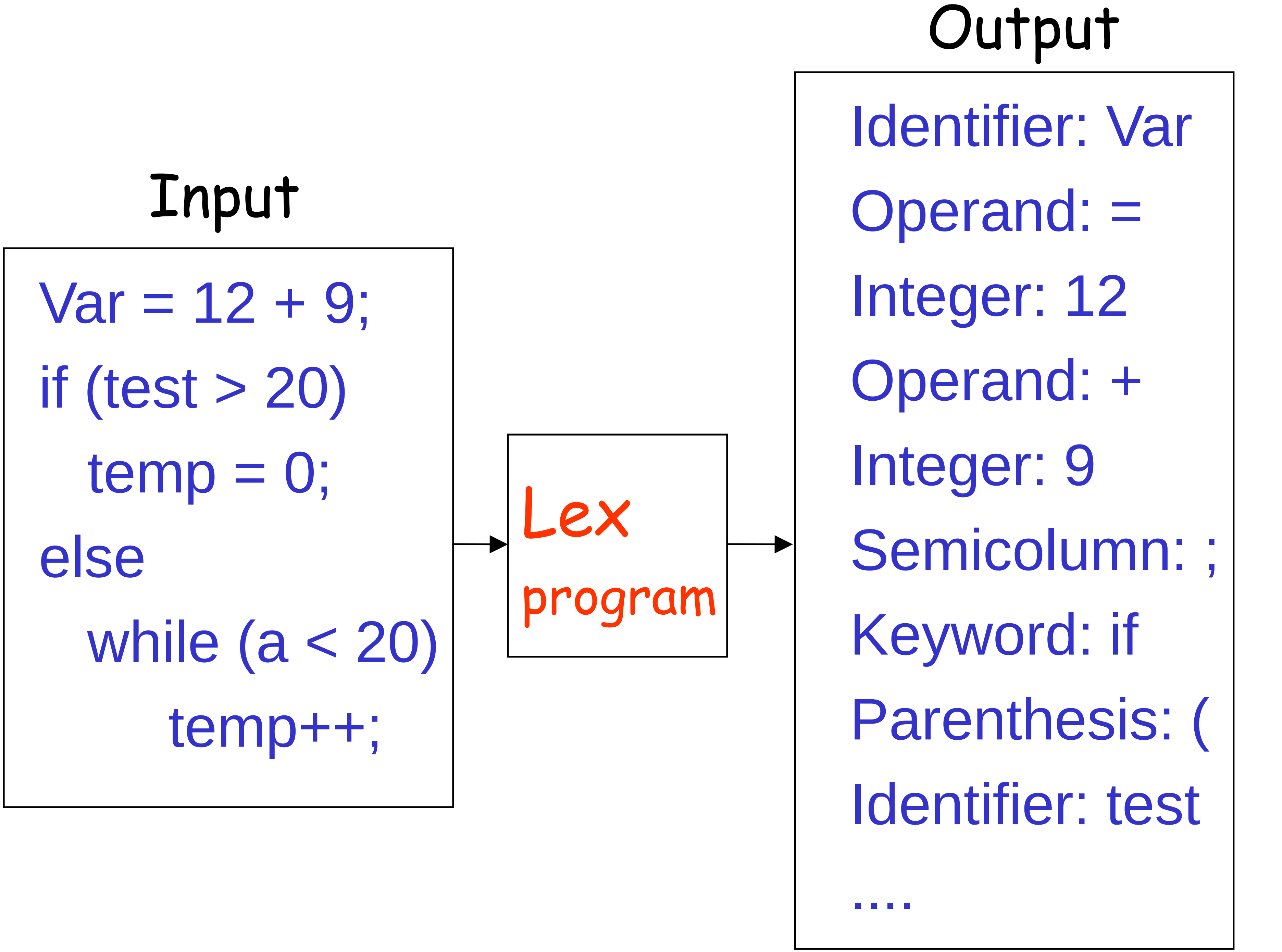
# Lex
IIT Patna
CSE Department

# Lex: a lexical analyzer

- A Lex program recognizes strings

- For each kind of string found
  the lex program takes an action



Input

```
Var = 12 + 9;
if (test > 20)
   temp = 0;
else
   while (a < 20)
       temp++;
```

