Compiler Design Lab(CO351)

Project Report

<u>Project 1: Scanner for C-language</u>

<u>Team details:</u>
Siddharth V, 15CO150
Akash Rao, 15CO202

Introduction

A **compiler** is computer software that transforms computer code written in one programming language (the source language) into another programming language (the target language). The name *compiler* is primarily used for programs that translate source code from a high-level programming language to a lower level language (e.g., assembly language, object code, or machine code) to create an executable program.

There are many different types of compilers.

- If the compiled program can run on a computer whose CPU or operating system is different from the one on which the compiler runs, the compiler is a *cross-compiler.*

- A *bootstrap* compiler is written in the language that it intends to compile.

A compiler is likely to perform many or all of the following operations:

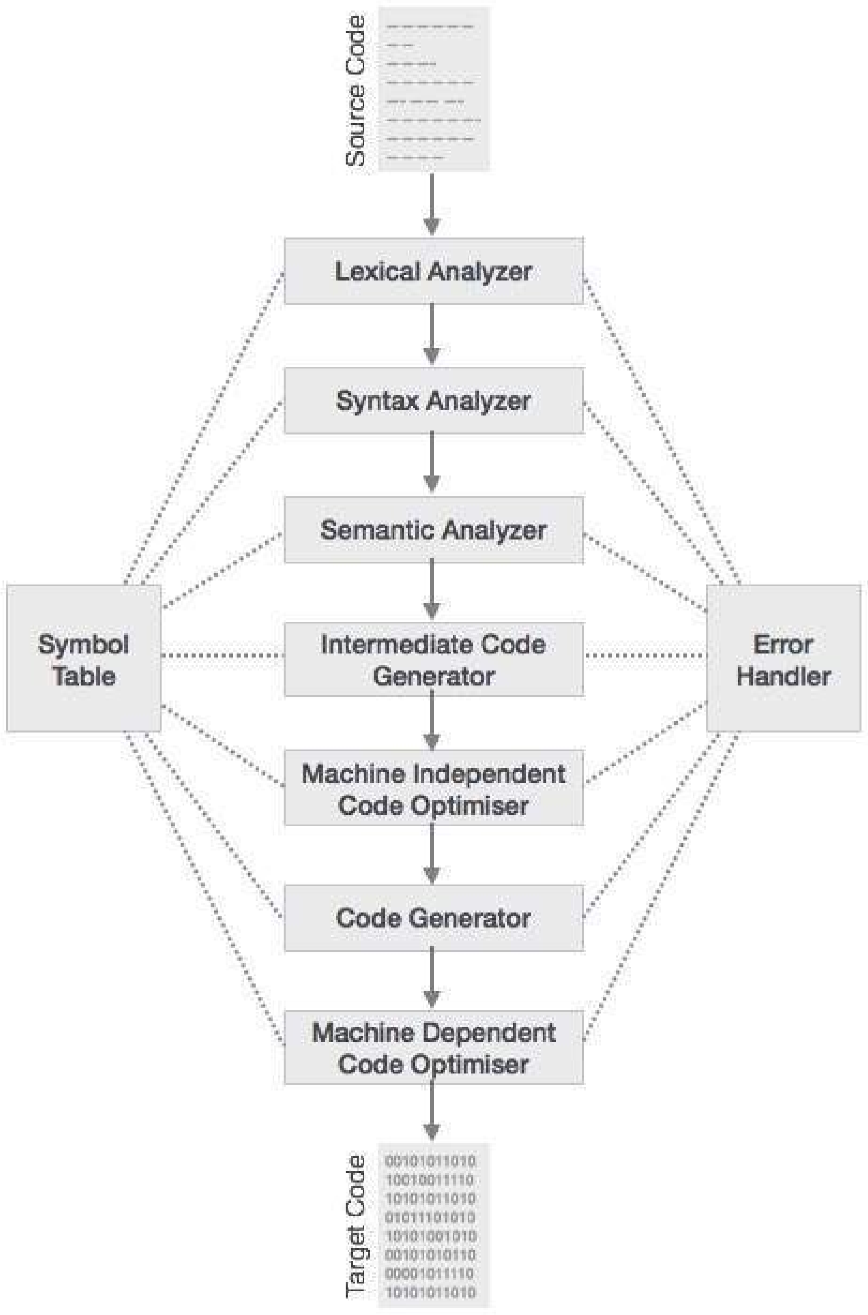
- Preprocessing
- Lexical analysis
- Parsing
- Semantic analysis (syntax-directed translation)
- Conversion of input programs to an intermediate representation
- Code optimization
- Code generation

Compilers implement these operations in phases that promote efficient design and correct transformations of source input to target output.

Compilers are not the only translators used to transform source programs. An interpreter is computer software that transforms and then executes the indicated operations. The translation process influences the design of computer languages which leads to a preference of compilation or interpretation. In practice, an interpreter can be implemented for compiled languages and compilers can be implemented for interpreted languages.

Compiler Design - Phases of Compiler

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.



The compiler has two modules namely front end and back end. Front-end constitutes of the Lexical analyzer, semantic analyzer, syntax analyzer and intermediate code generator. And the rest are assembled to form the back end.

- **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**

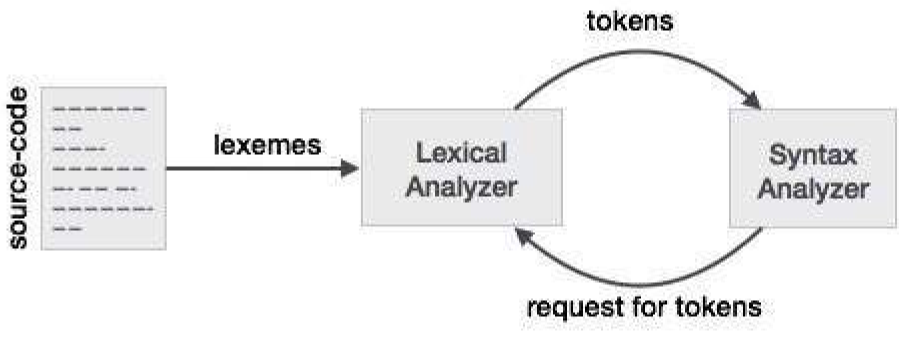
In this phase, the code generator takes the optimized representation of the intermediate code and maps it to the target machine language. The code generator translates the intermediate code into a sequence of (generally) relocatable machine code. Sequence of instructions of machine code performs the 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.

Compiler Design - Lexical Analysis

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.



Tokens

Lexemes are said to be a sequence of characters (alphanumeric) in a token. There are some predefined rules for every lexeme to be identified as a valid token. These rules are defined by grammar rules, by means of a pattern. A pattern explains what can be a token, and these patterns are defined by means of regular expressions.

In programming language the following can be considered as tokens

- **Identifiers:** names the programmer chooses;
- **Keywords:** names already in the programming language;
- **Separators(punctuators):** punctuation characters and paired-delimiters;
- **Operators:** symbols that operate on arguments and produce results;
- **Literals:** numeric, logical, textual, reference literals;
- **Comments:** line, block.

For example, in C language, the variable declaration line
int value = 100;
contains the tokens:
int (keyword), value (identifier), = (operator), 100 (constant) and ; (symbol).

Language

A language is considered as a finite set of strings over some finite set of alphabets. Computer languages are considered as finite sets, and mathematically set operations can be performed on them. Finite languages can be described by means of regular expressions.

Scanner

The first stage, the *scanner*, is usually based on a finite-state machine (FSM). It has encoded within it information on the possible sequences of characters that can be contained within any of the tokens it handles (individual instances of these character sequences are termed lexemes).

Evaluator

A lexeme, however, is only a string of characters known to be of a certain kind (e.g., a string literal, a sequence of letters). In order to construct a token, the lexical analyzer needs a second stage, the *evaluator*, which goes over the characters of the lexeme to produce a *value*. The lexeme's type combined with its value is what properly constitutes a token, which can be given to a parser.

Obstacles

Typically, tokenization occurs at the word level. However, it is sometimes difficult to define what is meant by a "word". Often a tokenizer relies on simple heuristics, for example:

- Punctuation and whitespace may or may not be included in the resulting list of tokens.
- All contiguous strings of alphabetic characters are part of one token; likewise with numbers.
- Tokens are separated by whitespace characters, such as a space or line break, or by punctuation characters.

