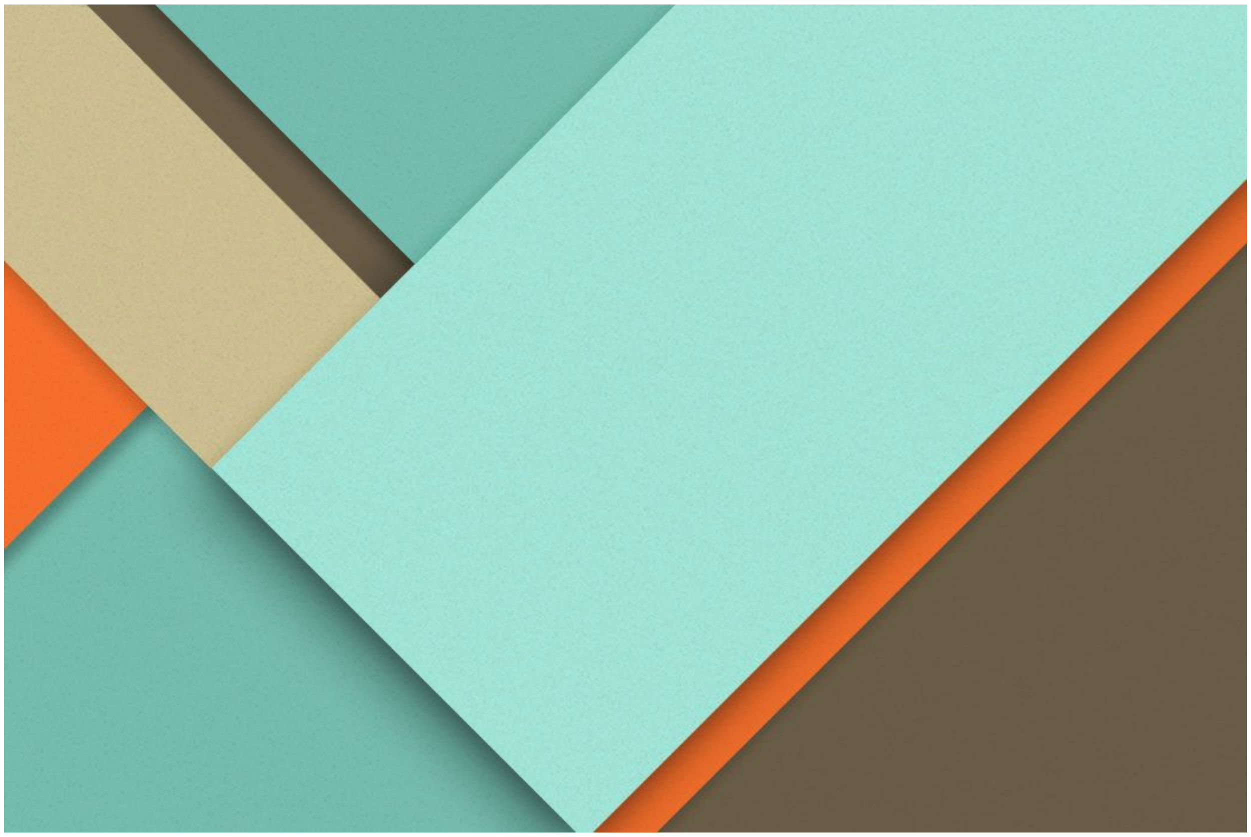


Phase 1 - Lexical Analysis Report

20th January, 2018

Sagar Bharadwaj - 15CO141
Aneesh Aithal - 15CO107

Introduction

The name **Compiler** is primarily used for programs that translate source code from a high-level programming language to a lower level language (Example : assembly language object code or machine code) to create an executable program.

In other words, compiler parses the code in source language and converts it into target language. The source language is usually a high level programming language and the target language is usually in machine understandable form.

Structure of Compiler

The structure of a compiler is divided into two main phases :

- Analysis Phase
- Synthesis Phase

Analysis Phase

The analysis part breaks up the source program into constituent pieces and imposes a grammatical structure on them. If the program is found to violate the grammar of the language, then, appropriate error messages are raised by the compiler in this phase. The analysis part also collects information about the source program and stores it in a data structure called a symbol table, which is passed along with the intermediate representation to the synthesis part.

The analysis phase is divided into the following four sub phases :

- **Lexical analysis** : Lexical analysis or tokenization is the process of converting a sequence of characters in a computer program into a sequence of tokens.
- **Syntax analysis** : Syntax analysis or parsing refers to the formal analysis of a string of tokens into its constituents, resulting in a parse tree showing their syntactic relation to each other.

- **Semantic analysis** : The semantic analyzer uses the syntax tree and the information in the symbol table to check the source program for semantic consistency with the language definition.
- **Intermediate code generation** : This phase involves generation of an explicit low-level or machine-like intermediate representation, which can be thought of as a program for an abstract machine.

Synthesis Phase

Synthesis phase involves the following two phases :

Code Optimisation : This phase involves optimisation of intermediate code to generate an optimised target code. Compilers specifically written for producing optimized build spend most time in this phase.

Code Generation : The code generator takes as input an intermediate representation of the source program and maps it into the target language.

Lexical Analysis

The lexical analyzer reads the stream of characters making up the source program and groups the characters into meaningful sequences called lexemes. The lexical analyser outputs a token for each lexeme it recognises. The token is of the form :

(token name, attribute value)

These sequence of tokens are passed on to the parser in the syntax phase. The symbol table is generated in this phase. Usually all the identifiers in a language are inserted into the symbol table.

The symbol table generated generally uses hash organisation. The symbol table is incrementally updated during the many phases of the compilation process.

