expression
void infixToPostfix(char s[]) {
     char result[1000];
     int resultIndex = 0;
     int len = strlen(s);
     char stack[1000];
     int stackIndex = -1;

     for (int i = 0; i < len; i++) {
          char c = s[i];
// If the scanned character is an operand, add it to the output string.
          if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c
>= '0' && c <= '9')) {
               result[resultIndex++] = c;
          }
          // If the scanned character is an ‘(‘, push it to the stack.
          else if (c == '(') {
               stack[++stackIndex] = c;
          }
```

```
        // If the scanned character is an ‘)’, pop and add to the
output string from the stack until an ‘(‘ is encountered.
        else if (c == ')') {
            while (stackIndex >= 0 && stack[stackIndex] != '(') {
                result[resultIndex++] = stack[stackIndex--];
            }
            stackIndex--; // Pop '('
        }
        // If an operator is scanned
        else {
            while (stackIndex >= 0 && (prec(s[i]) <
prec(stack[stackIndex]) || prec(s[i]) == prec(stack[stackIndex]) &&
associativity(s[i]) == 'L')) {
                result[resultIndex++] = stack[stackIndex--];
            }
            stack[++stackIndex] = c;
        }
    }
    // Pop all the remaining elements from the stack
    while (stackIndex >= 0) {
        result[resultIndex++] = stack[stackIndex--];
    }
    result[resultIndex] = '\0';
    printf("%s\n", result);
}
int main() {
   char exp[20];
    printf("Enter the expression: ");
    scanf("%[^\n]s", exp);
    // Function call
    infixToPostfix(exp);
    return 0;
}
```

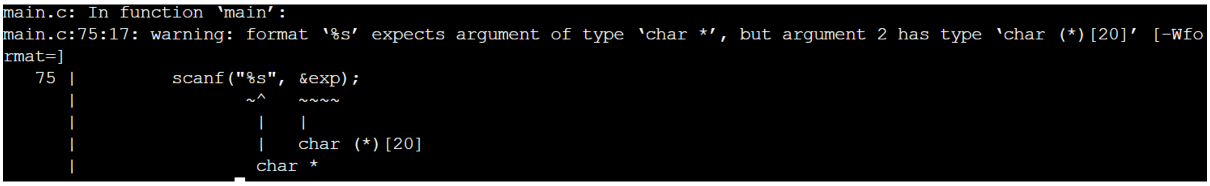
ERRORS WHILE RUNNING STACK PROGRAM:



```
main.c: In function ‘push’:
main.c:12:22: error: ‘x’ undeclared (first use in this function)
   12 |         scanf("%d", &x);
      |                      ^
main.c:12:22: note: each undeclared identifier is reported only once for each function it appears in
main.c:13:14: error: expected ‘;’ before ‘stack’
   13 |         top++
      |              ^
      |              ;
   14 |         stack[top]=x;
      |         ~~~~~
main.c: In function ‘pop’:
main.c:21:22: error: ‘x’ undeclared (first use in this function)
   21 |         scanf("%d", &x);
      |                      ^
```

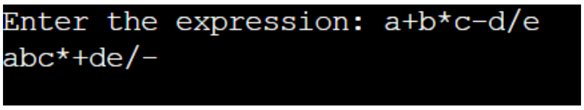


```
main.c: In function ‘main’:
main.c:73:10: error: array size missing in ‘exp’
   73 |     char exp[];
      |          ^~~
main.c:75:17: warning: format ‘%c’ expects argument of type ‘char *’, but argument 2 has type ‘char (*)[1]’ [-Wformat=]
   75 |         scanf("%c", &exp);
      |                ~^   ~~~~
      |                 |   |
      |                 |   char (*)[1]
      |                 char *
```



```
main.c: In function ‘main’:
main.c:75:17: warning: format ‘%s’ expects argument of type ‘char *’, but argument 2 has type ‘char (*)[20]’ [-Wfo
rmat=]
   75 |         scanf("%s", &exp);
      |                ~^   ~~~~
      |                 |   |
      |                 |   char (*)[20]
      |                 char *
```

OUTPUT:



```
Enter the expression: a+b*c-d/e
abc*+de/-
```

LAB 2: *CHARACTER COPYING*

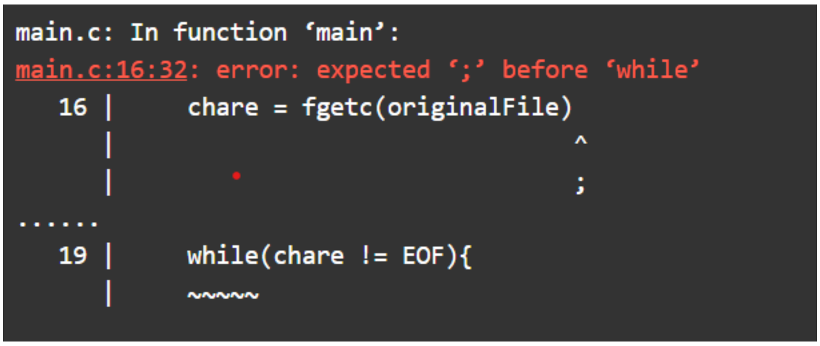
AIM: To open a file and copy the file character by character into another file.

SOFTWARE USED: VISUAL STUDIO CODE

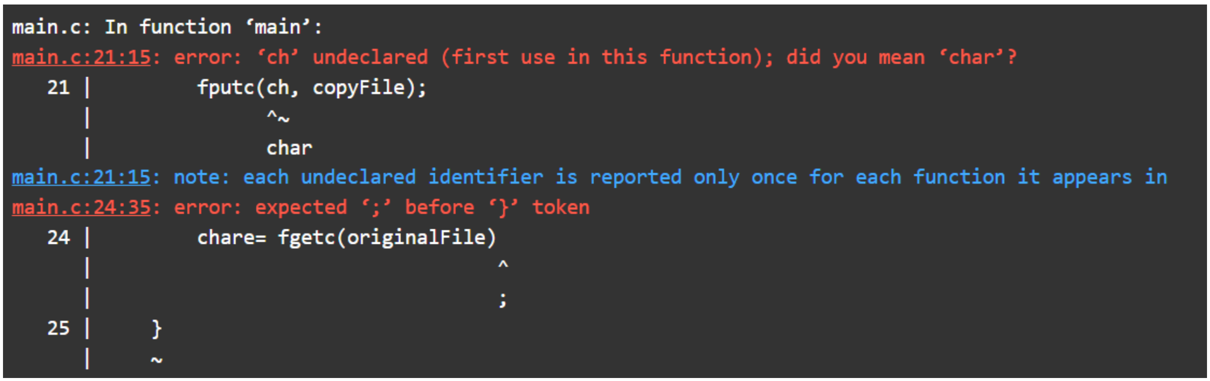
PROGRAM:

