# Lexical Analyzer Generator

Lex(Flex in recent implementation)

# What is Lex?

- The main job of a *lexical analyzer (scanner)* is to break up an input stream into *tokens*(**tokenize** input streams).
- *Ex :-*

```
a = b + c * d;
```

ID ASSIGN ID PLUS ID MULT ID SEMI

- Lex is an utility to help you rapidly generate your scanners

# Structure of Lex Program

- Lex source is separated into three sections by %% delimiters
- The general format of Lex source is

```
{definitions}
%%                                  (required)
{transition rules}
%%
                                    (optional)
{user Code}
```

-

```
%%
```

## Definitions

- Declarations of ordinary C variables ,constants and Libraries.

```
 %{
#include <math.h>
 #include <stdio.h>
   #include <stdlib.h>
%}
```

- flex definitions :-   name  definition

  Digit    [0-9]    (Regular Definition)

# Operators

`" \ [ ] ^ - ? . * + | ( ) $ / { } % < >`

- If they are to be used as text characters, an escape should be used

  `\$ = "$"`

  `\\ = "\"`

- Every character but *blank*, *tab* (`\t`), *newline* (`\n`) and the list above is always a text character

## Translation Rules

- The form of rules are:

```
Pattern                    { action }
```

The actions are C/C++ code.

```
[0-9]+          { return(Integer); } // RE
{DIGIT}+        { return(Integer); } // RD
```

# User Subroutines Section

- You can use your Lex routines in the same ways you use routines in other programming languages (Create functions, identifiers) .
- The section where main() is placed

```
%{
   void print(String x);
%}
%%
{letter}+    print(“ID”);
%%
main() {
   yylex();
}
void print(String x) {
   printf(x);
}
```

# An Overview of Lex

$$\begin{tikzcd}
\text{Lex source program} \arrow[r] & \boxed{\text{Lex compiler}} \arrow[r] & \text{lex.yy.c} \\
\text{lex.yy.c} \arrow[r] & \boxed{\text{C compiler}} \arrow[r] & \text{a.out} \\
\text{input} \arrow[r] & \boxed{\text{a.out}} \arrow[r] & \text{tokens}
\end{tikzcd}$$

# Lex Predefined Variables

- yytext -- a string containing the lexeme
- yyleng -- the length of the lexeme
- yyin -- the input stream pointer
  - the default input of default main() is **stdin**
- yyout -- the output stream pointer
  - the default output of default main() is **stdout**.

## Lex Library Routines

- yylex()
  - The default main() contains a call of yylex()
- yymore()
  - return the next token
- yyless(n)
  - retain the first n characters in yytext
- yywarp()
  - is called whenever Lex reaches an end-of-file
  - The default yywarp() always returns 1

# Review of Lex Predefined Variables

| Name | Function |
|---|---|
| `char *yytext` | pointer to matched string |
| `int yyleng` | length of matched string |
| `FILE *yyin` | input stream pointer |
| `FILE *yyout` | output stream pointer |
| `int yylex(void)` | call to invoke lexer, returns token |
| `char* yymore(void)` | return the next token |
| `int yyless(int n)` | retain the first n characters in yytext |
| `int yywrap(void)` | wrapup, return 1 if done, 0 if not done |
| `ECHO` | write matched string |
| `REJECT` | go to the next alternative rule |
| `INITAL` | initial start condition |
| `BEGIN` | condition switch start condition |

## Installation & Usage

# Installation

1) Download the Windows version of FLEX

2) Download a C/C++ Compiler DevCPP or Code::Blocks

3) Install all . It's recommended to install in folders WITHOUT spaces in their names.
I use '**C:\GNUWin32**' for FLEX and '**C:\** DevCPP'

5) Now add the BIN folders of both folders into your PATH variable. Incase you don't know how to go about this, see how to do it on Windows XP ,Windows Vista and Windows 7. You will have to add '**; C:\GNUWin32\bin;C:\Dev-Cpp**' to the end of your PATH

6) Open up a CMD prompt and type in the following

```
C:\flex --version
        flex version 2.5.4
C:\>gcc --version
      gcc (GCC) 3.4.2 (mingw-special)
      Copyright (C) 2004 Free Software Foundation, Inc.
      This is free software; see the source for copying conditions.
   There is NO warranty; not even for MERCHANTABILITY
   or FITNESS FOR A PARTICULAR PURPOSE.
```