Lex
program

Output

```
Identifier: Var
Operand: =
Integer: 12
Operand: +
Integer: 9
Semicolumn: ;
Keyword: if
Parenthesis: (
Identifier: test
....
```

In Lex strings are described with regular expressions

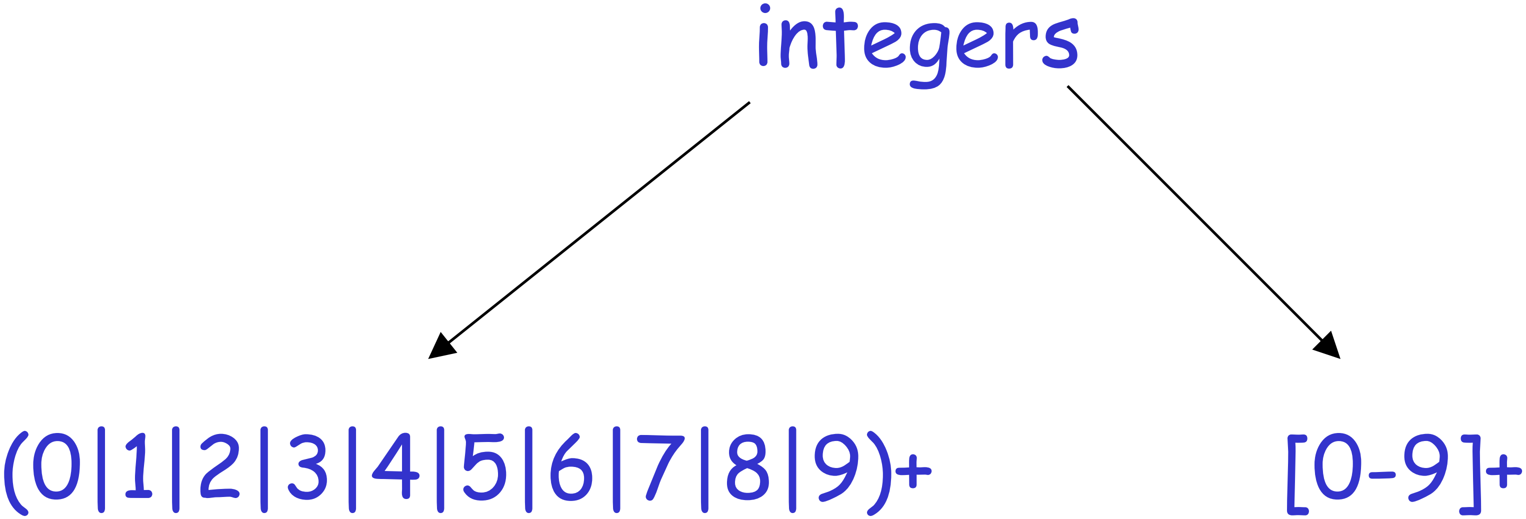
## Lex program

```
Regular expressions

    “+”
    “-”                  /* operators */
    “=”

    “if”
    “then”               /* keywords */
```

