



Module 02

Prof. Sukumar
Nandi

Objectives &
Outline

Lexical Analysis
Outline

Flex Specification
Sample

Regular Expressions

Common Errors

Line Count Example

Interactive Flex

Flex-Bison Flow

Start Conditions

Summary

Module 02: CS 348: Compilers

Lexical Analyzer Generator: Flex / Lex

Prof. Sukumar Nandi

Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati

sukumar@iitg.ac.in

February 22, 2023



Module Objectives

Module 02

Prof. Sukumar
Nandi

Objectives & Outline

Lexical Analysis
Outline

Flex Specification

Sample

Regular Expressions

Common Errors

Line Count Example

Interactive Flex

Flex-Bison Flow

Start Conditions

Summary

- Understand Lexical Analysis
- Understand Flex Specification



Module Outline

Module 02

Prof. Sukumar
Nandi

Objectives &
Outline

Lexical Analysis
Outline

Flex Specification
Sample

Regular Expressions

Common Errors

Line Count Example

Interactive Flex

Flex-Bison Flow

Start Conditions

Summary

- 1 Objectives & Outline
- 2 Lexical Analysis Outline
- 3 Flex Specification
 - Sample
 - Regular Expressions
 - Common Errors
 - Line Count Example
- 4 Interactive Flex
- 5 Flex-Bison Flow
- 6 Start Conditions
- 7 Summary



Lexical Analysis Algorithm

Module 02

Prof. Sukumar
Nandi

Objectives &
Outline

Lexical Analysis
Outline

Flex Specification

Sample

Regular Expressions

Common Errors

Line Count Example

Interactive Flex

Flex-Bison Flow

Start Conditions

Summary

- RE¹ for every Token Class
- Convert Regular Expression to an NFA²
- Convert NFA to DFA³
- Lexical Action for every final state of DFA

¹ Regular Expression

² Non-deterministic Finite Automata

³ Deterministic Finite Automata



Lexical Analysis Algorithm

Module 02

Prof. Sukumar Nandi

Objectives &
Outline

Lexical Analysis
Outline

Flex Specification
Sample

Regular Expressions

Common Errors

Line Count Example

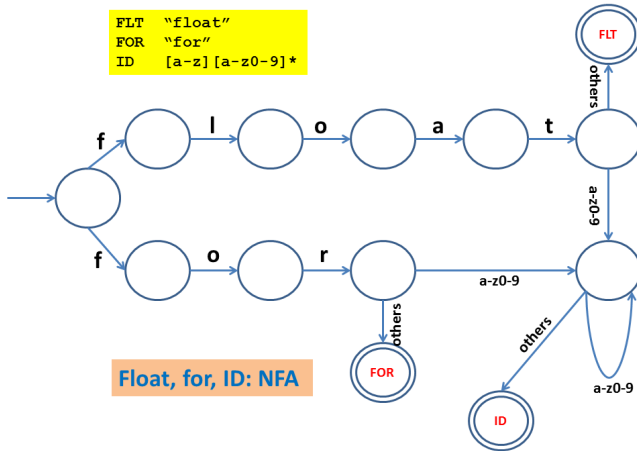
Interactive Flex

Flex-Bison Flow

Start Conditions

Summary

FLT "float"
FOR "for"
ID [a-z][a-z0-9]*



Float, for, ID: NFA

NFA Recognizer for a language having keywords "float" and "for" and identifiers starting with 'float' or 'for' (restrictive). Note that transitions on 'others' are look-ahead while all others are consumption.



Lexical Analysis Algorithm

Module 02

Prof. Sukumar Nandi

Objectives & Outline

Lexical Analysis Outline

Flex Specification

Sample

Regular Expressions

Common Errors

Line Count Example

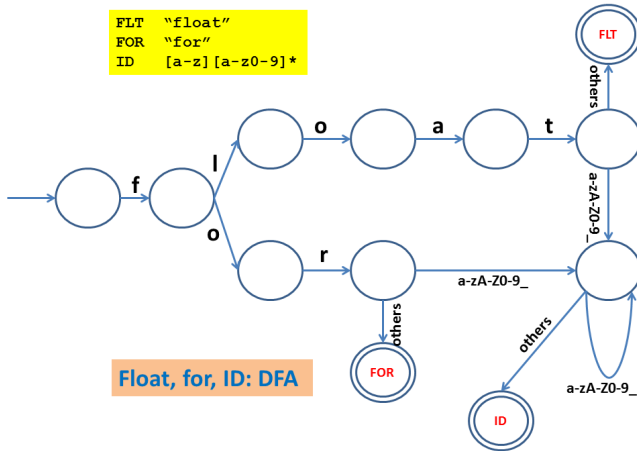
Interactive Flex

Flex-Bison Flow

Start Conditions

Summary

FLT "float"
FOR "for"
ID [a-z][a-z0-9]*



Float, for, ID: DFA

DFA Recognizer for a language having keywords "float" and "for" and identifiers starting with 'float' or 'for' (restrictive). Note that transitions on 'others' are look-ahead while all others are consumption.



