National Institute of Technology Karnataka

Surathkal

Compiler Design Lab

Report - 1

Bandaru Bharath Kumar – 15CO113

Aditya Bisht – 15CO104

Introduction

A compiler is a computer program that translates instruction text (source language) into a different language (target language) of instruction text. Typically, the source language is a high-level language, while the target code is machine code. Compilers normally converts into machine code for the machine it compiles in, but special compilers known as cross-compiler.

User writes the program in C and gives it to the compiler. Then the compiler will compile the code i.e. it converts the high-level language to machine level language. That converted code is executed to get the desired output.

Pre-processor:

A pre-processor is considered as a part of the compiler, it is a tool that produces input for the compilers.

Interpreter:

An interpreter translates high level language to machine level language. The difference lies in the way they read the source code. A compiler will read the whole source code at once, creates tokens, check for syntax, etc… while the interpreter will read the statement from the input converts it to an intermediate code, executes it and then takes the next statement for execution. If error occurs interpreter will stop execution and reports the error.

Linker:

Linker is a computer program that links and merges various object files together in order to make the file executable. The major task of a linker is to search and locate referenced module/routines in a program and to determine the memory location where these codes will be loaded.

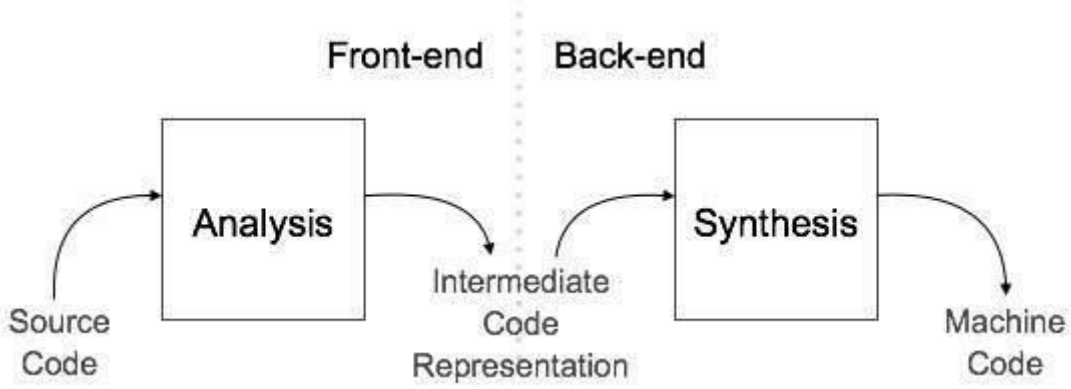
Loader:

Loader is a part of operating system and is responsible for loading executable files into memory and execute them. It calculates the size of a program (instructions and data) and creates memory space for it.

Cross – Compiler:

A compiler that runs on one platform and the generated code is capable of executed in another platform then it is called cross-compiler.

Phases Involved in a Compiler



There may be many phases involved in a compiler, however they can be assigned to one of the two stages:

1. ***The front end***: In this phase, the compiler verifies syntax and semantics according to the specific source language. In case of errors, warnings and highlighted erroneous code are shown. The input here is the source code and the output is an intermediate representation (IR) for further processing. More specifically, the intermediate outputs are token stream of the source code and sentences of the program.

2. ***The back end***: This phase performs further analysis, transformations and optimizations specific to the target CPU architecture. As a result, the target assembly code and register allocation is performed. Further optimization to ensure maximum utilization of the target hardware is done. The input here is the optimized IR and the output is the target code.

Pass: A pass refers to the traversal of a compiler through the entire program.

Phase: A phase of a compiler is a distinguishable stage, which takes input from the previous stage, processes and yields output that can be used as input for the next stage.

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.

In frontend there are four stages they are lexical analyser, syntax analyser, semantic analyser, intermediate code generator, machine independent code optimiser, machine dependent code optimiser.

