Your assignment is to write a program using yacc and lex that implements a calculator of C integer expressions. The calculator will process expressions until it encounters EOF or invalid syntax. Each calculation is terminated by a semicolon. Tokens can be separated by whitespace (but no comments). After each calculation, the calculator prints its result. Tokens include integer numbers and 26 predefined integer variables. There will be one variable corresponding to each of the lowercase letters in the alphabet. The table below defines the C operators you need to implement. The operators are listed in order of decreasing precedence. No operators may precede an assignment operator in a calculation except another assignment operator.

<table>
<thead>
<tr>
<th>symbols</th>
<th>comments</th>
<th>type</th>
<th>assoc</th>
</tr>
</thead>
<tbody>
<tr>
<td>()</td>
<td>parentheses</td>
<td>unary</td>
<td>n/a</td>
</tr>
<tr>
<td>++ --</td>
<td>increment, decrem</td>
<td>postfix</td>
<td>left</td>
</tr>
<tr>
<td>++ --</td>
<td>increment, decrem</td>
<td>prefix</td>
<td>right</td>
</tr>
<tr>
<td>-.</td>
<td>bitwise not, negation</td>
<td>unary</td>
<td>right</td>
</tr>
<tr>
<td>* / %</td>
<td>mult, div, rem</td>
<td>binary</td>
<td>left</td>
</tr>
<tr>
<td>+ -</td>
<td>add, sub</td>
<td>binary</td>
<td>left</td>
</tr>
<tr>
<td>&lt;&lt; &gt;&gt;</td>
<td>shifts</td>
<td>binary</td>
<td>left</td>
</tr>
<tr>
<td>&amp;</td>
<td>bitwise and</td>
<td>binary</td>
<td>left</td>
</tr>
<tr>
<td>^</td>
<td>bitwise xor</td>
<td>binary</td>
<td>left</td>
</tr>
<tr>
<td></td>
<td></td>
<td>bitwise or</td>
<td>binary</td>
</tr>
<tr>
<td>=</td>
<td>assignment</td>
<td>binary</td>
<td>right</td>
</tr>
</tbody>
</table>

Yacc does provide mechanisms for specifying the associativity and precedence of operators in an unambiguous grammar. However, you are not allowed to use these mechanisms. Instead you should define extra nonterminals and productions to enforce the specified associativity and precedence for this assignment.

Rather than having a default value of zero for each of the predefined variables, you will treat the default variable as having an unknown value. If a variable’s value is referenced without ever having been assigned a value, then the value of the expression should be designated as unknown. Likewise, the result of an operation is unknown if any of the operands are unknown. A variable’s value becomes unknown when it is assigned an unknown value.

In addition to calculating an expression, two other commands are supported. The first is to dump the values of the different variables when the keyword dump is detected. The second is to reset all of the values of the different variables to unknown when the keyword reset is encountered.

You need to create files called cexpr, scan.l, and makefile that will contain the parser, scanner and makefile. The makefile should make an executable called cexpr. You should attach all three of these files in a single e-mail message and mail them to whalley@cs.fsu.edu before the beginning of class on February 20. Please put COP5621 somewhere on the subject line of the message. To get you started, I have placed the ex.y, ex.l, and makefile files in the "whalley/cop5621ex" directory. You should rename the ex.y and ex.l files to cexpr and scan.l, respectively, and extend them with your solution for this assignment.
Example Input:

```plaintext
a = 55-3;  
b = c = a-42;  
a+b*c;  
dump  
reset  
c = 6;  
a = b;  
^D
```

Example Output:

```
52  
10  
152  
a: 52  
b: 10  
c: 10  
d: unknown  
e: unknown  
f: unknown  
g: unknown  
h: unknown  
i: unknown  
j: unknown  
k: unknown  
l: unknown  
m: unknown  
n: unknown  
o: unknown  
p: unknown  
q: unknown  
r: unknown  
s: unknown  
t: unknown  
u: unknown  
v: unknown  
w: unknown  
x: unknown  
y: unknown  
z: unknown  
6  
unknown  
Calculator off.
```