# Lex program

Regular expressions

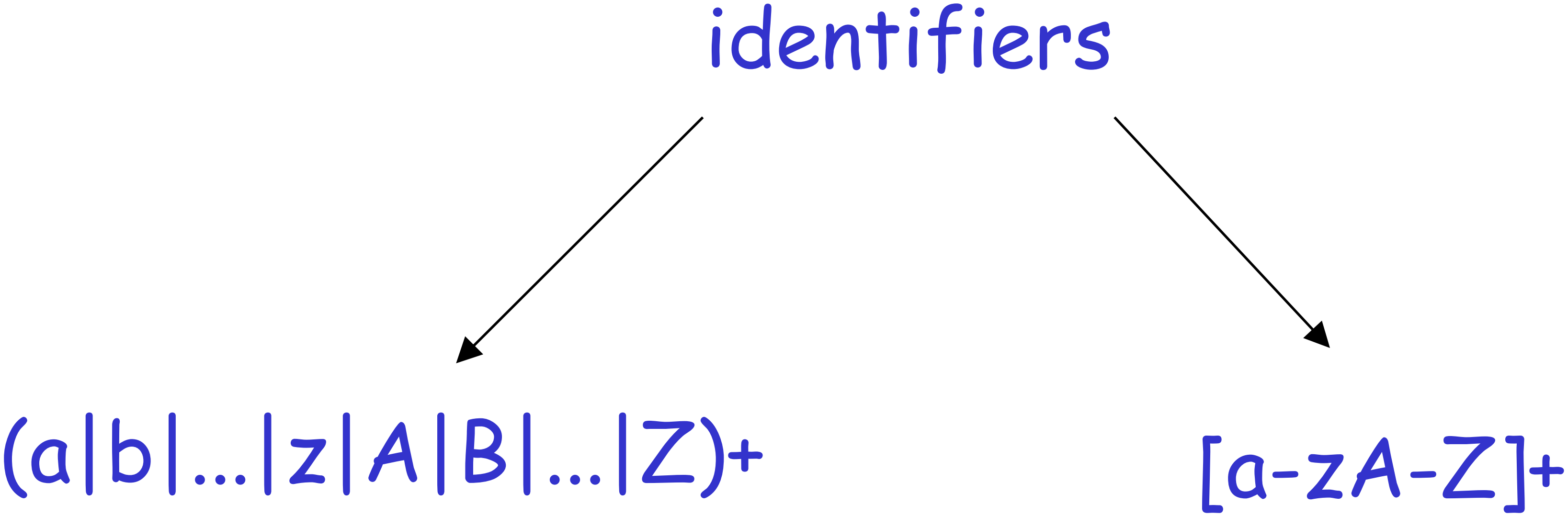
```
(0|1|2|3|4|5|6|7|8|9)+    /* integers */


(a|b|..|z|A|B|...|Z)+     /* identifiers */
```



integers

(0|1|2|3|4|5|6|7|8|9)+

[0-9]+



identifiers

(a|b|...|z|A|B|...|Z)+

[a-zA-Z]+

# Pattern Matching Primitives

| Metacharacter | Matches |
|---|---|
| . | Any character except new line |
| \n | Newline |
| * | Zero or more copies of the preceding expression |
| + | One or more copies of the preceding expression |
| ? | Zero or one copies of the preceding expression |
| ^ | Beginning of line |
| $ | End of line |
| a\|b | a or b |
| (ab)+ | One or more copies of ab (grouping) |
| "a+b" | Literal "a+b" |
| [] | Character class |

# Pattern Matching Examples

| Expression | Matches |
|---|---|
| abc | abc |
| abc* | ab, abc, abcc, abccc, ... |
| abc+ | abc, abcc, abccc, ... |
| a(bc)+ | bc, abcbc, abcbcbc, ... |
| a(bc)? | a, abc |
| [abc] | a, b, c |
| [a-z] | any letter, a through z |
| [a\-z] | a, -, z |
| [-az] | -, a, z |
| [A-Za-z0-9]+ | one or more alphanumeric characters |
| [ \t\n]+ | whitespace |
| [^ab] | anything except: a, b |
| [a^b] | a, ^, b |
| [a\|b] | a, \|, b |
| a\|b | a or b |

# LEX: Create and execute lex file

We have to first create a specification file (used to specify the tokenization rules, i.e regular expressions to represent the tokens of the language.)

Using vi editor we have to create .lex file.

