NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA, SURATHKAL

COMPILER LAB PROJECT-1

LEXICAL ANALYZER USING LEX

Submitted By:

Anuj Revankar(15CO109)

Rathan Kumar (15CO139)

Introduction

A compiler translates the code written in one language to some other language without changing the meaning of the program. It is also expected that a compiler should make the target code efficient and optimized in terms of time and space. The compilation process is a sequence of various phases. Each phase takes input from its previous stage, has its own representation of source program, and feeds its output to the next phase of the compiler. Let us understand the phases of a compiler.

Phases of compiler

Lexical Analysis

The first phase of scanner works as a text scanner. This phase scans the source code as a stream of characters and converts it into meaningful lexemes. Lexical analyzer represents these lexemes in the form of tokens as:

<token-name, attribute-value>

Syntax Analysis

The next phase is called the syntax analysis or parsing. It takes the token produced by lexical analysis as input and generates a parse tree (or syntax tree). In this phase, token arrangements are checked against the source code grammar, i.e. the parser checks if the expression made by the tokens is syntactically correct.

Semantic Analysis

Semantic analysis checks whether the parse tree constructed follows the rules of language. For example, assignment of values is between compatible data types, and adding string to an integer. Also, the semantic analyzer keeps track of identifiers, their types and expressions; whether identifiers are declared before use or not etc. The semantic analyzer produces an annotated syntax tree as an output.

Intermediate Code Generation

After semantic analysis the compiler generates an intermediate code of the source code for the target machine. It represents a program for some abstract machine. It is in between the high-level language and the machine language. This intermediate code should be generated in such a way that it makes it easier to be translated into the target machine code.

Code Optimization

The next phase does code optimization of the intermediate code. Optimization can be assumed as something that removes unnecessary code lines, and arranges the sequence of statements in order to speed up the program execution without wasting resources (CPU, memory).

Code Generation

In this phase, the code generator takes the optimized representation of the intermediate code and maps it to the target machine language. The code generator translates the intermediate code into a sequence of (generally) re-locatable machine code. Sequence of instructions of machine code performs the task as the intermediate code would do.

Symbol Table

It is a data-structure maintained throughout all the phases of a compiler. All the identifier's names along with their types are stored here. The symbol table makes it easier for the compiler to quickly search the identifier record and retrieve it. The symbol table is also used for scope management.

Lexical Analyzer

Lexical analysis is the first phase of a compiler. It takes the modified source code from language preprocessors that are written in the form of sentences. The lexical analyzer breaks these syntaxes into a series of tokens, by removing any whitespace or comments in the source code.

If the lexical analyzer finds a token invalid, it generates an error. The lexical analyzer works closely with the syntax analyzer. It reads character streams from the source code, checks for legal tokens, and passes the data to the syntax analyzer when it demands.

How Lexical Analyzer functions

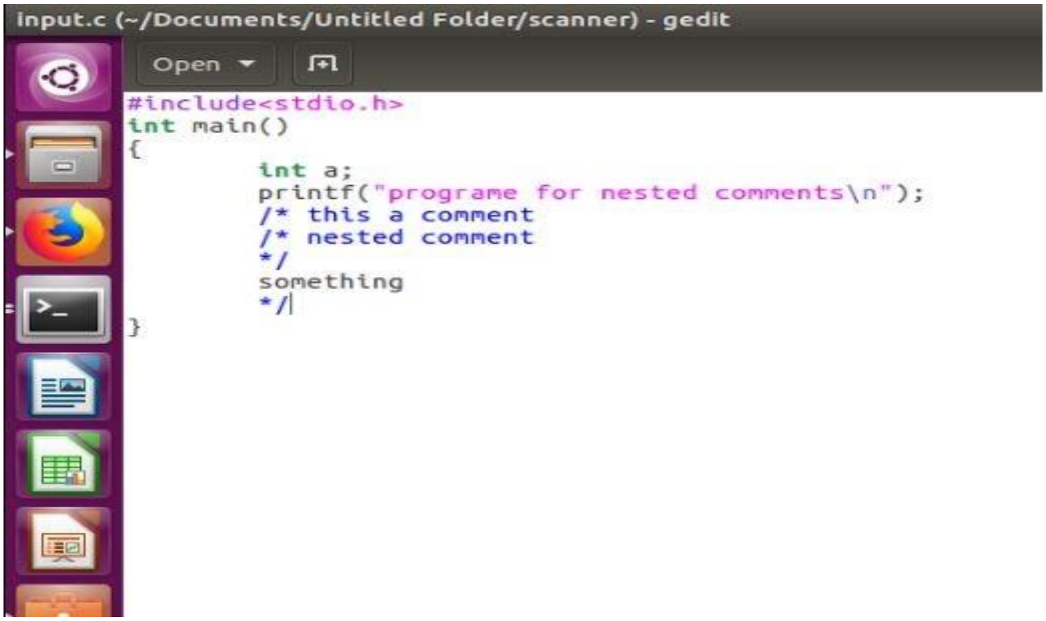
1. Tokenization .i.e Dividing the program into valid tokens.
2. Remove white space characters.
3. Remove comments.
4. It also provides help in generating error message by providing row number and column number.

Screenshots

Output 1

This Output demonstrates bracket mismatches, nested and unterminated comments.

