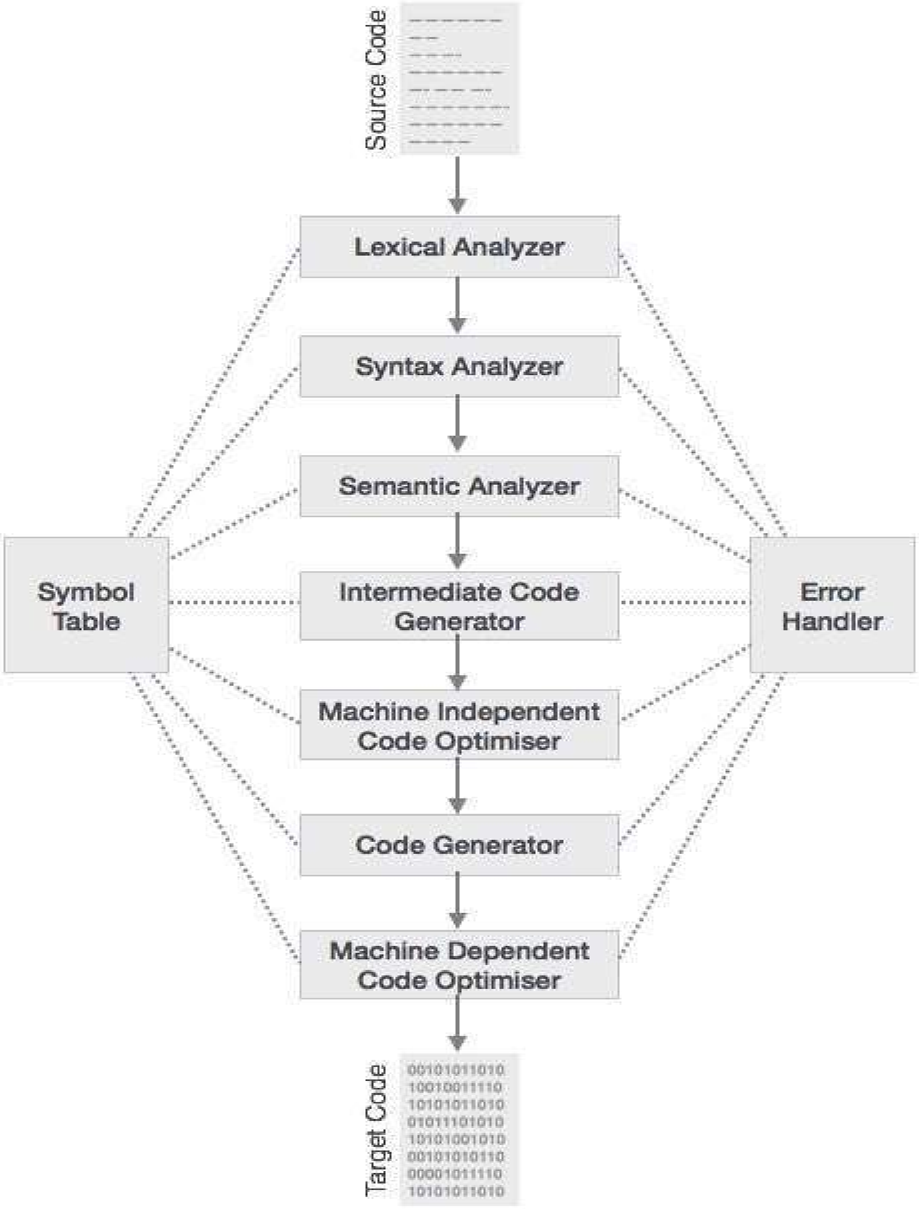
Introduction

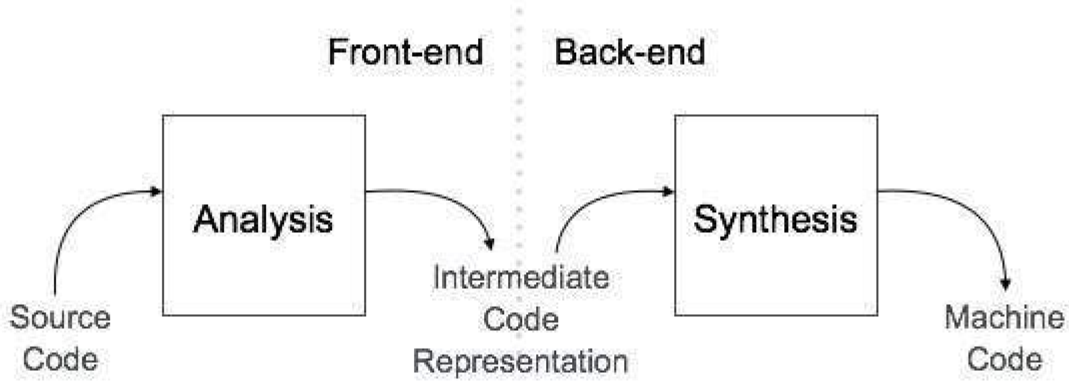
A compiler is a program that converts high-level language to assembly language. As a general practice, it is implemented in modules for two main reasons. One, to simplify implementation of each part. Secondly, it allows programmer to increase efficiency of a module independent of working of others. Thus, few basic modules are defined along with their inputs and outputs. They are as follows:



A compiler can broadly be divided into two phases based on the way they compile.

Analysis Phase

Known as the front-end of the compiler, the analysis phase of the compiler reads the source program, divides it into core parts and then checks for lexical, grammar and syntax errors.The analysis phase generates an intermediate representation of the source program and symbol table, which should be fed to the Synthesis phase as input.