Source Code(scanner.l)

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

    #define SYM_TBL 0
    #define CONST_TBL 1

    typedef struct node
    {
         char* name;
         int type;

         struct node* next;
    }node;

    node* sym_tbl[100];
    node* const_tbl[100];

    int hash(char* x, int M)
    {
        int i, sum;
        for (sum=0, i=0; i < strlen(x); i++)
        sum += x[i];
        return sum % M;
         }

         int lookup(char* x, int table)
         {
                 int idx = hash(x, 100);

                 node* t = NULL;

                 if(table==0)
                 {
         if(sym_tbl[idx]==NULL)
                 return 0;

         t = sym_tbl[idx];
         }
         else
         {
         if(const_tbl[idx]==NULL)
                 return 0;

         t = const_tbl[idx];
         }

                 while(t!=NULL)
                 {
                         if(strcmp(t->name, x)==0)
                                 return 1;
```

```
                        t = t->next;
                }

                return 0;

        }


        void insert(char* x, int type, int table)
        {
                if(lookup(x, table))
                        return;

                int idx = hash(x, 100);

                node* cell = (node*)malloc(sizeof(node));
                cell->name = (char*)malloc(strlen(x));
                strcpy(cell->name, x);
                cell->type = type;
                cell->next = NULL;

                node* t = NULL;

                if(table==0)
                {
        if(sym_tbl[idx]==NULL)
        {
                sym_tbl[idx] = cell;
                return;
        }

        t = sym_tbl[idx];
        }
        else
        {
        if(const_tbl[idx]==NULL)
        {
                const_tbl[idx] = cell;
                return;
        }

        t = const_tbl[idx];
        }

                while(t->next!=NULL)
                t = t->next;

                t->next = cell;
        }

        void display()
        {
                                                        printf("\n----------------------------\n\tSymbol
table\n----------------------------\n");
                printf("Value\t\t-\tType\n----------------------------\n");
                int i;

                for(i=0; i<100; i++)
                {
                        if(sym_tbl[i]==NULL)
                                continue;

                        node* t = sym_tbl[i];

                        while(t!=NULL)
                        {
                                printf("%s\t\t-\t%d\n", t->name, t->type);
                                t = t->next;
                        }

                }


                                                  printf("\n\n----------------------------\n\tConstant
table\n----------------------------\n");
                printf("Value\t\t-\tType\n----------------------------\n");

                for(i=0; i<100; i++)
                {
                        if(const_tbl[i]==NULL)
                                continue;

                        node* t = const_tbl[i];

                        while(t!=NULL)
                        {
                                printf("%s\t\t-\t%d\n", t->name, t->type);
                                t = t->next;
                        }
                }
        }
%}

%%

"//".*\n                {printf("Single line comment\n");}
"/*"([^*]|\*+[^*/])*"*/"  {printf("Multi line comment\n");}

\"(\\.|[^"\n\\])*\"                             {printf("String        const:\t\t%s\n",        yytext);
insert(yytext, 1, CONST_TBL); }
0[xX][0-9a-fA-F]+                               {printf("Int  const:\t\t%s\n",  yytext);  insert(yytext,
2, CONST_TBL); }
[0-9]+                                          {printf("Int  const:\t\t%s\n",  yytext);  insert(yytext,
2, CONST_TBL); }
(([0-9]+)|([0-9]*\.[0-9]+)([eE][-+]?[0-9]+)?)    {printf("Float        const:\t\t%s\n",        yytext);
insert(yytext, 3, CONST_TBL); }

'([^'\\\n]|\\.)'                                {printf("Char  const:\t\t%s\n",  yytext);  insert(yytext,
4, CONST_TBL); }
'                                               {printf("\n      Error:      Unterminated      Character
constant\n\n"); }
```

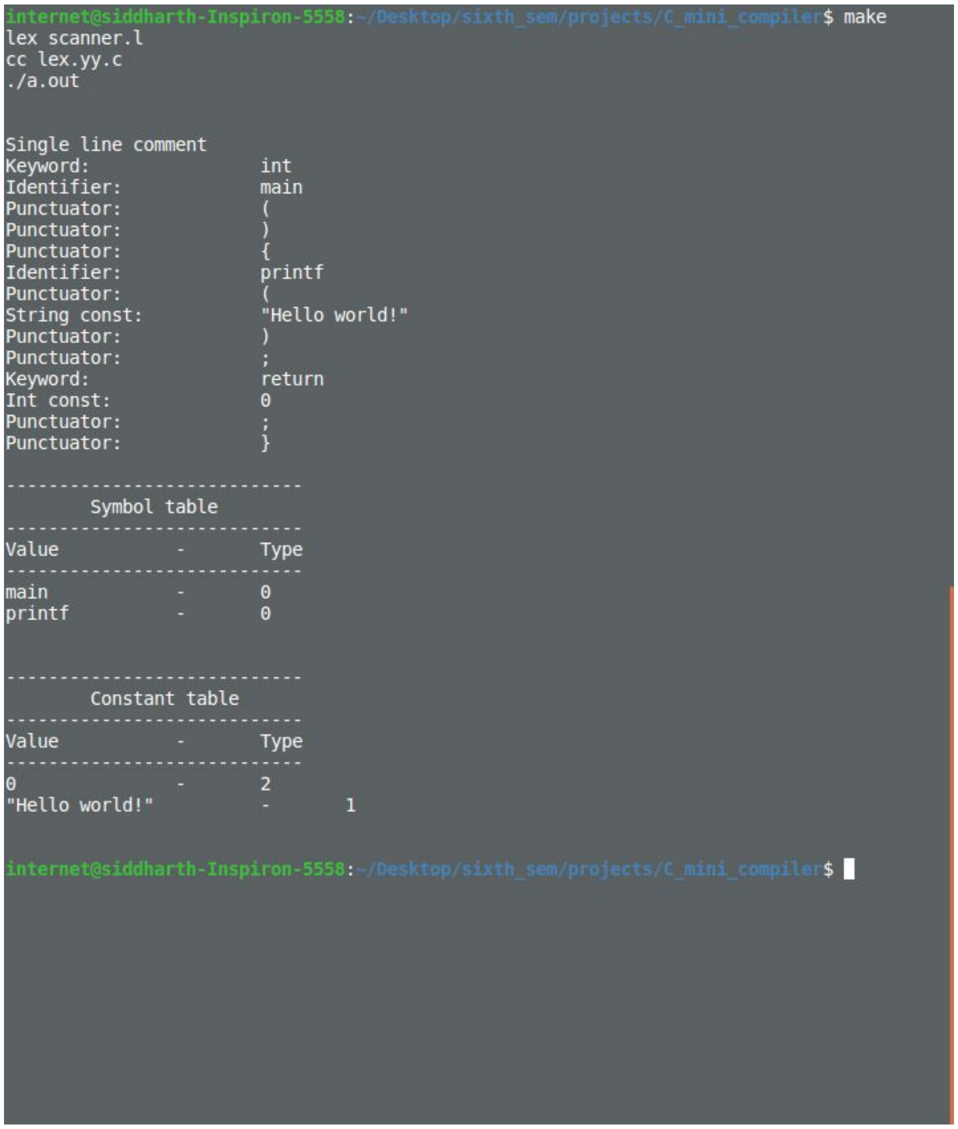
```
''                                                 {printf("Char const:\t\t%s\n", yytext); insert(yytext,
1, CONST_TBL); }
'([^'\\\n]|\\.)+'                                  {printf("\n Error: Character constant too long\n\n");
}

"auto"                    { printf("Keyword:\t\t%s\n", yytext); }
"break"                   { printf("Keyword:\t\t%s\n", yytext); }
"case"                    { printf("Keyword:\t\t%s\n", yytext); }
"char"                    { printf("Keyword:\t\t%s\n", yytext); }
"const"                   { printf("Keyword:\t\t%s\n", yytext); }
"continue"                { printf("Keyword:\t\t%s\n", yytext); }
"default"                 { printf("Keyword:\t\t%s\n", yytext); }
"do"                      { printf("Keyword:\t\t%s\n", yytext); }
"double"                  { printf("Keyword:\t\t%s\n", yytext); }
"else"                    { printf("Keyword:\t\t%s\n", yytext); }
"num"                     { printf("Keyword:\t\t%s\n", yytext); }
"extern"                  { printf("Keyword:\t\t%s\n", yytext); }
"float"                   { printf("Keyword:\t\t%s\n", yytext); }
"for"                     { printf("Keyword:\t\t%s\n", yytext); }
"goto"                    { printf("Keyword:\t\t%s\n", yytext); }
"if"                      { printf("Keyword:\t\t%s\n", yytext); }
"int"                     { printf("Keyword:\t\t%s\n", yytext); }
"long"                    { printf("Keyword:\t\t%s\n", yytext); }
"register"                { printf("Keyword:\t\t%s\n", yytext); }
"return"                  { printf("Keyword:\t\t%s\n", yytext); }
"short"                   { printf("Keyword:\t\t%s\n", yytext); }
"signed"                  { printf("Keyword:\t\t%s\n", yytext); }
"sizeof"                  { printf("Keyword:\t\t%s\n", yytext); }
"static"                  { printf("Keyword:\t\t%s\n", yytext); }
"struct"                  { printf("Keyword:\t\t%s\n", yytext); }
"switch"                  { printf("Keyword:\t\t%s\n", yytext); }
"typedef"                 { printf("Keyword:\t\t%s\n", yytext); }
"union"                   { printf("Keyword:\t\t%s\n", yytext); }
"unsigned"                { printf("Keyword:\t\t%s\n", yytext); }
"void"                    { printf("Keyword:\t\t%s\n", yytext); }
"volatile"                { printf("Keyword:\t\t%s\n", yytext); }
"while"                   { printf("Keyword:\t\t%s\n", yytext); }


