• the source text
• \( xy := x*100; \)  // multiply \( x \) by hundred
  // and assign to \( xy \)
• is handled as the sequence of tokens

<table>
<thead>
<tr>
<th>TokenType</th>
<th>TokenValue</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDENT</td>
<td>( xy )</td>
</tr>
<tr>
<td>ASSIGN</td>
<td>-----</td>
</tr>
<tr>
<td>IDENT</td>
<td>( x )</td>
</tr>
<tr>
<td>ARITH_OPRTR</td>
<td>*</td>
</tr>
<tr>
<td>INTEGER</td>
<td>100</td>
</tr>
<tr>
<td>SEMICOLON</td>
<td>-----</td>
</tr>
</tbody>
</table>

Example: Tokenize the input file
```c++
#include <iostream.h>
#include <fstream.h>

int main()
{
    char s[255];
    char ch;
    ifstream infile("in.dat");
    infile.get(s, 20, 'n');
    cout << "gcount=" << infile.gcount() << endl;
    infile.ignore(80, 'n');
    // get rid off chars until newline '\n'
    do
    {
        infile >> ch;
        if (infile.eof()) { cout << "Test\n"; break;}
    } while ( ch != 'n' );
}
```
Symbol Table

- Stores tokens and their attributes
  - types, values, and any other relevant associated information
- Used repeatedly during parsing of the tokenized input and object code generation
- Created during lexical analysis phase and modified during later phases
- Hashing is one way to efficiently store and access the entries in the symbol table

Yet another example – Tokenize the following

```plaintext
// Greatest Common Divisor
x := 8;
y := 12;
WHILE x != y DO
  IF x > y THEN x := x-y
  ELSE y := y-x
  FI
OD;
PRINT x
```

input to Lex:

```plaintext
%{// Greatest Common Divisor
  x := 8;
  y := 12;
  WHILE x != y DO
    IF x > y THEN x := x-y
    ELSE y := y-x
    FI
  OD;
  PRINT x
}%
```
Lex Primer

- Lex is a tool for creating lexical analyzers.
- Lexical analyzers tokenize input streams.
- Tokens are the terminals of a language.
- Regular expressions define tokens.

Usage Paradigm of Lex

Some Simple Lex Source Examples

- A minimum lex program:
  
  ```
  %%
  It only copies the input to the output unchanged.
  ```
- A trivial program to delete three spacing characters:
  
  ```
  %%
  [ \t\n];
  ```
- Another trivial example:
  
  ```
  %%
  [ \t$];
  It deletes from the input all blanks or tabs at the ends of lines.
  ```
A General Lex Source Example

```c
#include <stdio.h>

bin_digit [01]

{bin_digit}+ {  
    /* match all strings of 0's and 1's */  
    /* Print out message with matching */  
    /* text */  
    printf("BINARY: %s\n", yytext);  
  }  
  ((ab)*aa(ab)*bb(ab)* | (ab)*bb(ab)*aa(ab)*) {  
    /* match all strings over */  
    /* (a,b) containing aa and bb */  
    printf("AABB\n");  
  }  
  \n; /* ignore newlines */
```

```c
int main(int argc, char *argv[]) {  
    /* call yylex to use the generated lexer */  
    yylex();  
    /* make sure everything was printed */  
    fflush(yyout);  
    exit(0);  
}
```
Token Definitions
(Extended Regular Expression)

• Elementary Operations
  – single characters
    • except \^ \ . \ $ \ ^ \ [ \] - ? * + | ( ) / { } % < >
  – concatenation (put characters together)
    – alternation (a|b|c)
      • [ab] == a|b
      • [a-k] == a|b|c|...|i|j|k
      • [a-z0-9] == any letter or digit
      • [^a] == any character but a

• Elementary Operations (cont.)
  . matches any character except the newline
  * -- Kleene Closure
  + -- Positive Closure

• Examples:
    • note: without the quotes it could be any
      character
  – \[ \] is whitespace
    • (except CR),
      • There is a blank space character before the \t

• Special Characters:
  . -- matches any single character
    (except newline)
  * and \ -- quote the part as text
  \ -- tab
  \n -- newline
  \b -- backspace
  \' -- double quote
  \ \ -- \n
  ? -- this means the preceding was optional
  ab? == ab
  (ab)? == ab|c
• Special Characters (cont.)
  ^ -- means at the beginning of the line
      (unless it is inside of a [ ])
  $ means at the end of the line, same as /n
  [^ ] -- means anything except
      \("\)\" is a double quoted string
  (n,m) -- n through m occurrences
      a(1,3) is a or aa or aaa
  \{varName\} -- translation of varName from definition
  / -- matches only if followed by right part of /
      0/1 means the 0 of 01 but not 02 or 03 or ...
  ( ) -- grouping