Lexical Analysis Rules

Module 02

Prof. Sukumar
Nandi

Objectives &
Outline

Lexical Analysis
Outline

Flex Specification

Sample

Regular Expressions

Common Errors

Line Count Example

Interactive Flex

Flex-Bison Flow

Start Conditions

Summary

$\text{number} \rightarrow \text{digits optFrac optExp}$

$\text{digit} \rightarrow 0 \mid 1 \mid 2 \mid \dots \mid 9$

$\text{digits} \rightarrow \text{digit digit}^*$

$\text{optFrac} \rightarrow . \text{digit} \mid \epsilon$

$\text{optExp} \rightarrow (E (+ \mid - \mid \epsilon) \text{digit}) \mid \epsilon$

integer and float
constants

$\text{id} \rightarrow \text{letter} (\text{letter} \mid \text{digit})^*$

$\text{letter} \rightarrow A \mid B \mid C \dots \mid Z \mid a \mid b \mid c \dots \mid z$

$\text{digit} \rightarrow 0 \mid 1 \mid 2 \mid \dots \mid 9$

Character class



FSM for Integer and Floating Point Constants

Module 02

Prof. Sukumar
Nandi

Objectives &
Outline

Lexical Analysis
Outline

Flex Specification

Sample

Regular Expressions

Common Errors

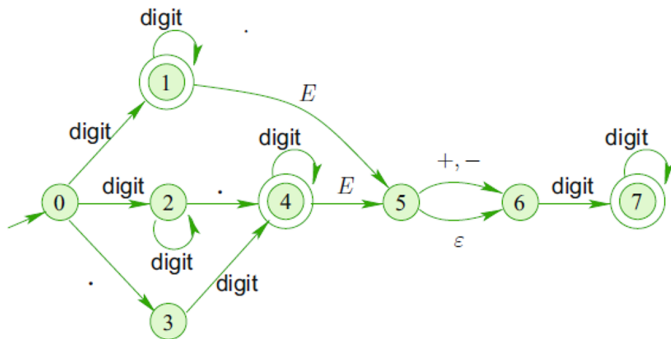
Line Count Example

Interactive Flex

Flex-Bison Flow

Start Conditions

Summary





Token Representation

Module 02

Prof. Sukumar Nandi

Objectives &
Outline

Lexical Analysis
Outline

Flex Specification
Sample

Regular Expressions
Common Errors

Line Count Example

Interactive Flex

Flex-Bison Flow

Start Conditions

Summary

Lexemes	Token Name	Attribute Value
Any ws	-	-
if	if	-
then	then	-
else	else	-
Any id	id	Pointer to ST
Any number	number	Pointer to ST
<	relop	LT
<=	relop	LE
==	relop	EQ
!=	relop	NE
>	relop	GT
>=	relop	GE