Code

```
%{
  #define RED   "\x1B[31m"
  #define RESET "\x1B[0m"
  #define GRN   "\x1B[32m"
  #define BLU   "\x1B[34m"

  int lookUpSymbolItem();
  void insertSymbolItem();

  const int symbolTableSize = 1000;
  typedef struct symbolItemStruct{
     char tokenValue[100];
     char tokenType[100];
     int lineNumber;
     struct symbolItemStruct* next;
  } symbolItem;
  symbolItem * symbolTable[1000];
  symbolItem * constantTable[1000];

  int lineNo = 1;
  int comment = 0;
%}

keyword
char|int|float|short|long|unsigned|signed|main|while|for|break|case|if|else|continue
|default|do|goto|return|void|struct|switch
number        [0-9]
letter        [a-zA-Z]
operator      [+-<>*=/!%^&.]
function      (_|{letter})({letter}|{number}|_)*"()"


%%
\/\/(.)*[\n]                    {lineNo++;}

[/][*]                         { printf("%-20s%20s%20d\n", "OPEN COMMENT",yytext, lineNo);
comment++; }
```

```
[*][/]                          { printf("%-20s%20s%20d\n", "CLOSE COMMENT",yytext, lineNo);
comment--; }

{function}                      { if(!comment) printf("%-20s%20s%20d\n", "FUNCTION", yytext,
lineNo); }

#(.*)                           { if(!comment) printf("%-20s%20s%20d\n","PREPROCESSOR", yytext,
lineNo); }

[\n]                            { lineNo++; }

[{]                             { if(!comment) printf("%-20s%20s%20d\n","LEFT BRACE", yytext,
lineNo); }
[}]                             { if(!comment) printf("%-20s%20s%20d\n", "RIGHT BRACE", yytext,
lineNo); }

\"[^"\n]*["\n]                      { if(!comment) {
                                    if(yytext[yyleng-1]!='"'){
                                        printf(RED "Error : Quote unbalanced at line number %d\n"
RESET,lineNo);
                                        lineNo++;
                                    }
                                    printf("%-20s%20s%20d\n", "STRING", yytext, lineNo);
                                    insertSymbolItem(yytext, "STRING", lineNo, 1);
                                  }
                                }

{keyword}                           { if(!comment){
                                    printf("%-20s%20s%20d\n", "KEYWORD", yytext, lineNo);
                                    // if(!lookUpSymbolItem(yytext))
                                    //     insertSymbolItem(yytext, "KEYWORD", lineNo, 0);
                                  }
                                }

[(]                             { if(!comment) printf("%-20s%20s%20d\n", "OPEN PARANTHESIS",
yytext, lineNo); }
[)]                             { if(!comment) printf("%-20s%20s%20d\n", "CLOSE PARANTHESIS",
yytext, lineNo); }
[\[\]]                          { if(!comment) printf("%-20s%20s%20d\n", "SQUARE BRACKETS",
yytext, lineNo); }
[,]                             { if(!comment) printf("%-20s%20s%20d\n", "COMMA", yytext, lineNo); }
```

```
[\t ]                     { ; }

[;]                      { if(!comment) printf("%-20s%20s%20d\n", "SEMICOLON", yytext,
lineNo); }

\'.\'                    { if(!comment) {
                           printf("%-20s%20s%20d\n", "CHARACTER", yytext, lineNo);
                           insertSymbolItem(yytext, "CHARACTER", lineNo, 1);
                           }
                         }

{number}+\.{number}+           { if(!comment) {
                           printf("%-20s%20s%20d\n", "FLOAT NUMBER", yytext, lineNo);
                           insertSymbolItem(yytext, "FLOAT", lineNo, 1);
                           }
                         }

{number}+                      { if(!comment) {
                           printf("%-20s%20s%20d\n", "NUMBER", yytext, lineNo);
                           insertSymbolItem(yytext, "INTEGER", lineNo, 1);
                           }
                         }

(_|{letter})({letter}|{number}|_)*  { if(!comment){
                           printf("%-20s%20s%20d\n", "IDENTIFIER", yytext, lineNo);
                           if(!lookUpSymbolItem(yytext))
                              insertSymbolItem(yytext, "IDENTIFIER", lineNo, 0);
                          }
                         }

{operator}                     { if(!comment) printf("%-20s%20s%20d\n", "OPERATOR", yytext,
lineNo); }

.                        { if(!comment) printf(RED "Error : Invalid Token %s at Line %d\n" RESET,
yytext, lineNo); }

%%
```

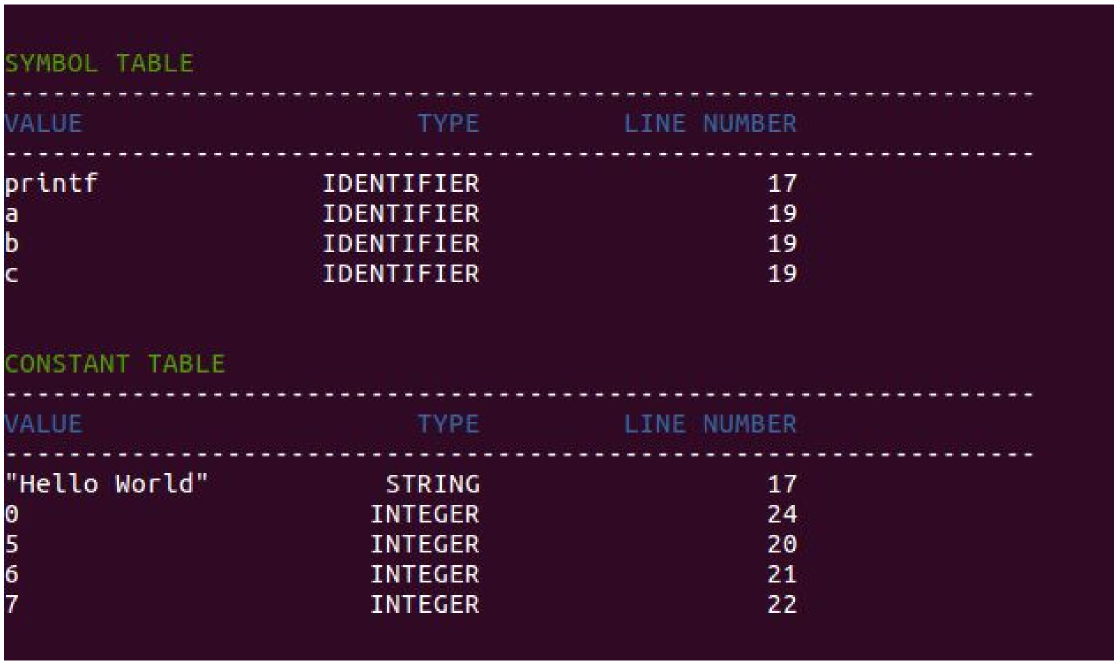
```
int yywrap(){
 return 1;
}

void initSymbolTable(){
  int i;
  for(i = 0;i<symbolTableSize; i++)
    symbolTable[i] = NULL;

  for(i = 0;i<symbolTableSize; i++)
    constantTable[i] = NULL;
}

int hash(unsigned char *str)
{
  unsigned long hashVar = 5381;
  int c;

  while (c = *str++)
    hashVar = (((hashVar << 5) + hashVar) + c)%1000;

  return hashVar;
}

symbolItem* createSymbolItem(char *tokenValue, char *tokenType, int lineNumber){
  symbolItem *item = (symbolItem*)malloc(sizeof(symbolItem));
  strcpy(item->tokenValue, tokenValue);
  strcpy(item->tokenType, tokenType);
  item->lineNumber = lineNumber;
  item->next = NULL;

  return item;
}

void insertSymbolItem(char *tokenValue, char *tokenType, int lineNumber, int tableno){
  int hashIndex = hash(tokenValue);

  symbolItem *item = createSymbolItem(tokenValue, tokenType, lineNumber);

  if(tableno == 0)
  {
    symbolItem * temp = symbolTable[hashIndex];
    while(temp!=NULL && temp->next!=NULL)
        temp = temp->next;

    if(temp == NULL)
        symbolTable[hashIndex] = item;
    else
        temp->next = item;
  }

  else
  {
    symbolItem * temp = constantTable[hashIndex];
    while(temp!=NULL && temp->next!=NULL)
        temp = temp->next;

    if(temp == NULL)
        constantTable[hashIndex] = item;
    else
        temp->next = item;
  }

}

int lookUpSymbolItem(char * tokenValue){
  int hashIndex = hash(tokenValue);

    symbolItem * temp = symbolTable[hashIndex];
    while(temp!=NULL && strcmp(tokenValue, temp->tokenValue)!=0)
        temp=temp->next;

    if(temp==NULL) return 0;
    else return 1;

}

void printSymbolItem(symbolItem * item){
  printf("%-20s%10s%20d\n",item->tokenValue, item->tokenType, item->lineNumber);
}

void showSymbolTable(){
  int i;
  printf("\n-----------------------------------------------------------------\n");
  printf(BLU "%-20s%10s%24s\n","VALUE","TYPE","LINE NUMBER" RESET);
  printf("-----------------------------------------------------------------\n");

  for(int i=0;i<symbolTableSize;i++){
    symbolItem* temp = symbolTable[i];
    while(temp!=NULL){
        printSymbolItem(temp);
        temp=temp->next;
    }
  }
}

void showConstantTable(){
  int i;
  printf("\n-----------------------------------------------------------------\n");
  printf(BLU "%-20s%10s%24s\n","VALUE","TYPE","LINE NUMBER" RESET);
  printf("-----------------------------------------------------------------\n");

  for(int i=0;i<symbolTableSize;i++){
    symbolItem* temp = constantTable[i];
    while(temp!=NULL){
        printSymbolItem(temp);
        temp=temp->next;
    }
  }
}

int main(int argc, char** argv){
  if(argc < 2){
    printf(RED "Pass input file as command line argument\n" RESET);
    exit(0);
  }
  initSymbolTable();
  yyin = fopen(argv[1], "r");
  printf("\n\n-----------------------------------------------------------------\n");
  printf(BLU "%-20s%20s%24s\n", "TOKEN VALUE", "TOKEN TYPE", "LINE NUMBER" RESET);
  printf("-----------------------------------------------------------------\n");
  yylex();
  if(comment)
    printf(RED "Error : Error in parsing comments" RESET);

  printf(GRN "\n\nSYMBOL TABLE" RESET);
  showSymbolTable();
  printf(GRN "\n\nCONSTANT TABLE" RESET);
  showConstantTable();
  printf("\n\n");
}
```