```
#include <stdio.h>
#include <string.h>
#define N 100

int main(){
    // CREATE CHAR FILE
    char chare[N];
    // OPEN ORIGINAL FILE
    FILE* original = fopen("file1.txt", "r");
    // OPEN FILE TO COPY IT TO
    FILE* copy = fopen("copy_file1.txt", "w");
    // GET FIRST CHARECTER
    chare[N] = fgetc(original);
    // CREATE A LOOP TILL END OD THE FILE
    while(chare[N] != EOF){
        // APPEND CHARECTER IN THE COPY FILE
        fputc(chare[N], copy);
        // GET NEXT CHARECTER IN ORIGINAL FILE
        chare[N]= fgetc(original);
    }
    // CLOSE BOTH FILE AND DISPLAY A MESSAGE
    fclose(original);
    fclose(copy);
    printf("/nFILE HAS BEEN COPIED");
    return 0;
}
```

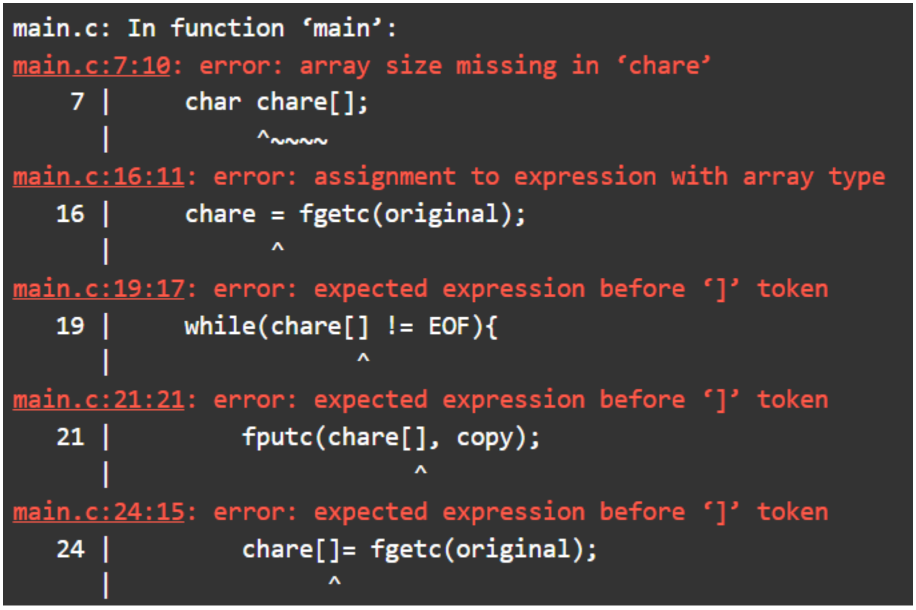
ERRORS:



```
main.c: In function ‘main’:
main.c:16:32: error: expected ‘;’ before ‘while’
   16 |     chare = fgetc(originalFile)
      |                                ^
      |                                ;
......
   19 |     while(chare != EOF){
      |     ~~~~~
```

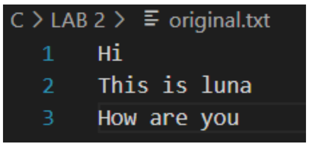


```
main.c: In function ‘main’:
main.c:21:15: error: ‘ch’ undeclared (first use in this function); did you mean ‘char’?
   21 |         fputc(ch, copyFile);
      |               ^~
      |               char
main.c:21:15: note: each undeclared identifier is reported only once for each function it appears in
main.c:24:35: error: expected ‘;’ before ‘}’ token
   24 |         chare= fgetc(originalFile)
      |                                   ^
      |                                   ;
   25 |     }
      |     ~
```

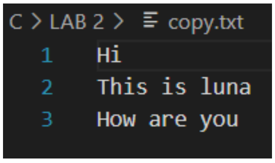


```
main.c: In function ‘main’:
main.c:7:10: error: array size missing in ‘chare’
    7 |     char chare[];
      |          ^~~~~
main.c:16:11: error: assignment to expression with array type
   16 |     chare = fgetc(original);
      |           ^
main.c:19:17: error: expected expression before ‘]’ token
   19 |     while(chare[] != EOF){
      |                 ^
main.c:21:21: error: expected expression before ‘]’ token
   21 |         fputc(chare[], copy);
      |                     ^
main.c:24:15: error: expected expression before ‘]’ token
   24 |         chare[]= fgetc(original);
      |               ^
```

OUPUT:



```
C > LAB 2 > original.txt
Hi
This is luna
How are you
```



```
C > LAB 2 > copy.txt
Hi
This is luna
How are you
```

LAB 3: *LEXICAL ANALYZER*

AIM: To create a program mimicking the function of a lexical analyzer.

SOFTRWARE USED: VISUAL STUDIO CODE

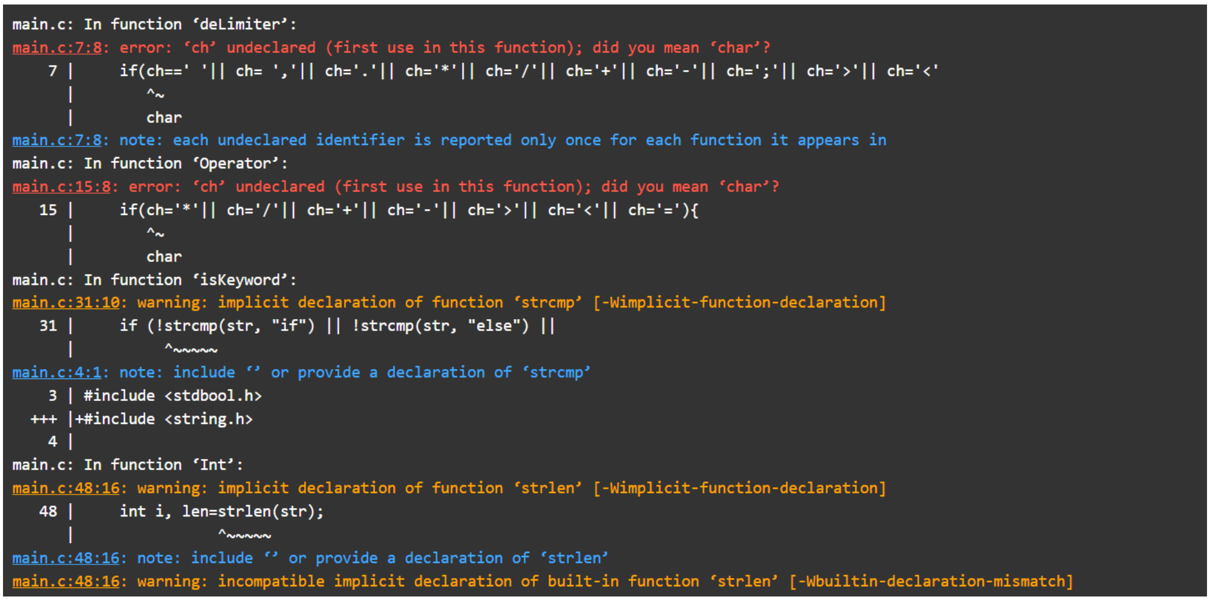
PROGRAM:

```
#include <stdio.h>
#define N 100
#include <stdbool.h>
#include <string.h>
#include <stdlib.h>
bool deLimiter(char ch){
if(ch==' '|| ch== ','||
ch=='.'|| ch=='*'|| ch=='/'||
ch=='+'|| ch=='-'|| ch==';'||
ch=='>'|| ch=='<'|| ch=='='||
ch=='('|| ch==')'|| ch=='['||
ch==']'|| ch=='{'|| ch=='}'){
        return true;
    }
    return false;
}
bool Operator(char ch){
    if(ch=='*'|| ch=='/'||
ch=='+'|| ch=='-'|| ch=='>'||
ch=='<'|| ch=='='){
        return true;
    }
    return false;
}
bool ValIden(char* str){
if(str[0]=='0'||str[0]=='1'||st
r[0]=='2'||str[0]=='3'||str[0]=
='4'||str[0]=='5'||str[0]=='6'|
|str[0]=='7'||str[0]=='8'||str[
0]=='9'|| deLimiter(str[0]==
true)) {
       return false;
   }
   return true;
}
bool isKeyword(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 Int(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 real(char *str){
    int i, len=strlen(str);
    bool deci= 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] == '-' && i > 0)){
            return false;
            if(str[i]=='.'){
                deci= true;
            }
        }
        return deci;
        }
}
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 parse(char* str)
{
    int left = 0, right = 0;
    int len = strlen(str);
while (right <= len && left <=
right) {
        if
(deLimiter(str[right]) ==
false)
            right++;
```

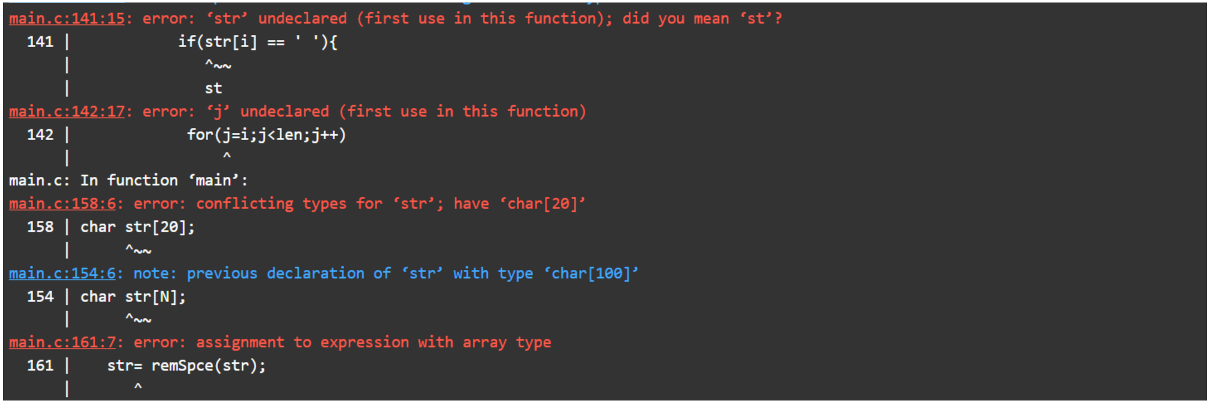
```
 if (deLimiter(str[right]) ==
true && left == right) {
 if (Operator(str[right]) ==
true)
 printf("'%c' IS AN
OPERATOR\n", str[right]);
            right++;
            left = right;
        } else if
(deLimiter(str[right]) == true
&& left != right||(right == len
&& left != right)) {
char* subStr = subString(str,
left, right - 1);
 if (isKeyword(subStr) == true)
printf("'%s' IS A KEYWORD\n",
subStr);

 else if (Int(subStr) == true)
printf("'%s' IS AN INTEGER\n",
subStr);
 else if (real(subStr) == true)
printf("'%s' IS A REAL
NUMBER\n", subStr);
else if (ValIden(subStr) ==
true && deLimiter(str[right -
1]) == false)
 printf("'%s' IS A VALID
IDENTIFIER\n", subStr);
else if (ValIden(subStr) ==
false && deLimiter(str[right -
1]) == false)
printf("'%s' IS NOT A VALID
IDENTIFIER\n", subStr);
    left = right;
        }
    }
    return;
}
char* remSpce(char* str){
    int i, len, j;
    len= strlen(str);
    for(i=0; i<len; i++){
           if(str[i] == ' '){
            for(j=i;j<len;j++)
        {
            str[j]=str[j+1];
        }
        len--;
        }
    }
    return str;
}
int main()
```

```
{
char str[N];
int i, n;
printf("Enter the string: ");
   scanf("%[^\n]%*c", str);
   str== remSpce(str);
   parse(str);
```

ERRORS:

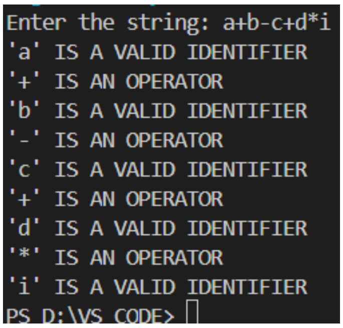


```
main.c: In function ‘deLimiter’:
main.c:7:8: error: ‘ch’ undeclared (first use in this function); did you mean ‘char’?
    7 |     if(ch==' '|| ch= ','|| ch='.'|| ch='*'|| ch='/'|| ch='+'|| ch='-'|| ch=';'|| ch='>'|| ch='<'
      |        ^~
      |        char
main.c:7:8: note: each undeclared identifier is reported only once for each function it appears in
main.c: In function ‘Operator’:
main.c:15:8: error: ‘ch’ undeclared (first use in this function); did you mean ‘char’?
   15 |     if(ch='*'|| ch='/'|| ch='+'|| ch='-'|| ch='>'|| ch='<'|| ch='='){
      |        ^~
      |        char
main.c: In function ‘isKeyword’:
main.c:31:10: warning: implicit declaration of function ‘strcmp’ [-Wimplicit-function-declaration]
   31 |     if (!strcmp(str, "if") || !strcmp(str, "else") ||
      |          ^~~~~~
main.c:4:1: note: include ‘’ or provide a declaration of ‘strcmp’
    3 | #include <stdbool.h>
  +++ |+#include <string.h>
    4 |
main.c: In function ‘Int’:
main.c:48:16: warning: implicit declaration of function ‘strlen’ [-Wimplicit-function-declaration]
   48 |     int i, len=strlen(str);
      |                ^~~~~~
main.c:48:16: note: include ‘’ or provide a declaration of ‘strlen’
main.c:48:16: warning: incompatible implicit declaration of built-in function ‘strlen’ [-Wbuiltin-declaration-mismatch]
```