Lexical Analysis:

A lexical analyser or scanner performs tokenization on a sequence of characters such as a computer program. Tokenization is a process of converting a sequence of characters into a sequence of tokens. A token is a string assigned with some meaning in a language. Lexical Analysis is the first stage in the front end of a compiler. A scanner also does other tasks, such as removing whitespaces and comments from the source program.

Lexical Analysis can be further divided into two stages - scanning and evaluating. In scanning, the source code is converted into syntactical units called lexemes and categorized into token classes. A lexeme is a sequence of characters that matches a pattern for a token. Types of tokens include identifiers, keywords, separators, operators, comments, etc. In evaluating, the lexemes are converted into processed values. This process is considered simpler when compared to parsing, etc. This can be performed using a lexer generator such as lex. A free implementation of lex found in Unix OS is called flex.

Syntax Analysis

The next phase is called the syntax analysis or **parsing**. It takes the token produced by lexical analysis as input. For syntax analysis, context-free grammars and the associated parsing techniques are to be used. 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.

The parse trees are used to show the structure of the sentence. There are many techniques for parsing the algorithms, the main techniques are top-down and bottom-up parsing.

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 analyser keeps track of identifiers, their types and expressions; whether identifiers are declared before use or not etc. The semantic analyser 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. The code generated in here should be such a way that it would be easier to be translated into target code.

Code optimization:

In this phase, the code is optimized. It removes the unwanted code lines, and arranges the sequence of statements in order to speed up the execution of the program.

Machine code generation:

The code generator takes the optimized representation of the intermediate code and then it converts it to target machine language.

Tokens:

Lexemes are said to be a sequence characters in tokens. There are some rules where every lexeme is identified. These rules are defined by means of a pattern. A pattern explains what can be a token and these are represented by regular expressions.

Keywords, constant, identifiers, numbers, operators and punctuations symbols are possible tokens to be identified.

Issues in Lexical Analyser:

It has two errors. They are lookahead, ambiguities. Lookahead is required to decide when one token will end and the next token will begin. The lexical analysis programs written with lex accept ambiguous specifications and choose the longest match possible at each input point.

Lexical errors:

A character sequences that cannot be scanned into any valid token is a lexical error.

Lexical errors are uncommon, but they still must be handled by scanner.

Code for Lexical Analyzer:

p3.l

```
%{
int comm=0;
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
int var_cnt=0;
void insert(char *yytext,char t,int l);
struct Storage
{
 char *name, token[20];
 int val;
 int li;
 struct Storage *next;
}*st,*head;
int braces=0;
int opbrc=0; //open braces ( )
int quo=0;
int line=1;
%}

D [0-9]
L [a-zA-Z]
O [-+*/=<>]
op "-="|"+="|"*="|"/="|"<<"|">>"|"<="|">="|"=="
WS [ \t\n|"\\"]
E [Ee][-+]?{D}+
PUN [,.;]
U [_]
Qu ["']
per [%&]
Keyword "break"|"case"|"continue"|"default"|"enum"|"register"|"return"|"sizeof"|"static"|"typedef"|"void"|"volatile"
Datatype "char"|"double"|"float"|"int"|"union"|"struct"
Loop "do"|"for"|"while"|"goto"
Condition "if"|"else"|"switch"

EDT "auto"|"short"|"long"|"const"|"extern"|"signed"|"unsigned"
FS "d"|"f"|"c"|"s"|"ld"|"lf"
SB "["|"]"
ws2 "\\"{Qu}

headerFunc "printf"|"scanf"
header "#include<"
close ".h>"
def "#define"|"# define"
String {Qu}({L}|{D}|{per}|{WS}|{O}|{ws2})*{Qu}
```