FSM for Logical Operators

Module 02

Prof. Sukumar Nandi

Objectives & Outline

Lexical Analysis Outline

Flex Specification Sample

Regular Expressions

Common Errors

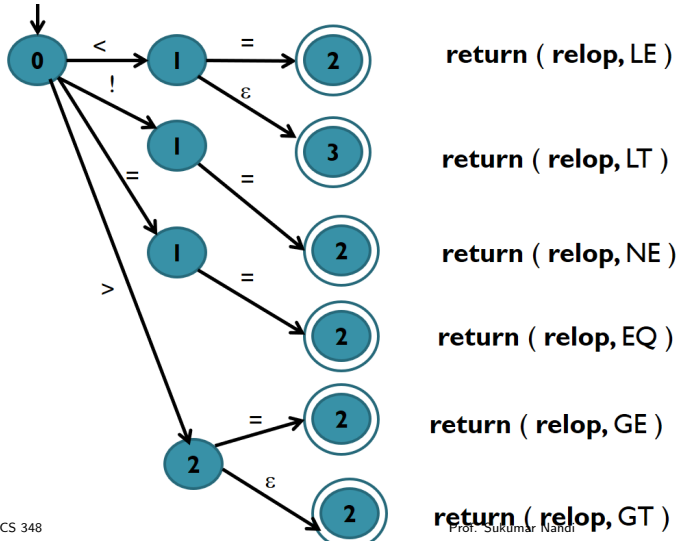
Line Count Example

Interactive Flex

Flex-Bison Flow

Start Conditions

Summary





Flex Flow

Module 02

Prof. Sukumar
Nandi

Objectives &
Outline

Lexical Analysis
Outline

Flex Specification

Sample

Regular Expressions

Common Errors

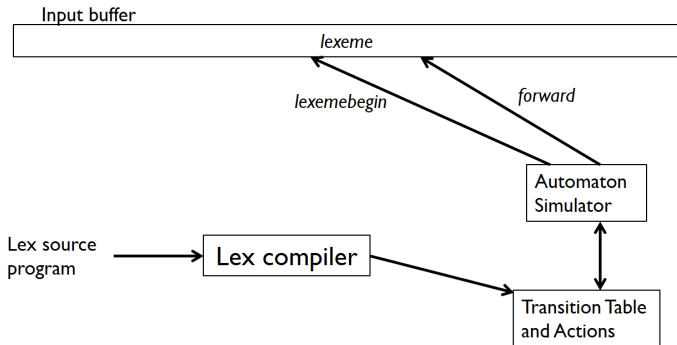
Line Count Example

Interactive Flex

Flex-Bison Flow

Start Conditions

Summary



Lex program → Transition table and actions → FA simulator



Our Sample for Flex

Module 02

Prof. Sukumar
Nandi

Objectives &
Outline

Lexical Analysis
Outline

Flex Specification
Sample

Regular Expressions

Common Errors

Line Count Example

Interactive Flex

Flex-Bison Flow

Start Conditions

Summary

- This is a simple block with declaration and expression statements
- We shall use this as a running example

```
{  
    int x;  
    int y;  
    x = 2;  
    y = 3;  
    x = 5 + y * 4;  
}
```



Structure of Flex Specs

Module 02

Prof. Sukumar
Nandi

Objectives &
Outline

Lexical Analysis
Outline

Flex Specification

Sample

Regular Expressions

Common Errors

Line Count Example

Interactive Flex

Flex-Bison Flow

Start Conditions

Summary

Declarations

%%

Translation rule

%%

Auxiliary functions



Flex Specs for our sample

Module 02

Prof. Sukumar
Nandi

Objectives &
Outline

Lexical Analysis
Outline

Flex Specification

Sample

Regular Expressions

Common Errors

Line Count Example

Interactive Flex

Flex-Bison Flow

Start Conditions

Summary

- C Declarations and definitions
- Definitions of Regular Expressions
- Definitions of Rules & Actions
- C functions

```
%{  
/* C Declarations and Definitions */  
%}  
  
/* Regular Expression Definitions */  
INT      "int"  
ID       [a-z][a-z0-9]*  
PUNC     [;]  
CONST    [0-9]+  
WS       [ \t\n]  
  
/* Definitions of Rules & Actions */  
%%  
{INT}    { printf("<KEYWORD, int>\n"); /* Keyword Rule */ }  
{ID}     { printf("<ID, %s\n", yytext); /* Identifier Rule & yytext points to lexeme */}  
"+"      { printf("<OPERATOR, +>\n"); /* Operator Rule */ }  
"*"      { printf("<OPERATOR, *>\n"); /* Operator Rule */ }  
"="      { printf("<OPERATOR, =>\n"); /* Operator Rule */ }  
"{"      { printf("<SPECIAL SYMBOL, {>\n"); /* Scope Rule */ }  
"}"      { printf("<SPECIAL SYMBOL, }>\n"); /* Scope Rule */ }  
{PUNC}   { printf("<PUNCTUATION, ;>\n"); /* Statement Rule */ }  
{CONST}  { printf("<INTEGER CONSTANT, %s>\n",yytext); /* Literal Rule */ }  
{WS}     /* White-space Rule */ ;  
  
%%  
  
/* C functions */  
main() { yylex(); /* Flex Engine */ }
```



Flex I/O for our sample

Module 02

Prof. Sukumar
Nandi

Objectives &
Outline

Lexical Analysis
Outline

Flex Specification

Sample

Regular Expressions

Common Errors

Line Count Example

Interactive Flex

Flex-Bison Flow

Start Conditions

Summary

I/P Character Stream

```
{  
    int x;  
    int y;  
    x = 2;  
    y = 3;  
    x = 5 + y * 4;  
}
```

