CASE WESTERN RESERVE UNIVERSITY
Case School of Engineering
Department of Electrical Engineering and Computer Science
EECS 337 Systems Programming (Compiler Design)
Fall 2010
Assignment #6 Due: October 5, 2010

Part 1: Reading and Exercises
Compilers - Principles, Techniques, & Tools, 2nd Edition, Sections 4.5, 4.6, 4.9
Homework Exercises – problems 4.5.3 (b), 4.6.8 (a,b) and laboratory assignment.

Introduction

An example of a math calculator program using yacc is found on page 292 of the book. The software uses a C version of yylex() to provide tokens and attributes to the parser instead of a lex generated scanner. Notice the coding of yylex() on page 289 is not the same as the version found on page 292. You should also notice that the version on page 292 is non-functional and does not provide the correct tokens to the parser. In this assignment you are provided with a working version of yylex() and you are NOT allowed to change the source code of yylex() to implement the assignment. In this assignment you design a method to encode decimal, hexadecimal, octal and floating point numbers adding only parser productions to the original code on page 292.

In the ansi_c compiler the precedence of the operators is handled by the sequence of the productions (look at primary_expr), where in the calculator program the parser uses the yacc %left %right statements to generate precedence. The precedence statements are located in the top of the yacc file just below the %token statements. The precedence is defined going down the page, from lower to higher precedence. The binary operators are referenced to the left (a op b) where the unary operators are referenced to the right (op b). For example: (-2+-3) where minus is used as the unary operator and plus is used as the binary operator.

The book version of the calculator program supports DOUBLES as the yylval data type and only one character or digit is returned by the scanner. On a syntax error the software closes the standard IO streams and exits.

In this assignment yylval is redefined using a structure definition to hold type, long and double values. This changes the size of the yacc stack from a single scalar value to a larger direct reference structure definition. So while the size of the yacc stack is bigger in this implementation it does provide direct memory references for increased speed compared to indirect address pointer implementations, like found in the ansi_c compiler.
The math operators (op) are:

'+'	addition
'-'	subtraction
'*'	multiple
'/'	divide
'-'	unary minus

For the math instructions if all the input operands (CONSTANTS) are LONG then the result is a LONG. If any of the operands are a DOUBLE then the result is a DOUBLE. The data conversion methods are shown below.

LONG = LONG op LONG
DOUBLE = DOUBLE op LONG
DOUBLE = LONG op DOUBLE
DOUBLE = DOUBLE op DOUBLE
LONG = op LONG
DOUBLE = op DOUBLE

The logical bit-wise operators are:

'|'	IOR
'^'	XOR
'&'	AND
'%'	MOD
'~'	unary invert

For the logical bit-wise operators all the DOUBLE input operands are converted into LONG and the results are LONG. The data conversion methods are shown below.

LONG = LONG op LONG
LONG = DOUBLE op LONG
LONG = LONG op DOUBLE
LONG = DOUBLE op DOUBLE
LONG = op LONG
LONG = op DOUBLE

The table below compares the methods of defining the YYSTYPE and yylval over the last few assignments. The three basic methods defined are typically found in most implementations of yacc generated programs. In Assignment 04 the yylval is used to hold an immediate value and YYSTYPE is defined as a integer. In Assignment 05 the yylval is used to hold either an immediate value or a pointer to dynamic memory and YYSTYPE is defined as a union structure. The pointer to memory is an indirect memory access (->). In this assignment the yylval is used to hold the calculator values and YYSTYPE is defined as a structure. A structure definition provides a direct memory reference (.) and is faster than an indirect memory reference. See the following table.

Assignment:	04	05	06

YYSTYPE:	integer:	union:
		int token;
		CLIPS *clips;
			structure:
			int type;
			long lvalue;
			double dvalue;

yylval usage:	value	value or indirect	direct
yacc stack size:	small	medium	large

To achieve maximum execution speed the YYSTYPE data structure is defined using a structure definition and the math and logic operations are coded directly in-line. The trade-off for increased speed is more source code to maintain and a larger executable image. This model allows yacc generated programs to be used in soft real time applications such as on-line interpreters and processing stages in communication systems.

In this assignment you shall implement the functions to encode constant values in the parser instead of using lex and a scanner function. You will add to the calculator grammar WITHOUT CHANGING ANY OF THE SOURCE CODE FOR THE C FUNCTION yylex(). The encoder functions are normally found in the lexical scanner but in this assignment the functions to encode constant values shall be accomplished using additional productions to the calculator grammar. You will need to use the math library (-lm) and it has already been included in the new Makefile. You should be able to support: integers (ddd), octal (0ooo), hexadecimal (0xhhh), and floating point numbers (d.de-d). The current version defined in the book supports doubles as one character.

Example: (2 + 3) * (2. + 3.)

As you add to the grammar and test the actions you should not have any shift/reduce errors and SHALL NOT have any REDUCE/REDUCE errors. The error messages are printed by yacc when there is a problem with the productions. You can allow a shift/reduce error but you can NOT allow a reduce/reduce error! (A correct solution does not have any errors.) Reduce/reduce errors occur when the parser can not determine which production to reduce based on the input tokens. This will result in a error being generated in the meaning of the program. Remember to keep most productions with the non-terminal symbols to the left and the terminal symbols to the right.