p3.l

```
String {Qu}({L}|{D}|{per}|{WS}|{O}|{ws2})*{Qu}
IFR ({L}|{U})({L}|{D})*
array {IFR}{SB}{D}*{SB}
wl {D}({L}|{D})*
%%
"\n" line++;
"/*" { if(comm==0) comm++; }
"*/" if(comm==1){ comm--; } else {  printf("Error: Encountered a */ before /*\n"); }

{header}{L}*{close}                          { insert(yytext,'P',line);}
{def}                                        { insert(yytext,'P',line);}

{Keyword}                                    {if(comm==0) { insert(yytext,'k',line);}}
{Datatype}                                   {if(comm==0){ insert(yytext,'d',line);} }
{Loop}                                       {if(comm==0){ insert(yytext,'l',line);} }
{Condition}                                  {if(comm==0){ insert(yytext,'c',line);} }
{EDT}+" "+{EDT}*+" "+{Datatype}              {if(comm==0){ insert(yytext,'e',line);} }

{headerFunc}                                 {if(comm==0){  insert(yytext,'h',line);}}

({D}+)+"."+{D}*                              {if(comm==0){ insert(yytext,'C',line);} }
{D}*                                         {if(comm==0){  insert(yytext,'C',line);}}
{String}                                     {if(comm==0){ insert(yytext,'s',line);} }


{Qu}+{IFR} { {printf(" %s Error Quotes not ended at line %d \n",yytext,line);}}
{IFR}+{Qu}+{IFR} { {printf(" %s Error Wrong usage of Quotes at line %d\n",yytext,line);}}
{IFR}+{Qu} { {printf(" %s Error Quotes not started at line %d\n",yytext,line);}}

{wl}  { {printf(" %s Wrong language used\n",yytext);} }
{IFR}                                        {if(comm==0&&quo==0){  insert(yytext,'v',line);}}

{IFR}+"."+{IFR}                              {if(comm==0){  insert(yytext,'v',line);} }
{PUN}                                        { if(comm==0)  insert(yytext,'p',line);}
"{"                                          {  braces++;  insert(yytext,'p',line);}
"}"                                          {  braces--;  insert(yytext,'p',line);}
"("                                          {  opbrc++;  insert(yytext,'p',line);}
")"                                          {  opbrc--; insert(yytext,'p',line);}
{Qu} {}

"--"                                         {if(comm==0){ insert(yytext,'o',line);} }
"++"                                         {if(comm==0){ insert(yytext,'o',line);} }

{per}                                        {if(comm==0){ insert(yytext,'o',line);}}
{op}                                         {if(comm==0){ insert(yytext,'o',line);}}
```

p3.l

```
{op}                                    {if(comm==0){ insert(yytext,'o',line);}}
{O}                                     {if(comm==0){ insert(yytext,'o',line);}}

{WS} {}
{SB} {if(comm==0){ insert(yytext,'p',line);} }
. { printf(" %s Errors at line %d\n",yytext,line);}
%%

int main()
{
yyin=fopen("test case 5.txt","r");
yyout=fopen("output 5.txt","w");
fprintf(yyout,"\n Symbol Table Format is:\n \tLexeme\t\t\t\tToken\t\t\tLine No\t\t\tAttribute Value\n");
yylex();
if(comm>0) { printf("Comments does not end"); }
}
int yywrap()
{
return(1);
}

void insert(char *yytext,char t,int l)
{
  int len1 = strlen(yytext);
  int i;
    char token[20];
    struct Storage *symbol,*temp,*nextptr;
    nextptr=head;
    switch(t)
    {
        case 'k':
        case 'd':
        case 'l':
        case 'e':
        case 'c':
            strcpy(token,"Keyword");
            break;

        case 'C':
            strcpy(token,"Constant");
            break;

        case 's':
            strcpy(token,"String");
            break;
```