O/P Token Stream

```
<SPECIAL SYMBOL, {>  
<KEYWORD, int> <ID, x> <PUNCTUATION, ;>  
<KEYWORD, int> <ID, y> <PUNCTUATION, ;>  
<ID, x> <OPERATOR, ==> <INTEGER CONSTANT, 2> <PUNCTUATION, ;>  
<ID, y> <OPERATOR, ==> <INTEGER CONSTANT, 3> <PUNCTUATION, ;>  
<ID, x> <OPERATOR, ==> <INTEGER CONSTANT, 5> <OPERATOR, +>  
<ID, y> <OPERATOR, *> <INTEGER CONSTANT, 4> <PUNCTUATION, ;>  
<SPECIAL SYMBOL, }>
```

- Every token is a doublet showing the token class and the specific token information
- The output is generated as one token per line. It has been rearranged here for better readability



Variables in Flex

Module 02

Prof. Sukumar
Nandi

Objectives &
Outline

Lexical Analysis
Outline

Flex Specification

Sample

Regular Expressions

Common Errors

Line Count Example

Interactive Flex

Flex-Bison Flow

Start Conditions

Summary

<code>yylex()</code>	Flex generated lexer driver
<code>yyin</code>	File pointer to Flex input
<code>yyout</code>	File pointer to Flex output
<code>yytext</code>	Pointer to Lexeme
<code>yylen</code>	Length of the Lexeme



Regular Expressions – Basic

Module 02

Prof. Sukumar
Nandi

Objectives &
Outline

Lexical Analysis
Outline

Flex Specification
Sample

Regular Expressions
Common Errors

Line Count Example

Interactive Flex

Flex-Bison Flow

Start Conditions

Summary

Expr.	Meaning
x	Character x
.	Any character except newline
[xyz]	Any characters amongst x, y or z.
[a-z]	Denotes any letter from a through z
[^0-9]	Stands for any character which is not a decimal digit, including new-line
\x	If x is an a, b, f, n, r, t, or v, then the ANSI-C interpretation of \x. Otherwise, a literal x (used to escape operators such as *)
\0	A NULL character
\num	Character with octal value num
\xnum	Character with hexadecimal value num
"string"	Match the literal string. For instance "/"* denotes the character / and then the character *, as opposed to /* denoting any number of slashes
<<EOF>>	Match the end-of-file



Regular Expressions - Operators

Module 02

Prof. Sukumar
Nandi

Objectives &
Outline

Lexical Analysis
Outline

Flex Specification
Sample

Regular Expressions
Common Errors
Line Count Example

Interactive Flex

Flex-Bison Flow

Start Conditions

Summary

Expr.	Meaning
(r)	Match an r; parentheses are used to override precedence
rs	Match the regular expression r followed by the regular expression s. This is called <i>concatenation</i>
r s	Match either an r or an s. This is called <i>alternation</i>
{ <i>abbreviation</i> }	Match the expansion of the abbreviation definition. Instead of: %% [a-zA-Z_][a-zA-Z0-9_]* return IDENTIFIER; %%
	Use id [a-zA-Z_][a-zA-Z0-9_]* %% {id} return IDENTIFIER; %%



Regular Expressions - Operators

Module 02

Prof. Sukumar
Nandi

Objectives &
Outline

Lexical Analysis
Outline

Flex Specification
Sample

Regular Expressions

Common Errors

Line Count Example

Interactive Flex

Flex-Bison Flow

Start Conditions

Summary

Expr.

Meaning

quantifiers

r^*	zero or more r 's
r^+	one or more r 's
$r^?$	zero or one r 's
$r\{[num]\}$	num times r
$r\{min,[max]\}$	Anywhere from min to max (defaulting to no bound) r 's
r/s	Match an r but only if it is followed by an s . This type of pattern is called <i>trailing context</i> .

For example: Distinguish $DO1J=1,5$ (a for loop where I runs from 1 to 5) from $DO1J=1.5$ (a definition/assignment of the floating variable $DO1J$ to 1.5) in FORTRAN. Use