Input



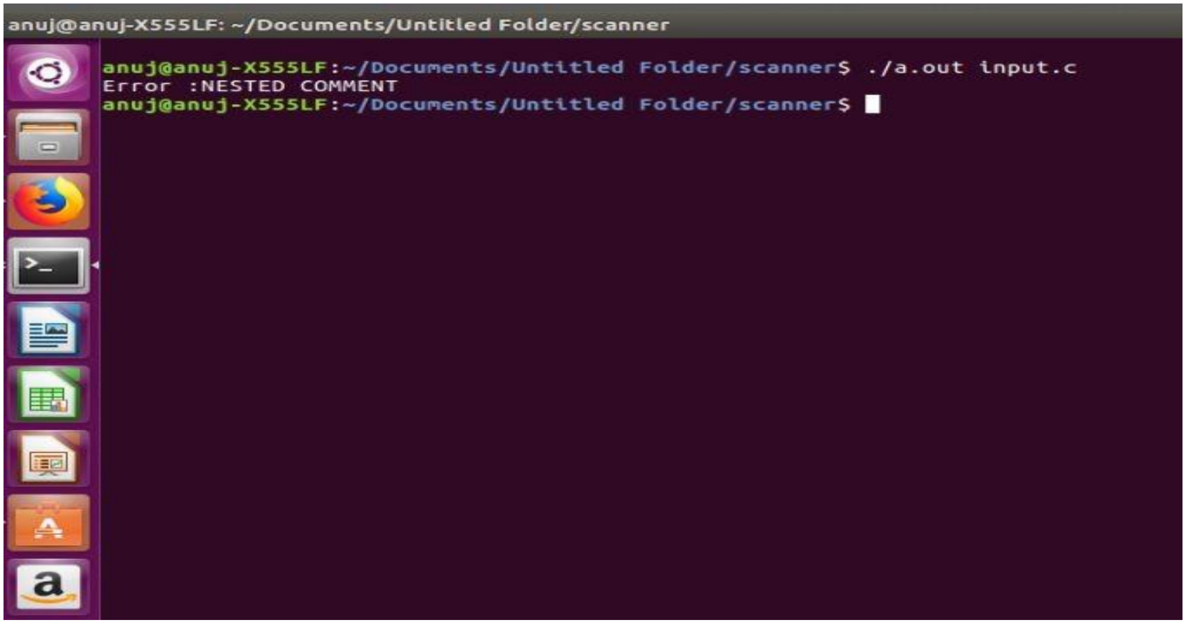
```
#include<stdio.h>
int main()
{
        int a;
        printf("programe for nested comments\n");
        /* this a comment
        /* nested comment
        */
        something
        */
}
```

Expected output

ERROR: COMMENT DOES NOT END

ERROR: NESTED COMMENT

Output

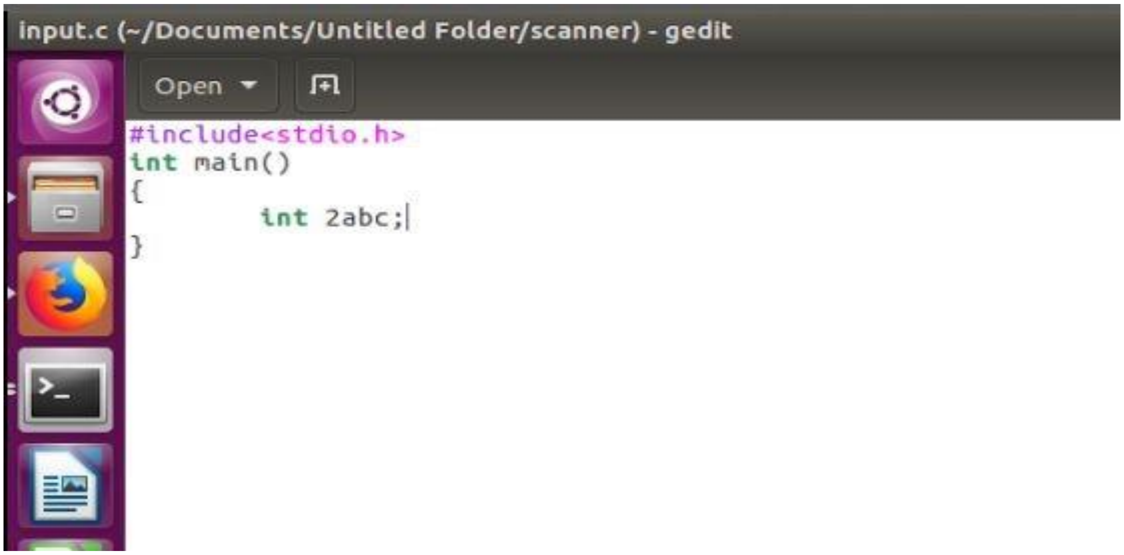


```
anuj@anuj-X555LF:~/Documents/Untitled Folder/scanner$ ./a.out input.c
Error :NESTED COMMENT
anuj@anuj-X555LF:~/Documents/Untitled Folder/scanner$ 
```

Output 2

This Output demonstrates bad Tokens.

Input

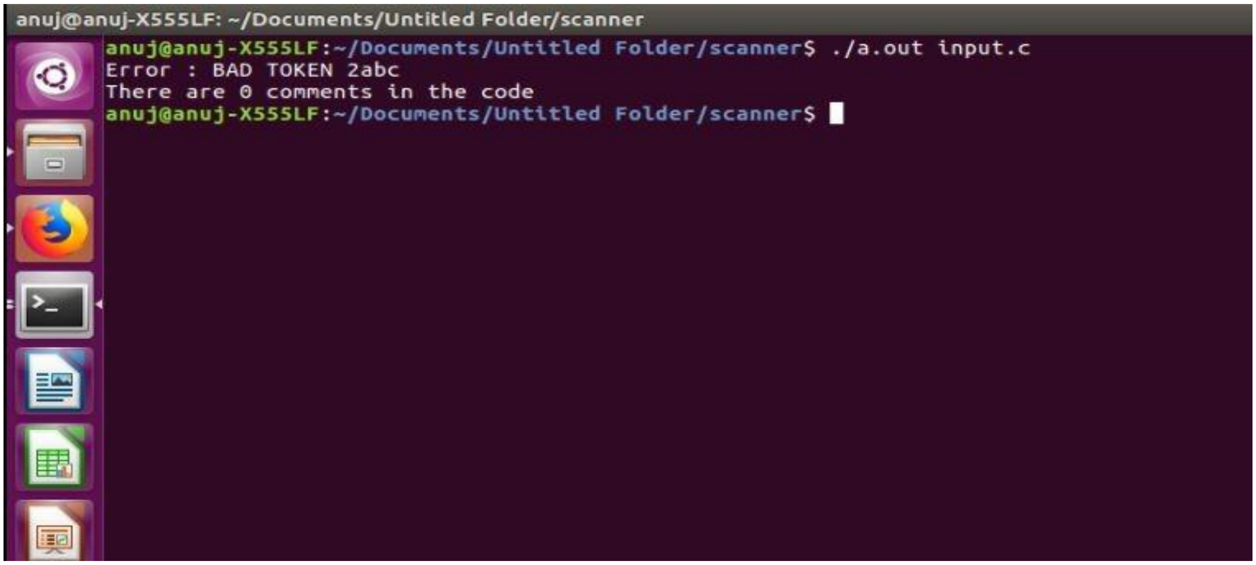


```
#include<stdio.h>
int main()
{
        int 2abc;
}
```

