# CS 4850 Fall 2015 Flex Assignment

## Objectives

* Learn to use a state machine generator lex or flex
  + Regular expressions in, C code out
* Learn to write C programs in more than one file
  + include header files only, to interface with lex.yy.c
* Set up way to debug what the lexical analyzer is returning to the higher level parser
* Learn to use both yytext and yyval

## Basic Plan

Use the program Flex to generate a C file from your grammar (\*.l) file with a state machine which parses regular expressions and executes your C code for each match. “lex.yy.c”

Write a C program which is compiled along with the output C file (lex.yy.c) from Flex which tests the parser.

DO NOT INCLUDE YOUR C FILE IN lex.yy.c, or the other way around, you must compile both files and link them into an executable.

Learn how to use .h header files with externals to have more than one file in your program and what is defined where using flex/bison.

<http://flex.sourceforge.net/manual/>

Your test program will take the place of the file generated by Bison/Yacc which you will use later.

Your program will loop

Call yylex() which will return **an integer** which should be a Token constant **#define**d by you

You will print a string version of the Token and the value lex matched

So let’s say you have **#define** INTEGER 23

The flex below will, if it reads “156”

[0-9]+ yylval.integer = atoi(yytext); return INTEGER;

return 23

With the string 156 stored in yytext and integer 156 stored in yylval.integer

Your program will use the 23 to look up the string “INTEGER” in an array of strings or part of the switch statement

And print: Token INTEGER, Value 156, Text >156<

Where the **Value is from yyval** and Text from yytext the >< are to show start and stop of the string

“ Text >%s< “, yytext

If the return token is string or symbol save it in an array so you have a list of all the strings and symbols to print at the end. Passing strings between flex and bison has been the hardest problem in past semesters so solve that problem now. Pay attention to memory used for each string.

When flex returns EOF print the following

A list of all the Strings

Another list of all the Symbols

## Details

You should have a grammar that returns a different token for each of the following

Negative or positive

Float (examples -123.0 or 123.0 or 0.123E-3) no +123.0, no 123.0F f

Integer (examples -123 or 123) no +, no 123L or l

The binary operators with one token (BIOPERATOR) yyval.string must be character array big enough that you can strcpy into.

+ addition, - subtraction, \* multiplication, / division, % modulus, ^ raise to power (+, -, \*, /, %, ^)

< less than, <=, >, >=, == equal to, != not equal to

= assignment

AND, OR, NAND, NOR

Unary operators (one token UNOPERATOR)

! Boolean not

Strings

anything enclosed in “ “ do not return the enclosing “ “ and respect \” as not end of string

Symbols

Names that follow C rules for names

Skip anything else

### yytext vs yyval

You use both.

Yytext will have the string lex matched as in the example above. flex does this.

To use yyval you will need a switch on the return value of yylex() which prints the token’s string and the value from the correct field of yyval.

Bison requires you to use yyval which is a struct defined as well as the TOKEN values at the top of bison with a syntax like:

%token <floatval> FLOAT

%token <intval> INTEGER

Bison generates into y.tab.h #defines for FLOAT and INTEGER as well as the struct yyval{ int intval; double floatval; }

In lex your grammer would be

[0-9]+ yyval.intval = atoi(yytext); return INTEGER;

**You must define yyval and use it in your flex,** this is in a header file included in all your C files.

**You write** this header file for this test program, **later Bison** will write it.

## Rules

Define your token constants and yyval in a header file named

y.tab.h

which is included at the top of your program source file and lex.yy.c

This is the file that Bison generates if you use the –d switch.

Your program must compile on the CS department’s Linux machines C-208 with no –l switches in the compile command.

You can upload and test on your CS account.

## Submit

Your .l flex grammar/input file

Your .c driver/test file

A makefile

DO NOT SUBMIT the file generated by flex