**Use of Special Symbols in Lex Regular Expressions**

Lex (or Flex) uses **regular expressions (regex)** to define patterns for lexical analysis. Operators like +, \*, /, ?, |, and [] help create patterns to match **identifiers, numbers, operators, and other tokens**.

**1. + (One or More)**

The + operator means **one or more occurrences** of the preceding character or group.

**Example**

[a-zA-Z]+

* Matches **one or more** letters (e.g., hello, World, xYz).
* Will **not** match 123 (since it has no letters).

**Practical Usage**

✔ Identifiers: [a-zA-Z\_][a-zA-Z0-9\_]\*  
✔ Numbers: [0-9]+ (matches 1, 23, 456)

**2. \* (Zero or More)**

The \* operator means **zero or more occurrences** of the preceding character or group.

**Example**

[a-zA-Z]\*

* Matches **zero or more** letters.
* Matches: hello, abc, xyz, or even an empty string.

**Practical Usage**

✔ Whitespace Handling: [ \t\n]\* (matches spaces, tabs, or newlines)  
✔ Multi-line Comments: /\\*([^\*]|\\*+[^\*/])\*\\*+/

**3. ? (Zero or One)**

The ? operator means **zero or one occurrence** of the preceding character or group.

**Example**

[a-zA-Z]?[0-9]+

* Matches **optional** letter before numbers.
* Matches: x123, A456, 789 (with or without letters at the start).

**4. . (Any Single Character)**

The . operator matches **any single character** except a newline.

**Example**

.+

* Matches **any sequence** of characters except a newline.

**Practical Usage**

✔ Any character: .\* (matches everything except newlines)  
✔ Single-line Comments: //.\*

**5. | (Alternation OR)**

The | operator means **either this OR that**.

**Example**

if|else|while|for

* Matches any of if, else, while, or for.

**Practical Usage**

✔ Matching Keywords: int|float|char|double  
✔ Matching Operators: [+\-\*/]

**6. [] (Character Class)**

Square brackets [] define a **set of characters**, meaning **any one** of the enclosed characters can match.

**Example**

[aeiou]

* Matches **any single vowel** (a, e, i, o, u).

**More Examples**

[a-z] // Matches any lowercase letter

[A-Z] // Matches any uppercase letter

[0-9] // Matches any digit

[a-zA-Z] // Matches any letter

**Practical Usage**

✔ Identifiers: [a-zA-Z\_][a-zA-Z0-9\_]\*  
✔ Numbers: [0-9]+

**7. [^ ] (Negation)**

The [^ ] operator means **"match any character NOT in this set"**.

**Example**

[^0-9]

* Matches any character that **is NOT a digit**.

**Practical Usage**

✔ Removing everything except letters: [a-zA-Z]+  
✔ Matching non-whitespace: [^ \t\n]

**8. () (Grouping)**

Parentheses () **group multiple characters together** to apply operators like +, \*, ?.

**Example**

(abc)+

* Matches **"abc" one or more times**: (abc, abcabc, abcabcabc).

**Practical Usage**

✔ Floating point numbers: [0-9]+(\.[0-9]+)?  
✔ Multi-line comments: /\\*([^\*]|\\*+[^\*/])\*\\*+/

**9. \ (Escape Character)**

The \ backslash is used to **escape special characters**.

**Example**

\\*

* Matches **a literal \*** instead of using it as an operator.

**Practical Usage**

✔ Matching +, \*, ?: \+, \\*, \?  
✔ Matching . literally: \.

**Examples in a Full Lex Program**

%{

#include <stdio.h>

%}

%%

// Keywords

if|else|while|for { printf("Keyword: %s\n", yytext); }

// Identifiers

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

// Numbers (Integers and Floats)

[0-9]+ { printf("Integer: %s\n", yytext); }

[0-9]+\\.[0-9]+ { printf("Float: %s\n", yytext); }

// Operators

[+\-\*/=] { printf("Operator: %s\n", yytext); }

// Ignore Whitespace

[ \t\n]+ { /\* Skip \*/ }

**Lexical Rules and Their Structure in Lex**

Lexical rules define how an input text is analyzed and categorized into meaningful tokens. Each rule consists of a **pattern** and an associated **action**.