Expected output

ERROR: BAD TOKEN

Output

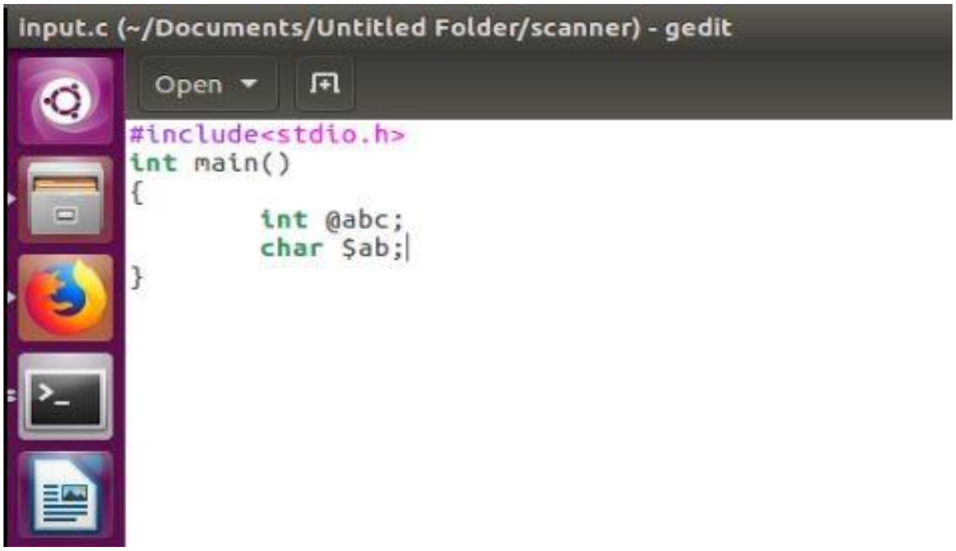


```
anuj@anuj-X555LF:~/Documents/Untitled Folder/scanner$ ./a.out input.c
Error : BAD TOKEN 2abc
There are 0 comments in the code
anuj@anuj-X555LF:~/Documents/Untitled Folder/scanner$
```

Output 3

This Output demonstrates unidentified tokens.

Input



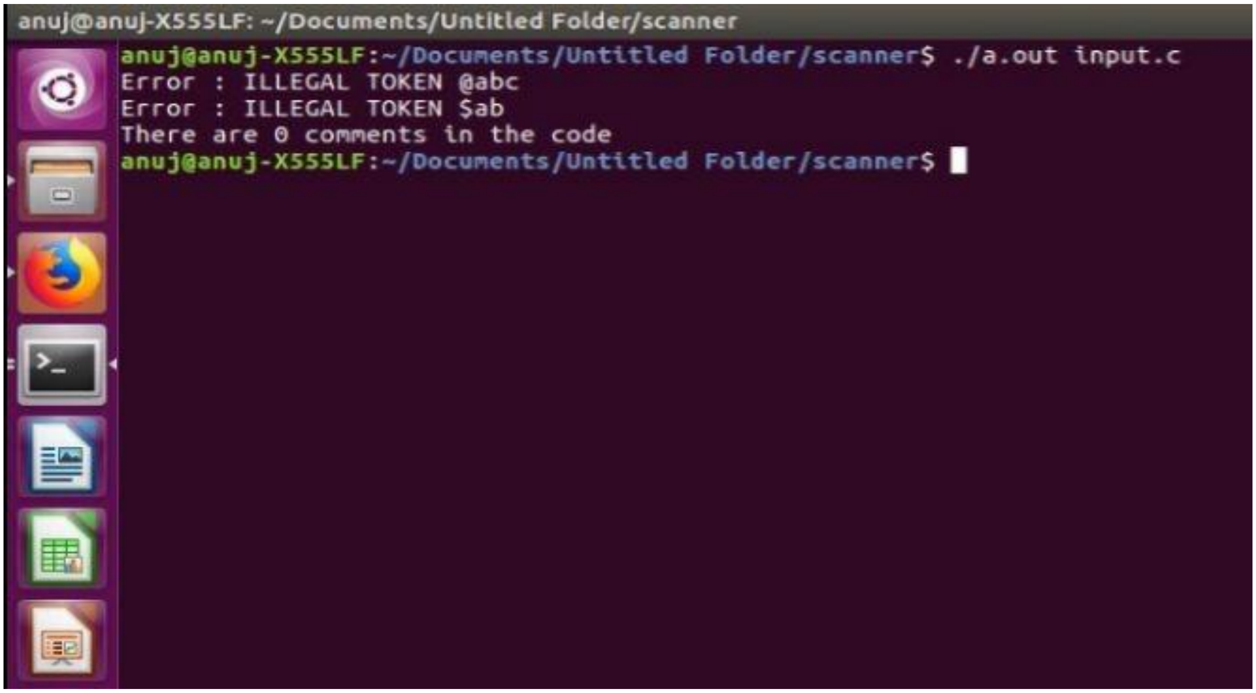
input.c (~/Documents/Untitled Folder/scanner) - gedit

```
#include<stdio.h>
int main()
{
        int @abc;
        char $ab;
}
```