Synthesis Phase

Known as the back-end of the compiler, the synthesis phase generates the target program with the help of intermediate source code representation and symbol table.

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.

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

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.

TOKEN: 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.

The Longest Match Rule states that the lexeme scanned should be determined based on the longest match among all the tokens available.

The lexical analyzer also follows rule priority where a reserved word, e.g., a keyword, of a language is given priority over user input. That is, if the lexical analyzer finds a lexeme that matches with any existing reserved word, it should generate an error.

Code: *scanner.l* (lex file to be used to generate scanner)

```
/* Authors: Ritwick Mishra, Abhilash Sanap        Date: 18-01-18*/
%{
#include <stdio.h>
#include <string.h>
#define SIZE 1000
void insert_hash(char *,char *,int);
void display_hash();
struct DataItem {
        char* text;
        char* type;
        int lineno;
        struct DataItem * next;
        int attr;
};
struct DataItem* hashArray[SIZE];                   /*Hash Table*/
struct DataItem* item;
int v=0;
%}
alpha [a-zA-Z_]
digit [0-9]
%%
[ \t]           ;                                   //Removes spaces and tabs
[ \n]   { yylineno = yylineno + 1;}
int             {ECHO; insert_hash("int","Datatype",yylineno);}
char            {ECHO; insert_hash("char","Datatype",yylineno);}
void            {ECHO; insert_hash("void","Datatype",yylineno);}
while           {ECHO; insert_hash("while","Keyword",yylineno);}
if              {ECHO; insert_hash("if","Keyword",yylineno);}
else            {ECHO; insert_hash("else","Keyword",yylineno);}
printf          {ECHO; insert_hash("printf","Function",yylineno);}
scanf           {ECHO; insert_hash(yytext,"Function",yylineno);}
^"#include ".+ ;
return          {ECHO; insert_hash(yytext,"Keyword",yylineno);}
{digit}+        {ECHO; insert_hash(yytext,"Integer",yylineno);}
{alpha}({alpha}|{digit})* {ECHO; insert_hash(yytext,"Identifier",yylineno);}
        /*Special Characters*/
"{"             {ECHO; insert_hash("{","Sp. Char",yylineno);}
"}"             {ECHO; insert_hash("{","Sp. Char",yylineno);}
"["             {ECHO; insert_hash("{","Sp. Char",yylineno);}
"]"             {ECHO; insert_hash("{","Sp. Char",yylineno);}
";"             {ECHO; insert_hash("{","Sp. Char",yylineno);}
"("             {ECHO; insert_hash("{","Sp. Char",yylineno);}
")"             {ECHO; insert_hash("{","Sp. Char",yylineno);}
"&"             {ECHO; insert_hash("{","Sp. Char",yylineno);}
"|"             {ECHO; insert_hash("{","Sp. Char",yylineno);}
"!"             {ECHO; insert_hash("{","Sp. Char",yylineno);}
```