```
main.c:141:15: error: ‘str’ undeclared (first use in this function); did you mean ‘st’?
  141 |            if(str[i] == ' '){
      |               ^~~
      |               st
main.c:142:17: error: ‘j’ undeclared (first use in this function)
  142 |             for(j=i;j<len;j++)
      |                 ^
main.c: In function ‘main’:
main.c:158:6: error: conflicting types for ‘str’; have ‘char[20]’
  158 | char str[20];
      |      ^~~
main.c:154:6: note: previous declaration of ‘str’ with type ‘char[100]’
  154 | char str[N];
      |      ^~~
main.c:161:7: error: assignment to expression with array type
  161 |    str= remSpce(str);
      |       ^
```

OUTPUT:



```
Enter the string: a+b-c+d*i
'a' IS A VALID IDENTIFIER
'+' IS AN OPERATOR
'b' IS A VALID IDENTIFIER
'-' IS AN OPERATOR
'c' IS A VALID IDENTIFIER
'+' IS AN OPERATOR
'd' IS A VALID IDENTIFIER
'*' IS AN OPERATOR
'i' IS A VALID IDENTIFIER
PS D:\VS CODE> 
```

LAB 4: *REMOVING LEFT RECURSION*

AIM: To create a program that removes left recursion from a production.

SOFTRWARE USED: VISUAL STUDIO

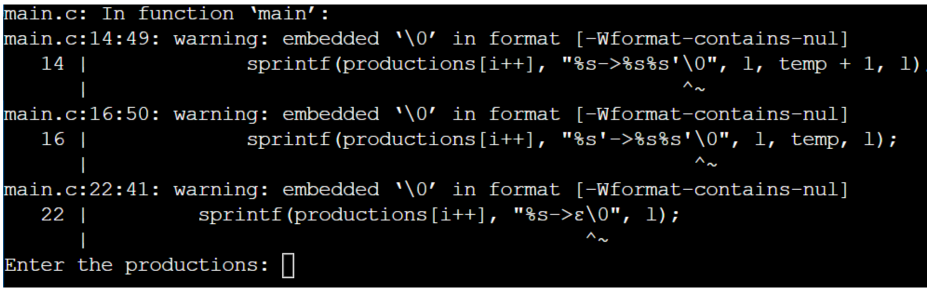
LANGUAGE USED: C

CODE:

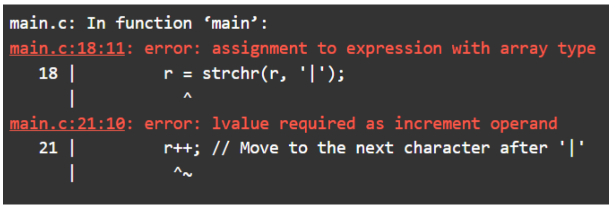
```
#include<stdio.h>
#include<string.h>
#define SIZE 10

int main() {
    char non_terminal; // Represents the non-terminal symbol
    char beta, alpha; // Represents the symbols in the production
    int num; // Number of productions
    char production[10][SIZE]; // Array to store productions
    int index = 3;
// Starting index of the production right-hand side
    printf("Enter Number of Production : ");
    scanf("%d", &num);
    printf("Enter the grammar as E->E-T :\n");
    for (int i = 0; i < num; i++) {
        scanf("%s", production[i]);
// Taking input of each production
    }
    for (int i = 0; i < num; i++) {
        printf("\nGRAMMAR : : : %s", production[i]);
// Printing the grammar productions
        non_terminal = production[i][0];
// First character is the non-terminal symbol
        if (non_terminal == production[i][index]) {
// Checking for left recursion
            alpha = production[i][index + 1]; // Next character is
alpha
            printf(" is left recursive.\n");
            while (production[i][index] != 0 && production[i][index]
!= '|')
                index++; // Finding the end of the left recursion
            if (production[i][index] != 0) {
                beta = production[i][index + 1];
// Character after '|' is beta
                printf("Grammar without left recursion:\n");
                printf("%c->%c%c\'|null", non_terminal, beta,
non_terminal);
                printf("\n%c\'->%c%c\'|E\n", non_terminal, alpha,
non_terminal); // Printing the new productions
```

```
            } else
                printf(" can't be reduced\n");
        } else
            printf(" is not left recursive.\n");
        index = 3; // Resetting index for next production
    }
}
```

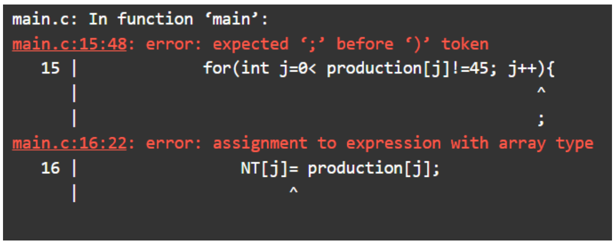
ERRORS:



```
main.c: In function ‘main’:
main.c:14:49: warning: embedded ‘\0’ in format [-Wformat-contains-nul]
   14 |               sprintf(productions[i++], "%s->%s%s'\0", l, temp + 1, l)
      |                                                   ^~
main.c:16:50: warning: embedded ‘\0’ in format [-Wformat-contains-nul]
   16 |               sprintf(productions[i++], "%s'->%s%s'\0", l, temp, l);
      |                                                    ^~
main.c:22:41: warning: embedded ‘\0’ in format [-Wformat-contains-nul]
   22 |           sprintf(productions[i++], "%s->ε\0", l);
      |                                           ^~
Enter the productions: 
```

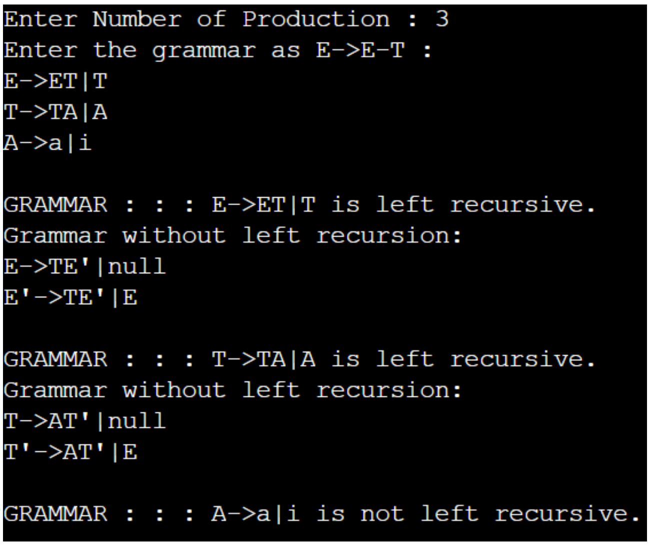