Expected output

ERROR: ILLEGAL TOKEN

Output



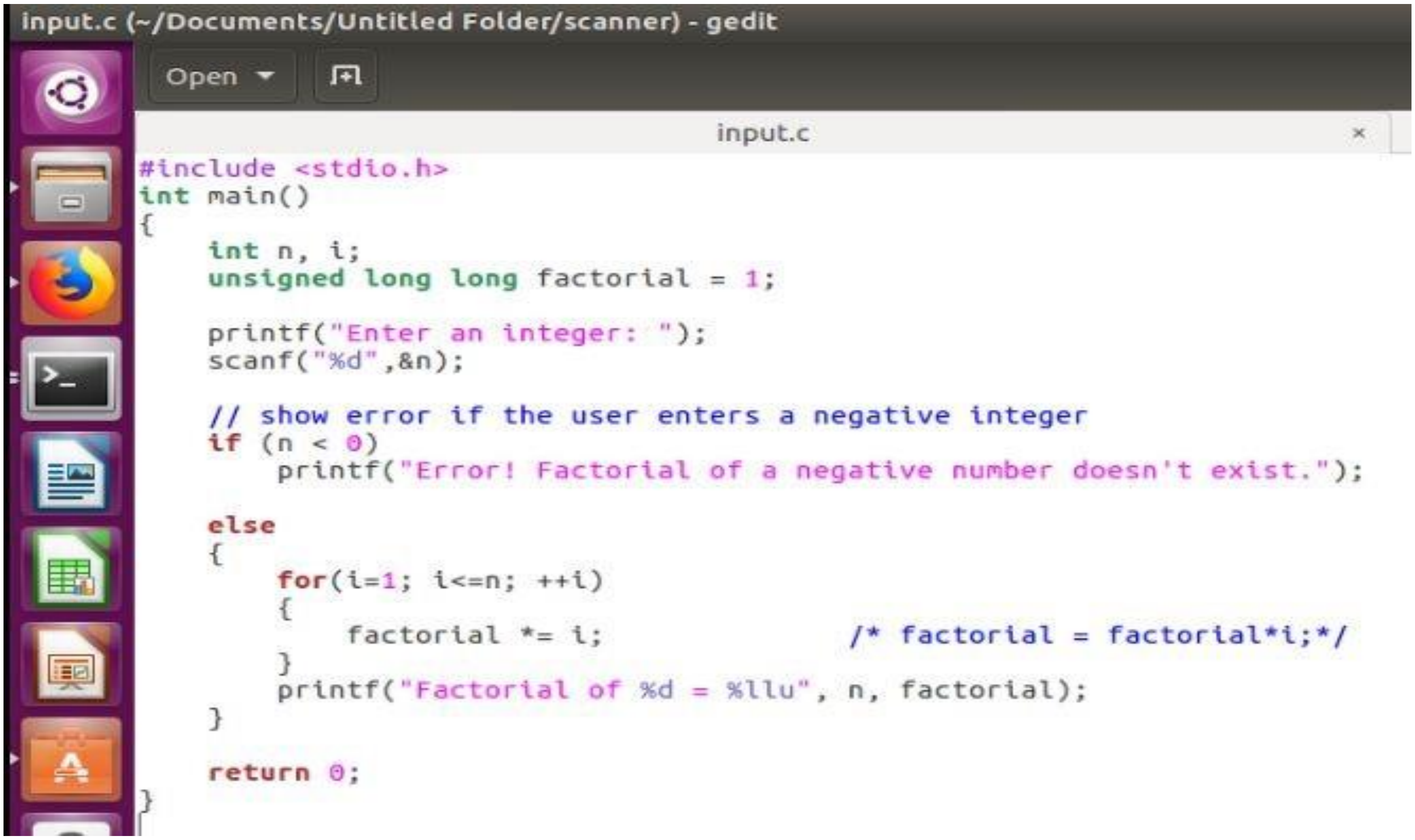
anuj@anuj-X555LF: ~/Documents/Untitled Folder/scanner

```
anuj@anuj-X555LF:~/Documents/Untitled Folder/scanner$ ./a.out input.c
Error : ILLEGAL TOKEN @abc
Error : ILLEGAL TOKEN $ab
There are 0 comments in the code
anuj@anuj-X555LF:~/Documents/Untitled Folder/scanner$
```

Output 4

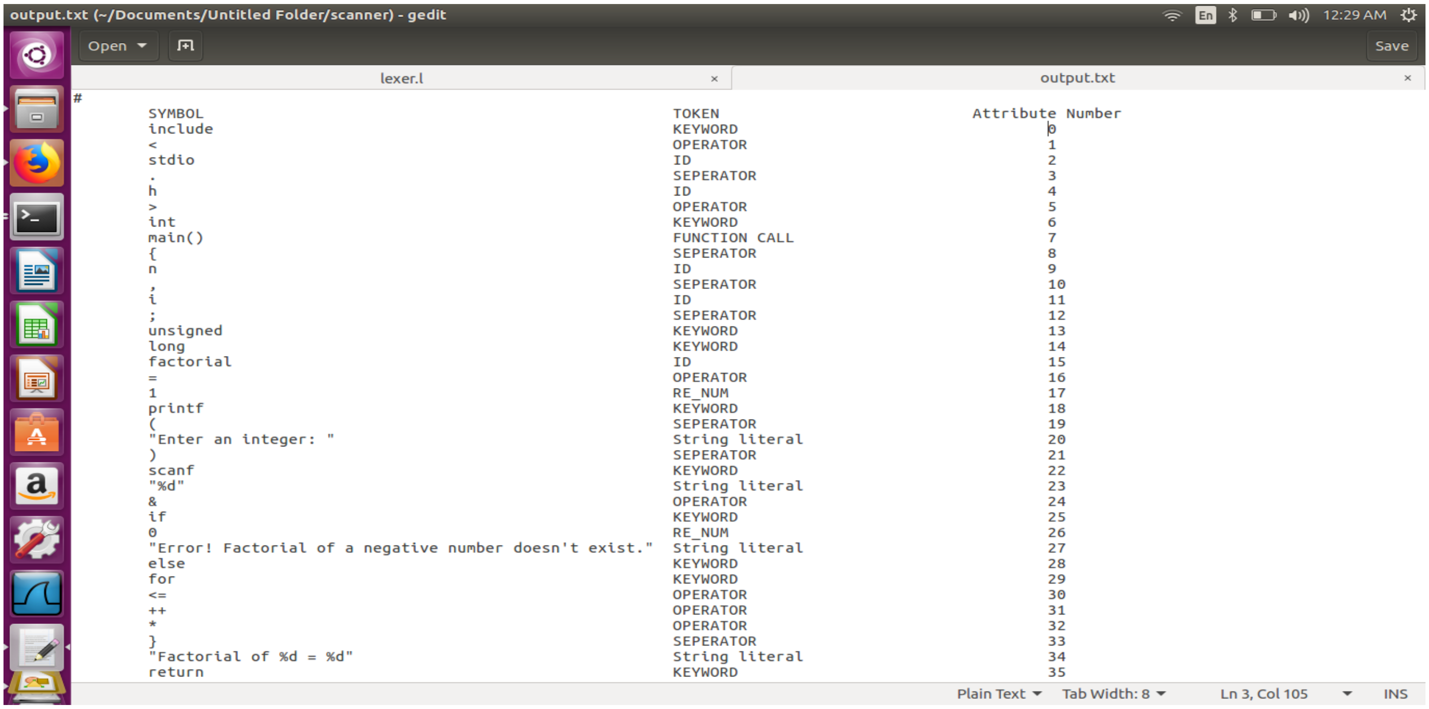
This Output has no errors. The Expected output shows the contents of the Expected output file. The Expected output file contains the symbol-constant table and information about comment lines.