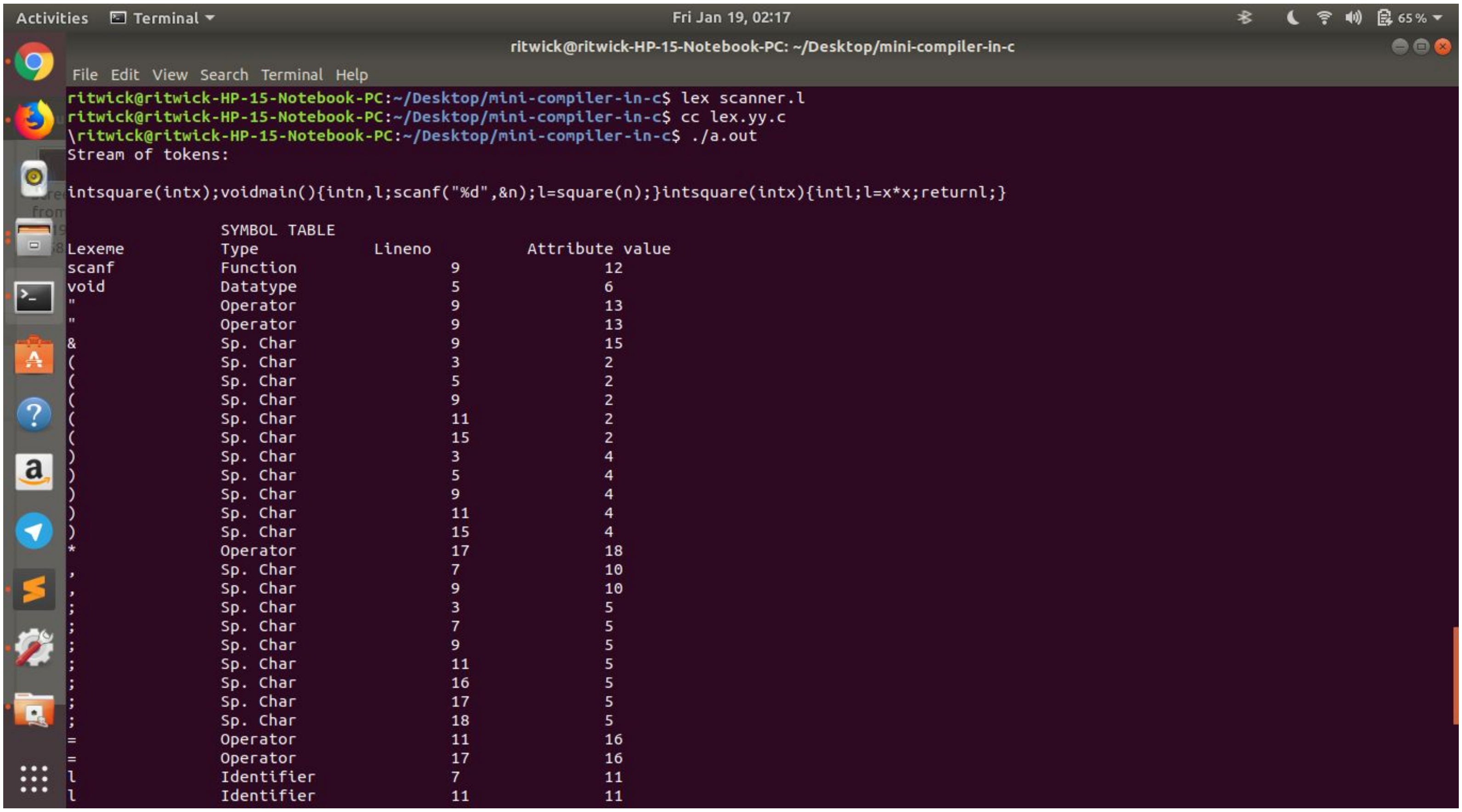
```
","             {ECHO; insert_hash("{","Sp. Char",yylineno);}
"."             {ECHO; insert_hash("{","Sp. Char",yylineno);}
"%c"            {ECHO; insert_hash("{","Sp. Char",yylineno);}
"%d"            {ECHO; insert_hash("{","Sp. Char",yylineno);}
        /*Operators*/
"+"             {ECHO; insert_hash(yytext,"Operator",yylineno);}
"-"             {ECHO; insert_hash(yytext,"Operator",yylineno);}
"*"             {ECHO; insert_hash(yytext,"Operator",yylineno);}
"%"             {ECHO; insert_hash(yytext,"Operator",yylineno);}
"="             {ECHO; insert_hash(yytext,"Operator",yylineno);}
"=="            {ECHO; insert_hash(yytext,"Operator",yylineno);}
">="            {ECHO; insert_hash(yytext,"Operator",yylineno);}
"<="            {ECHO; insert_hash(yytext,"Operator",yylineno);}
">"             {ECHO; insert_hash(yytext,"Operator",yylineno);}
"<"             {ECHO; insert_hash(yytext,"Operator",yylineno);}
"&&"            {ECHO; insert_hash(yytext,"Operator",yylineno);}
"||"            {ECHO; insert_hash(yytext,"Operator",yylineno);}
"\""            {ECHO; insert_hash(yytext,"Operator",yylineno);}
\\              {ECHO; insert_hash(yytext,"Operator",yylineno);}
        /*Comments*/
\/\/.*  ;
\/\*(.*\n)*.*\*\/       ;
.                       {printf("\nIllegal token at line no : %d\nExiting...\n\n",yylineno);
                        return -1;
                        }

%%
int hashCode(char* key) {
  unsigned int i,hash=7;
  for(i=0;i<strlen(key);++i){
     hash=hash*31+key[i];
  }
  return hash % SIZE;
}
void insert_hash(char* text,char* type,int lineno) {
        int len,len2;
        len= strlen(text);
        len2= strlen(type);
  int hashIndex = hashCode(text);
  if(hashArray[hashIndex] != NULL) {
          struct DataItem * head = hashArray[hashIndex];
          int temp = head->attr;
          struct DataItem * new;
          new = (struct DataItem *)malloc(sizeof(struct DataItem));
          new-> type = (char *)malloc(len2*sizeof(char));
          strcpy(new->type,type);
          new->lineno= lineno;
          new-> text = (char *)malloc(len*sizeof(char));
```

```
             strcpy(new->text,text);
             new->attr = temp;
             while(head->next!=NULL){
                    head=head->next;
             }
             head->next =new;
   }
   else{
   hashArray[hashIndex] = (struct DataItem*)malloc(sizeof(struct DataItem));
   hashArray[hashIndex]-> type = (char *)malloc(len2*sizeof(char));
   strcpy(hashArray[hashIndex]->type,type);
   hashArray[hashIndex]-> lineno= lineno;
   hashArray[hashIndex]-> text = (char *)malloc(len*sizeof(char));
   strcpy(hashArray[hashIndex]->text,text);
   hashArray[hashIndex]->next = NULL;
   hashArray[hashIndex]->attr = v++;
}
}
void display_hash() {
   int i = 0;
  printf("Lexeme\t\tType\t\tLineno\t\tAttribute value\n");
   for(i = 0; i<SIZE; i++) {
      if(hashArray[i] != NULL){
        struct DataItem * head = hashArray[i];
        printf("%s\t\t%s\t\t%d\t\t%d\n",head->text,head->type,head->lineno,head->attr);
        while(head->next!=NULL){
          head=head->next;
          printf("%s\t\t%s\t\t%d\t\t%d\n",head->text,head->type,head->lineno,head->attr);
        }
      }
   }
     printf("\n");
}
int main(){
          yyin = fopen("tc1.c","r");
          printf("Stream of tokens:\n\n");
          yylex();
          printf("\n\n\t\tSYMBOL TABLE\n");
          display_hash();
          return 0;
}
int yywrap(){
          return 1;
}
```