Testcases

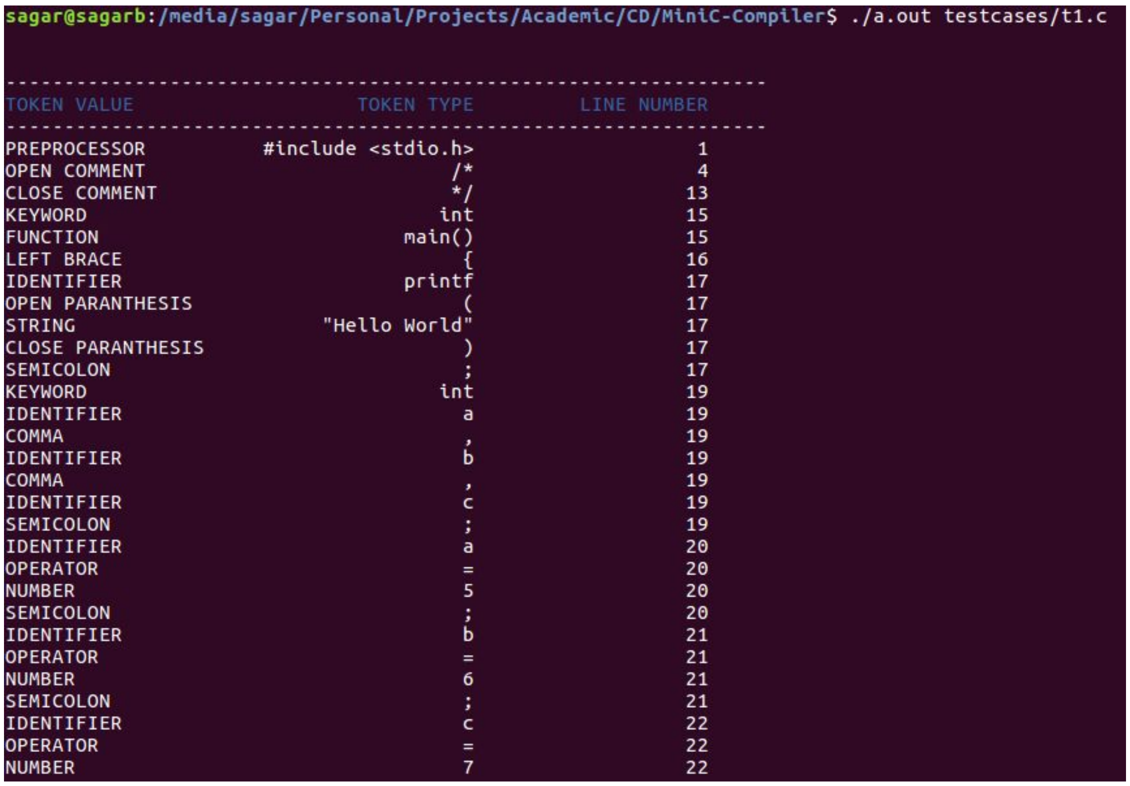
Test Case 1 - Code

```
#include <stdio.h>

// T1
/*
A very Basic Program
Tests for :
 - Comment removal (Both Single line and multi line)
 - Basic Tokenisation
   - Keywords
   - Identifiers
   - Strings
   - Function Calls
*/

int main()
{
   printf("Hello World");

   int a,b,c;
   a = 5;
   b = 6;
   c = 7;
   b = a + c;
   return 0;
}
```

Test Case 1 - Screenshot



```
SYMBOL TABLE
------------------------------------------------------------------------
VALUE                         TYPE          LINE NUMBER
------------------------------------------------------------------------
printf                  IDENTIFIER                      17
a                       IDENTIFIER                      19
b                       IDENTIFIER                      19
c                       IDENTIFIER                      19


CONSTANT TABLE
------------------------------------------------------------------------
VALUE                         TYPE          LINE NUMBER
------------------------------------------------------------------------
"Hello World"               STRING                      17
0                          INTEGER                      24
5                          INTEGER                      20
6                          INTEGER                      21
7                          INTEGER                      22
```

