CASE WESTERN RESERVE UNIVERSITY
Case School of Engineering
Department of Electrical Engineering and Computer Science
EECS 337 Systems Programming (Compiler Design)
Fall 2010
Solution #3 September 14, 2010

Download the files from blackboard from under the Solutions #3 directory. You should have four files: main.c, scan.l, gram1.y and gram2.y. This version of scan.l is slightly different from the original version in that it returns the binary value of the ascii code to the parser instead of the ascii code. To return the value of the ascii code use *yytext or yytext[0] and subtract the value of ascii zero ('0'). Example: yylval = *yytext – '0';

Then copy either gram1.y or gram2.y to gram.y and build the universe compiler. Both versions build a decimal number from the input1234.txt file using left reduce or right reduce productions. The file gram1.y is a left reduce grammar and gram2.y is a right reduce grammar. Notice the method to build a decimal number. In gram1.y it is: $$ = $1 * 10 + $2; where in gram2.y it is: $$ = $1 * power + $2; power = power * 10; In the second version a state variable for the decimal power is required to construct the final value.

When you run the compiler you will see the values printed to the console window.
For gram1.y the output is:

```
[doldham@ivhm-1 universe]$ ./universe +d < input1234.txt
for caseid start time: Wed Sep 17 08:02:24 2008
T_UNIVERSE: $1=1
universes: $1=1 T_UNIVERSE: $2=2
universes: $1=12 T_UNIVERSE: $2=3
universes: $1=123 T_UNIVERSE: $2=4
universes: $1=1234
```

For gram2.y the output is:

```
[doldham@ivhm-1 universe]$ ./universe +d < input1234.txt
for caseid start time: Wed Sep 17 07:58:38 2008
T_UNIVERSE: $1=4
T_UNIVERSE: $1=3 universes: $2=4
T_UNIVERSE: $1=2 universes: $2=34
T_UNIVERSE: $1=1 universes: $2=234
universes: $1=1234
```

Notice in gram1.y the values appear in the order of the input file where in gram2.y the values appear in reverse order. In the first example the values are pushed and popped off the stack where in the second example all the input values are pushed onto the stack then popped off in reverse order. Both methods produce the same output but the first example keeps the yacc stack to a minimum size during execution. YACC is designed as a left reduce parser and therefore using the productions of gram1.y provides the best utilization of the yacc stack.