To compile the .lex file give below command

```
lex filename.lex
```

It will generate lex.yy.c file

Using cc compiler to create object file:

```
cc lex.yy.c -o objectfile -ll
```

Now run the objectfile.

```
./ objectfile
```

# LEX File

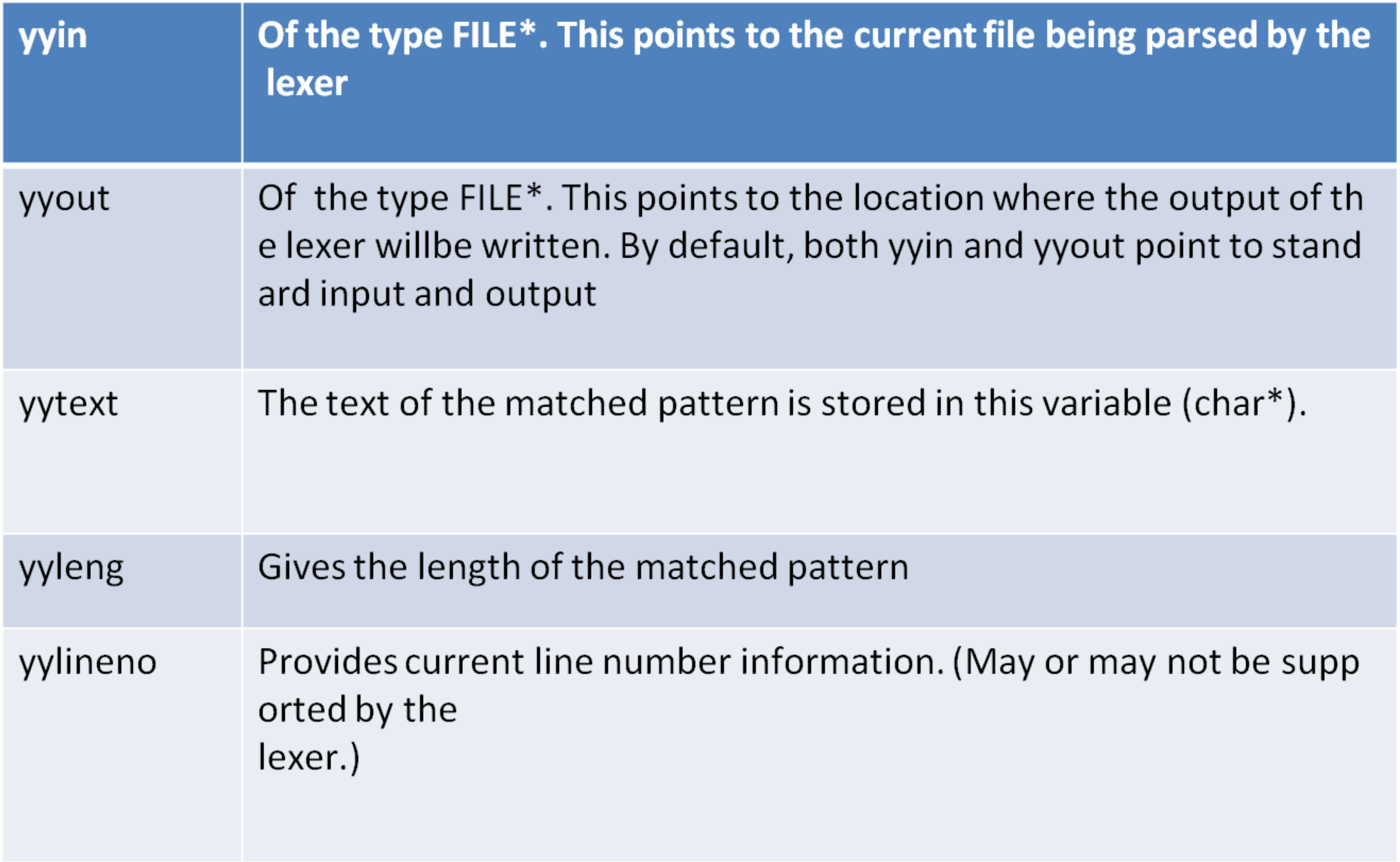
In .lex file there will be three sections

- Definitions section
- Rule section
- User Code section

# LEX Sample program

```
%{
/*
*/
int lineno=1;

%}
//Definition section
line .*₩n
%%

{line} { printf("%5d%s",lineno,yytext); }

//rule section
%%

main()
{
yylex();
return 0;
}
//User code section
```