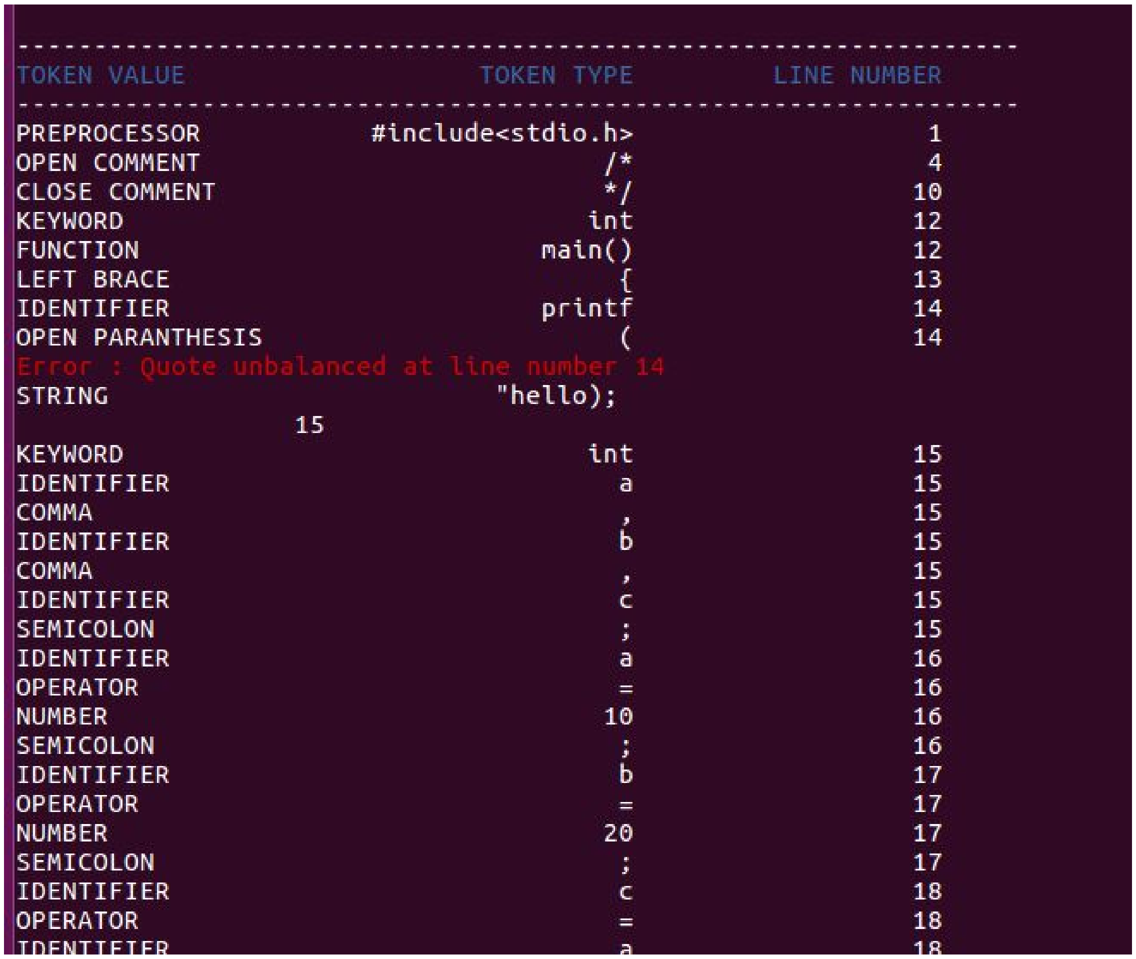


```
sagar@sagarb:/media/sagar/Personal/Projects/Academic/CD/MiniC-Compiler$ ./a.out testcases/t1.c


----------------------------------------------------------------
TOKEN VALUE                     TOKEN TYPE          LINE NUMBER
----------------------------------------------------------------
PREPROCESSOR          #include <stdio.h>                   1
OPEN COMMENT                          /*                   4
CLOSE COMMENT                         */                  13
KEYWORD                              int                  15
FUNCTION                          main()                  15
LEFT BRACE                             {                  16
IDENTIFIER                        printf                  17
OPEN PARANTHESIS                       (                  17
STRING                     "Hello World"                  17
CLOSE PARANTHESIS                      )                  17
SEMICOLON                              ;                  17
KEYWORD                              int                  19
IDENTIFIER                             a                  19
COMMA                                  ,                  19
IDENTIFIER                             b                  19
COMMA                                  ,                  19
IDENTIFIER                             c                  19
SEMICOLON                              ;                  19
IDENTIFIER                             a                  20
OPERATOR                               =                  20
NUMBER                                 5                  20
SEMICOLON                              ;                  20
IDENTIFIER                             b                  21
OPERATOR                               =                  21
NUMBER                                 6                  21
SEMICOLON                              ;                  21
IDENTIFIER                             c                  22
OPERATOR                               =                  22
NUMBER                                 7                  22
```

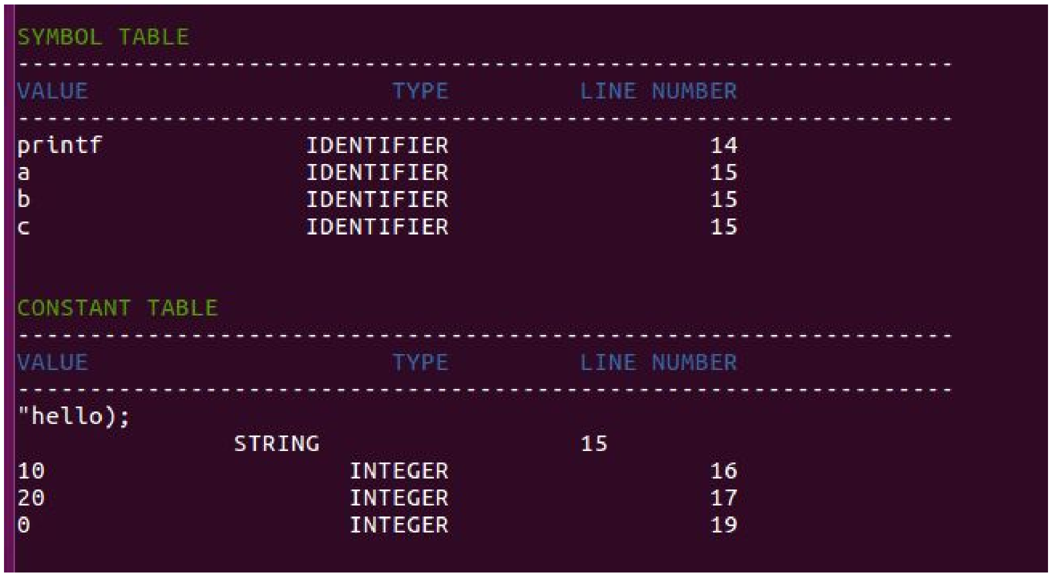
Test Case 2 - Code

```
// T2
/*
Test case to test following errors :
- Missing quotes

Extened token support for :
    - Operators including Arithmetic, assignment and Comma
*/

int main()
{
    printf("hello);
    int a,b,c;
    a = 10;
    b = 20;
    c = a+b;
    return 0;
}
```

Test Case 2 - Screenshot



```
-----------------------------------------------------------------
TOKEN VALUE                        TOKEN TYPE          LINE NUMBER
-----------------------------------------------------------------
PREPROCESSOR           #include<stdio.h>                    1
OPEN COMMENT                          /*                    4
CLOSE COMMENT                         */                   10
KEYWORD                              int                   12
FUNCTION                          main()                   12
LEFT BRACE                             {                   13
IDENTIFIER                        printf                   14
OPEN PARANTHESIS                       (                   14
Error : Quote unbalanced at line number 14
STRING                           "hello);
                  15
KEYWORD                              int                   15
IDENTIFIER                             a                   15
COMMA                                  ,                   15
IDENTIFIER                             b                   15
COMMA                                  ,                   15
IDENTIFIER                             c                   15
SEMICOLON                              ;                   15
IDENTIFIER                             a                   16
OPERATOR                               =                   16
NUMBER                                10                   16
SEMICOLON                              ;                   16
IDENTIFIER                             b                   17
OPERATOR                               =                   17
NUMBER                                20                   17
SEMICOLON                              ;                   17
IDENTIFIER                             c                   18
OPERATOR                               =                   18
```