```
main.c: In function ‘main’:
main.c:18:11: error: assignment to expression with array type
   18 |         r = strchr(r, '|');
      |           ^
main.c:21:10: error: lvalue required as increment operand
   21 |         r++; // Move to the next character after '|'
      |          ^~
```



```
main.c: In function ‘main’:
main.c:15:48: error: expected ‘;’ before ‘)’ token
   15 |             for(int j=0< production[j]!=45; j++){
      |                                                ^
      |                                                ;
main.c:16:22: error: assignment to expression with array type
   16 |                NT[j]= production[j];
      |                      ^
```

OUTPUT:



```
Enter Number of Production : 3
Enter the grammar as E->E-T :
E->ET|T
T->TA|A
A->a|i

GRAMMAR : : : E->ET|T is left recursive.
Grammar without left recursion:
E->TE'|null
E'->TE'|E

GRAMMAR : : : T->TA|A is left recursive.
Grammar without left recursion:
T->AT'|null
T'->AT'|E

GRAMMAR : : : A->a|i is not left recursive.
```

LAB 5: *REMOVING LEFT FACTORING*

AIM: To create a program that removes left factoring from a production.

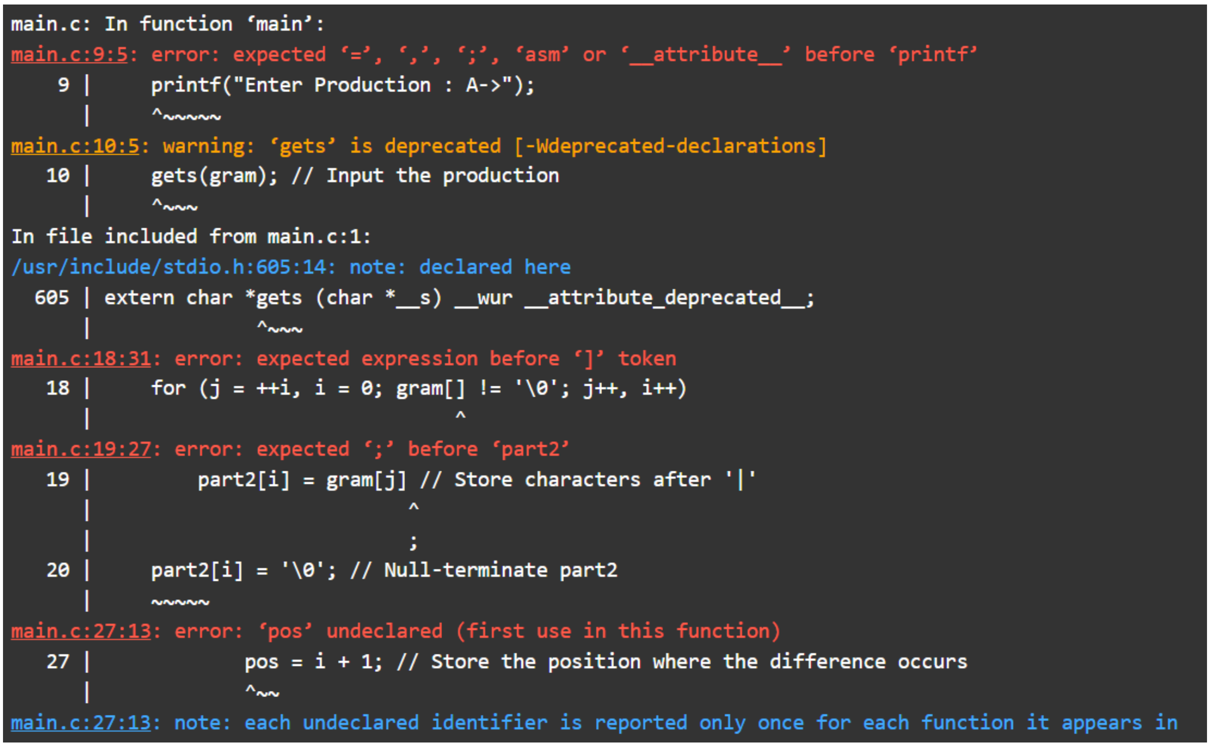
SOFTRWARE USED: VISUAL STUDIO

LANGUAGE USED: C

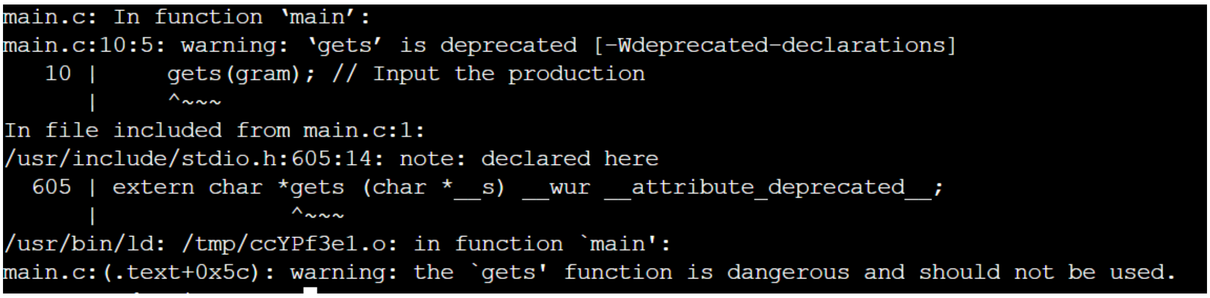
CODE:

```
#include<stdio.h>
#include<string.h>
int main() {
    char gram[20], part1[20], part2[20], modifiedGram[20],
newGram[20], tempGram[20];
    int i, j = 0, k = 0, l = 0, pos;
    // Prompt user to input production
    printf("Enter Production : A->");
    fgets(gram, 20, stdin); // Input the production

    // Split the production into two parts (part1 and part2)
separated by '|'
    for (i = 0; gram[i] != '|'; i++, j++)
        part1[j] = gram[i]; // Store characters before '|'
    part1[j] = '\0'; // Null-terminate part1