p3.l

```
132             strcpy(token,"String");
133             break;
134         case 'a':
135         case 'v':
136         case 'I':
137          case 'h':
138             strcpy(token,"Identifier");
139             break;
140
141         case 'u':
142             strcpy(token,"User defined function");
143             break;
144
145         case 'p':
146             strcpy(token,"Punctuator");
147             break;
148
149         case 'o':
150             strcpy(token,"Operator");
151             break;
152
153         case 'P':
154             strcpy(token,"Pre processor");
155             break;
156     }
157
158         for(i=0;i<var_cnt;i++,nextptr=nextptr->next)
159         {
160             symbol = nextptr;
161             if(strcmp(symbol->name,yytext)==0)
162                 break;
163         }
164         if(i==var_cnt)
165         {
166             temp = (struct Storage*)malloc(sizeof(struct Storage));
167             temp->name = (char*)malloc((len1+1)*sizeof(char));
168             strcpy(temp->name,yytext);
169             strcpy(temp->token,token);
170             temp->val = i;
171             temp->li = l;
172             temp->next = NULL;
173             if(var_cnt==0)
174                 head = temp;
175             else
176                 symbol->next = temp;
```

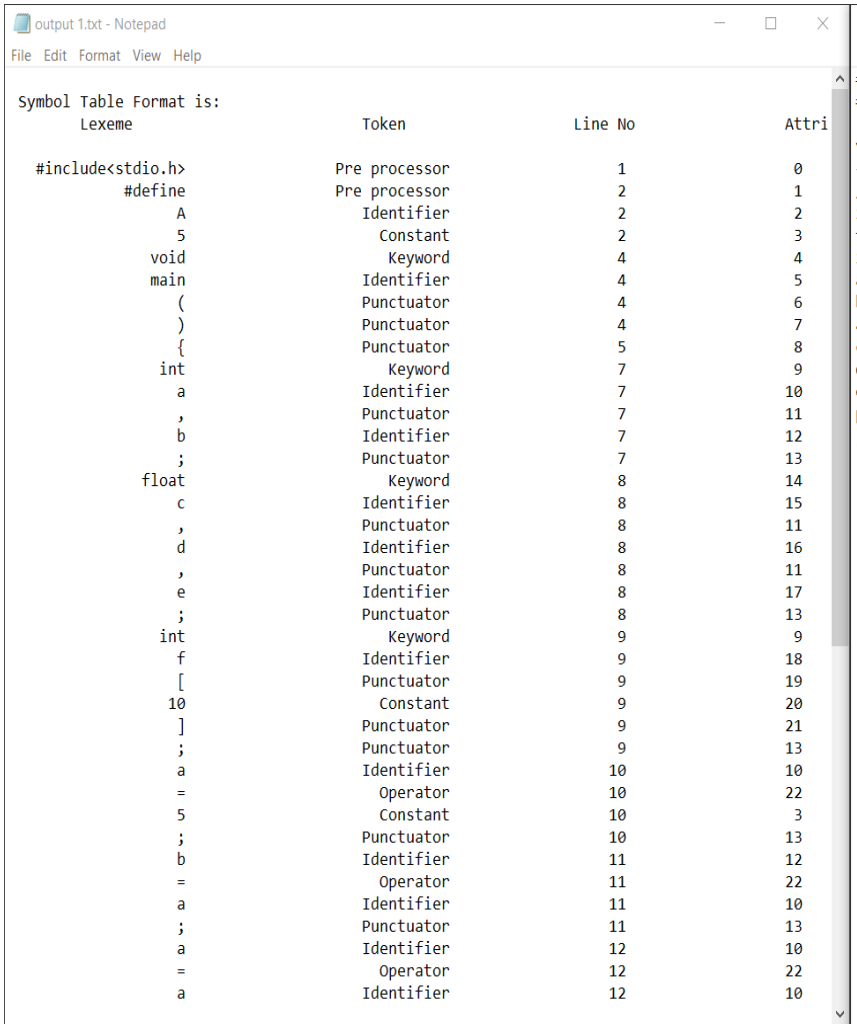
Line 148, Column 1

p3.l

```
159         {
160             symbol = nextptr;
161             if(strcmp(symbol->name,yytext)==0)
162                 break;
163         }
164         if(i==var_cnt)
165         {
166             temp = (struct Storage*)malloc(sizeof(struct Storage));
167             temp->name = (char*)malloc((len1+1)*sizeof(char));
168             strcpy(temp->name,yytext);
169             strcpy(temp->token,token);
170             temp->val = i;
171             temp->li = l;
172             temp->next = NULL;
173             if(var_cnt==0)
174                 head = temp;
175             else
176                 symbol->next = temp;
177             var_cnt++;
178 
179         }
180       fprintf(yyout,"\n%20s%30.30s%20d%20d",yytext,token,l,i);
181 }
182 
183 
```

Outputs:

Test case 1:



output 1.txt - Notepad

File Edit Format View Help

```
Symbol Table Format is:
       Lexeme                    Token              Line No              Attri

  #include<stdio.h>          Pre processor              1                  0
           #define          Pre processor              2                  1
                 A             Identifier              2                  2
                 5               Constant              2                  3
              void                Keyword              4                  4
              main             Identifier              4                  5
                 (             Punctuator              4                  6
                 )             Punctuator              4                  7
                 {             Punctuator              5                  8
               int                Keyword              7                  9
                 a             Identifier              7                 10
                 ,             Punctuator              7                 11
                 b             Identifier              7                 12
                 ;             Punctuator              7                 13
             float                Keyword              8                 14
                 c             Identifier              8                 15
                 ,             Punctuator              8                 11
                 d             Identifier              8                 16
                 ,             Punctuator              8                 11
                 e             Identifier              8                 17
                 ;             Punctuator              8                 13
               int                Keyword              9                  9
                 f             Identifier              9                 18
                 [             Punctuator              9                 19
                10               Constant              9                 20
                 ]             Punctuator              9                 21
                 ;             Punctuator              9                 13
                 a             Identifier             10                 10