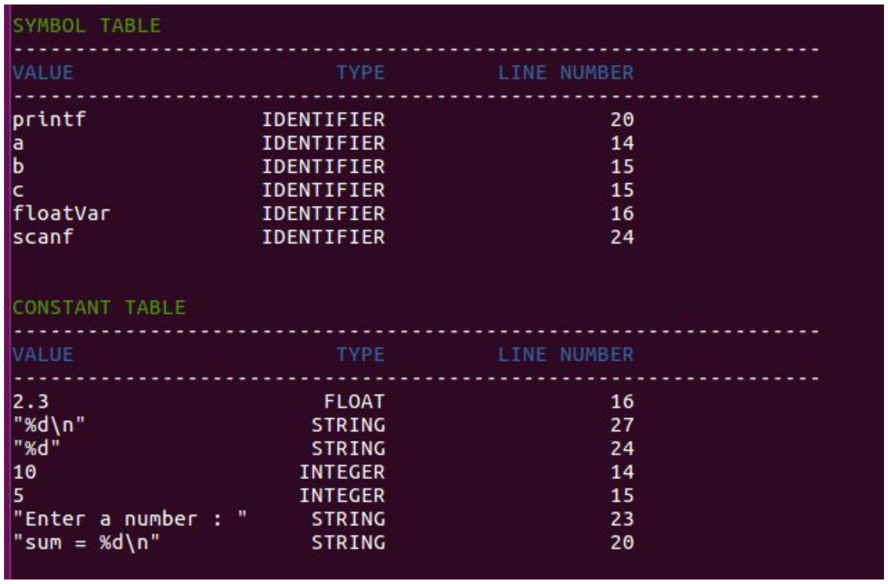


```
SYMBOL TABLE
-------------------------------------------------------------
VALUE                       TYPE         LINE NUMBER
-------------------------------------------------------------
printf            IDENTIFIER                 14
a                 IDENTIFIER                 15
b                 IDENTIFIER                 15
c                 IDENTIFIER                 15


CONSTANT TABLE
-------------------------------------------------------------
VALUE                       TYPE         LINE NUMBER
-------------------------------------------------------------
"hello);
             STRING                  15
10                   INTEGER                 16
20                   INTEGER                 17
0                    INTEGER                 19
```

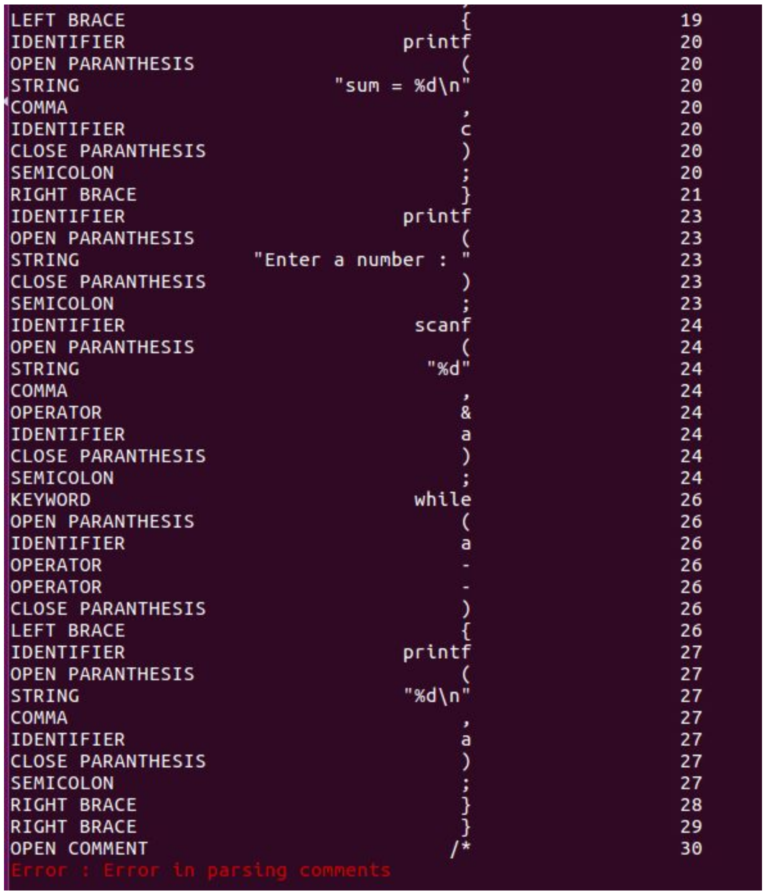
Test Case 3 - Code

```
#include <stdio.h>

// T3
/*
Extended support for datatypes :
    - short and long int
    - float
Added support for while loop
Errors include :
    - Unclosed comment
*/
int main()
{
    short int a = 10;
    long long b = 5, c;
    float floatVar = 2.3;
    c = a + b;

    while(a--){
        printf("sum = %d\n", c);
    }

    printf("Enter a number : ");
    scanf("%d",&a);

    while(a--){
        printf("%d\n",a);
    }
}
/* Hello this is a sample comment
```

Test Case 3 - Screenshot



```
SYMBOL TABLE
-----------------------------------------------------------------
VALUE                     TYPE         LINE NUMBER
-----------------------------------------------------------------
printf              IDENTIFIER                  20
a                   IDENTIFIER                  14
b                   IDENTIFIER                  15
c                   IDENTIFIER                  15
floatVar            IDENTIFIER                  16
scanf               IDENTIFIER                  24


CONSTANT TABLE
-----------------------------------------------------------------
VALUE                     TYPE         LINE NUMBER
-----------------------------------------------------------------
2.3                      FLOAT                  16
"%d\n"                  STRING                  27
"%d"                    STRING                  24
10                     INTEGER                  14
5                      INTEGER                  15
"Enter a number : "     STRING                  23
"sum = %d\n"            STRING                  20
```