Input



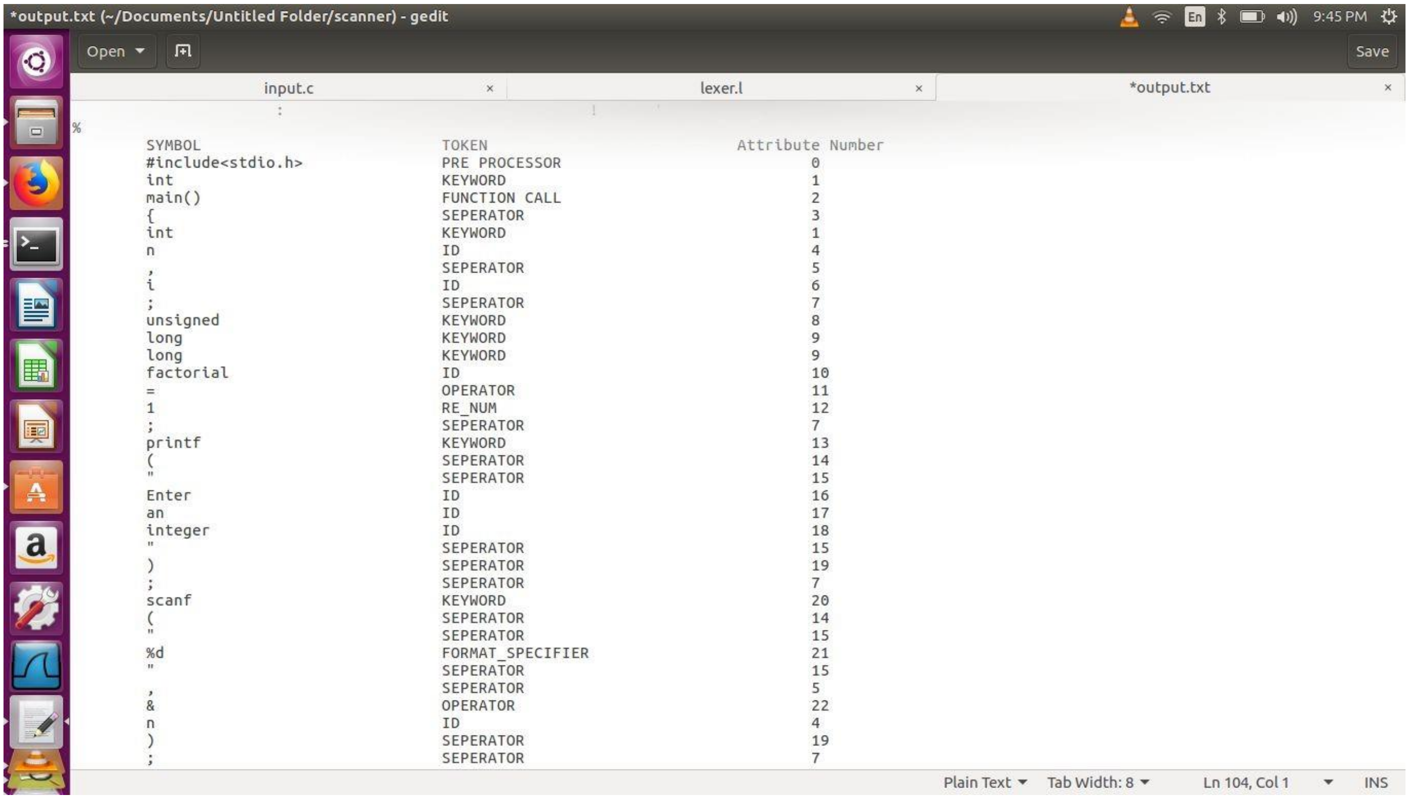
```
#include <stdio.h>
int main()
{
    int n, i;
    unsigned long long factorial = 1;

    printf("Enter an integer: ");
    scanf("%d",&n);

    // show error if the user enters a negative integer
    if (n < 0)
        printf("Error! Factorial of a negative number doesn't exist.");

    else
    {
        for(i=1; i<=n; ++i)
        {
            factorial *= i;              /* factorial = factorial*i;*/
        }
        printf("Factorial of %d = %llu", n, factorial);
    }

    return 0;
}
```