                 =               Operator             10                 22
                 5               Constant             10                  3
                 ;             Punctuator             10                 13
                 b             Identifier             11                 12
                 =               Operator             11                 22
                 a             Identifier             11                 10
                 ;             Punctuator             11                 13
                 a             Identifier             12                 10
                 =               Operator             12                 22
                 a             Identifier             12                 10
```

test case 1.txt - Notepad

File Edit Format View Help

```
#include<stdio.h>
#define A 5

void main()
{
/* Normal comments*/
int a,b;
float c,d,e;
int f[10];
a=5;
b=a;
a=a+b;
c=10.32;
d=12.33;
e=c-d;
printf("%d %f",a,e);
}
```

Anaconda Prompt

```
(lex) G:\Coding files\lex>lex p3.l
p3.l:92: warning, rule cannot be matched

(lex) G:\Coding files\lex>lex p3.l

(lex) G:\Coding files\lex>gcc lex.yy.c

(lex) G:\Coding files\lex>a.exe

(lex) G:\Coding files\lex>
```

Test case 2:

output 2.txt - Notepad

File Edit Format View Help

```
Symbol Table Format is:
      Lexeme                    Token                Line No              Attri

  #include<stdio.h>          Pre processor              1                   0
              void                Keyword                3                   1
              main              Identifier               3                   2
                 (              Punctuator               3                   3
                 )              Punctuator               3                   4
                 {              Punctuator               4                   5
             float                 Keyword               5                   6
                 a              Identifier               5                   7
                 ,              Punctuator               5                   8
                 b              Identifier               5                   9
                 ,              Punctuator               5                   8
                 c              Identifier               5                  10
                 ,              Punctuator               5                   8
                 e              Identifier               5                  11
                 ;              Punctuator               5                  12
              char                 Keyword               6                  13
                 d              Identifier               6                  14
                 [              Punctuator               6                  15
                10                Constant               6                  16
                 ]              Punctuator               6                  17
                 =                Operator               6                  18
           "Hello"                  String               6                  19
                 ;              Punctuator               6                  12
                 a              Identifier               7                   7
                 =                Operator               7                  18
               5.3                Constant               7                  20
                 ;              Punctuator               7                  12
                 b              Identifier               8                   9
                 =                Operator               8                  18
              2.65                Constant               8                  21
                 ;              Punctuator               8                  12
                 c              Identifier               9                  10
                 =                Operator               9                  18
                 a              Identifier               9                   7
                 +                Operator               9                  22
                 b              Identifier               9                   9
                 ;              Punctuator               9                  12
             scanf              Identifier              10                  23
```