$DO/[A-Z0-9]^*=[A-Z0-9]^*$

r	Match an r at the beginning of a line
$r\$$	Match an r at the end of a line



Wrong Flex Specs for our sample

- Rules for ID and INT have been swapped.
- No keyword can be tokenized as keyword now.

```
%{
/* C Declarations and Definitions */
%}

/* Regular Expression Definitions */
INT      "int"
ID       [a-z][a-z0-9]*
PUNC     [;]
CONST    [0-9]+
WS       [ \t\n]

%%

{ID}      { printf("<ID, %s>\n", yytext); /* Identifier Rule */}
{INT}     { printf("<KEYWORD, \"int\">\n"); /* Keyword Rule */ }
"+"       { printf("<OPERATOR, ++>\n"); /* Operator Rule */ }
"*"       { printf("<OPERATOR, *>\n"); /* Operator Rule */ }
"="       { printf("<OPERATOR, ==>\n"); /* Operator Rule */ }
"{"       { printf("<SPECIAL SYMBOL, {>\n"); /* Scope Rule */ }
"}"       { printf("<SPECIAL SYMBOL, }>\n"); /* Scope Rule */ }
{PUNC}    { printf("<PUNCTUATION, ;>\n"); /* Statement Rule */ }
{CONST}   { printf("<INTEGER CONSTANT, %s>\n",yytext); /* Literal Rule */ }
{WS}      /* White-space Rule */ ;
%%

main() {
    yylex(); /* Flex Engine */
}
```



Wrong Flex I/O for our sample

Module 02

Prof. Sukumar
Nandi

Objectives &
Outline

Lexical Analysis
Outline

Flex Specification

Sample

Regular Expressions

Common Errors

Line Count Example

Interactive Flex

Flex-Bison Flow

Start Conditions

Summary

I/P Character Stream

```
{  
    int x;  
    int y;  
    x = 2;  
    y = 3;  
    x = 5 + y * 4;  
}
```

O/P Token Stream

```
<SPECIAL SYMBOL, {>  
<ID, int> <ID, x> <PUNCTUATION, ;>  
<ID, int> <ID, y> <PUNCTUATION, ;>  
<ID, x> <OPERATOR, ==> <INTEGER CONSTANT, 2> <PUNCTUATION, ;>  
<ID, y> <OPERATOR, ==> <INTEGER CONSTANT, 3> <PUNCTUATION, ;>  
<ID, x> <OPERATOR, ==> <INTEGER CONSTANT, 5> <OPERATOR, +>  
<ID, y> <OPERATOR, *> <INTEGER CONSTANT, 4> <PUNCTUATION, ;>  
<SPECIAL SYMBOL, }>
```

- Both int's have been taken as ID!



Count Number of Lines – Flex Specs

Module 02

Prof. Sukumar
Nandi

Objectives &
Outline

Lexical Analysis
Outline

Flex Specification

Sample

Regular Expressions

Common Errors

Line Count Example

Interactive Flex

Flex-Bison Flow

Start Conditions

Summary

```
/* C Declarations and definitions */
%{
    int charCount = 0, wordCount = 0, lineCount = 0;
}%

/* Definitions of Regular Expressions */
word  [^ \t\n]+          /* A word is a seq. of char. w/o a white space */

/* Definitions of Rules \& Actions */
%%
{word}  { wordCount++; charCount += yyleng; /* Any character other than white space */ }
[\n]    { charCount++; lineCount++;        /* newline character */ }
.       { charCount++;                      /* space and tab characters */ }
%%

/* C functions */
main() {
    yylex();
    printf("Characters: %d Words: %d Lines %d\n",charCount, wordCount, lineCount);
}
```



Count Number of Lines – lex.yy.c

Module 02

Prof. Sukumar
Nandi

Objectives &
Outline

Lexical Analysis
Outline

Flex Specification

Sample

Regular Expressions

Common Errors

Line Count Example

Interactive Flex

Flex-Bison Flow

Start Conditions

Summary

```
char *yytext;
int charCount = 0, wordCount = 0, lineCount = 0; /* C Declarations and definitions */
/* Definitions of Regular Expressions & Definitions of Rules & Actions */
int yylex (void) { /** The main scanner function which does all the work. */
// ...
    if ( ! (yy_start) ) (yy_start) = 1;    /* first start state */
    if ( ! yyin ) yyin = stdin;
    if ( ! yyout ) yyout = stdout;
// ...
    while ( 1 ) {          /* loops until end-of-file is reached */
// ..
        yy_current_state = (yy_start);
yy_match: // ...
yy_find_action: // ...
do_action:
    switch ( yy_act ) { /* beginning of action switch */
        case 0: /* must back up */ // ...
        case 1: { wordCount++; charCount += yyleng; } YY_BREAK
        case 2: { charCount++; lineCount++; } YY_BREAK
        case 3: { charCount++; } YY_BREAK
        case 4: ECHO; YY_BREAK
        case YY_STATE_EOF(INITIAL): yyterminate();
        case YY_END_OF_BUFFER:
        default: YY_FATAL_ERROR("fatal flex scanner internal error--no action found" );
    } /* end of action switch */
} /* end of scanning one token */
} /* end of yylex */
main() { /* C functions */
    yylex();
    printf("Characters: %d Words: %d Lines %d\n",charCount, wordCount, lineCount);
}
```