[a-zA-Z_][a-zA-Z0-9_]*    {printf("Identifier:\t\t%s\n", yytext);  insert(yytext, 0, SYM_TBL); }

"*/"                      { printf("\n Error: Unexpected end of comment\n\n"); }
"/*"                      { printf("\n Error: Unterminated Multi line comment\n\n"); }

"--"                      { printf("Operator:\t\t%s\n", yytext); }
"&&"                      { printf("Operator:\t\t%s\n", yytext); }
"||"                      { printf("Operator:\t\t%s\n", yytext); }
"=="                      { printf("Operator:\t\t%s\n", yytext); }
">="                      { printf("Operator:\t\t%s\n", yytext); }
"<="                      { printf("Operator:\t\t%s\n", yytext); }
"!="                      { printf("Operator:\t\t%s\n", yytext); }

"!"                       { printf("Punctuator:\t\t%s\n", yytext); }
"%"                       { printf("Punctuator:\t\t%s\n", yytext); }
"^"                       { printf("Punctuator:\t\t%s\n", yytext); }
"&"                       { printf("Punctuator:\t\t%s\n", yytext); }
```

```
"*"                         { printf("Punctuator:\t\t%s\n", yytext); }
"("                         { printf("Punctuator:\t\t%s\n", yytext); }
")"                         { printf("Punctuator:\t\t%s\n", yytext); }
"-"                         { printf("Punctuator:\t\t%s\n", yytext); }
"+"                         { printf("Punctuator:\t\t%s\n", yytext); }
"="                         { printf("Punctuator:\t\t%s\n", yytext); }
"{"                         { printf("Punctuator:\t\t%s\n", yytext); }
"}"                         { printf("Punctuator:\t\t%s\n", yytext); }
"|"                         { printf("Punctuator:\t\t%s\n", yytext); }
"~"                         { printf("Punctuator:\t\t%s\n", yytext); }
"["                         { printf("Punctuator:\t\t%s\n", yytext); }
"]"                         { printf("Punctuator:\t\t%s\n", yytext); }
";"                         { printf("Punctuator:\t\t%s\n", yytext); }

":"                         { printf("Punctuator:\t\t%s\n", yytext); }
"<"                         { printf("Punctuator:\t\t%s\n", yytext); }
">"                         { printf("Punctuator:\t\t%s\n", yytext); }
"?"                         { printf("Punctuator:\t\t%s\n", yytext); }
","                         { printf("Punctuator:\t\t%s\n", yytext); }
"."                         { printf("Punctuator:\t\t%s\n", yytext); }
"/"                         { printf("Punctuator:\t\t%s\n", yytext); }
"#"                         { printf("Punctuator:\t\t%s\n", yytext); }

"\n"                        { /*printf("\n");*/ }
" "                         { /*printf(" ");*/ }
"\t"                        { /*printf("\t");*/ }

"\""                        {printf("\n Error: Unmatched quotation\n\n");}
.                           { printf("\n Error: Invalid token \n\n"); }

%%

int yywrap()
{
    return 1;
}

int main()
{

    printf("\n\n");

    yyin = fopen("test_cases/6.c", "r");

    int token;
    while(yylex());


    display();

    printf("\n\n");

    return 0;
}
```

Test Cases

Test case 1

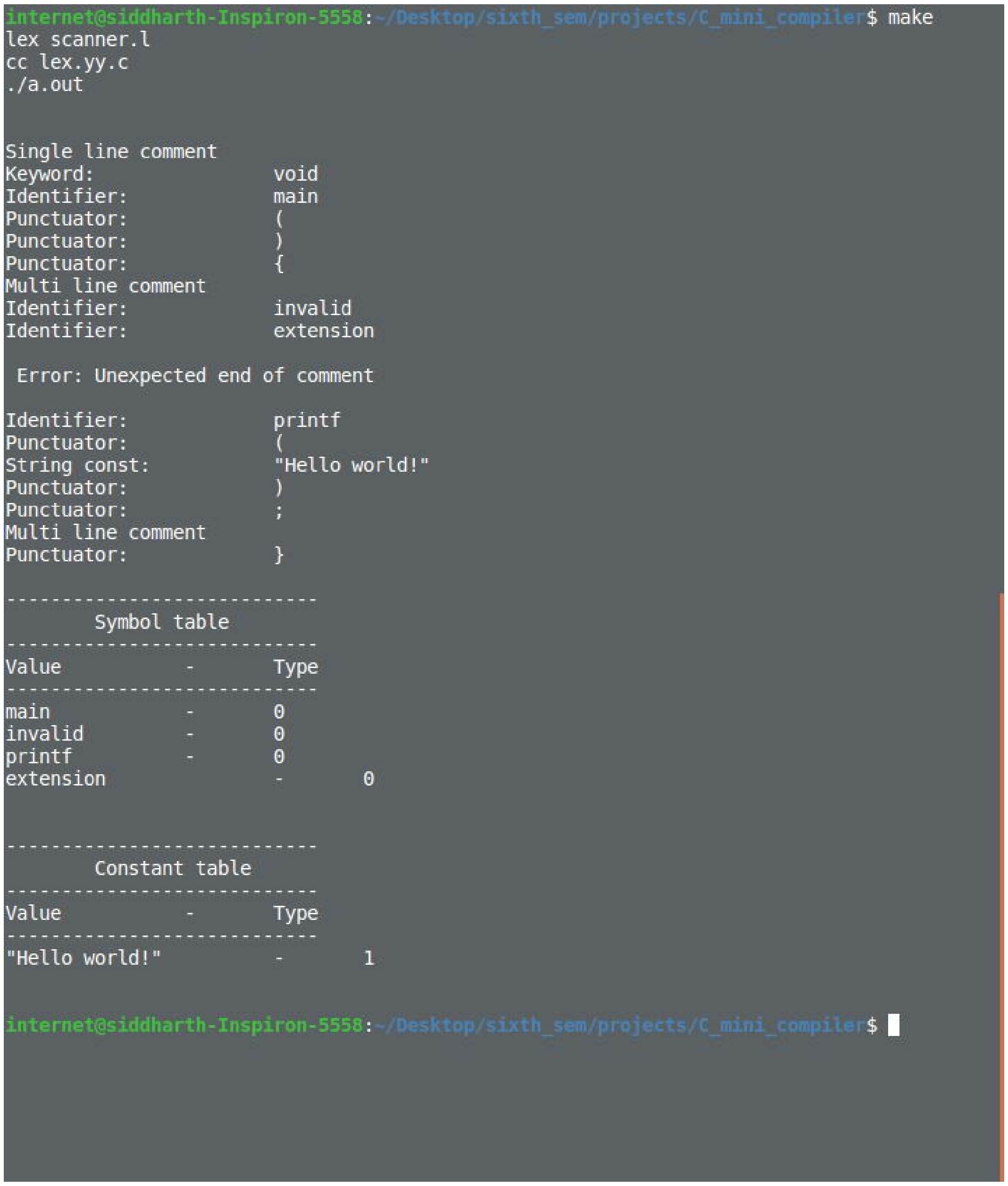
```c
//Hello world program

int main()
{
    printf("Hello world!");

    return 0;
}
```



```
internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/C_mini_compiler$ make
lex scanner.l
cc lex.yy.c
./a.out


Single line comment
Keyword:                int
Identifier:             main
Punctuator:             (
Punctuator:             )
Punctuator:             {
Identifier:             printf
Punctuator:             (
String const:           "Hello world!"
Punctuator:             )
Punctuator:             ;
Keyword:                return
Int const:              0
Punctuator:             ;
Punctuator:             }

----------------------------
        Symbol table
----------------------------
Value           -       Type
----------------------------
main            -       0
printf          -       0


----------------------------
        Constant table
----------------------------
Value           -       Type
----------------------------
0               -       2
"Hello world!"          -       1


internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/C_mini_compiler$ 
```

Test case 2

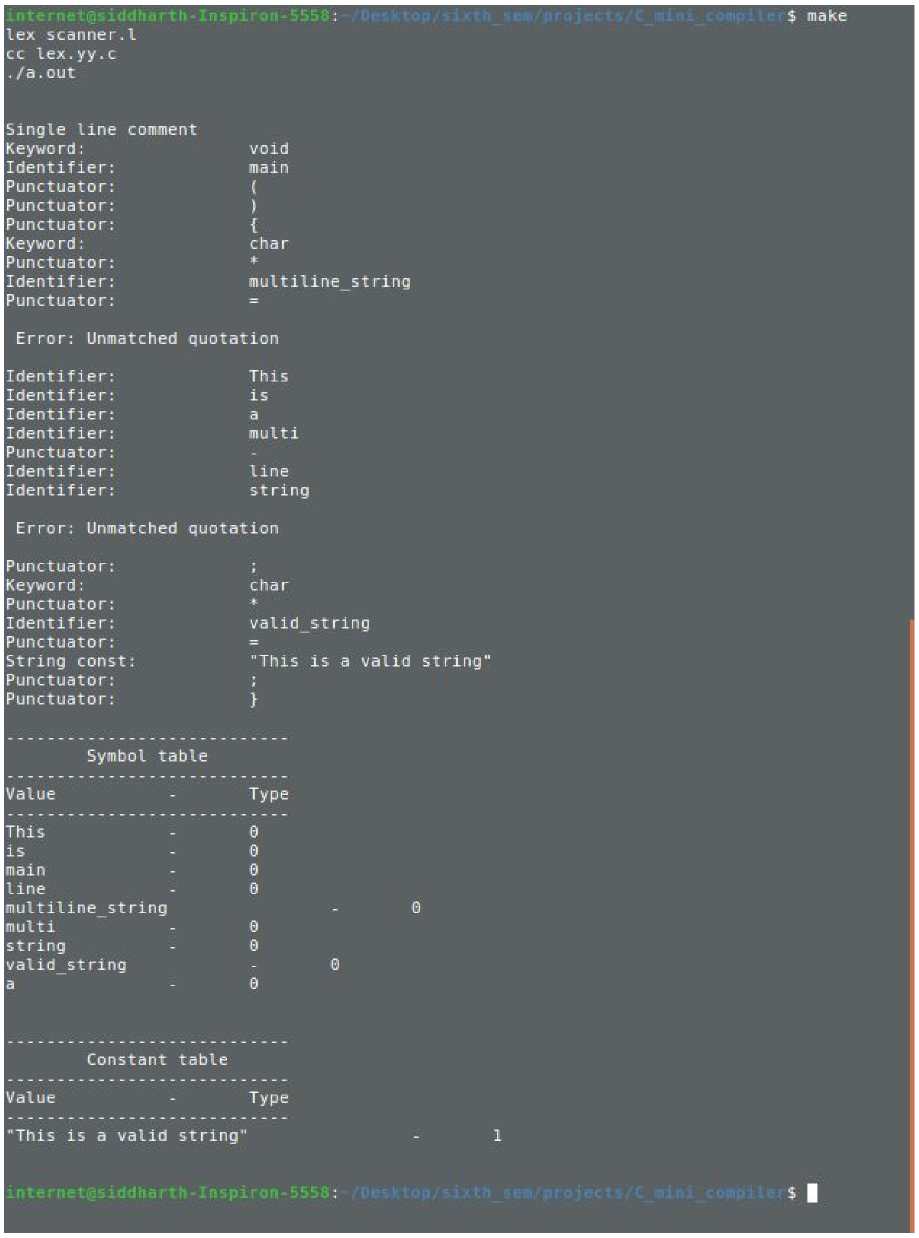
```
//Nested comment
void main()
{
    /* The /* comment ends here */ invalid extension */

    printf("Hello world!");

    /* This is an /* valid comment */
}
```



```
internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/C_mini_compiler$ make
lex scanner.l
cc lex.yy.c
./a.out


Single line comment
Keyword:                void
Identifier:             main
Punctuator:             (
Punctuator:             )
Punctuator:             {
Multi line comment
Identifier:             invalid
Identifier:             extension

 Error: Unexpected end of comment

Identifier:             printf
Punctuator:             (
String const:           "Hello world!"
Punctuator:             )
Punctuator:             ;
Multi line comment
Punctuator:             }

----------------------------
        Symbol table
----------------------------
Value           -       Type
----------------------------
main            -       0
invalid         -       0
printf          -       0
extension               -       0


----------------------------
        Constant table
----------------------------
Value           -       Type
----------------------------
"Hello world!"          -       1


internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/C_mini_compiler$ 
```

Test case 3

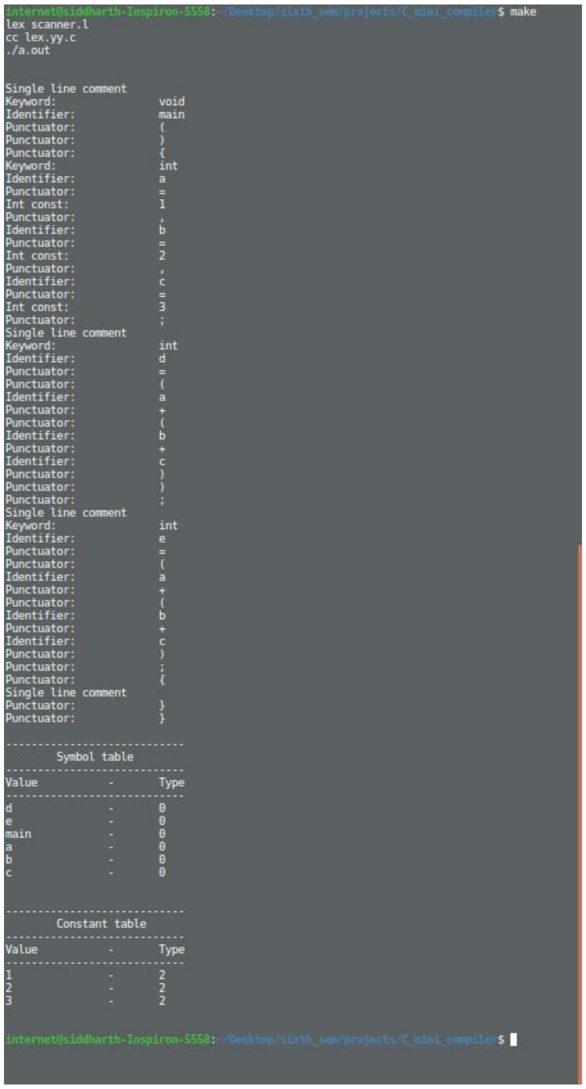
```
//Multi-line string
void main()
{
    char *multiline_string = "This is a
                              multi-line string";

    char *valid_string = "This is a valid string";

}
```



```
internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/C_mini_compiler$ make
lex scanner.l
cc lex.yy.c
./a.out


Single line comment
Keyword:                void
Identifier:             main
Punctuator:             (
Punctuator:             )
Punctuator:             {
Keyword:                char
Punctuator:             *
Identifier:             multiline_string
Punctuator:             =

 Error: Unmatched quotation

Identifier:             This
Identifier:             is
Identifier:             a
Identifier:             multi
Punctuator:             -
Identifier:             line
Identifier:             string

 Error: Unmatched quotation

Punctuator:             ;
Keyword:                char
Punctuator:             *
Identifier:             valid_string
Punctuator:             =
String const:           "This is a valid string"
Punctuator:             ;
Punctuator:             }

----------------------------
        Symbol table
----------------------------
Value           -       Type
----------------------------
This            -       0
is              -       0
main            -       0
line            -       0
multiline_string                -       0
multi           -       0
string          -       0
valid_string            -       0
a               -       0


----------------------------
        Constant table
----------------------------
Value           -       Type
----------------------------
"This is a valid string"                -       1


internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/C_mini_compiler$ 
```

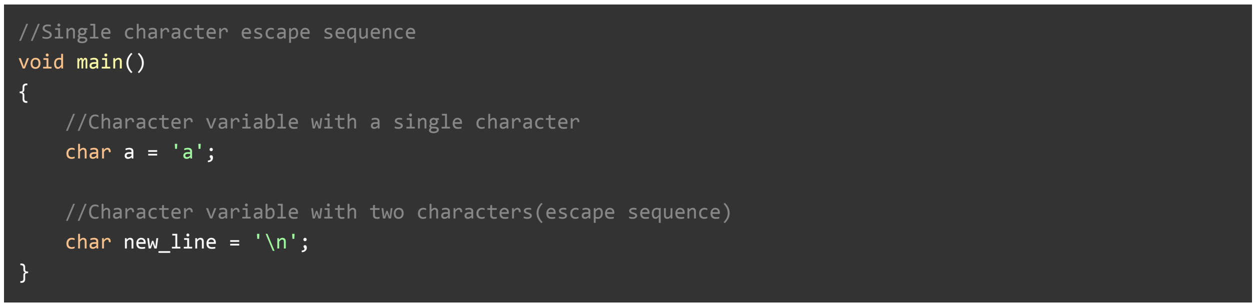
Test case 4

```
//Unbalanced paranthesis and curly braces
void main()
{
    int a = 1, b = 2, c = 3;

    //Unbalaced paranthesis
    int d = (a + (b+c));