# LEX variable



| yyin | Of the type FILE*. This points to the current file being parsed by the lexer |
|---|---|
| yyout | Of the type FILE*. This points to the location where the output of the lexer willbe written. By default, both yyin and yyout point to standard input and output |
| yytext | The text of the matched pattern is stored in this variable (char*). |
| yyleng | Gives the length of the matched pattern |
| yylineno | Provides current line number information. (May or may not be supported by the lexer.) |

# LEX function

| **yylex()** | **The function that starts the analysis. It is automatically generated by Lex.** |
|---|---|
| yywrap() | This function is called when end of file (or input) is encountered. I |
| yyless(int n) | This function can be used to push back all but first ‘n’ characters of the read token. |
| yymore | This function tells the lexer to append the next token to the current token. |

# Each regular expression
# has an associated action (in C code)

## Examples:

| Regular expression | Action |
|---|---|
| \n | linenum++; |
| [0-9]+ | prinf(“integer”); |
| [a-zA-Z]+ | printf(“identifier”); |

Default action: ECHO;

Prints the string identified to the output

# A small lex program

```
%%
[ \t\n]                 ;   /*skip spaces*/

[0-9]+                  printf(“Integer\n”);

[a-zA-Z]+               printf(“Identifier\n”);
```

Input

```
1234   test

var  566     78

9800
```

Output

```
Integer
Identifier
Identifier
Integer
Integer
Integer
```

# Another program

```
%{
int linenum = 1;
%}

%%

[ \t]           ;    /*skip spaces*/
\n              linenum++;

[0-9]+          prinf(“Integer\n”);

[a-zA-Z]+       printf(“Identifier\n”);

 .              printf(“Error in line: %d\n”,
                       linenum);
```

Input

```
1234   test
var  566     78
9800   +
temp
```

Output

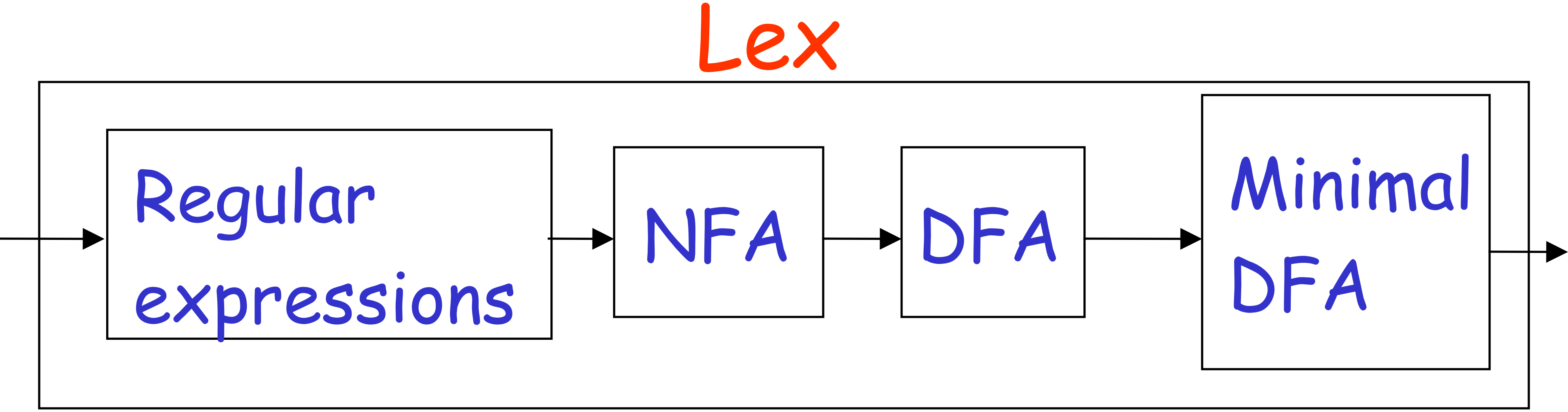
```
Integer
Identifier
Identifier
Integer
Integer
Integer
Error in line: 3
Identifier
```

# Lex matches the longest input string

Example: Regular Expressions “if”
“ifend”

| | | |
|---|---|---|
| Input: | ifend | if |
| Matches: | “ifend” | “if” |

# Internal Structure of Lex



The final states of the DFA are associated with actions