    // Increment i to move past '|' and continue copying
characters into part2
    for (j = ++i, i = 0; gram[j] != '\0'; j++, i++)
        part2[i] = gram[j]; // Store characters after '|'
    part2[i] = '\0'; // Null-terminate part2

    // Find common prefix between part1 and part2 and store it in
modifiedGram
    for (i = 0; i < strlen(part1) || i < strlen(part2); i++) {
        if (part1[i] == part2[i]) {
            modifiedGram[k] = part1[i]; // Common prefix
characters
            k++;
            pos = i + 1; // Store the position where the
difference occurs
        }
    }

    // Store the remaining characters in part1 after the common
prefix in newGram
    for (i = pos, j = 0; part1[i] != '\0'; i++, j++) {
        newGram[j] = part1[i];
    }
    // Add '|' to separate newGram from part2
    newGram[j++] = '|';

    // Store the remaining characters in part2 after the common
prefix in newGram
    for (i = pos; part2[i] != '\0'; i++, j++) {
        newGram[j] = part2[i];
    }

    // Replace the common prefix in modifiedGram with 'X'
    modifiedGram[k] = 'X';
    modifiedGram[++k] = '\0'; // Null-terminate modifiedGram

    newGram[j] = '\0'; // Null-terminate newGram

    // Print the modified production
    printf("\n A->%s|null", modifiedGram);
    printf("\n X->%s\n", newGram);
}
```

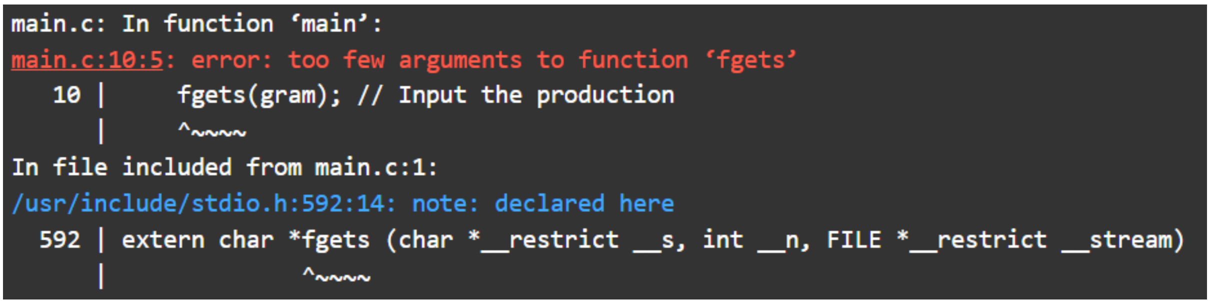
ERRORS:



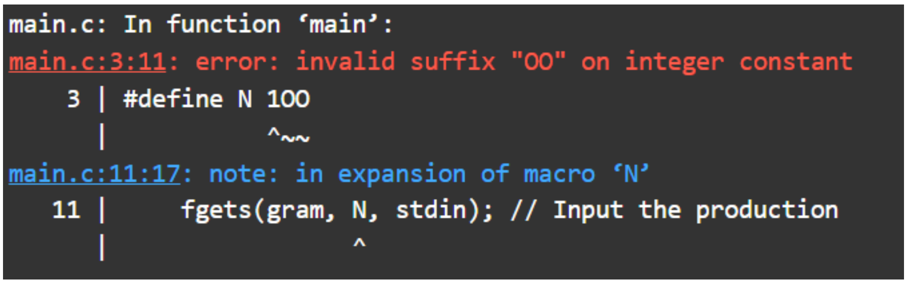
```
main.c: In function ‘main’:
main.c:9:5: error: expected ‘=’, ‘,’, ‘;’, ‘asm’ or ‘__attribute__’ before ‘printf’
    9 |     printf("Enter Production : A->");
      |     ^~~~~~
main.c:10:5: warning: ‘gets’ is deprecated [-Wdeprecated-declarations]
   10 |     gets(gram); // Input the production
      |     ^~~~
In file included from main.c:1:
/usr/include/stdio.h:605:14: note: declared here
  605 | extern char *gets (char *__s) __wur __attribute_deprecated__;
      |              ^~~~
main.c:18:31: error: expected expression before ‘]’ token
   18 |     for (j = ++i, i = 0; gram[] != '\0'; j++, i++)
      |                               ^
main.c:19:27: error: expected ‘;’ before ‘part2’
   19 |         part2[i] = gram[j] // Store characters after '|'
      |                           ^
      |                           ;
   20 |     part2[i] = '\0'; // Null-terminate part2
      |     ~~~~~
main.c:27:13: error: ‘pos’ undeclared (first use in this function)
   27 |             pos = i + 1; // Store the position where the difference occurs
      |             ^~~
main.c:27:13: note: each undeclared identifier is reported only once for each function it appears in
```



```
main.c: In function ‘main’:
main.c:10:5: warning: ‘gets’ is deprecated [-Wdeprecated-declarations]
   10 |     gets(gram); // Input the production
      |     ^~~~
In file included from main.c:1:
/usr/include/stdio.h:605:14: note: declared here
  605 | extern char *gets (char *__s) __wur __attribute_deprecated__;
      |              ^~~~
/usr/bin/ld: /tmp/ccYPf3e1.o: in function `main':
main.c:(.text+0x5c): warning: the `gets' function is dangerous and should not be used.
```

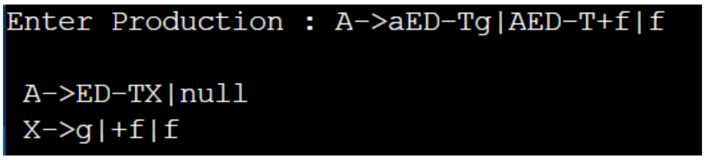


```
main.c: In function ‘main’:
main.c:10:5: error: too few arguments to function ‘fgets’
   10 |     fgets(gram); // Input the production
      |     ^~~~~
In file included from main.c:1:
/usr/include/stdio.h:592:14: note: declared here
  592 | extern char *fgets (char *__restrict __s, int __n, FILE *__restrict __stream)
      |              ^~~~~
```



```
main.c: In function ‘main’:
main.c:3:11: error: invalid suffix "OO" on integer constant
    3 | #define N 1OO
      |           ^~~
main.c:11:17: note: in expansion of macro ‘N’
   11 |     fgets(gram, N, stdin); // Input the production
      |                 ^
```

OUTPUT:



```
Enter Production : A->aED-Tg|AED-T+f|f

 A->ED-TX|null
 X->g|+f|f
```

LAB 6: *FIRST AND FOLLOW*

AIM: To create a program that finds the first and follow of a Non terminal.

SOFTRWARE USED: VISUAL STUDIO

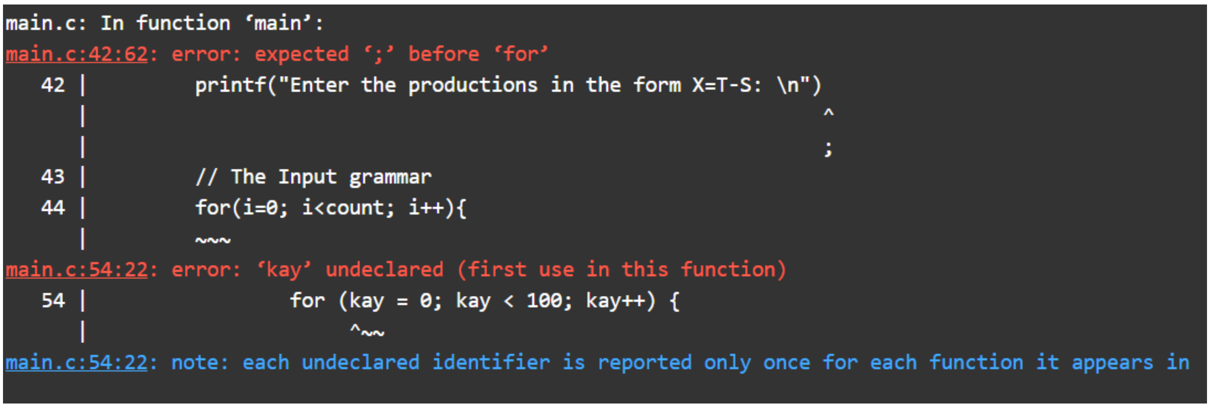
LANGUAGE USED: C

CODE:

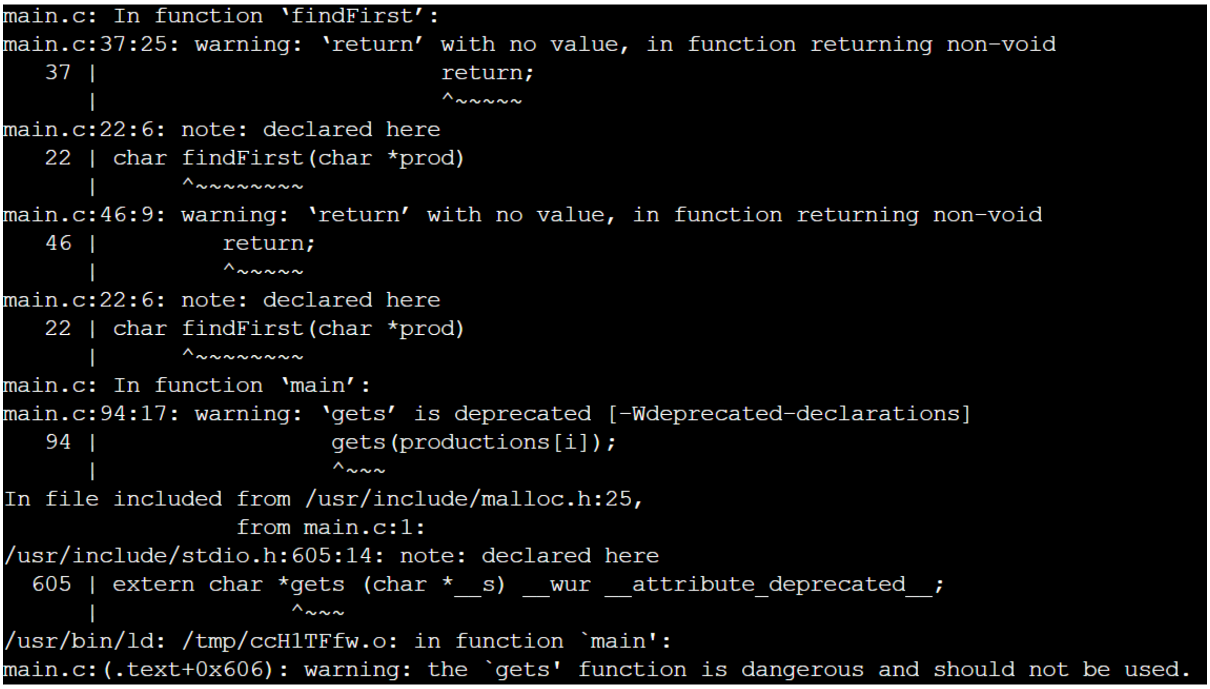
```
#include <ctype.h>
#include <stdio.h>
#include <string.h>
// Functions used to find Follow
void grammarfollow(char, int, int);
void follow(char c);
// Function used to find First
void find_first(char, int, int);
int count, n = 0;
// Stores the final resultof the First Sets
char final_first[10][100];
// Stores the final resultof the Follow Sets
char final_follow[10][100];
int m = 0;
// Stores the production rules
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;
    printf("Enter the no of productions: ");
    scanf("%d", &count);
    printf("Enter the productions: as E=T-X: \n");
    for(i=0; i<=count; i++){
        fgets(production[i], 10, stdin);
    }
    int ff;
    char done[count];
    int ptr = -1;
    // Initializing the final_first array
    for (k = 0; k < count; k++) {
        for (ff= 0; ff< 100; ff++) {
```