    //Balanced paranthesis
    int e = (a + (b+c);
    {
        //Balanced block
    }
}
```

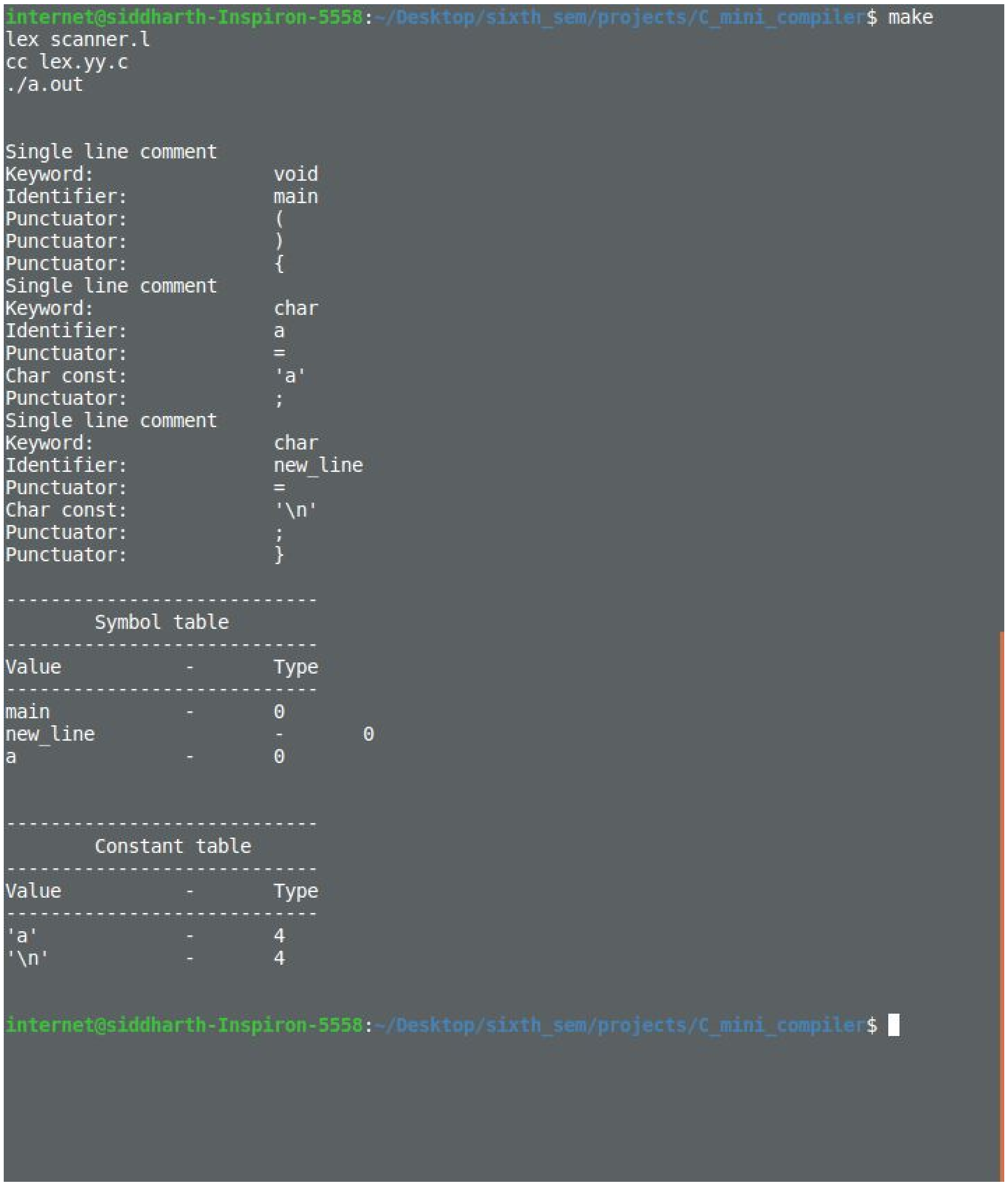


```
internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/C_mini_compiler$ make
lex scanner.l
cc lex.yy.c
./a.out


Single line comment
Keyword:                void
Identifier:             main
Punctuator:             (
Punctuator:             )
Punctuator:             {
Keyword:                int
Identifier:             a
Punctuator:             =
Int const:              1
Punctuator:             ,
Identifier:             b
Punctuator:             =
Int const:              2
Punctuator:             ,
Identifier:             c
Punctuator:             =
Int const:              3
Punctuator:             ;
Single line comment
Keyword:                int
Identifier:             d
Punctuator:             =
Punctuator:             (
Identifier:             a
Punctuator:             +
Punctuator:             (
Identifier:             b
Punctuator:             +
Identifier:             c
Punctuator:             )
Punctuator:             )
Punctuator:             ;
Single line comment
Keyword:                int
Identifier:             e
Punctuator:             =
Punctuator:             (
Identifier:             a
Punctuator:             +
Punctuator:             (
Identifier:             b
Punctuator:             +
Identifier:             c
Punctuator:             )
Punctuator:             ;
Punctuator:             {
Single line comment
Punctuator:             }
Punctuator:             }

----------------------------
        Symbol table
----------------------------
Value           -       Type
----------------------------
d               -       0
e               -       0
main            -       0
a               -       0
b               -       0
c               -       0


----------------------------
        Constant table
----------------------------
Value           -       Type
----------------------------
1               -       2
2               -       2
3               -       2


internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/C_mini_compiler$ 
```

Test case 5



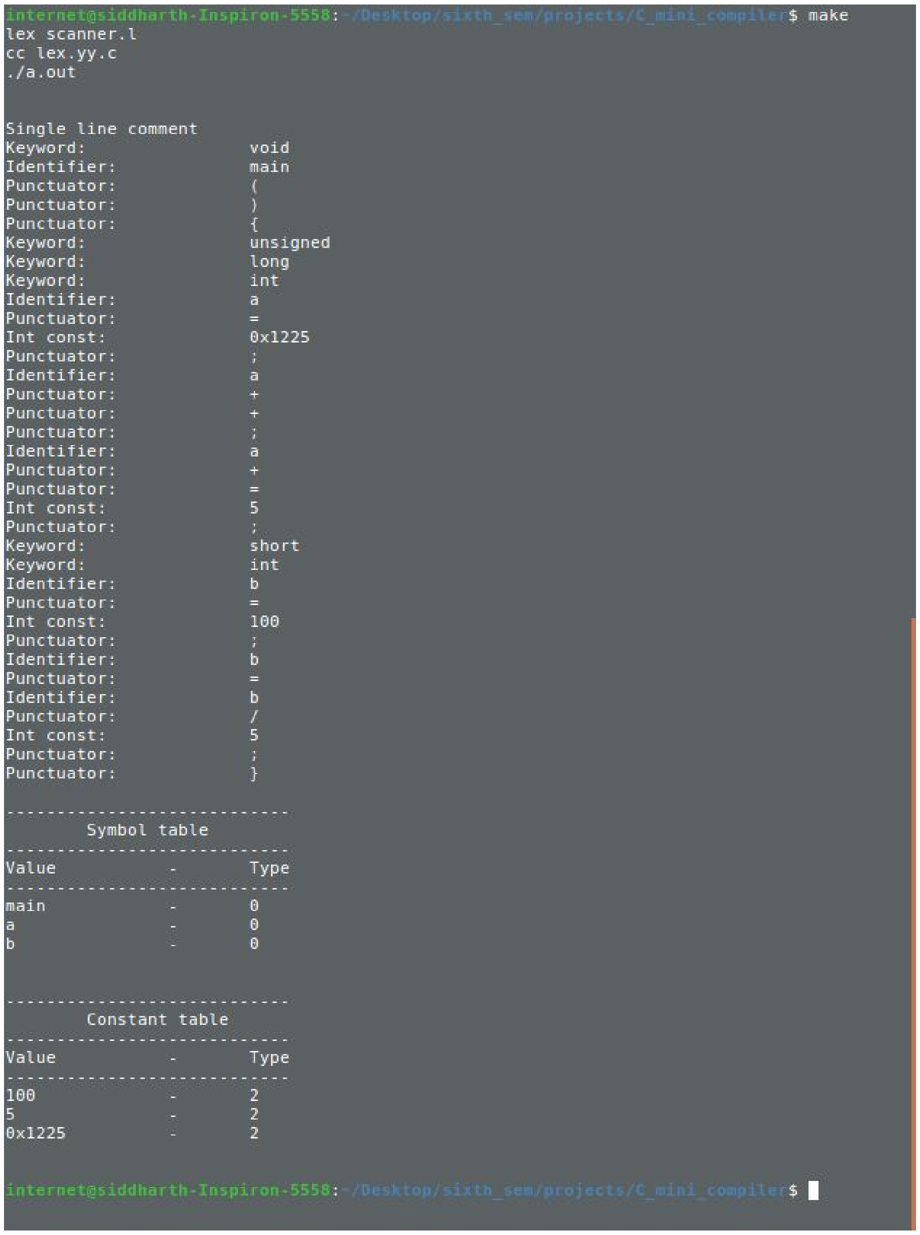
```
//Single character escape sequence
void main()
{
    //Character variable with a single character
    char a = 'a';

    //Character variable with two characters(escape sequence)
    char new_line = '\n';
}
```



```
internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/C_mini_compiler$ make
lex scanner.l
cc lex.yy.c
./a.out


Single line comment
Keyword:                void
Identifier:             main
Punctuator:             (
Punctuator:             )
Punctuator:             {
Single line comment
Keyword:                char
Identifier:             a
Punctuator:             =
Char const:             'a'
Punctuator:             ;
Single line comment
Keyword:                char
Identifier:             new_line
Punctuator:             =
Char const:             '\n'
Punctuator:             ;
Punctuator:             }

----------------------------
        Symbol table
----------------------------
Value           -       Type
----------------------------
main            -       0
new_line                -       0
a               -       0