**Structure of a Lexical Rule**

PATTERN { ACTION }

* **PATTERN**: A regular expression defining what kind of text to match.
* **ACTION**: The corresponding C code to execute when the pattern matches.

**List of Functions Used in Lexical Rules**

In the provided Lex programs, the following functions are used:

| **Function** | **Purpose** |
| --- | --- |
| yylex() | Lex's built-in function that scans the input and applies rules. |
| yytext | A special Lex variable that holds the matched text. |
| yyleng | Stores the length of the matched text. |
| yyin | File pointer for input stream. |
| yywrap() | Called when the end of input is reached (returns 1 to indicate EOF). |
| fprintf(FILE \*, format, args...) | Writes formatted output to a file. |
| fopen(filename, mode) | Opens a file with the given mode (e.g., "r", "w"). |
| fclose(FILE \*) | Closes an opened file. |
| printf(format, args...) | Prints output to the console. |

**How to Write Lexical Rules**

**Basic Components**

1. **Single Character Rule**  
   Matches and processes a single character.
2. . { printf("Character: %s\n", yytext); }
3. **Keyword Matching**  
   Detects and prints keywords.
4. if|else|for|while { printf("Keyword: %s\n", yytext); }
5. **Identifier Detection (Variable Names)**  
   Matches valid C/C++ identifiers.
6. [a-zA-Z\_][a-zA-Z0-9\_]\* { printf("Identifier: %s\n", yytext); }
7. **Integer and Float Detection**  
   Recognizes numbers and differentiates integers from floats.
8. [0-9]+ { printf("Integer: %s\n", yytext); }
9. [0-9]+\\.[0-9]+ { printf("Float: %s\n", yytext); }
10. **Whitespace Handling**  
    Matches spaces, tabs, and newlines.
11. [ \t\n] { /\* Ignore whitespace \*/ }
12. **Comment Removal**  
    Removes single-line and multi-line comments.
13. //.\* { /\* Ignore single-line comments \*/ }
14. /\\\*([^\*]|\\\*+[^\*/])\*\\\*+/ { /\* Ignore multi-line comments \*/ }
15. **HTML Tag Extraction**  
    Captures and stores HTML tags.
16. /<[^>]+>/ { fprintf(outFile, "%s\n", yytext); }

%%

int main() {

yylex(); // Calls the lexer

return 0;

}

int yywrap() {

return 1;

}

**Summary of Operators**

| **Operator** | **Description** | **Example** | **Matches** |
| --- | --- | --- | --- |
| + | One or more | [0-9]+ | 1, 23, 456 |
| \* | Zero or more | [a-z]\* | abc, xyz, "" (empty string) |
| ? | Zero or one | [0-9]? | 1, "", 4 |
| . | Any single character | a.b | axb, a1b, a@b |
| ` | ` | Alternation (OR) | `if |
| [] | Character class | [aeiou] | a, e, i, o, u |
| [^] | Negation | [^0-9] | Any non-digit |
| () | Grouping | (abc)+ | abc, abcabc |
| \ | Escape special characters | \. | . literal |

**Conclusion**

* **Lex uses regular expressions** to define tokens.
* **Operators like +, \*, ?, |, and []** help build patterns.
* **Use \ to escape special characters** when needed.
* **Parentheses () allow grouping for complex patterns**.

**Explanation of the Code:**

1. **Header Inclusion**:
2. %{
3. #include <stdio.h>
4. %}
   * This section includes the standard I/O library (stdio.h), which is required for input and output operations.
5. **Lexical Rules Section**:
6. %%
7. \n { lines++; }
8. [ ] { spaces++; }
9. \t { tabs++; }
10. . { others++; }
11. %%
    * \n → Matches a newline character (Enter key). Increments the lines counter.
    * [ ] → Matches a space character. Increments the spaces counter.
    * \t → Matches a tab character. Increments the tabs counter.
    * . → Matches any other character. Increments the others counter.
12. **Global Variable Declaration**:
13. int lines = 0, spaces = 0, tabs = 0, others = 0;
    * Initializes counters for lines, spaces, tabs, and other characters.
14. **Main Function**:
15. int main() {
16. printf("Enter text (Press Ctrl+D to end input on Unix/Linux or Ctrl+Z on Windows):\n");
17. yylex(); // Calls the lexer to process input
18. printf("Lines: %d\n", lines);
19. printf("Spaces: %d\n", spaces);
20. printf("Tabs: %d\n", tabs);
21. printf("Other characters: %d\n", others);
22. return 0;
23. }
    * Prompts the user to enter text.
    * Calls yylex(), which starts lexical analysis.
    * After input processing, prints the counts of lines, spaces, tabs, and other characters.
24. **yywrap Function**:
25. int yywrap() {
26. return 1;
27. }
    * yywrap() is required by flex. It returns 1 to indicate that there is no more input to process.