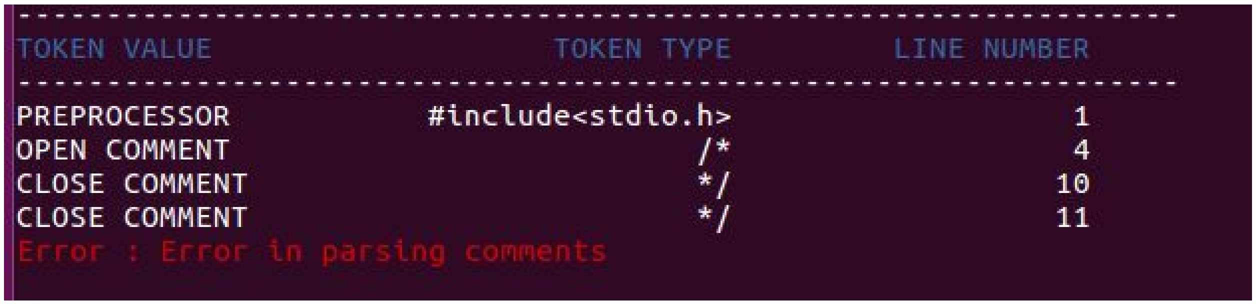


```
LEFT BRACE                                {                  19
IDENTIFIER                           printf                  20
OPEN PARANTHESIS                          (                  20
STRING                         "sum = %d\n"                  20
COMMA                                     ,                  20
IDENTIFIER                                c                  20
CLOSE PARANTHESIS                         )                  20
SEMICOLON                                 ;                  20
RIGHT BRACE                               }                  21
IDENTIFIER                           printf                  23
OPEN PARANTHESIS                          (                  23
STRING                  "Enter a number : "                  23
CLOSE PARANTHESIS                         )                  23
SEMICOLON                                 ;                  23
IDENTIFIER                            scanf                  24
OPEN PARANTHESIS                          (                  24
STRING                                 "%d"                  24
COMMA                                     ,                  24
OPERATOR                                  &                  24
IDENTIFIER                                a                  24
CLOSE PARANTHESIS                         )                  24
SEMICOLON                                 ;                  24
KEYWORD                               while                  26
OPEN PARANTHESIS                          (                  26
IDENTIFIER                                a                  26
OPERATOR                                  -                  26
OPERATOR                                  -                  26
CLOSE PARANTHESIS                         )                  26
LEFT BRACE                                {                  26
IDENTIFIER                           printf                  27
OPEN PARANTHESIS                          (                  27
STRING                               "%d\n"                  27
COMMA                                     ,                  27
IDENTIFIER                                a                  27
CLOSE PARANTHESIS                         )                  27
SEMICOLON                                 ;                  27
RIGHT BRACE                               }                  28
RIGHT BRACE                               }                  29
OPEN COMMENT                             /*                  30
Error : Error in parsing comments
```

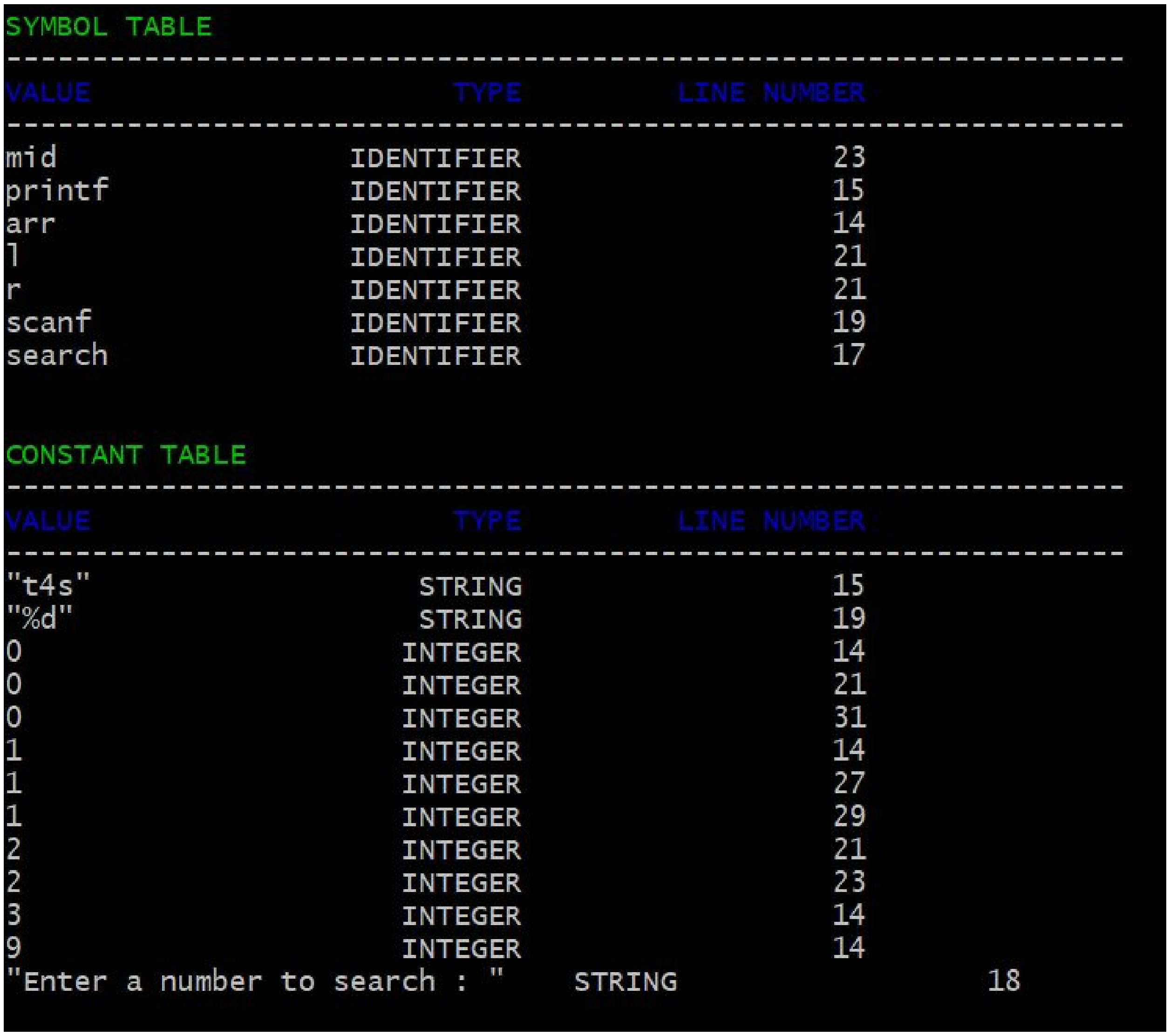
Test Case 4 - Code

```
// T4
/*
Suppport extended for :
    - Combined declaration and definition of arrays
    - Array subscript operator ([])
Errors :
    - Extra comment closing token
*/
*/

int main()
{
    int arr[3] = {-1,0,9};
    printf("t4s");

    int search;
    printf("Enter a number to search : ");
    scanf("%d", &search);

    int l = 0,r = 2;
    while(l<=r){
        int mid = (l+r)/2;
        if(arr[mid] == search)
            break;
        else if(arr[mid]<search)
            r = mid - 1;
        else
            l = mid + 1;
    }
    return 0;
}
```

Test Case 4 - Screenshot



```
-----------------------------------------------------------------
TOKEN VALUE                     TOKEN TYPE          LINE NUMBER
-----------------------------------------------------------------
PREPROCESSOR           #include<stdio.h>                      1
OPEN COMMENT                          /*                      4
CLOSE COMMENT                         */                     10
CLOSE COMMENT                         */                     11
Error : Error in parsing comments
```