----------------------------
        Constant table
----------------------------
Value           -       Type
----------------------------
'a'             -       4
'\n'            -       4


internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/C_mini_compiler$ 
```

Test case 6

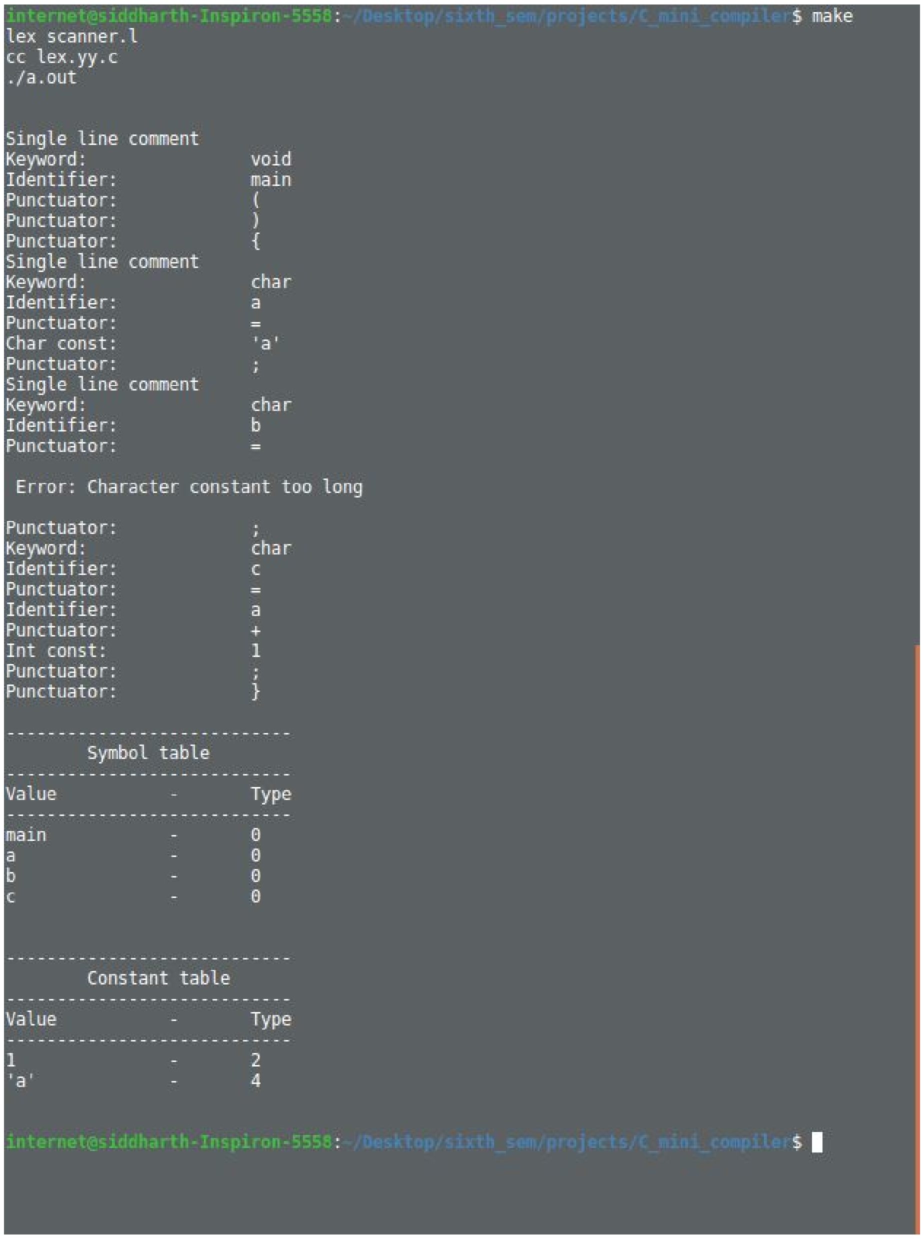
```
//Integer constants and operations
void main()
{
    unsigned long int a = 0x1225;
    a++;
    a += 5;
    short int b = 100;
    b = b/5;
}
```



```
internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/C_mini_compiler$ make
lex scanner.l
cc lex.yy.c
./a.out


Single line comment
Keyword:                void
Identifier:             main
Punctuator:             (
Punctuator:             )
Punctuator:             {
Keyword:                unsigned
Keyword:                long
Keyword:                int
Identifier:             a
Punctuator:             =
Int const:              0x1225
Punctuator:             ;
Identifier:             a
Punctuator:             +
Punctuator:             +
Punctuator:             ;
Identifier:             a
Punctuator:             +
Punctuator:             =
Int const:              5
Punctuator:             ;
Keyword:                short
Keyword:                int
Identifier:             b
Punctuator:             =
Int const:              100
Punctuator:             ;
Identifier:             b
Punctuator:             =
Identifier:             b
Punctuator:             /
Int const:              5
Punctuator:             ;
Punctuator:             }

----------------------------
        Symbol table
----------------------------
Value           -       Type
----------------------------
main            -       0
a               -       0
b               -       0


----------------------------
        Constant table
----------------------------
Value           -       Type
----------------------------
100             -       2
5               -       2
0x1225          -       2


internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/C_mini_compiler$
```

Test case 7

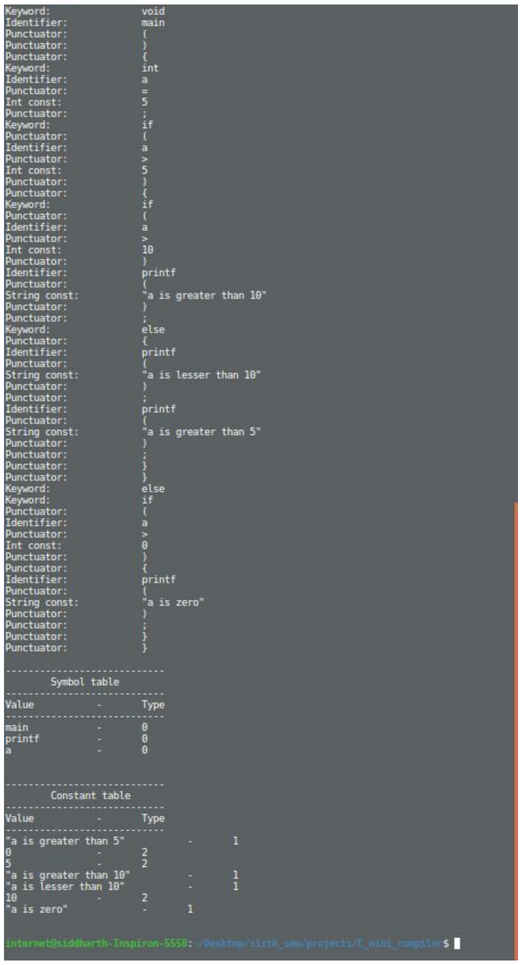
```
//Character constants and operations
void main()
{
    //valid character
    char a = 'a';
    //invalid character
    char b = 'bc';
    char c = a + 1;
}
```



```
internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/C_mini_compiler$ make
lex scanner.l
cc lex.yy.c
./a.out


Single line comment
Keyword:                void
Identifier:             main
Punctuator:             (
Punctuator:             )
Punctuator:             {
Single line comment
Keyword:                char
Identifier:             a
Punctuator:             =
Char const:             'a'
Punctuator:             ;
Single line comment
Keyword:                char
Identifier:             b
Punctuator:             =

 Error: Character constant too long

Punctuator:             ;
Keyword:                char
Identifier:             c
Punctuator:             =
Identifier:             a
Punctuator:             +
Int const:              1
Punctuator:             ;
Punctuator:             }

----------------------------
        Symbol table
----------------------------
Value           -       Type
----------------------------
main            -       0
a               -       0
b               -       0
c               -       0


----------------------------
        Constant table
----------------------------
Value           -       Type
----------------------------
1               -       2
'a'             -       4


internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/C_mini_compiler$ 
```

Test case 8

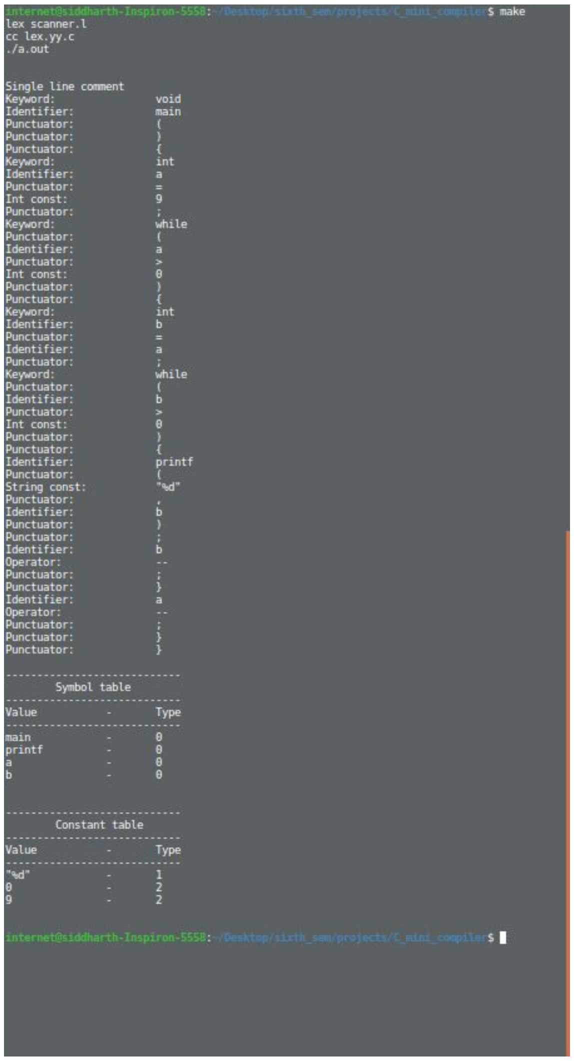
```
//Nested if-else
void main()
{
    int a = 5;
    if(a>5)
    {
         if(a>10)
                 printf("a is greater than 10");
         else
         {
                 printf("a is lesser than 10");
                 printf("a is greater than 5");
         }
    }
    else if(a>0)
    {
         printf("a is zero");
    }
}
```