Output with string literals and unique symbols in symbol table

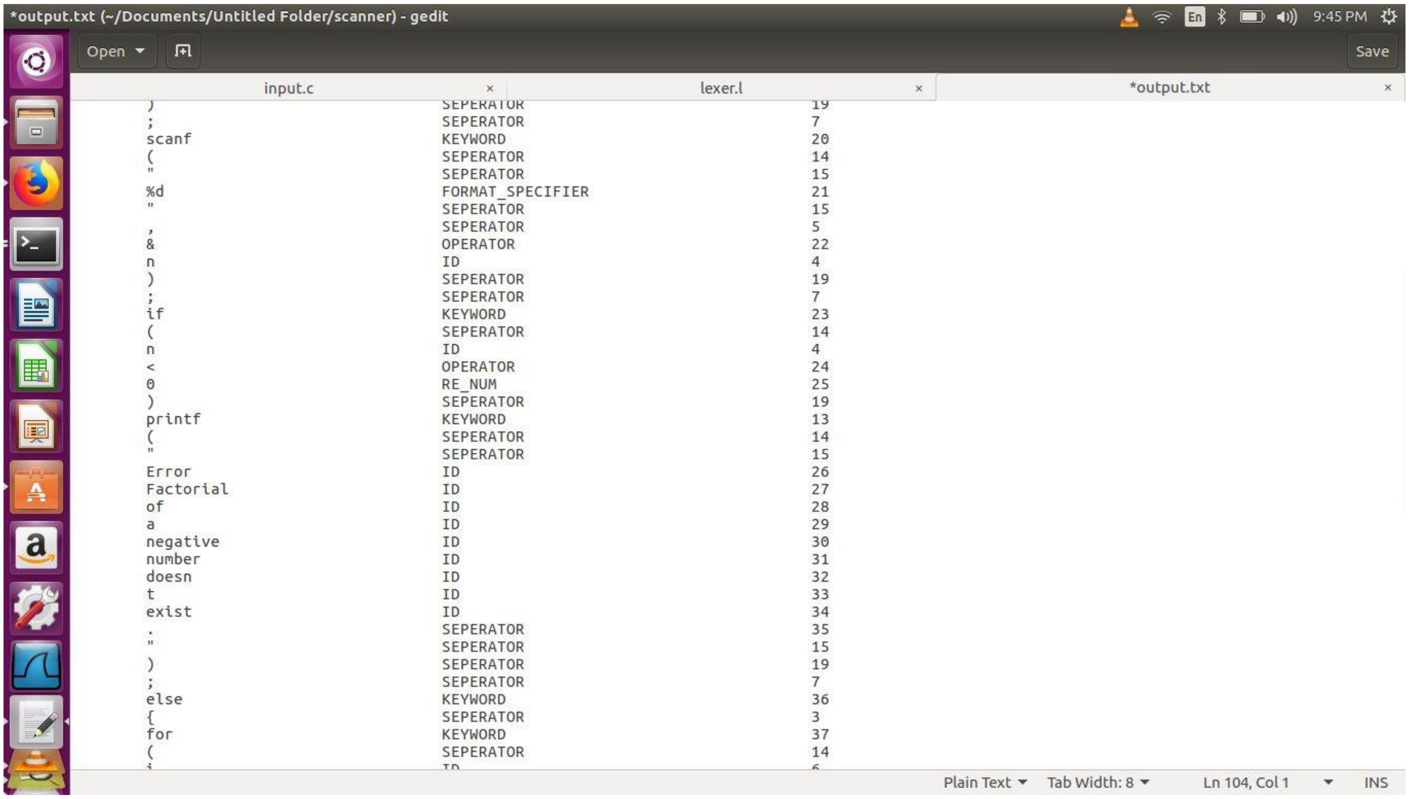


```
#
        SYMBOL                                                  TOKEN                           Attribute Number
        include                                                 KEYWORD                                 0
        <                                                       OPERATOR                                1
        stdio                                                   ID                                      2
        .                                                       SEPERATOR                               3
        h                                                       ID                                      4
        >                                                       OPERATOR                                5
        int                                                     KEYWORD                                 6
        main()                                                  FUNCTION CALL                           7
        {                                                       SEPERATOR                               8
        n                                                       ID                                      9
        ,                                                       SEPERATOR                               10
        i                                                       ID                                      11
        ;                                                       SEPERATOR                               12
        unsigned                                                KEYWORD                                 13
        long                                                    KEYWORD                                 14
        factorial                                               ID                                      15
        =                                                       OPERATOR                                16
        1                                                       RE_NUM                                  17
        printf                                                  KEYWORD                                 18
        (                                                       SEPERATOR                               19
        "Enter an integer: "                                    String literal                          20
        )                                                       SEPERATOR                               21
        scanf                                                   KEYWORD                                 22
        "%d"                                                    String literal                          23
        &                                                       OPERATOR                                24
        if                                                      KEYWORD                                 25
        0                                                       RE_NUM                                  26
        "Error! Factorial of a negative number doesn't exist."  String literal                          27
        else                                                    KEYWORD                                 28
        for                                                     KEYWORD                                 29
        <=                                                      OPERATOR                                30
        ++                                                      OPERATOR                                31
        *                                                       OPERATOR                                32
        }                                                       SEPERATOR                               33
        "Factorial of %d = %d"                                  String literal                          34
        return                                                  KEYWORD                                 35
```

