```
Main()
   /*Goal → Expr */
   word ← NextWord();
   if (Expr() and word = eof)
     then proceed to next step
     else return false

Expr()
   /* Expr → Term Expr' */
   if (Term() = false)
     then return false
     else return EPrime()

EPrime()
   /* Expr' → + Term Expr' */
   /* Expr' → - Term Expr' */
   if (word = + or word = - ) then
     word ← NextWord()
     if (Term() = false)
      then return false
      else return EPrime()
   /* Expr' -> ε */
   return true

Term()
   /* Term → Factor Term' */
   if (Factor() = false)
     then return false
     else return TPrime()
```

```
TPrime()
 /* Term' -> * Factor Term' */
 /* Term' -> / Factor Term' */
 if (word = * or word = /) then
    word ← NextWord()
    if (Factor() = false)
       then return false
       else return TPrime()
 /* Term' -> e */
 return true

Factor ()
 /* Factor -> ( Expr ) */
 if (word = ( ) then
    word <= NextWord()
    if (Expr() = false)
       then return false
       else if (word != ) ) then
            report syntax error
            return false

 /* Factor → num */
 /* Factor → ident */
 else if (word != null &&
          word != ident) then
    report syntax error
    return false

 word ← NextWord()
 return true
```

Figure 3.7 Recursive Descent Parser for Expressions from Cooper & Torczon, *Engineering a Compiler*