```
final_first[k][ff] = '!';
        }
    }
    int point1 = 0, point2, xxx;
    for (k = 0; k < count; k++) {
        c = production[k][0];
        point2 = 0;
        xxx = 0;
        // Checking if the first of c has already been calculated
        for (ff= 0; ff<= ptr; ff++)
            if (c == done[ff])
                xxx = 1;
        if (xxx == 1)
            continue;
        // Calling function
find_first(c, 0, 0);
        ptr += 1;
        // Adding c to the calculated list
        done[ptr] = c;
        printf("\n First(%c) = { ", c);
final_first[point1][point2++] = c;
        // Printing the First Sets of the given grammar
        for (i = 0 + jm; i < n; i++) {
            int fs = 0, chk = 0;
            for (fs = 0; fs< point2; fs++) {
                if (first[i] == final_first[point1][fs]) {
                    chk = 1;
                    break;
                }
            }
            if (chk == 0) {
                printf("%c, ", first[i]);
final_first[point1][point2++] = first[i];
            }
        }
        printf("}\n");
        jm = n;
        point1++;
    }
    printf("\n");
    printf("================================="\n\n");
    char donee[count];
    ptr = -1;
    // Initializing the final_follow array
    for (k = 0; k < count; k++) {
        for (ff= 0; ff< 100; ff++) {
final_follow[k][ff] = '!';
        }
    }
    point1 = 0;
    int land = 0;
    for (e = 0; e < count; e++) {
        ck = production[e][0];
        point2 = 0;
        xxx = 0;
        // Checking if Follow of ckhas already been calculated
        for (ff= 0; ff<= ptr; ff++)
            if (ck == donee[ff])
                xxx = 1;
        if (xxx == 1)
            continue;
        land += 1;
        // Function call
        follow(ck);
        ptr += 1;
        // Adding ck to the calculated list
        donee[ptr] = ck;
        printf(" Follow(%c) = { ", ck);
final_follow[point1][point2++] = ck;
        // Printing the Follow Sets of the given grammar
        for (i = 0 + km; i < m; i++) {
            int fs= 0, chk = 0;
            for (fs= 0; fs< point2; fs++) {
                if (f[i] == final_follow[point1][fs]) {
                    chk = 1;
                    break;
                }
            }
            if (chk == 0) {
                printf("%c, ", f[i]);
final_follow[point1][point2++] = f[i];
            }
        }
        printf(" }\n\n");
        km = m;
        point1++;
    }
}
void follow(char c)
{
    int i, j;
    // Adding "$" to the Follow Setof the start symbol
    if (production[0][0] == c) {
        f[m++] = '$';
    }
    for (i = 0; i < 10; i++) {
        for (j = 2; j < 10; j++) {
            if (production[i][j] == c) {
                if (production[i][j + 1] != '\0') {
                    // Calculate the first of the next non-terminal
in the production
grammarfollow(production[i][j + 1], i,
                                (j + 2));
                }
                if (production[i][j + 1] == '\0'
&& c != production[i][0]) {
 // Calculate the Follow of thenon-terminal in the L.H.S. of
theproduction
                    follow(production[i][0]);
                }
            }
        }
    }
}

void find_first(char c, int q1, int q2)
{
    int j;
    // The case where we will encounter a terminal
    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)) {
      // Recursion to calculate the First new non-terminal we
encounter after
                    // epsilon
                    find_first(production[q1][q2], q1,
                              (q2 + 1));
                }
                else
                    first[n++] = '#';
            }
            else if (!isupper(production[j][2])) {
                first[n++] = production[j][2];
            }
            else {
// Recursion to calculate First ofthe new non-terminal we encounterat
the beginning
                find_first(production[j][2], j, 3);
            }
        }
    }
}
void grammarfollow(char c, int c1, int c2)
{
    int k;
    // The case where we will encountera terminal
    if (!(isupper(c)))
        f[m++] = c;
    else {
        int i = 0, j = 1;
        for (i = 0; i < count; i++) {
            if (final_first[i][0] == c)
                break;
        }
// Including the First set of the non-terminal in the Follow ofthe
original query
        while (final_first[i][j] != '!') {
            if (final_first[i][j] != '#') {
                f[m++] = final_first[i][j];
            }
            else {
                if (production[c1][c2] == '\0') {
            // The case where we will reach theend of the production
                    follow(production[c1][0]);
                }
                else {
                    // Recursion to the next symbolin case we
encounter a "#"
grammarfollow(production[c1][c2], c1, c2 + 1);
                }
            }
            j++;
        }
    }
}
```

ERRORS:

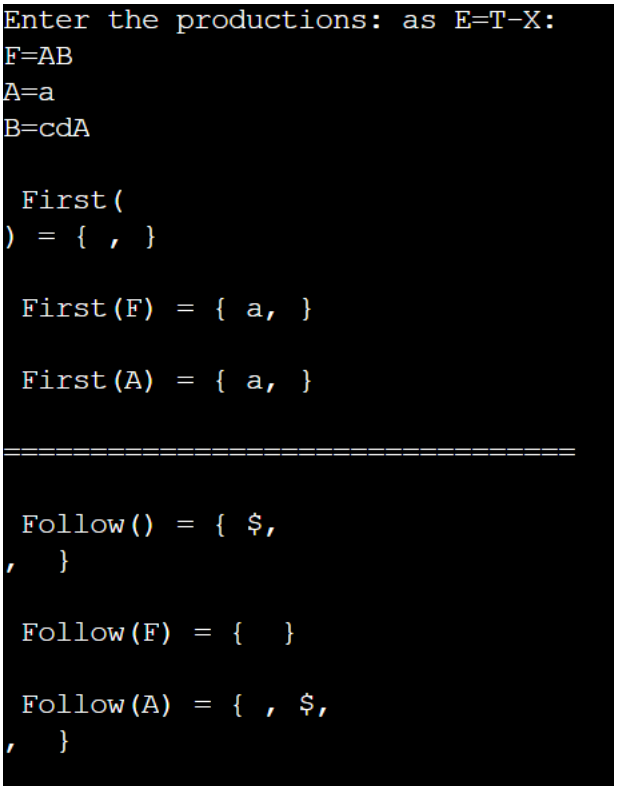


```
main.c: In function ‘main’:
main.c:42:62: error: expected ‘;’ before ‘for’
   42 |         printf("Enter the productions in the form X=T-S: \n")
      |                                                              ^
      |                                                              ;
   43 |         // The Input grammar
   44 |         for(i=0; i<count; i++){
      |         ~~~
main.c:54:22: error: ‘kay’ undeclared (first use in this function)
   54 |                 for (kay = 0; kay < 100; kay++) {
      |                      ^~~
main.c:54:22: note: each undeclared identifier is reported only once for each function it appears in
```



```
main.c: In function ‘findFirst’:
main.c:37:25: warning: ‘return’ with no value, in function returning non-void
   37 |                         return;
      |                         ^~~~~~
main.c:22:6: note: declared here
   22 | char findFirst(char *prod)
      |      ^~~~~~~~~
main.c:46:9: warning: ‘return’ with no value, in function returning non-void
   46 |         return;
      |         ^~~~~~
main.c:22:6: note: declared here
   22 | char findFirst(char *prod)
      |      ^~~~~~~~~
main.c: In function ‘main’:
main.c:94:17: warning: ‘gets’ is deprecated [-Wdeprecated-declarations]
   94 |                 gets(productions[i]);
      |                 ^~~~
In file included from /usr/include/malloc.h:25,
                 from main.c:1:
/usr/include/stdio.h:605:14: note: declared here
  605 | extern char *gets (char *__s) __wur __attribute_deprecated__;
      |              ^~~~
/usr/bin/ld: /tmp/ccH1TFfw.o: in function `main':
main.c:(.text+0x606): warning: the `gets' function is dangerous and should not be used.
```

OUTPUT:



```
Enter the productions: as E=T-X:
F=AB
A=a
B=cdA

 First(
) = { , }

 First(F) = { a, }

 First(A) = { a, }

=================================


 Follow() = { $,
,  }

 Follow(F) = {  }

 Follow(A) = { , $,
,  }
```