Modes of Flex Operations

Module 02

Prof. Sukumar
Nandi

Objectives &
Outline

Lexical Analysis
Outline

Flex Specification
Sample

Regular Expressions

Common Errors

Line Count Example

Interactive Flex

Flex-Bison Flow

Start Conditions

Summary

Flex can be used in two modes:

- **Non-interactive:** Call `yyllex()` only once. It keeps spitting the tokens till the end-of-file is reached. So the actions on the rules do not have `return` and falls through in the `switch` in `lex.yy.c`.
This is convenient for small specifications. But does not work well for large programs because:
 - Long stream of spitted tokens may need a further tokenization while processed by the parser
 - At times tokenization itself, or at least the information update in the actions for the rules, may need information from the parser (like pointer to the correctly scoped symbol table)
- **Interactive:** Repeatedly call `yyllex()`. Every call returns one token (after taking the actions for the rule matched) that is consumed by the parser and `yyllex()` is again called for the next token. This lets parser and lexer work hand-in-hand and also eases information interchange between the two.



Flex Specs (non-interactive) for our sample

Module 02

Prof. Sukumar
Nandi

Objectives &
Outline

Lexical Analysis
Outline

Flex Specification

Sample

Regular Expressions

Common Errors

Line Count Example

Interactive Flex

Flex-Bison Flow

Start Conditions

Summary

- C Declarations and definitions
- Definitions of Regular Expressions
- Definitions of Rules & Actions
- C functions

```
%{  
/* C Declarations and Definitions */  
%}  
/* Regular Expression Definitions */  
INT      "int"  
ID       [a-z][a-z0-9]*  
PUNC     [;]  
CONST    [0-9]+  
WS       [ \t\n]  
/* Definitions of Rules & Actions */  
%%  
{INT}    { printf("<KEYWORD, int>\n"); /* Keyword Rule */ }  
{ID}     { printf("<ID, %s>\n", yytext); /* Identifier Rule */ }  
"+"      { printf("<OPERATOR, +>\n"); /* Operator Rule */ }  
"*"      { printf("<OPERATOR, *>\n"); /* Operator Rule */ }  
"="      { printf("<OPERATOR, =>\n"); /* Operator Rule */ }  
"{"      { printf("<SPECIAL SYMBOL, {>\n"); /* Scope Rule */ }  
"}"      { printf("<SPECIAL SYMBOL, }>\n"); /* Scope Rule */ }  
{PUNC}   { printf("<PUNCTUATION, ;>\n"); /* Statement Rule */ }  
{CONST}  { printf("<INTEGER CONSTANT, %s>\n",yytext); /* Literal Rule */ }  
{WS}     /* White-space Rule */ ;  
%%  
/* C functions */  
main() { yylex(); /* Flex Engine */ }
```



Flex Specs (interactive) for our sample

Module 02

Prof. Sukumar
Nandi

Objectives &
Outline

Lexical Analysis
Outline

Flex Specification

Sample

Regular Expressions

Common Errors

Line Count Example

Interactive Flex

Flex-Bison Flow

Start Conditions

Summary

```
%{
#define INT 10
#define ID 11
#define PLUS 12
#define MULT 13
#define ASSIGN 14
#define LBRACE 15
#define RBACE 16
#define CONST 17
#define SEMICOLON 18
}%
```

```
INT "int"
ID [a-z][a-z0-9]*
PUNC [;]
CONST [0-9]+
WS [ \t\n]
```

```
%%
{INT} { return INT; }
{ID} { return ID; }
"+" { return PLUS; }
"*" { return MULT; }
"=" { return ASSIGN; }
{" { return LBRACE; }
"}" { return RBACE; }
{PUNC} { return SEMICOLON; }
{CONST} { return CONST; }
{WS} { /* Ignore
        whitespace */ }
%%
```

```
main() { int token;
        while (token = yylex()) {
            switch (token) {
                case INT: printf("<KEYWORD, %d, %s>\n",
                                token, yytext); break;
                case ID: printf("<IDENTIFIER, %d, %s>\n",
                                token, yytext); break;
                case PLUS: printf("<OPERATOR, %d, %s>\n",
                                token, yytext); break;
                case MULT: printf("<OPERATOR, %d, %s>\n",
                                token, yytext); break;
                case ASSIGN: printf("<OPERATOR, %d, %s>\n",
                                token, yytext); break;
                case LBRACE: printf("<SPECIAL SYMBOL, %d, %s>\n",
                                token, yytext); break;
                case RBACE: printf("<SPECIAL SYMBOL, %d, %s>\n",
                                token, yytext); break;
                case SEMICOLON: printf("<PUNCTUATION, %d, %s>\n",
                                token, yytext); break;
                case CONST: printf("<INTEGER CONSTANT, %d, %s>\n",
                                token, yytext); break;
            }
        }
    }
```

- Input is taken from stdin. It can be changed by opening the file in main() and setting the file pointer to yyin.
- When the lexer will be integrated with the YACC generated parser, the yyparse() therein will call yylex() and the main() will call yyparse().