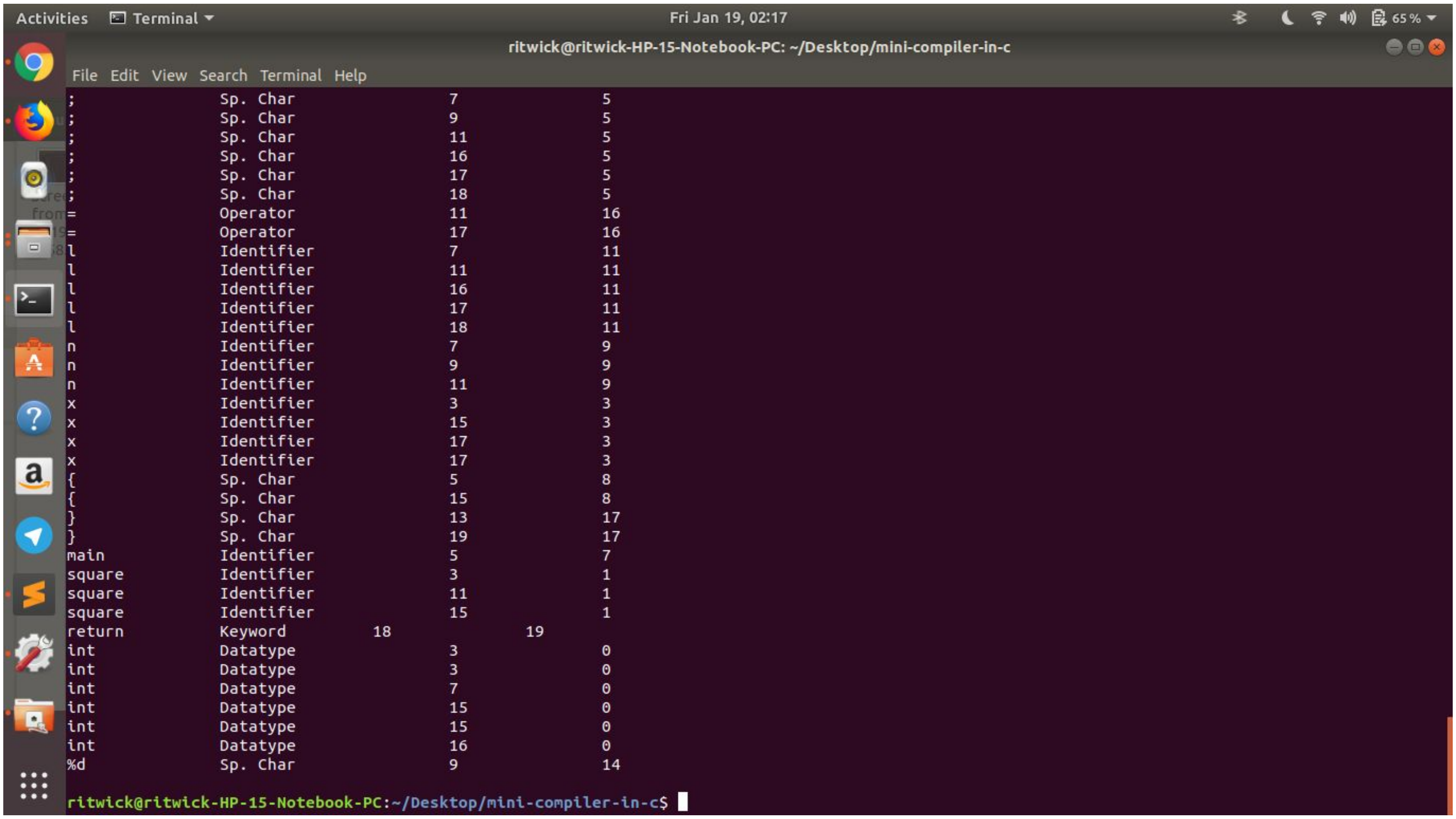
Test case 1 (no errors):

```
#include <stdio.h>
//function prototype
int square (int x);
/*hello
        hi
        goodbye*/
void main(){
        //int variables
        int n,l;
        scanf ("%d", &n );
        //function call
        l=square(n);
}
int square (int x){                    //function definition
        int l;
        l= x * x;
        return l;
}
```

Output:



```
ritwick@ritwick-HP-15-Notebook-PC:~/Desktop/mini-compiler-in-c$ lex scanner.l
ritwick@ritwick-HP-15-Notebook-PC:~/Desktop/mini-compiler-in-c$ cc lex.yy.c
\ritwick@ritwick-HP-15-Notebook-PC:~/Desktop/mini-compiler-in-c$ ./a.out
Stream of tokens:

intsquare(intx);voidmain(){intn,l;scanf("%d",&n);l=square(n);}intsquare(intx){intl;l=x*x;returnl;}

                SYMBOL TABLE
Lexeme          Type            Lineno          Attribute value
scanf           Function                9               12
void            Datatype                5               6
"               Operator                9               13
"               Operator                9               13
&               Sp. Char                9               15
(               Sp. Char                3               2
(               Sp. Char                5               2
(               Sp. Char                9               2
(               Sp. Char                11              2
(               Sp. Char                15              2
)               Sp. Char                3               4
)               Sp. Char                5               4
)               Sp. Char                9               4
)               Sp. Char                11              4
)               Sp. Char                15              4
*               Operator                17              18
,               Sp. Char                7               10
,               Sp. Char                9               10
;               Sp. Char                3               5
;               Sp. Char                7               5
;               Sp. Char                9               5
;               Sp. Char                11              5
;               Sp. Char                16              5
;               Sp. Char                17              5
;               Sp. Char                18              5
=               Operator                11              16
=               Operator                17              16
l               Identifier              7               11
l               Identifier              11              11
```

