Steven Diviney
08462267
CS3071

1 Attributed Translation Grammar

Using {} to denote iteration and (..) to denote semantic actions as per Coco/R

$$< Calc > \Rightarrow \{Ident_p :=< Expr >_q; \}(.Assign\ s, t.)$$
$$s \Leftarrow q \qquad t \Leftarrow p$$

$$< Calc > \Rightarrow \{display :=< Expr >_p; [: hex; | : oct; |;]_q\}(.Display\ s, t.)$$
$$s \Leftarrow q \qquad t \Leftarrow p$$

$$< Calc > \Rightarrow halt(.Halt.)$$

$$< Ident >_p \Rightarrow < ident >_q$$
$$p \Leftarrow q$$

$$< Expr >_p \Rightarrow < Term >_q + < Term >_r\ (.Add\ s, t, u.)$$
$$s \Leftarrow q \qquad t \Leftarrow r \qquad newt(p, u)$$

$$< Expr >_p \Rightarrow < Term >_q - < Term >_r\ (.Sub\ s, t, u.)$$
$$s \Leftarrow q \qquad t \Leftarrow r \qquad newt(p, u)$$

$$< Expr >_p \Rightarrow < Term >_q$$
$$p \Leftarrow q$$

$$< Term >_p \Rightarrow < Factor >_q * < Factor >_r\ (.Mulu\ s, t, u.)$$
$$s \Leftarrow q \qquad t \Leftarrow r \qquad newt(p, u)$$

$$< Term >_p \Rightarrow < Factor >_q / < Factor >_r\ (.Div\ s, t, u.)$$
$$s \Leftarrow q \qquad t \Leftarrow r \qquad newt(p, u)$$

$$< Term >_p \Rightarrow < Factor >_q \,\%\, < Factor >_r \; (.Mod\, s, t, u.)$$
$$s \Leftarrow q \qquad t \Leftarrow r \qquad newt(p, u)$$

$$< Term >_p \Rightarrow < Factor >_q$$
$$p \Leftarrow q$$

$$< Factor >_p \Rightarrow < Expon >_q \,\hat{}\, < Expon >_r \; (.Power\, s, t, u.)$$
$$s \Leftarrow q \qquad t \Leftarrow r \qquad newt(p, u)$$

$$< Factor >_p \Rightarrow < Expon >_q$$
$$p \Leftarrow q$$

$$< Expon >_p \Rightarrow ident_q$$
$$p \Leftarrow q$$

$$< Expon >_p \Rightarrow (Expr_q)$$
$$p \Leftarrow q$$

A draft attributed translation grammar was designed based on the inputs and operations to be performed by the interpreter. The goal was to construct a basic calculator able to process simple arithmetic expressions a single line at a time. Digits and identifiers are defined as tokens, whereas the operator symbols are part of the ATG. The calculator also required some basic memory to store variables. A simple map between the variables name and it's value was used. After initial construction and testing, the grammar was refined into that which is presented above. The draft grammar lacked any operator precedence and additional productions were need to implement this. The grammar is constructed so that symbols with higher precedence occur further down the derivation tree, insuring that they are evaluated first. The grammar was also constructed so as to give identical results to the example in the assignment, but additional tests were also performed. It is assumed that an expression starts with an assignment or the word "display". The length of these statements is limited only by the input buffer, not the grammar. "halt" stops the interpreter.
Compile.cpp contains the programs main method. A line is read from the user and passed to a new Scanner object which is then passed to a new Parser object. The input is then parsed. New Scanner and Parser objects are created with every new line. This seemed like the simplest way to read from stdin without having to press CTRL-D after a line was entered. A map is also created, it's reference passed to each new Parser as they are created. This serves as the symbol table.