**Explanation of the Code:**

1. **Header Inclusion**:
2. %{
3. #include <stdio.h>
4. %}
   * This section includes the standard I/O library (stdio.h) to use functions like printf.
5. **Lexical Rules Section**:
6. %%
7. [a-zA-Z\_][a-zA-Z0-9\_]\* { printf("Valid Identifier: %s\n", yytext); }
8. [^a-zA-Z0-9\_] { /\* Ignore other characters \*/ }
9. %%
   * [a-zA-Z\_] → The first character of a valid C/C++ identifier must be a letter (uppercase/lowercase) or an underscore (\_).
   * [a-zA-Z0-9\_]\* → The following characters can be letters, digits, or underscores.
   * yytext → This is a built-in Lex variable that holds the current matched pattern.
   * [^a-zA-Z0-9\_] → Matches any character that is *not* a valid identifier character (used here to ignore non-identifiers).
10. **Main Function**:
11. int main() {
12. printf("Enter text (Press Ctrl+D to end input on Unix/Linux or Ctrl+Z on Windows):\n");
13. yylex();
14. return 0;
15. }
    * Prompts the user to input text.
    * Calls yylex() to process the input and find valid identifiers.

**Explanation of the Code:**

1. **Header Inclusion**:
2. %{
3. #include <stdio.h>
4. %}
   * Includes the standard I/O library (stdio.h) to use functions like printf.
5. **Lexical Rules Section**:
6. %%
7. [0-9]+ { printf("Integer: %s\n", yytext); }
8. [0-9]+\\.[0-9]+ { printf("Float: %s\n", yytext); }
9. [^0-9\\.] { /\* Ignore other characters \*/ }
10. %%
    * [0-9]+ → Matches an integer (one or more digits).
    * [0-9]+\\.[0-9]+ → Matches a floating-point number (digits before and after a decimal point).
    * yytext → A built-in Lex variable that holds the current matched pattern.
    * [^0-9\\.] → Matches and ignores any character that is *not* a digit or a decimal point.
11. **Main Function**:
12. int main() {
13. printf("Enter text (Press Ctrl+D to end input on Unix/Linux or Ctrl+Z on Windows):\n");
14. yylex();
15. return 0;
16. }
    * Prompts the user to enter text.
    * Calls yylex() to process the input and identify numbers.
17. **yywrap Function**:
18. int yywrap() {

**Explanation of the Code:**

1. **Header Inclusion**:
2. %{
3. #include <stdio.h>
4. #include <string.h>
5. %}
   * Includes stdio.h for standard input/output operations.
   * Includes string.h for string comparison functions.
6. **Keyword Checking Function**:
7. const char \*keywords[] = {"int", "float", "if", "else", "while", "return", "void", "char"};
8. int is\_keyword(char \*word) {
9. for (int i = 0; i < sizeof(keywords)/sizeof(keywords[0]); i++) {
10. if (strcmp(word, keywords[i]) == 0)
11. return 1;
12. }
13. return 0;
14. }
    * Defines an array of C/C++ keywords.
    * is\_keyword() function checks if a given word is a keyword.
15. **Lexical Rules Section**:
16. %%
17. // Operators
18. [+\\-\*/=<>!&|]+ { printf("Operator: %s\n", yytext); }
19. // Separators
20. [(){}\\[\\],;] { printf("Separator: %s\n", yytext); }
21. // Identifiers and Keywords
22. [a-zA-Z\_][a-zA-Z0-9\_]\* {
23. if (is\_keyword(yytext))
24. printf("Keyword: %s\n", yytext);
25. else
26. printf("Identifier: %s\n", yytext);
27. }
28. // Ignore whitespace
29. [ \\t\\n]+ { /\* Ignore whitespace \*/ }
30. %%
    * **Operators**: Matches mathematical and logical operators (+, -, \*, /, =, <, >, !, &, |).
    * **Separators**: Matches common separators ((), {}, [], ,, ;).
    * **Identifiers & Keywords**:
      + If the token matches a keyword from the keywords[] list, it prints as a "Keyword".
      + Otherwise, it is recognized as an "Identifier".
    * **Whitespace Handling**: Skips spaces, tabs, and newlines.
31. **Main Function**:
32. int main() {
33. printf("Enter text (Press Ctrl+D to end input on Unix/Linux or Ctrl+Z on Windows):\n");
34. yylex();
35. return 0;
36. }
    * Prompts the user to enter text.
    * Calls yylex() to process the input and tokenize it.
37. **yywrap Function**:
38. int yywrap() {