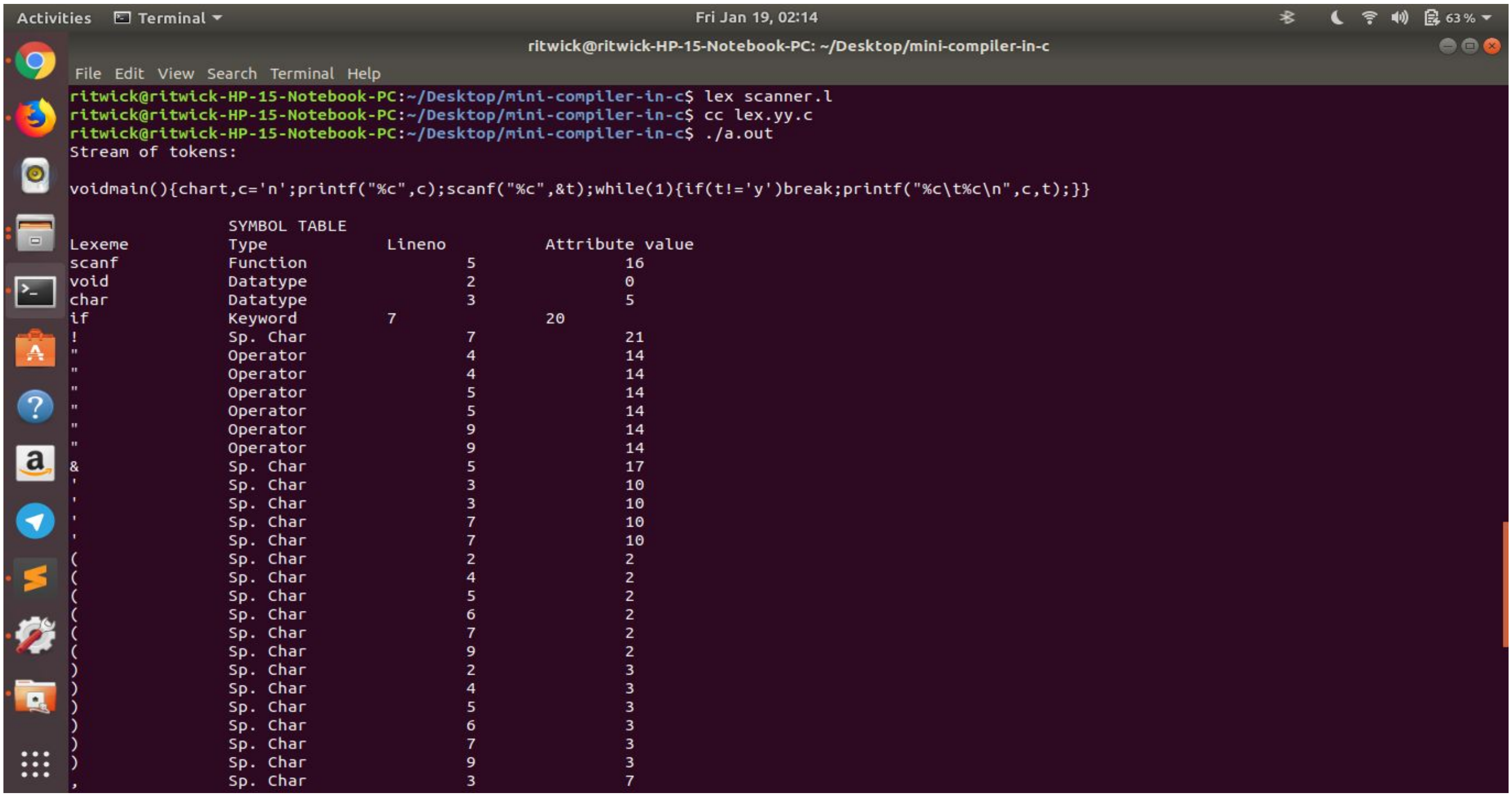

Activities
Terminal
Fri Jan 19, 02:17
65 %
ritwick@ritwick-HP-15-Notebook-PC: ~/Desktop/mini-compiler-in-c
File Edit View Search Terminal Help

```
;               Sp. Char                7               5
;               Sp. Char                9               5
;               Sp. Char                11              5
;               Sp. Char                16              5
;               Sp. Char                17              5
;               Sp. Char                18              5
=               Operator                11              16
=               Operator                17              16
l               Identifier              7               11
l               Identifier              11              11
l               Identifier              16              11
l               Identifier              17              11
l               Identifier              18              11
n               Identifier              7               9
n               Identifier              9               9
n               Identifier              11              9
x               Identifier              3               3
x               Identifier              15              3
x               Identifier              17              3
x               Identifier              17              3
{               Sp. Char                5               8
{               Sp. Char                15              8
}               Sp. Char                13              17
}               Sp. Char                19              17
main            Identifier              5               7
square          Identifier              3               1
square          Identifier              11              1
square          Identifier              15              1
return          Keyword         18              19
int             Datatype                3               0
int             Datatype                3               0
int             Datatype                7               0
int             Datatype                15              0
int             Datatype                15              0
int             Datatype                16              0
%d              Sp. Char                9               14

ritwick@ritwick-HP-15-Notebook-PC:~/Desktop/mini-compiler-in-c$ 
```