## Usage

- First Go to Directory Contains Files
- To run Lex on a source file, type

  ```
  flex (lex source file.l)
  ```
- It produces a file named lex.yy.c which is a C program for the lexical analyzer.
- To compile lex.yy.c, type

  ```
  gcc lex.yy.c
  ```
- To run the lexical analyzer program, type

  ```
  a.exe < input file > output file
  ```

## Examples

- Ex1

```
%{
int numChars = 0, numWords = 0, numLines = 0;
%}
%%
\n              {numLines++; numChars++;}
[^ \t\n]+       {numWords++; numChars += yyleng;}
.               {numChars++;}
%%
int main() {
yylex();
printf("%d\t%d\t%d\n", numChars, numWords, numLines);
}
    int yywrap(){
            return 1;
            }
```

## Example on Decaf

- Definitions Section

```
%{
/* need this for the call to atof() below */
#include <math.h>
 #include <stdio.h>
   #include <stdlib.h>
%}
DIGIT                                      [0-9]
LITTER                                     [a-zA-Z]
C                                         [//]
S                                         [ ]
L                                         [\n]
ID                                        {LITTER}({LITTER}|{DIGIT}|_)*
double                                   {DIGIT}+(\.{DIGIT}+)?([Ee][+-]?{DIGIT}+)?
hex                                       (0[xX])({DIGIT}|[a-fA-F])*
String                                    (\"({LITTER}|{S})*\")
Sym                                       [~!@#$%^&*()_+|><"""{}.,]
Comment                                  (({LITTER}|{DIGIT}|{Sym}|{S}|{L})*)
Comment2                                 ({C}+({LITTER}|{DIGIT}|{Sym}|{S})*{L})
%%
```

## Example on Decaf

- Rule Section

```
{DIGIT}+                          {
                                   printf( "An integer: %s (%d)\n", yytext,atoi( yytext ) );

                                   }


{double}                           {
                                            printf( "A double: %s (%g)\n", yytext,
                                                        atof( yytext ) );
                                   }


{hex}                               {
                                            printf( "A Hexadecimal: %s \n", yytext );
                                    }


{String}                           {
                                            printf( "A String: %s \n", yytext );
                       }
"/*"{Comment}+"*/"                             /* eat up one-line comments */

{Comment2}                                    /* eat up one-line comments */
```

## Example on Decaf

- Rule Section Cont~

```
void|int|double|bool|string|class|interface|null|this|extends|implements|for|while|if|else|return|break
|new|NewArray|Print|ReadInteger|ReadLine|true|false
                                        {    printf( "A keyword: %s\n", yytext );    }

{ID}                                    {    printf( "An identifier: %s\n", yytext );  }

"+"|"-"|"*"|"/"|"="|"=="|"<"|"<="|">"|">="|"!="|"&&"|"||"|"!"|";"|","|"."|"["|"]"|"("|")"|"{"|"}"
                                        {  printf( "An operator: %s\n", yytext );   }

"{"[^}\n]*"}"     /* eat up one-line comments */

[ \t\n]+          /* eat up whitespace */


.            {printf( "Unrecognized character: %s\n", yytext ); }

%%
```

## Example on Decaf

- User Code Section

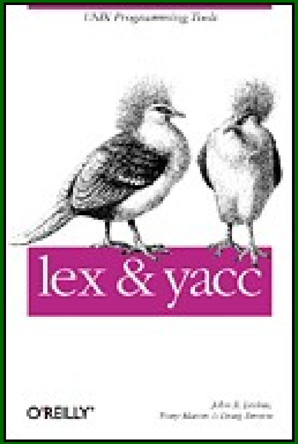
```
main( argc, argv )
int argc;
char **argv;
   {
   ++argv, --argc;   /* skip over program name */
   if ( argc > 0 )
         yyin = fopen( argv[0], "r" );
   else
         yyin = stdin;

   yylex();
   }

int yywrap(){
            return 1;
            }
```

## Run The Decaf Example

Run It With Me ☺

# Reference Books

- lex & yacc 2nd_edition John R. Levine, Tony Mason, Doug Brown, O'reilly



- Mastering Regular Expressions
  - by Jeffrey E.F. Friedl
  - O'Reilly
  - ISBN: 1-56592-257-3