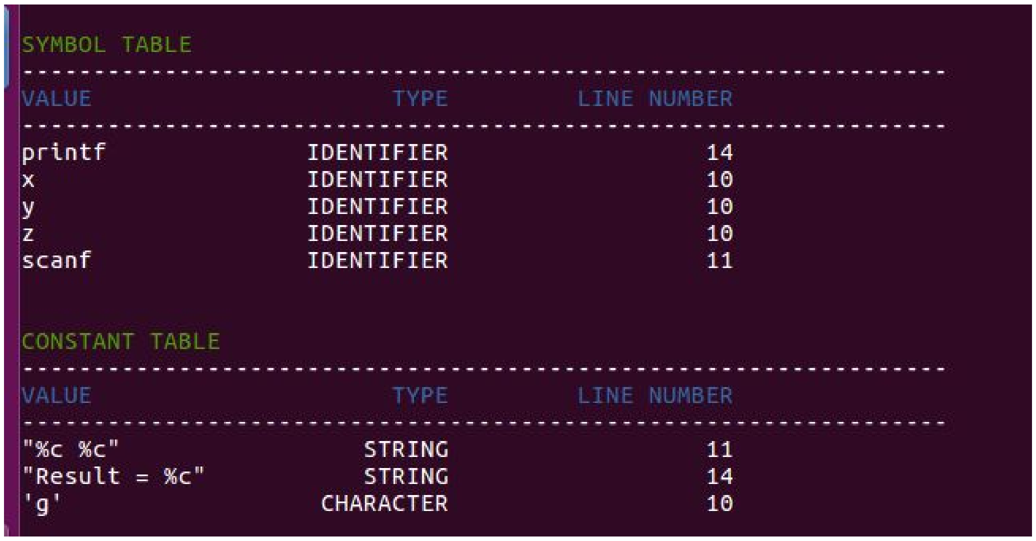


```
SYMBOL TABLE
------------------------------------------------------------------
VALUE                     TYPE          LINE NUMBER
------------------------------------------------------------------
mid                 IDENTIFIER                   23
printf              IDENTIFIER                   15
arr                 IDENTIFIER                   14
l                   IDENTIFIER                   21
r                   IDENTIFIER                   21
scanf               IDENTIFIER                   19
search              IDENTIFIER                   17


CONSTANT TABLE
------------------------------------------------------------------
VALUE                     TYPE          LINE NUMBER
------------------------------------------------------------------
"t4s"                   STRING                   15
"%d"                    STRING                   19
0                      INTEGER                   14
0                      INTEGER                   21
0                      INTEGER                   31
1                      INTEGER                   14
1                      INTEGER                   27
1                      INTEGER                   29
2                      INTEGER                   21
2                      INTEGER                   23
3                      INTEGER                   14
9                      INTEGER                   14
"Enter a number to search : "    STRING                   18
```

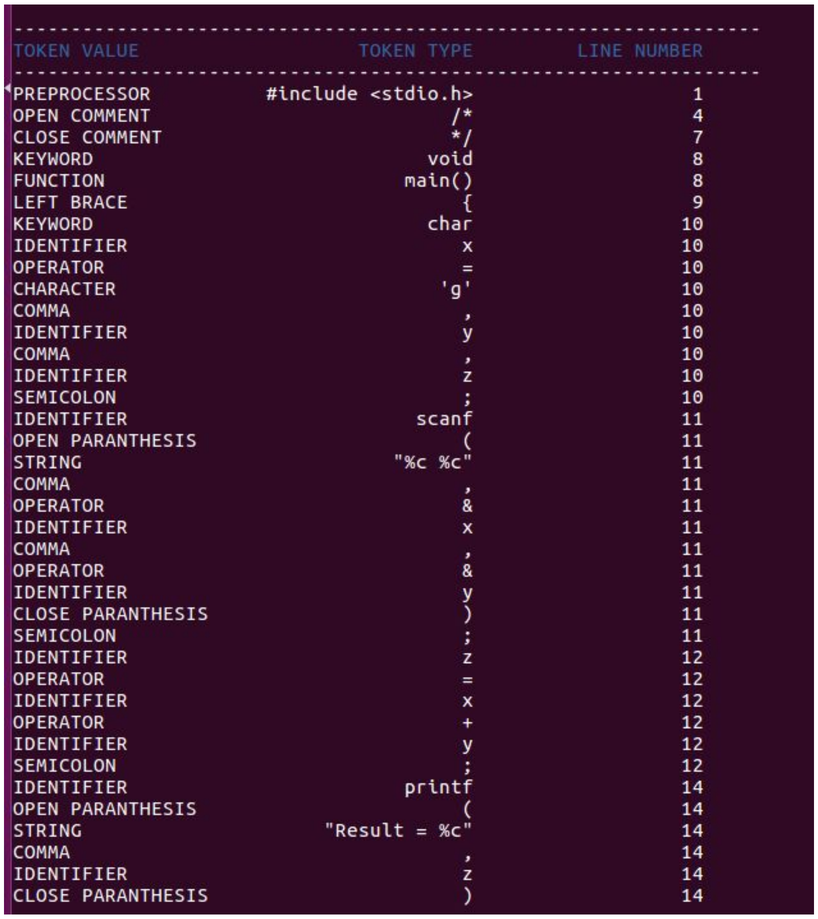
Test Case 5 - Code

```
// T5
/*
Errors include :
    - Unbalanced quotes
*/
void main()
{
    char x = 'g', y, z;
    scanf("%c %c",&x, &y);
    z = x + y;

    printf("Result = %c", z);
}
```

Test Case 5 - Screenshot



```
SYMBOL TABLE
-----------------------------------------------------------------------
VALUE                       TYPE          LINE NUMBER
-----------------------------------------------------------------------
printf                IDENTIFIER                   14
x                     IDENTIFIER                   10
y                     IDENTIFIER                   10
z                     IDENTIFIER                   10
scanf                 IDENTIFIER                   11


CONSTANT TABLE
-----------------------------------------------------------------------
VALUE                       TYPE          LINE NUMBER
-----------------------------------------------------------------------
"%c %c"                   STRING                   11
"Result = %c"             STRING                   14
'g'                    CHARACTER                   10
```