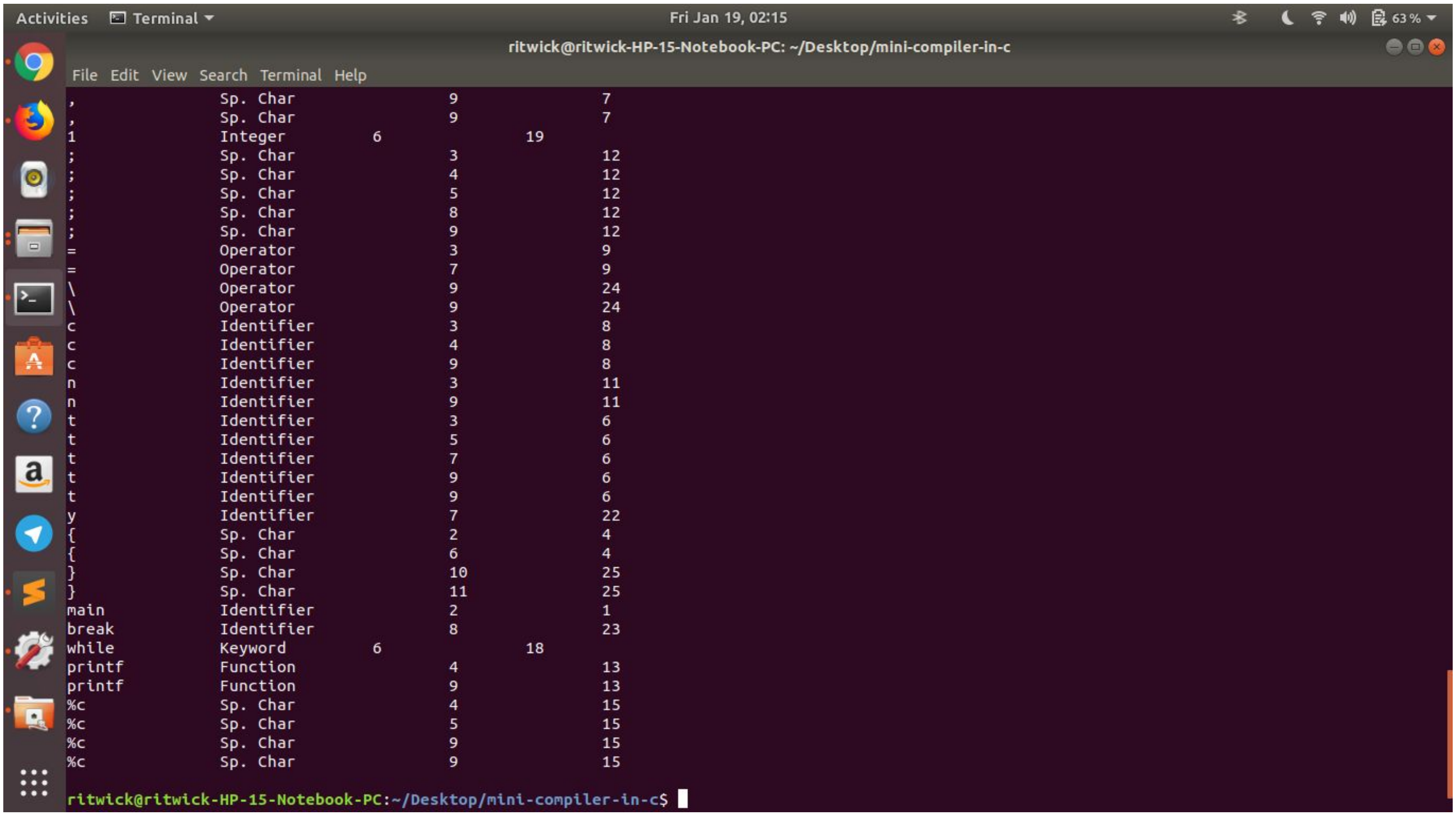
Test case 2 (no errors):

```
#include <stdio.h>
 void main(){
         char t,c = 'n';
         printf("%c",c);
         scanf("%c",&t);
         while (1){
                 if(t!='y')
                         break;
                 printf("%c\t%c\n",c,t);
         }
 }
```

Output:


Activities
Terminal
Fri Jan 19, 02:14
63 %
ritwick@ritwick-HP-15-Notebook-PC: ~/Desktop/mini-compiler-in-c
File Edit View Search Terminal Help

```
ritwick@ritwick-HP-15-Notebook-PC:~/Desktop/mini-compiler-in-c$ lex scanner.l
ritwick@ritwick-HP-15-Notebook-PC:~/Desktop/mini-compiler-in-c$ cc lex.yy.c
ritwick@ritwick-HP-15-Notebook-PC:~/Desktop/mini-compiler-in-c$ ./a.out
Stream of tokens:

voidmain(){chart,c='n';printf("%c",c);scanf("%c",&t);while(1){if(t!='y')break;printf("%c\t%c\n",c,t);}}

                SYMBOL TABLE
Lexeme          Type            Lineno          Attribute value
scanf           Function                5               16
void            Datatype                2               0
char            Datatype                3               5
if              Keyword         7               20
!               Sp. Char                7               21
"               Operator                4               14
"               Operator                4               14
"               Operator                5               14
"               Operator                5               14
"               Operator                9               14
"               Operator                9               14
&               Sp. Char                5               17
'               Sp. Char                3               10
'               Sp. Char                3               10
'               Sp. Char                7               10
'               Sp. Char                7               10
(               Sp. Char                2               2
(               Sp. Char                4               2
(               Sp. Char                5               2
(               Sp. Char                6               2
(               Sp. Char                7               2
(               Sp. Char                9               2
)               Sp. Char                2               3
)               Sp. Char                4               3
)               Sp. Char                5               3
)               Sp. Char                6               3
)               Sp. Char                7               3
)               Sp. Char                9               3
,               Sp. Char                3               7
```