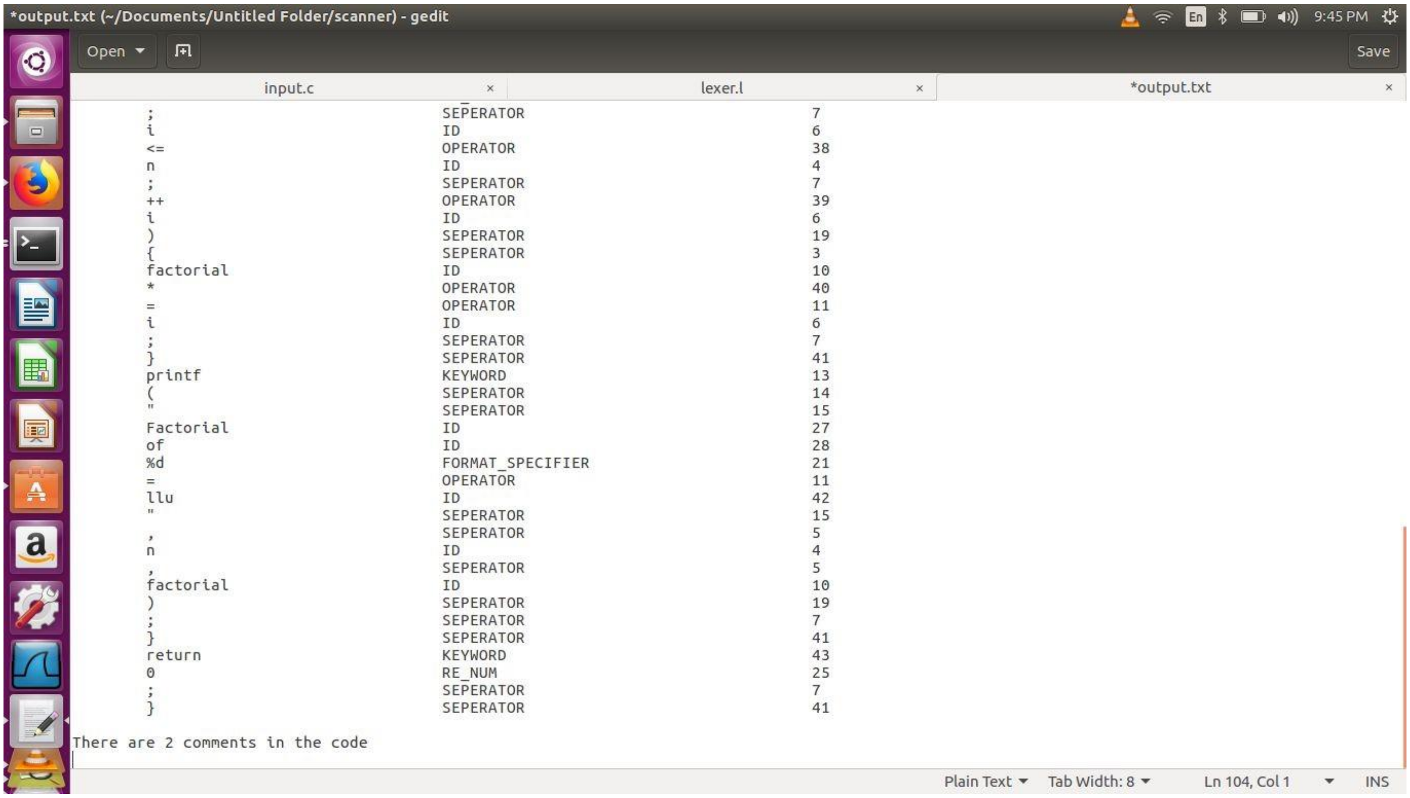
Output with format specifiers and describing the re-occurrence of symbols in the program



```
%
        SYMBOL                          TOKEN                                   Attribute Number
        #include<stdio.h>               PRE PROCESSOR                                   0
        int                             KEYWORD                                         1
        main()                          FUNCTION CALL                                   2
        {                               SEPERATOR                                       3
        int                             KEYWORD                                         1
        n                               ID                                              4
        ,                               SEPERATOR                                       5
        i                               ID                                              6
        ;                               SEPERATOR                                       7
        unsigned                        KEYWORD                                         8
        long                            KEYWORD                                         9
        long                            KEYWORD                                         9
        factorial                       ID                                              10
        =                               OPERATOR                                        11
        1                               RE_NUM                                          12
        ;                               SEPERATOR                                       7
        printf                          KEYWORD                                         13
        (                               SEPERATOR                                       14
        "                               SEPERATOR                                       15
        Enter                           ID                                              16
        an                              ID                                              17
        integer                         ID                                              18
        "                               SEPERATOR                                       15
        )                               SEPERATOR                                       19
        ;                               SEPERATOR                                       7
        scanf                           KEYWORD                                         20
        (                               SEPERATOR                                       14
        "                               SEPERATOR                                       15
        %d                              FORMAT_SPECIFIER                                21
        "                               SEPERATOR                                       15
        ,                               SEPERATOR                                       5
        &                               OPERATOR                                        22
        n                               ID                                              4
        )                               SEPERATOR                                       19
        ;                               SEPERATOR                                       7
```



```
        )                               SEPERATOR                                       19
        ;                               SEPERATOR                                       7
        scanf                           KEYWORD                                         20
        (                               SEPERATOR                                       14
        "                               SEPERATOR                                       15
        %d                              FORMAT_SPECIFIER                                21
        "                               SEPERATOR                                       15
        ,                               SEPERATOR                                       5
        &                               OPERATOR                                        22
        n                               ID                                              4
        )                               SEPERATOR                                       19
        ;                               SEPERATOR                                       7
        if                              KEYWORD                                         23
        (                               SEPERATOR                                       14
        n                               ID                                              4
        <                               OPERATOR                                        24
        0                               RE_NUM                                          25
        )                               SEPERATOR                                       19
        printf                          KEYWORD                                         13
        (                               SEPERATOR                                       14
        "                               SEPERATOR                                       15
        Error                           ID                                              26
        Factorial                       ID                                              27
        of                              ID                                              28
        a                               ID                                              29
        negative                        ID                                              30
        number                          ID                                              31
        doesn                           ID                                              32
        t                               ID                                              33
        exist                           ID                                              34
        .                               SEPERATOR                                       35
        "                               SEPERATOR                                       15
        )                               SEPERATOR                                       19
        ;                               SEPERATOR                                       7
        else                            KEYWORD                                         36
        {                               SEPERATOR                                       3
        for                             KEYWORD                                         37
        (                               SEPERATOR                                       14
        i                               ID                                              6
```



```
        ;                             SEPERATOR                          7
        i                             ID                                 6
        <=                            OPERATOR                           38
        n                             ID                                 4
        ;                             SEPERATOR                          7
        ++                            OPERATOR                           39
        i                             ID                                 6
        )                             SEPERATOR                          19
        {                             SEPERATOR                          3
        factorial                     ID                                 10
        *                             OPERATOR                           40
        =                             OPERATOR                           11
        i                             ID                                 6
        ;                             SEPERATOR                          7
        }                             SEPERATOR                          41
        printf                        KEYWORD                            13
        (                             SEPERATOR                          14
        "                             SEPERATOR                          15
        Factorial                     ID                                 27
        of                            ID                                 28
        %d                            FORMAT_SPECIFIER                   21
        =                             OPERATOR                           11
        llu                           ID                                 42
        "                             SEPERATOR                          15
        ,                             SEPERATOR                          5
        n                             ID                                 4
        ,                             SEPERATOR                          5
        factorial                     ID                                 10
        )                             SEPERATOR                          19
        ;                             SEPERATOR                          7
        }                             SEPERATOR                          41
        return                        KEYWORD                            43
        0                             RE_NUM                             25
        ;                             SEPERATOR                          7
        }                             SEPERATOR                          41

There are 2 comments in the code
```