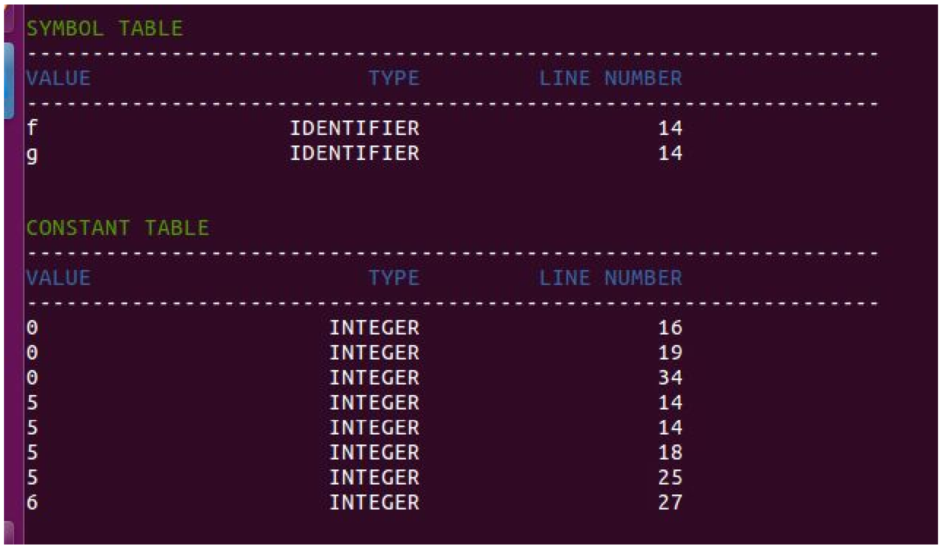


```
-----------------------------------------------------------------
TOKEN VALUE                     TOKEN TYPE          LINE NUMBER
-----------------------------------------------------------------
PREPROCESSOR          #include <stdio.h>                      1
OPEN COMMENT                          /*                      4
CLOSE COMMENT                         */                      7
KEYWORD                             void                      8
FUNCTION                          main()                      8
LEFT BRACE                             {                      9
KEYWORD                             char                     10
IDENTIFIER                             x                     10
OPERATOR                               =                     10
CHARACTER                            'g'                     10
COMMA                                  ,                     10
IDENTIFIER                             y                     10
COMMA                                  ,                     10
IDENTIFIER                             z                     10
SEMICOLON                              ;                     10
IDENTIFIER                         scanf                     11
OPEN PARANTHESIS                       (                     11
STRING                           "%c %c"                     11
COMMA                                  ,                     11
OPERATOR                               &                     11
IDENTIFIER                             x                     11
COMMA                                  ,                     11
OPERATOR                               &                     11
IDENTIFIER                             y                     11
CLOSE PARANTHESIS                      )                     11
SEMICOLON                              ;                     11
IDENTIFIER                             z                     12
OPERATOR                               =                     12
IDENTIFIER                             x                     12
OPERATOR                               +                     12
IDENTIFIER                             y                     12
SEMICOLON                              ;                     12
IDENTIFIER                        printf                     14
OPEN PARANTHESIS                       (                     14
STRING                     "Result = %c"                     14
COMMA                                  ,                     14
IDENTIFIER                             z                     14
CLOSE PARANTHESIS                      )                     14
```

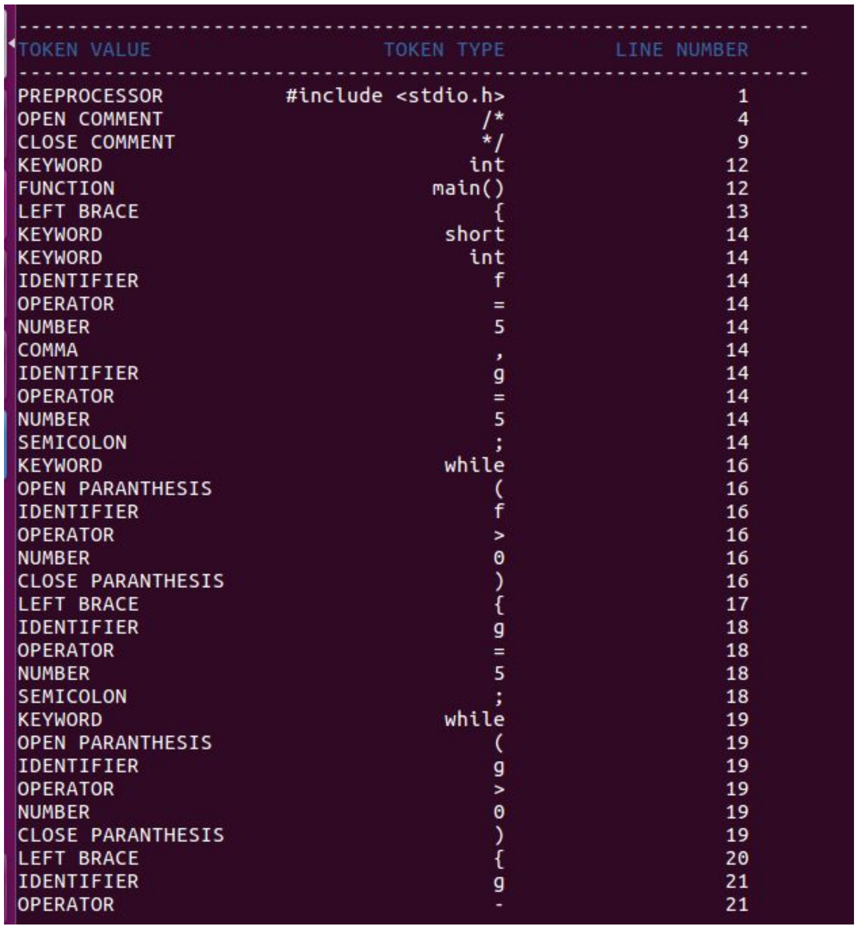
Test Case 6 - Code

```
#include <stdio.h>

// T6
/*
Support extended for :
    - Nested while loops
    - If conditional statements
    - Nested conditional statemets
*/


int main()
{
    short int f = 5, g = 5;

    while(f>0)
    {
        g = 5;
        while(g > 0)
        {
            g--;
        }
        f--;
    }
    if(f==5){
        g++;
        if(g==6){
            f++;
        }
        else{
            g++;
        }
    }
    return 0;
}
```

Test Case 6 - Screenshot



```
SYMBOL TABLE
-----------------------------------------------------------------------
VALUE                        TYPE          LINE NUMBER
-----------------------------------------------------------------------
f                    IDENTIFIER                         14
g                    IDENTIFIER                         14

CONSTANT TABLE
-----------------------------------------------------------------------
VALUE                        TYPE          LINE NUMBER
-----------------------------------------------------------------------
0                       INTEGER                         16
0                       INTEGER                         19
0                       INTEGER                         34
5                       INTEGER                         14
5                       INTEGER                         14
5                       INTEGER                         18
5                       INTEGER                         25
6                       INTEGER                         27
```



```
-----------------------------------------------------------------
TOKEN VALUE                    TOKEN TYPE          LINE NUMBER
-----------------------------------------------------------------
PREPROCESSOR          #include <stdio.h>                    1
OPEN COMMENT                          /*                    4
CLOSE COMMENT                         */                    9
KEYWORD                              int                   12
FUNCTION                          main()                   12
LEFT BRACE                             {                   13
KEYWORD                            short                   14
KEYWORD                              int                   14
IDENTIFIER                             f                   14
OPERATOR                               =                   14
NUMBER                                 5                   14
COMMA                                  ,                   14
IDENTIFIER                             g                   14
OPERATOR                               =                   14
NUMBER                                 5                   14
SEMICOLON                              ;                   14
KEYWORD                            while                   16
OPEN PARANTHESIS                       (                   16
IDENTIFIER                             f                   16
OPERATOR                               >                   16
NUMBER                                 0                   16
CLOSE PARANTHESIS                      )                   16
LEFT BRACE                             {                   17
IDENTIFIER                             g                   18
OPERATOR                               =                   18
NUMBER                                 5                   18
SEMICOLON                              ;                   18
KEYWORD                            while                   19
OPEN PARANTHESIS                       (                   19
IDENTIFIER                             g                   19
OPERATOR                               >                   19
NUMBER                                 0                   19
CLOSE PARANTHESIS                      )                   19
LEFT BRACE                             {                   20
IDENTIFIER                             g                   21
OPERATOR                               -                   21
```