**Explanation of the Code:**

1. **Header Inclusion and Variable Initialization**:
2. %{
3. #include <stdio.h>
4. int char\_count = 0, word\_count = 0, whitespace\_count = 0;
5. %}
   * Includes stdio.h for standard input/output operations.
   * Declares three counters:
     + char\_count → Counts total characters.
     + word\_count → Counts total words.
     + whitespace\_count → Counts total whitespace characters.
6. **Lexical Rules Section**:
7. %%
8. // Count characters (including whitespace and punctuation)
9. . { char\_count++; }
10. // Count whitespace characters (spaces, tabs, newlines)
11. [ \\t\\n] { whitespace\_count++; }
12. // Count words (a sequence of non-whitespace characters)
13. [a-zA-Z0-9\_]+ { word\_count++; char\_count += yyleng; }
14. %%
    * . → Matches any character and increments char\_count.
    * [ \t\n] → Matches spaces, tabs, and newlines, and increments whitespace\_count.
    * [a-zA-Z0-9\_]+ → Matches a word (a sequence of letters, digits, or underscores).
      + Increments word\_count for each word.
      + Adds the length of the word (yyleng) to char\_count to ensure accurate counting.
15. **Main Function**:
16. int main() {
17. FILE \*file = fopen("Input.txt", "r");
18. if (!file) {
19. printf("Error: Could not open Input.txt\n");
20. return 1;
21. }
22. yyin = file;
23. yylex();
24. fclose(file);
25. printf("Total Characters: %d\n", char\_count);
26. printf("Total Words: %d\n", word\_count);
27. printf("Total Whitespaces: %d\n", whitespace\_count);
28. return 0;
29. }
    * Opens "Input.txt" for reading.
    * If the file cannot be opened, prints an error message and exits.
    * Sets yyin to the file pointer so yylex() reads from the file.
    * Calls yylex() to process the file.
    * Closes the file after processing.
    * Prints the total counts of characters, words, and whitespaces.
30. **yywrap Function**:
31. int yywrap() {
32. return 1;
33. }

**Explanation of the Code:**

1. **Header Inclusion and File Pointer Declaration**:
2. %{
3. #include <stdio.h>
4. FILE \*outFile;
5. %}
   * Includes stdio.h for file handling.
   * Declares a file pointer outFile for writing the processed output.
6. **Lexical Rules Section**:
7. %%
8. // Match whitespace sequences (spaces, tabs, newlines) and replace with a single space
9. [ \\t\\n]+ { fprintf(outFile, " "); }
10. // Match and copy all other characters to the output file
11. . { fprintf(outFile, "%s", yytext); }
12. %%
    * [ \t\n]+ → Matches one or more whitespace characters (spaces, tabs, newlines) and replaces them with a single space (" ").
    * . → Matches all other characters and writes them to Output.txt without modification.
13. **Main Function**:
14. int main() {
15. FILE \*inFile = fopen("Input.txt", "r");
16. if (!inFile) {
17. printf("Error: Could not open Input.txt\n");
18. return 1;
19. }
20. outFile = fopen("Output.txt", "w");
21. if (!outFile) {
22. printf("Error: Could not open Output.txt\n");
23. fclose(inFile);
24. return 1;
25. }
27. yyin = inFile;
28. yylex();
30. fclose(inFile);
31. fclose(outFile);
32. printf("Processing complete. Check Output.txt\n");
33. return 0;
34. }
    * Opens "Input.txt" for reading.
    * If "Input.txt" cannot be opened, prints an error and exits.
    * Opens "Output.txt" for writing.
    * If "Output.txt" cannot be opened, prints an error, closes "Input.txt", and exits.
    * Sets yyin to the input file and calls yylex() to process the file.
    * Closes both files after processing.
    * Prints a message indicating successful completion.
35. **yywrap Function**:
36. int yywrap() {
37. return 1;
38. }
    * Required by flex, returns 1 to indicate the end of input.