CODE

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <string.h>
int lines=0; int
max=0; int c=0; int
comments=0;
%}

LINE \n
letter [a-zA-Z]
 digit[0-9]
 op "&&"|"<"|">"|"<="|">="|"="|"+"|"-
"|"?"|"*"|"/"|"||"
keyword
"if"|"else"|"int"|"char"|"scanf"|"printf"|"switch"|"return"|"struct"|"do"|"while"|"void"|"for"|"float"|"main"|"inc
lude"|"auto"|"break"|"case"|"const"|"continue"|"default"|"double"|"enum"|"extern"|"goto"|"long"|"register"|"r
eturn"|"short"|"signed"|"sizeof"|"static"|"typedef"|"union"|"unsigned"|"volatile"

Id {letter}({letter}|{digit})*
func {Id}"("("int "|"float "){Id}")"
func_call {Id}"("")"
point "*"{Id}
```

```
array {Id}"["{digit}+"]"
preprocessor "#"("include"|"define ")(("<"{Id}".h>")|({Id}" "{digit}))
comment "/*"
endcomment "*/"
scomment "//"
bad "@"|"$"
all ({Id}|{digit}|{op}|{array}|{keyword})*

%%

{comment} {if(c==1){printf("\n nested comment error\n");exit(0);}else{c=1;}}

{endcomment} {c=0;comments++;}

{scomment}  if(c==0)c=2;comments++;

{LINE}  if(c==2)c=0;

({digit}|{op})+{letter}+  if(c==0)printf("Error : BAD TOKEN %s\n",yytext);

{all}*{bad}+{all}*  if(c==0)printf("Error : ILLEGAL TOKEN %s\n",yytext);



{preprocessor} if(c==0)Insert("PRE PROCESSOR",yytext,Search(yytext));

{func} if(c==0)Insert("FUNCTION" ,yytext,Search(yytext));
{func_call} if(c==0)Insert("FUNCTION CALL",yytext,Search(yytext));
{array} if(c==0)Insert("ARRAY",yytext,Search(yytext));

{digit}+("E"("+"|"-")?{digit}+)? if(c==0)Insert("RE_NUM",yytext,Search(yytext));

{digit}+"."{digit}+("E"("+"|"-")?{digit}+)? if(c==0)Insert("FLOAT",yytext,Search(yytext));
{point} if(c==0)Insert("POINTER" , yytext,Search(yytext));
{keyword}   if(c==0)Insert("KEYWORD",yytext,Search(yytext));

"\a"|"\\n"|"\\b"|"\t"|"\\t"|"\b"|"\\a" Insert("ESCAPE",yytext,Search(yytext));

{Id} if(c==0)Insert("ID",yytext,Search(yytext));

"&&"|"<"|">"|"<="|">="|"="|"+"|"-"|"?"|"*"|"/"|"&"|"||" if(c==0)Insert("OPERATOR",yytext,Search(yytext));

"{"|"}"|"["|"]"|"("|")"|"."|"\""|"\\"|";"|"," if(c==0)Insert("SEPERATOR",yytext,Search(yytext));


"%d"|"%s"|"%c"|"%f"|"%e" if(c==0)Insert("FORMAT_SPECIFIER",yytext,Search(yytext));

%%
```

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

int size=0; void
Insert(char[],char[],int); void
Display(); int Search(char[]);

struct SymbTab
{
 char label[100],symbol[100];
int addr; struct SymbTab
*next;}; struct SymbTab
*first,*last;

int main(int argc,char *argv[]){
yyin=fopen(argv[1],"r");
  ++lines;
  FILE *file=fopen("output.txt","w");
yyout=file;   yylex();    fclose(yyin);

  Display();
fclose(yyout);
return 0;
}  void Insert(char l[100],char a[100],int
op)
{   int
n;
  struct SymbTab *p;
p=malloc(sizeof(struct SymbTab));
strcpy(p->label,l);   strcpy(p-
>symbol,a);   p->addr=op;
  p->next=NULL;
if(size==0){
first=p;     last=p;
  }
  else{     last-
>next=p;     last=p;
  }
  size++;
}
int Search(char l[100])
{
               int abc=0; struct
               SymbTab *qw;
               if(size==0)
```

```
        {
                return 0;
        }
        qw=first;
        int i;
        for(i=0;i<size;i++)
        {
                if(strcmp(qw->symbol,l)==0)
                {
                        abc=qw->addr;
                        return abc;
                }
                else
                {
                        qw=qw->next;
                }
 }
        max++;
        return max;
}

void Display()
{   int
i;
  struct SymbTab *p;
p=first;
  printf("There are %d comments in the code\n",comments);
//printf("\n\tSYMBOL\t\t\t\tTOKEN\t\t\t\tAttribute Number\n");
fprintf(yyout,"\n\tSYMBOL\t\t\t\tTOKEN\t\t\t\tAttribute Number\n");
for(i=0;i<size;i++)
   {
    //printf("\t%s\t\t\t\t%s\t\t\t\t%d\n",p->symbol,p->label,p->addr); fprintf(yyout,"\t%s\t\t\t\t%s\t\t\t\t%d\n",p-
        >symbol,p->label,p->addr);
    p=p->next;
   }
}
```