COSC 561

cexpr - calculator for C integer expressions

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.

symbols	comments	type	assoc
( )	parentheses	unary	n/a
~	bitwise not	unary	right
−	negation	unary	right
*/%	mult, div, rem	binary	left
+ −	add, sub	binary	left
<< >>	shifts	binary	left
&	bitwise and	binary	left
^	bitwise xor	binary	left
|	bitwise or	binary	left
= += -= *= /= %= <<= >>= &= ^= |=	assignment	binary	right

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. In addition, you have to detect error conditions, which include integer overflow and divide by zero. When there is an error condition, you will print out an appropriate message indicating the first type of error encountered rather than the value after the calculation. You will not perform any assignments in a calculation after encountering an error.

In addition to calculating an expression, two other commands are supported. The first is to dump the values of the different variables when the command **dump;** is detected. The second is to clear all of the values of the different variables to zero when the command **clear;** is encountered. All variables should be zero at the beginning of the execution.

You need to create files called cexpr.y, scan.l, and Makefile that will contain the parser, scanner and Makefile. The Makefile should make an executable called cexpr. To get you started, I have placed ex.y, ex.l, and an example Makefile in a starter package on the Canvas site for this course (in the PA3 directory in the Assignments section). You should extend these files for this assignment. Additionally, the starter package includes the sample input as well as a reference executable (`ref_cexpr`) that you can use to test alternative inputs. You will also submit a project report, similar to the report you submitted for PA1 and PA2. Specifically, you will submit a one page (single-spaced) document that describes:

1. (in your own words) the program you set out to write,
2. your approach (i.e. design and relevant implementation details) for writing this program,
3. how you debugged and tested your solution, and
4. any issues you had in completing the assignment.

You should upload a gzipped tar file (created with `tar cvzf ...`) with your source files and a pdf of your report to the Canvas course website by 11:59pm on the assignment due date. Partial credits will be given for incomplete efforts. However, a program that does not compile or run will get 0 points. Point breakdown is below:

- lex scanner works properly (20)
- yacc parser works for each command / expression (50)
- precedence / associativity of each operator is correct (10)
- error checking (overflow, dividebyzero) (10)
- project report (10)

```
Example Input:
a = 55-3;
b = c = a-42;
a+b*c;
dump;
clear;
c = 6;
a = b;
a = 10000;
b = 100000;
a*b;
a*b*10;
c/d;
d/c;
^D

Example Output:

52
10
152
a: 52
b: 10
c: 10
d: 0
e: 0
f: 0
g: 0
h: 0
i: 0
j: 0
k: 0
l: 0
m: 0
n: 0
o: 0
p: 0
q: 0
r: 0
s: 0
t: 0
u: 0
v: 0
w: 0
x: 0
y: 0
z: 0
6
0
10000
100000
1000000000
overflow
dividebyzero
0

Calculator off.
```