To implement the solution you need to understand the values returned from yylex(). The only token defined by the parser (yacc.y) is NUMBER and returns the value in yylval. The parser uses ASCII characters for everything else. The math, logic and parentheses operators are all ASCII characters. The yylex() function reads a single character and compares it to a number of patterns to determine the return operation. On reading an EOF the function returns EOF. On reading a space or tab character the function skips to reading the next character. On reading a dollar sign ('$' = keyboard exit character) the function returns EOF. On reading a CR ('\n') the function returns CR. On reading any digit ('0'-'9') it sets yylval to the binary value and returns NUMBER. On reading anything else, it returns the character to the parser. The operations for yylex() are defined in the table below.

input:	yylval:	return:	comment:
space		continue	skip space
tab		continue	skip tab
'$'		EOF	keyboard exit symbol
'\n'		'\n'	CR
[0-9]	value	NUMBER	encode single digit
[.]		char	everything else
EOF		EOF	end of file

The book version of yylex() is switched out of the source code using the preprocessor #ifdef junk #endif statements. It is normal to compile switch out obsolete or broken code and leave the code in the file to allow programmers the ability to compare old to new versions.

Part 2: Laboratory

From a console window, make a directory on your computer in your EECS337 directory under your Case ID and call it hw06.
mkdir ~/EECS337/caseid/hw06/   ; where caseid is YOUR Case ID, enter all in lower case

Change directory to the hw06 directory.
cd ~/EECS337/caseid/hw06/

Download a copy of: hw06_caseid.tar file to the hw06 directory from http://blackboard.case.edu/ in the EECS337 homework assignment area.

To untar the tar file type the command.
tar xvf hw06_caseid.tar

The following files will be created in the current working directory:
hw06_test.sh

calc.c
book1.y
book2.y
Makefile
math1.calc
math2.calc
math3.calc
math4.calc
math5.calc
math6.calc
math7.calc
math8.calc
math9.calc

To test the original book version:
cp book1.y yacc.y
make clean
make
To test the version from the command line type:
./calc and enter a math expression: (**2 + 3) * (4 + 5)**
You should get the result printed to the standard output.
Notice on a syntax error the software exits: **2 + .2**
To exit the command line type: **$**
To test with yydebug statements type:
./calc +yydebug and enter a math expression as before.

Now test the updated book version:
cp book2.y yacc.y
make clean
make
To test the version from the command line type:
./calc and enter a math expression: (**2 + 3) * (4 + 5)**
You should get the result printed to the standard output.
Notice on a syntax error the software DOES NOT exit: **2 + .2**
To exit the command line type: **$**

You are now ready to solve the laboratory assignment.

Part 3: Laboratory Assignment

Edit main.c and change all caseids to your caseid and save the main.c file.

Edit the yacc.y file and without changing yylex(), add the productions to support: integers (ddd), octal (0ooo), hexadecimal (0xhhh), and floating point numbers (d.de-d). Test your calculator using each of the test files by typing the commands:
./calc < math1.calc
./calc < math2.calc
./calc < math3.calc
./calc < math4.calc
./calc < math5.calc
./calc < math6.calc
./calc < math7.calc
./calc < math8.calc
./calc < math9.calc

When all your lab assignments have been completed execute the homework script file "./hw06_test.sh". You may need to change the mode of the file to an executable using the command "chmod 755 *.sh". The script file shall remove the old object files, make all source programs and run the tests. To redirect the standard error (stderr) and standard output (stdout) to a file use the following command.
./hw06_test.sh &> hw06_test.txt

Print out the hw06_test.txt file and put your name, assignment number, date on it and turn it in with your homework exercises. Consider using the **a2ps** utility for printing your output to help minimize the number of sheets. Copy your yacc.y file to hw06_CaseID.y using your Case ID and post the file to the digital drop box on blackboard.case.edu under the EECS337 section.
cp yacc.y hw06_caseid.y

The final source code files are listed below.

Your final directory structure for the calc compiler should be as below (using your Case ID):

```
EECS337/caseid/hw06/hw06_test.sh
EECS337/caseid/hw06/hw06_test.txt
EECS337/caseid/hw06/Makefile
EECS337/caseid/hw06/calc.c
EECS337/caseid/hw06/book1.y
EECS337/caseid/hw06/book2.y
EECS337/caseid/hw06/yacc.y          /* with your productions to encode constant values */
EECS337/caseid/hw06/hw06_caseid.y   /* cp yacc.y hw06_caseid.y and submit to digital drop */
EECS337/caseid/hw06/math1.calc
EECS337/caseid/hw06/math2.calc
EECS337/caseid/hw06/math3.calc
EECS337/caseid/hw06/math4.calc
EECS337/caseid/hw06/math5.calc
EECS337/caseid/hw06/math6.calc
EECS337/caseid/hw06/math7.calc
EECS337/caseid/hw06/math8.calc
EECS337/caseid/hw06/math9.calc
```