```
Keyword:                void
Identifier:             main
Punctuator:             (
Punctuator:             )
Punctuator:             {
Keyword:                int
Identifier:             a
Punctuator:             =
Int const:              5
Punctuator:             ;
Keyword:                if
Punctuator:             (
Identifier:             a
Punctuator:             >
Int const:              5
Punctuator:             )
Punctuator:             {
Keyword:                if
Punctuator:             (
Identifier:             a
Punctuator:             >
Int const:              10
Punctuator:             )
Identifier:             printf
Punctuator:             (
String const:           "a is greater than 10"
Punctuator:             )
Punctuator:             ;
Keyword:                else
Punctuator:             {
Identifier:             printf
Punctuator:             (
String const:           "a is lesser than 10"
Punctuator:             )
Punctuator:             ;
Identifier:             printf
Punctuator:             (
String const:           "a is greater than 5"
Punctuator:             )
Punctuator:             ;
Punctuator:             }
Punctuator:             }
Keyword:                else
Keyword:                if
Punctuator:             (
Identifier:             a
Punctuator:             >
Int const:              0
Punctuator:             )
Punctuator:             {
Identifier:             printf
Punctuator:             (
String const:           "a is zero"
Punctuator:             )
Punctuator:             ;
Punctuator:             }
Punctuator:             }

----------------------------
        Symbol table
----------------------------
Value           -       Type
----------------------------
main            -       0
printf          -       0
a               -       0


----------------------------
        Constant table
----------------------------
Value           -       Type
----------------------------
"a is greater than 5"           -       1
0               -       2
5               -       2
"a is greater than 10"          -       1
"a is lesser than 10"           -       1
10              -       2
"a is zero"     -       1

internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/C_mini_compiler$
```

Test case 9

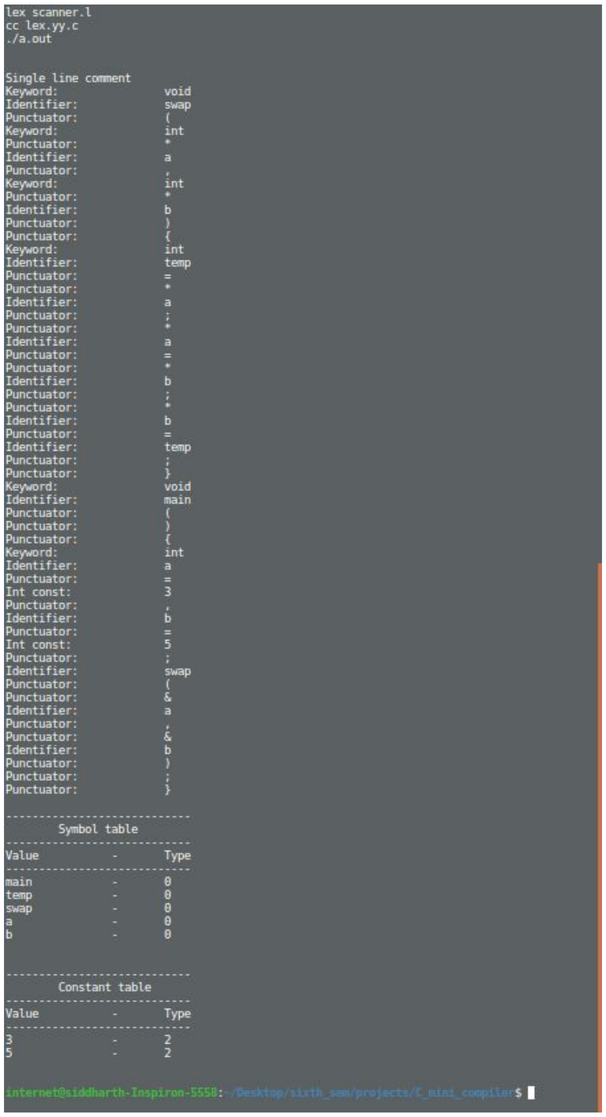
```
//Nested while

void main()
{
    int a=9;
    while(a>0)
    {
        int b = a;
        while(b>0)
        {
            printf("%d", b);
            b--;
        }
        a--;
    }
}
```



```
internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/C_mini_compiler$ make
lex scanner.l
cc lex.yy.c
./a.out


Single line comment
Keyword:                void
Identifier:             main
Punctuator:             (
Punctuator:             )
Punctuator:             {
Keyword:                int
Identifier:             a
Punctuator:             =
Int const:              9
Punctuator:             ;
Keyword:                while
Punctuator:             (
Identifier:             a
Punctuator:             >
Int const:              0
Punctuator:             )
Punctuator:             {
Keyword:                int
Identifier:             b
Punctuator:             =
Identifier:             a
Punctuator:             ;
Keyword:                while
Punctuator:             (
Identifier:             b
Punctuator:             >
Int const:              0
Punctuator:             )
Punctuator:             {
Identifier:             printf
Punctuator:             (
String const:           "%d"
Punctuator:             ,
Identifier:             b
Punctuator:             )
Punctuator:             ;
Identifier:             b
Operator:               --
Punctuator:             ;
Punctuator:             }
Identifier:             a
Operator:               --
Punctuator:             ;
Punctuator:             }
Punctuator:             }

----------------------------
        Symbol table
----------------------------
Value           -       Type
----------------------------
main            -       0
printf          -       0
a               -       0
b               -       0


----------------------------
        Constant table
----------------------------
Value           -       Type
----------------------------
"%d"            -       1
0               -       2
9               -       2


internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/C_mini_compiler$ 
```

Test case 10

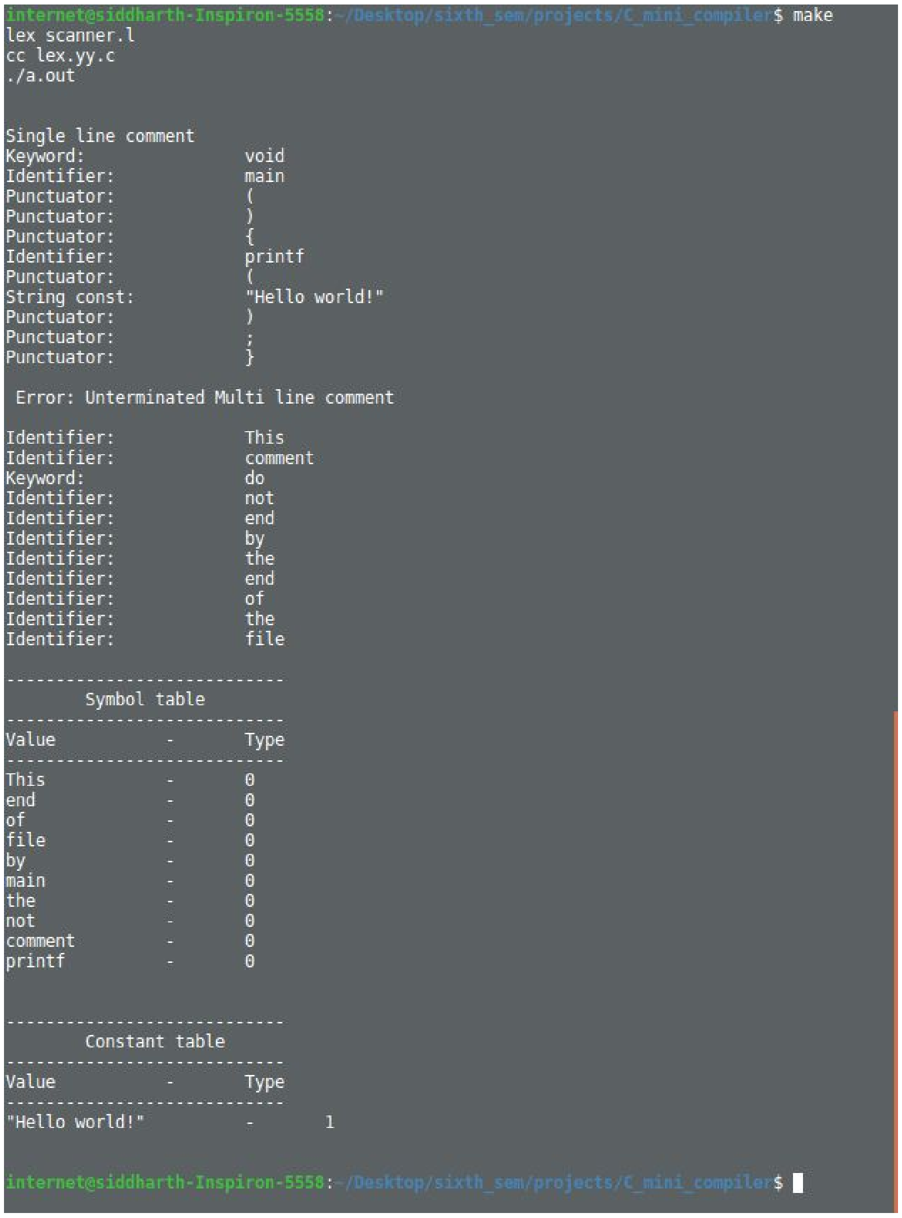
```
//Function call and pointer