test case 2.txt - Notepad

File Edit Format View Help

```
#include<stdio.h>

void main()
{
float a,b,c,e;
char d[10] = "Hello";
a=5.3;
b=2.65;
c=a+b;
scanf("%f",&e);

if(e>5.0)
 printf("%f",c);
else if(e<5.0)
 printf("%s",d);
else
 printf("Nothing");
}
```

Anaconda Prompt

```
(lex) G:\Coding files\lex>lex p3.1

(lex) G:\Coding files\lex>gcc lex.yy.c

(lex) G:\Coding files\lex>a.exe

(lex) G:\Coding files\lex>
```

Test case 3:

output 3.txt - Notepad

File Edit Format View Help

```
Symbol Table Format is:
       Lexeme                    Token                Line No              Attri

  #include<stdio.h>          Pre processor                1                  0
            struct                Keyword                  3                  1
            values             Identifier                  3                  2
                 {             Punctuator                  4                  3
               int                Keyword                  5                  4
                cd             Identifier                  5                  5
                 ;             Punctuator                  5                  6
                 }             Punctuator                  6                  7
                 ;             Punctuator                  6                  6
              void                Keyword                  8                  8
              main             Identifier                  8                  9
                 (             Punctuator                  8                 10
                 )             Punctuator                  8                 11
                 {             Punctuator                  9                  3
   long long float                Keyword                 10                 12
                 d             Identifier                 10                 13
                 ,             Punctuator                 10                 14
                 c             Identifier                 10                 15
                 =               Operator                 10                 16
                 0               Constant                 10                 17
                 ;             Punctuator                 10                  6
            struct                Keyword                 11                  1
            values             Identifier                 11                  2
                 v             Identifier                 11                 18
                 ;             Punctuator                 11                  6
               int                Keyword                 12                  4
                 i             Identifier                 12                 19
                 =               Operator                 12                 16
                 3               Constant                 12                 20
                 ;             Punctuator                 12                  6
              v.cd             Identifier                 13                 21
                 =               Operator                 13                 16
                 5               Constant                 13                 22
                 ;             Punctuator                 13                  6
             scanf             Identifier                 14                 23
                 (             Punctuator                 14                 10
             "%llf"                 String                 14                 24
                 ,             Punctuator                 14                 14
```

test case 3.txt - Notepad

File Edit Format View Help

```
#include<stdio.h>

struct values
{
int cd;
};

void main()
{
long long float d,c=0;
struct values v;
int i=3;
v.cd=5;
scanf("%llf",&d);
while(d--)
 {
   c++;
 }
while(i--)
{
  while(d>0)
  {
   d--;
   v.cd++;
  }
}

printf("%llf",c);
}
```

Anaconda Prompt

```
(lex) G:\Coding files\lex>lex p3.l

(lex) G:\Coding files\lex>gcc lex.yy.c

(lex) G:\Coding files\lex>a.exe

(lex) G:\Coding files\lex>gcc lex.yy.c

(lex) G:\Coding files\lex>lex p3.l

(lex) G:\Coding files\lex>gcc lex.yy.c

(lex) G:\Coding files\lex>a.exe

(lex) G:\Coding files\lex>
```

Test case 4:

output 4.txt - Notepad

File Edit Format View Help

Symbol Table Format is:

Lexeme	Token	Line No	Attri
#include<stdio.h>	Pre processor	1	0
Void	Identifier	3	1
main	Identifier	3	2
(	Punctuator	3	3
)	Punctuator	3	4
{	Punctuator	4	5
This	Identifier	5	6
is	Identifier	5	7
error	Identifier	5	8
char	Keyword	6	9
a	Identifier	6	10
[	Punctuator	6	11
10	Constant	6	12
]	Punctuator	6	13
=	Operator	6	14
;	Punctuator	6	15
int	Keyword	7	16
;	Punctuator	7	15
printf	Identifier	8	17
(	Punctuator	8	3
)	Punctuator	8	4
;	Punctuator	8	15
printf	Identifier	9	17
(	Punctuator	9	3
"hello /" hi "	String	9	18
)	Punctuator	9	4
;	Punctuator	9	15
}	Punctuator	10	19

test case 4.txt - Notepad

File Edit Format View Help

```
#include<stdio.h>

Void main()
{
/*Nested /* comments. */ This is error*/
char a[10] = "asdas;
int a"b;
printf("hello);
printf("hello /" hi ");
}
```

Anaconda Prompt

```
(lex) G:\Coding files\lex>gcc lex.yy.c

(lex) G:\Coding files\lex>a.exe

(lex) G:\Coding files\lex>lex p3.l

(lex) G:\Coding files\lex>gcc lex.yy.c

(lex) G:\Coding files\lex>a.exe
Error: Encountered a */ before /*
 "asdas Error Quotes not ended at line 6
 a"b Error Wrong usage of Quotes at line 7
 "hello Error Quotes not ended at line 8

(lex) G:\Coding files\lex>
```

Test Case 5:

output 5.txt - Notepad

File Edit Format View Help

```
Symbol Table Format is:
       Lexeme                       Token                 Line No              Attri

  #include<stdio.h>              Pre processor                1                   0
               void                   Keyword                 3                   1
               main                Identifier                 3                   2
                  (               Punctuator                  3                   3
                  )               Punctuator                  3                   4
                  {               Punctuator                  4                   5
                int                  Keyword                  5                   6
                  a                 Identifier                5                   7
                  [               Punctuator                  5                   8
                 10                 Constant                  5                   9
                  ]               Punctuator                  5                  10
                  ;               Punctuator                  5                  11
                int                  Keyword                  6                   6
                  ,               Punctuator                  6                  12
                 _l                 Identifier                6                  13
                  ;               Punctuator                  6                  11
                 _l                 Identifier                7                  13
                  =                  Operator                 7                  14
                 15                 Constant                  7                  15
                  ;               Punctuator                  7                  11
                  =                  Operator                 8                  14
                 10                 Constant                  8                   9
                  ;               Punctuator                  8                  11
             printf                Identifier                9                  16
                  (               Punctuator                  9                   3
               "%d"                    String                 9                  17
                  ,               Punctuator                  9                  12
                 _l                 Identifier                9                  13
                  )               Punctuator                  9                   4
                  ;               Punctuator                  9                  11
             printf                Identifier                10                  16
                  (               Punctuator                  10                  3
"hsbda \" asd \" sa"                    String                 10                  18
                  )               Punctuator                  10                  4
                  ;               Punctuator                  10                  11
                  }               Punctuator                  11                  19
               ygvg                Identifier                12                  20
```

test case 5.txt - Notepad

File Edit Format View Help

```
#include<stdio.h>

void main()
{
int a[10];
int 0b,_l;
_l=15;
0b=10;
printf("%d",_l);
printf("hsbda \" asd \" sa");
}
/* /* These comments are wrong */ygvg*/
/* /* These are valid comments */
```

Anaconda Prompt

```
(lex) G:\Coding files\lex>lex p3.l

(lex) G:\Coding files\lex>gcc lex.yy.c

(lex) G:\Coding files\lex>a.exe
 0b Wrong language used
 0b Wrong language used
Error: Encountered a */ before /*

(lex) G:\Coding files\lex>
```