**Explanation of the Code:**

1. **Header Inclusion and File Pointer Declaration**:
2. %{
3. #include <stdio.h>
4. FILE \*outFile;
5. %}
   * Includes stdio.h for standard input/output operations.
   * Declares a file pointer outFile for writing the processed output.
6. **Lexical Rules Section**:
7. %%
8. // Remove single-line comments (//...\\n)
9. // Matches '//' followed by any characters until a newline
10. // and does not write them to output
11. //.\* { /\* Ignore single-line comment \*/ }
12. // Remove multi-line comments (/\* ... \*/)
13. /\\\*([^\*]|\\\*+[^\*/])\*\\\*+/ { /\* Ignore multi-line comment \*/ }
14. // Match and copy all other characters to the output file
15. . { fprintf(outFile, "%s", yytext); }
16. %%
    * //.\* → Matches and removes single-line comments (starting with // and ending at a newline).
    * /\\*([^\*]|\\*+[^\*/])\*\\*+/ → Matches and removes multi-line comments (starting with /\* and ending with \*/).
    * . → Matches all other characters and writes them to out.c.
17. **Main Function**:
18. int main() {
19. FILE \*inFile = fopen("input.c", "r");
20. if (!inFile) {
21. printf("Error: Could not open input.c\n");
22. return 1;
23. }
24. outFile = fopen("out.c", "w");
25. if (!outFile) {
26. printf("Error: Could not open out.c\n");
27. fclose(inFile);
28. return 1;
29. }
31. yyin = inFile;
32. yylex();
34. fclose(inFile);
35. fclose(outFile);
36. printf("Processing complete. Check out.c\n");
37. return 0;
38. }
    * Opens "input.c" for reading.
    * Opens "out.c" for writing.
    * If a file cannot be opened, prints an error and exits.
    * Sets yyin to the input file and calls yylex() to process the file.
    * Closes both files after processing.
    * Prints a success message.
39. **yywrap Function**:
40. int yywrap() {
41. return 1;

**Explanation of the Code:**

1. **Header Inclusion and File Pointer Declaration**:
2. %{
3. #include <stdio.h>
4. FILE \*outFile;
5. %}
   * Includes stdio.h for standard input/output operations.
   * Declares a file pointer outFile for writing the extracted HTML tags.
6. **Lexical Rules Section**:
7. %%
8. // Match HTML tags and write them to output file
9. /<[^>]+>/ { fprintf(outFile, "%s\n", yytext); }
10. // Ignore all other content
11. .|\n { /\* Ignore non-tag content \*/ }
12. %%
    * /<[^>]+>/ → Matches any HTML tag (anything enclosed in < and >).
    * Writes matched tags to the output file with fprintf(outFile, "%s\n", yytext);.
    * .| \n → Matches and ignores all non-tag content (text, attributes, etc.).
13. **Main Function**:
14. int main() {
15. char inputFileName[100], outputFileName[100];
17. printf("Enter the input HTML file name: ");
18. scanf("%s", inputFileName);
20. printf("Enter the output text file name: ");
21. scanf("%s", outputFileName);
23. FILE \*inFile = fopen(inputFileName, "r");
24. if (!inFile) {
25. printf("Error: Could not open %s\n", inputFileName);
26. return 1;
27. }
29. outFile = fopen(outputFileName, "w");
30. if (!outFile) {
31. printf("Error: Could not open %s\n", outputFileName);
32. fclose(inFile);
33. return 1;
34. }
36. yyin = inFile;
37. yylex();
39. fclose(inFile);
40. fclose(outFile);
41. printf("Processing complete. Extracted tags are stored in %s\n", outputFileName);
42. return 0;
43. }
    * Prompts the user to enter an HTML input file name and an output file name.
    * Opens the input file for reading and the output file for writing.
    * If files cannot be opened, prints an error and exits.
    * Calls yylex() to process the HTML file and extract tags.
    * Closes both files after processing.