void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

void main()
{
    int a=3, b=5;
    swap(&a, &b);
}
```



```
lex scanner.l
cc lex.yy.c
./a.out


Single line comment
Keyword:                void
Identifier:             swap
Punctuator:             (
Keyword:                int
Punctuator:             *
Identifier:             a
Punctuator:             ,
Keyword:                int
Punctuator:             *
Identifier:             b
Punctuator:             )
Punctuator:             {
Keyword:                int
Identifier:             temp
Punctuator:             =
Punctuator:             *
Identifier:             a
Punctuator:             ;
Punctuator:             *
Identifier:             a
Punctuator:             =
Punctuator:             *
Identifier:             b
Punctuator:             ;
Punctuator:             *
Identifier:             b
Punctuator:             =
Identifier:             temp
Punctuator:             ;
Punctuator:             }
Keyword:                void
Identifier:             main
Punctuator:             (
Punctuator:             )
Punctuator:             {
Keyword:                int
Identifier:             a
Punctuator:             =
Int const:              3
Punctuator:             ,
Identifier:             b
Punctuator:             =
Int const:              5
Punctuator:             ;
Identifier:             swap
Punctuator:             (
Punctuator:             &
Identifier:             a
Punctuator:             ,
Punctuator:             &
Identifier:             b
Punctuator:             )
Punctuator:             ;
Punctuator:             }

----------------------------
        Symbol table
----------------------------
Value           -       Type
----------------------------
main            -       0
temp            -       0
swap            -       0
a               -       0
b               -       0


----------------------------
        Constant table
----------------------------
Value           -       Type
----------------------------
3               -       2
5               -       2

internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/C_mini_compiler$
```

Test case 11

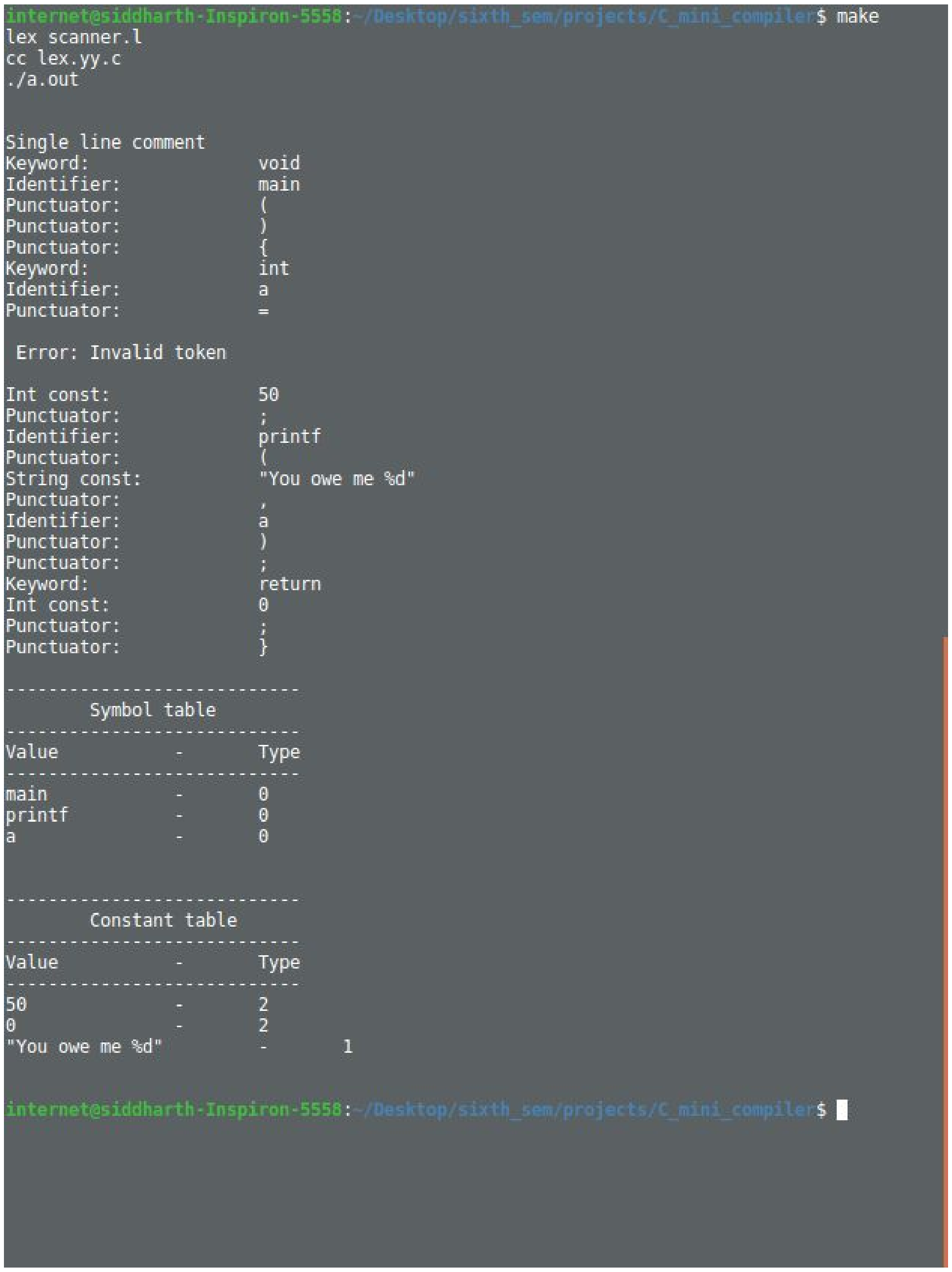
```
//Comments that dont end until end of the file

void main()
{

    printf("Hello world!");

}

/* This comment do not end by the end of the file
```



```
internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/C_mini_compiler$ make
lex scanner.l
cc lex.yy.c
./a.out


Single line comment
Keyword:                void
Identifier:             main
Punctuator:             (
Punctuator:             )
Punctuator:             {
Identifier:             printf
Punctuator:             (
String const:           "Hello world!"
Punctuator:             )
Punctuator:             ;
Punctuator:             }

 Error: Unterminated Multi line comment

Identifier:             This
Identifier:             comment
Keyword:                do
Identifier:             not
Identifier:             end
Identifier:             by
Identifier:             the
Identifier:             end
Identifier:             of
Identifier:             the
Identifier:             file

----------------------------
        Symbol table
----------------------------
Value           -       Type
----------------------------
This            -       0
end             -       0
of              -       0
file            -       0
by              -       0
main            -       0
the             -       0
not             -       0
comment         -       0
printf          -       0


----------------------------
        Constant table
----------------------------
Value           -       Type
----------------------------
"Hello world!"          -       1


internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/C_mini_compiler$
```

Test case 12

```
//Invalid tokens
void main()
{
    int a = $50;
    printf("You owe me %d",a);
    return 0;
}
```



```
internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/C_mini_compiler$ make
lex scanner.l
cc lex.yy.c
./a.out


Single line comment
Keyword:                void
Identifier:             main
Punctuator:             (
Punctuator:             )
Punctuator:             {
Keyword:                int
Identifier:             a
Punctuator:             =

 Error: Invalid token

Int const:              50
Punctuator:             ;
Identifier:             printf
Punctuator:             (
String const:           "You owe me %d"
Punctuator:             ,
Identifier:             a
Punctuator:             )
Punctuator:             ;
Keyword:                return
Int const:              0
Punctuator:             ;
Punctuator:             }

----------------------------
        Symbol table
----------------------------
Value           -       Type
----------------------------
main            -       0
printf          -       0
a               -       0


----------------------------
        Constant table
----------------------------
Value           -       Type
----------------------------
50              -       2
0               -       2
"You owe me %d"         -       1


internet@siddharth-Inspiron-5558:~/Desktop/sixth_sem/projects/C_mini_compiler$ 
```