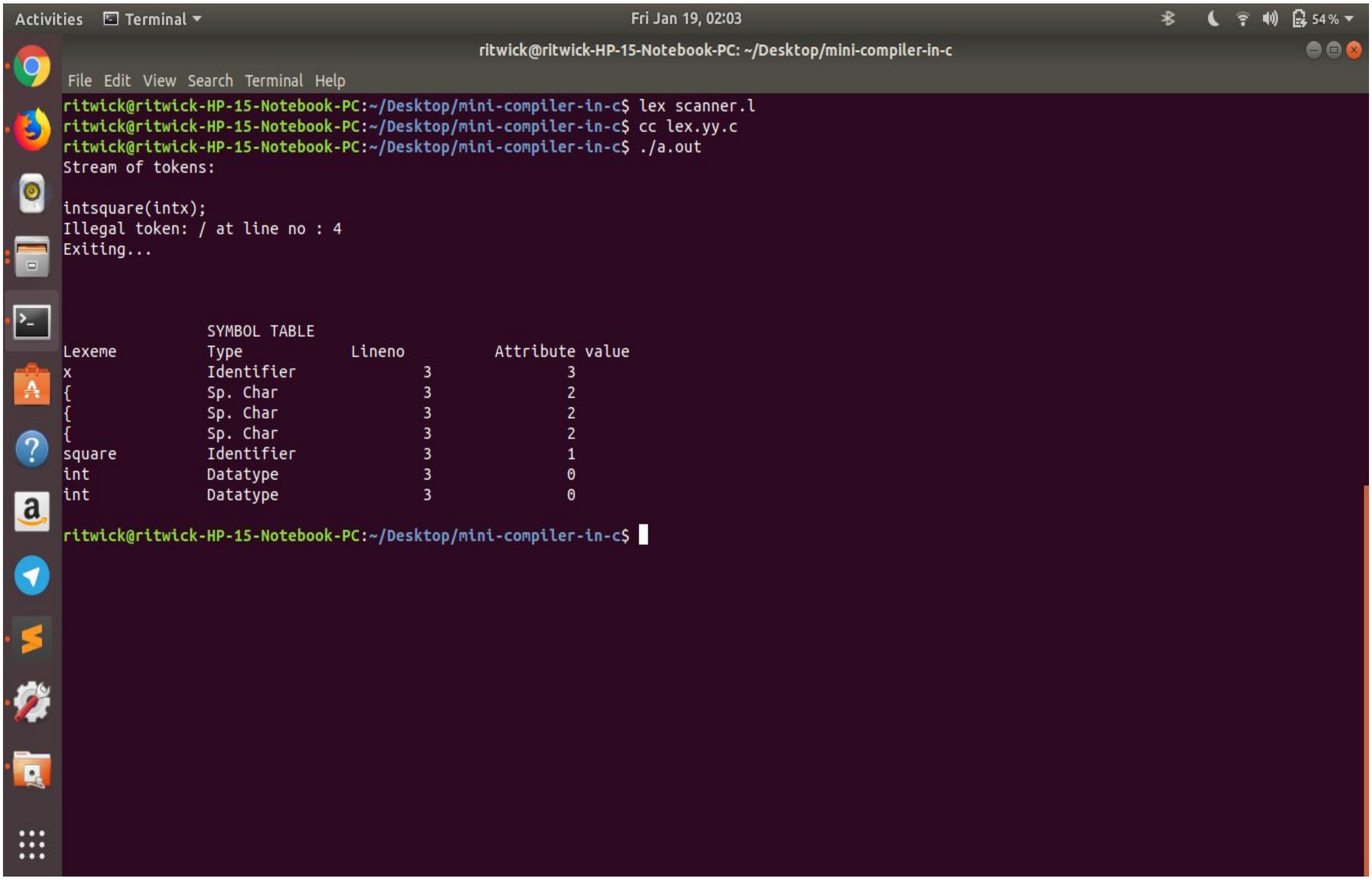

Activities
Terminal
Fri Jan 19, 02:15
63 %
ritwick@ritwick-HP-15-Notebook-PC: ~/Desktop/mini-compiler-in-c
File Edit View Search Terminal Help

```
,               Sp. Char                9               7
,               Sp. Char                9               7
1               Integer         6               19
;               Sp. Char                3               12
;               Sp. Char                4               12
;               Sp. Char                5               12
;               Sp. Char                8               12
;               Sp. Char                9               12
=               Operator                3               9
=               Operator                7               9
\               Operator                9               24
\               Operator                9               24
c               Identifier              3               8
c               Identifier              4               8
c               Identifier              9               8
n               Identifier              3               11
n               Identifier              9               11
t               Identifier              3               6
t               Identifier              5               6
t               Identifier              7               6
t               Identifier              9               6
t               Identifier              9               6
y               Identifier              7               22
{               Sp. Char                2               4
{               Sp. Char                6               4
}               Sp. Char                10              25
}               Sp. Char                11              25
main            Identifier              2               1
break           Identifier              8               23
while           Keyword         6               18
printf          Function                4               13
printf          Function                9               13
%c              Sp. Char                4               15
%c              Sp. Char                5               15
%c              Sp. Char                9               15
%c              Sp. Char                9               15

ritwick@ritwick-HP-15-Notebook-PC:~/Desktop/mini-compiler-in-c$
```

Test case 3 (error: comment reaches end of file without closing):

```
#include <stdio.h>
//function prototype
int square (int x);
/*hello
        hi
        goodbye
int main(){                         //function definition
        int l;
        l= 2 * 2;
        printf("%d",l);
        return 1;
}
```

Output:

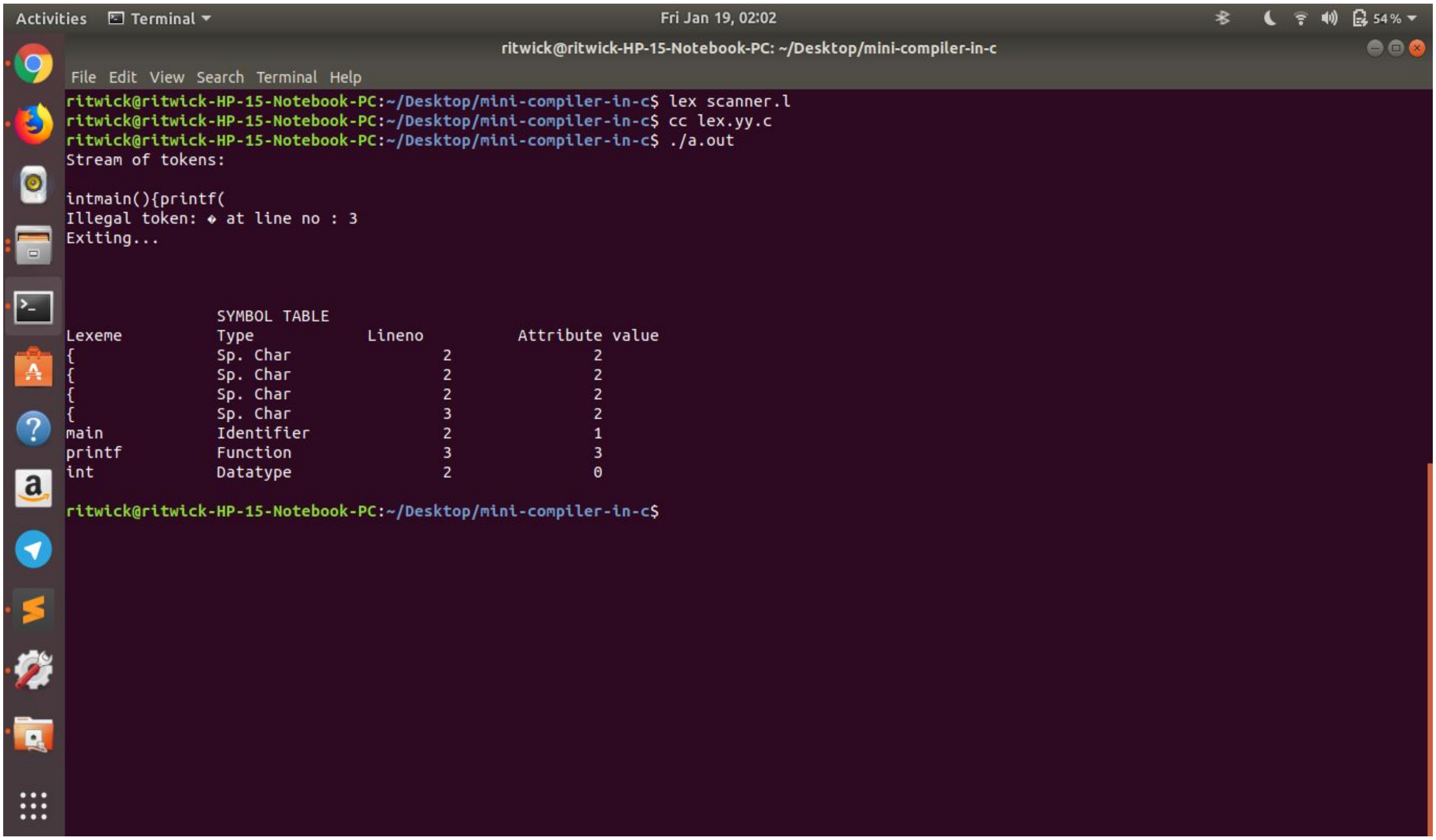

Activities
Terminal
Fri Jan 19, 02:03
54 %
ritwick@ritwick-HP-15-Notebook-PC: ~/Desktop/mini-compiler-in-c
File Edit View Search Terminal Help

```
ritwick@ritwick-HP-15-Notebook-PC:~/Desktop/mini-compiler-in-c$ lex scanner.l
ritwick@ritwick-HP-15-Notebook-PC:~/Desktop/mini-compiler-in-c$ cc lex.yy.c
ritwick@ritwick-HP-15-Notebook-PC:~/Desktop/mini-compiler-in-c$ ./a.out
Stream of tokens:

intsquare(intx);
Illegal token: / at line no : 4
Exiting...



                SYMBOL TABLE
Lexeme          Type            Lineno          Attribute value
x               Identifier              3               3
{               Sp. Char                3               2
{               Sp. Char                3               2
{               Sp. Char                3               2
square          Identifier              3               1
int             Datatype                3               0
int             Datatype                3               0

ritwick@ritwick-HP-15-Notebook-PC:~/Desktop/mini-compiler-in-c$ 
```

Test case 4 (error: unrecognised token):

```
#include <stdio.h>
int main(){
        printf(“%s”,` ^\n);
        return 0;
}
```

Output:



Activities
Terminal
Fri Jan 19, 02:02
54 %
ritwick@ritwick-HP-15-Notebook-PC: ~/Desktop/mini-compiler-in-c
File Edit View Search Terminal Help

```
ritwick@ritwick-HP-15-Notebook-PC:~/Desktop/mini-compiler-in-c$ lex scanner.l
ritwick@ritwick-HP-15-Notebook-PC:~/Desktop/mini-compiler-in-c$ cc lex.yy.c
ritwick@ritwick-HP-15-Notebook-PC:~/Desktop/mini-compiler-in-c$ ./a.out
Stream of tokens:

intmain(){printf(
Illegal token: � at line no : 3
Exiting...



                SYMBOL TABLE
Lexeme          Type            Lineno          Attribute value
{               Sp. Char                2               2
{               Sp. Char                2               2
{               Sp. Char                2               2
{               Sp. Char                3               2
main            Identifier              2               1
printf          Function                3               3
int             Datatype                2               0

ritwick@ritwick-HP-15-Notebook-PC:~/Desktop/mini-compiler-in-c$
```