Flex I/O (interactive) for our sample

Module 02

Prof. Sukumar
Nandi

Objectives &
Outline

Lexical Analysis
Outline

Flex Specification

Sample

Regular Expressions

Common Errors

Line Count Example

Interactive Flex

Flex-Bison Flow

Start Conditions

Summary

I/P Character Stream

```
{  
    int x;  
    int y;  
    x = 2;  
    y = 3;  
    x = 5 + y * 4;  
}
```

#define	INT	10
#define	ID	11
#define	PLUS	12
#define	MULT	13
#define	ASSIGN	14
#define	LBRACE	15
#define	RBRACE	16
#define	CONST	17
#define	SEMICOLON	18

O/P Token Stream

```
<SPECIAL SYMBOL, 15, {>  
<KEYWORD, 10, int>  
<IDENTIFIER, 11, x>  
<PUNCTUATION, 18, ;>  
<KEYWORD, 10, int>  
<IDENTIFIER, 11, y>  
<PUNCTUATION, 18, ;>  
<IDENTIFIER, 11, x>  
<OPERATOR, 14, =>  
<INTEGER CONSTANT, 17, 2>  
<PUNCTUATION, 18, ;>  
<IDENTIFIER, 11, y>  
<OPERATOR, 14, =>  
<INTEGER CONSTANT, 17, 3>  
<PUNCTUATION, 18, ;>  
<IDENTIFIER, 11, x>  
<OPERATOR, 14, =>  
<INTEGER CONSTANT, 17, 5>  
<OPERATOR, 12, +>  
<IDENTIFIER, 11, y>  
<OPERATOR, 13, *>  
<INTEGER CONSTANT, 17, 4>  
<PUNCTUATION, 18, ;>  
<SPECIAL SYMBOL, 16, }>
```

- Every token is a triplet showing the token class, token manifest constant and the specific token information.



Managing Symbol Table

Module 02

Prof. Sukumar
Nandi

Objectives &
Outline

Lexical Analysis
Outline

Flex Specification

Sample

Regular Expressions

Common Errors

Line Count Example

Interactive Flex

Flex-Bison Flow

Start Conditions

Summary

```
%{  
    struct symbol {  
        char *name;  
        struct ref *reflist;  
    };  
    struct ref {  
        struct ref *next;  
        char *filename;  
        int flags;  
        int lineno;  
    };  
  
    #define NHASH 100  
    struct symbol symtab[NHASH];  
    struct symbol *lookup(char *);  
    void addref(int, char*, char*, int);  
%}
```



First Flex Program

Module 02

Prof. Sukumar
Nandi

Objectives &
Outline

Lexical Analysis
Outline

Flex Specification
Sample

Regular Expressions

Common Errors

Line Count Example

Interactive Flex

Flex-Bison Flow

Start Conditions

Summary

```
$ flex myLex.l
$ cc lex.yy.c -ll
$ ./a.out
...
$
```

Check the flex library name in your system. You may need:

```
$ flex myLex.l
$ cc lex.yy.c -lfl
$ ./a.out
...
$
```



Flex-Bison Flow

Module 02

Prof. Sukumar
Nandi

Objectives &
Outline

Lexical Analysis
Outline

Flex Specification

Sample

Regular Expressions

Common Errors

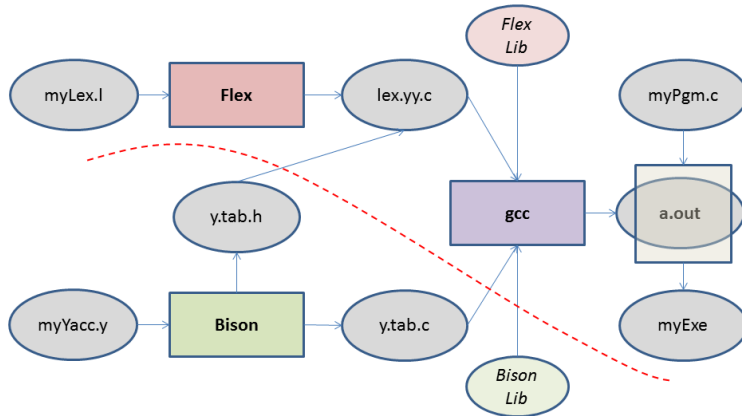
Line Count Example

Interactive Flex

Flex-Bison Flow

Start Conditions

Summary





Start Condition in Flex

Module 02

Prof. Sukumar
Nandi

Objectives &
Outline

Lexical Analysis
Outline

Flex Specification
Sample

Regular Expressions

Common Errors

Line Count Example

Interactive Flex

Flex-Bison Flow

Start Conditions

Summary

Flex provides a mechanism for conditionally activating rules. Any rule whose pattern is prefixed with `<sc>` will only be active when the scanner is in the start condition named `sc`. For example,

```
<STRING>[~"]*           { /* eat up the string body ... */  
                           ...  
                           }
```

will be active only when the scanner is in the `STRING` start condition, and

```
<INITIAL,STRING,QUOTE>\. { /* handle an escape ... */  
                           ...  
                           }
```

will be active only when the current start condition is either `INITIAL`, `STRING`, or `QUOTE`.

Source: https://ftp.gnu.org/old-gnu/Manuals/flex-2.5.4/html_node/flex_11.html



Start Condition in Flex - Specs

Module 02

Prof. Sukumar
Nandi

Objectives &
Outline

Lexical Analysis
Outline

Flex Specification
Sample

Regular Expressions

Common Errors

Line Count Example

Interactive Flex

Flex-Bison Flow

Start Conditions

Summary

- *Declaration*: Declared in the definitions section of the input
 - *BEGIN Action*: A start condition is activated using the BEGIN action. Until the next BEGIN action is executed, rules with the given start condition will be active and rules with other start conditions will be inactive
 - *Inclusive Start Conditions*: Use unindented lines beginning with '%s' followed by a list of names. If the start condition is inclusive, then rules with no start conditions at all will also be active
 - *Exclusive Start Conditions*: Use unindented lines beginning with '%x' followed by a list of names. If it is exclusive, then only rules qualified with the start condition will be active
- A set of rules contingent on the same exclusive start condition describe a scanner which is independent of any of the other rules in the flex input. Because of this, exclusive start conditions make it easy to specify mini-scanners which scan portions of the input that are syntactically different from the rest (for example, comments)

Source: https://ftp.gnu.org/old-gnu/Manuals/flex-2.5.4/html_node/flex_11.html



Start Condition in Flex - Example

Module 02

Prof. Sukumar
Nandi

Objectives &
Outline

Lexical Analysis
Outline

Flex Specification
Sample

Regular Expressions

Common Errors

Line Count Example

Interactive Flex

Flex-Bison Flow

Start Conditions

Summary

The set of rules:

```
%s example
%%
<example>foo    do_something();
bar             something_else();
```

is equivalent to

```
%x example
%%
<example>foo    do_something();
<INITIAL,example>bar    something_else();
```

Without the `<INITIAL,example>` qualifier, the bar pattern in the second example wouldn't be active (that is, couldn't match) when in start condition example. If we just used `<example>` to qualify bar, though, then it would only be active in example and not in INITIAL, while in the first example it's active in both, because in the first example the example start condition is an inclusive (`%s`) start condition.

Source: https://ftp.gnu.org/old-gnu/Manuals/flex-2.5.4/html_node/flex_11.html



Handling Comments

Module 02

Prof. Sukumar
Nandi

Objectives &
Outline

Lexical Analysis
Outline

Flex Specification
Sample

Regular Expressions

Common Errors

Line Count Example

Interactive Flex

Flex-Bison Flow

Start Conditions

Summary

```
%x comment
```

```
%%
```

```
int line_num = 1;
```

```
"/*
```

```
BEGIN(comment);
```

```
<comment>[~*\n]*
```

```
/* eat anything that's not a '*' */
```

```
<comment>"*"+[~*/\n]*
```

```
/* eat up '*'s not followed by '/'s */
```

```
<comment>\n
```

```
++line_num;
```

```
<comment>"*"+"/"
```

```
BEGIN(INITIAL);
```

Source: https://ftp.gnu.org/old-gnu/Manuals/flex-2.5.4/html_node/flex_11.html



Module Summary

Module 02

Prof. Sukumar
Nandi

Objectives &
Outline

Lexical Analysis
Outline

Flex Specification

Sample

Regular Expressions

Common Errors

Line Count Example

Interactive Flex

Flex-Bison Flow

Start Conditions

Summary

- Lexical Analysis process is introduced
- Flex specification for Lexical Analyzer generation is discussed in depth
- Flow of Flex and Bison explained
- Special Flex feature of Start Condition discussed