

# CmpE 320 – Spring 2008 – Assignment 1 Questions & Solutions

## Question 1: Double Fun

### Part A (25 pts)

Give an EBNF description for the double scalars used in MATLAB. The main data type is the double, so the integer syntax (as opposed to other languages) is also behaved as a double in MATLAB. Here are some valid examples for you:

- 5
- -20.
- .2
- 2.75E11
- 10.e-8
- -+-5.78E-6

As shown in the examples “e” and “E” are used for the exponent part, there can be a minus sign in the integer part or the exponent part and the dot can be used only in the base part. At most one sign operator can be used for the exponent part whereas any number of sign operators can be used for the base part as in the last example.

### Answer

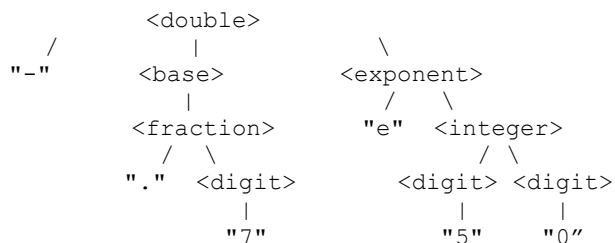
```
<double>  -> { ("+"|"-" ) } <base> [<exponent>]
<base>     -> [<integer>] <fraction>
           | <integer> ["."]
<integer> -> <digit> {<digit>}
<fraction> -> "." <integer>
<exponent> -> ("e"|"E") [ ("+"|"-" ) ] <integer>
<digit>    -> ("0"|"1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9")
```

### Part B (20 pts)

Given the grammar you wrote in the previous part, show the parse tree (10 pts) and the *rightmost* derivation (10 pts) of the following double: `-.7e50`

### Answer

#### Parse Tree



#### Derivation

```
<double> => - <base> <exponent>
<double> => - <base> e <integer>
<double> => - <base> e <digit> <digit>
<double> => - <base> e <digit> 0
<double> => - <base> e50
<double> => - <fraction> e50
<double> => -. <digit> e50
<double> => -.7e50
```

## Question 2: Confused (10 pts)

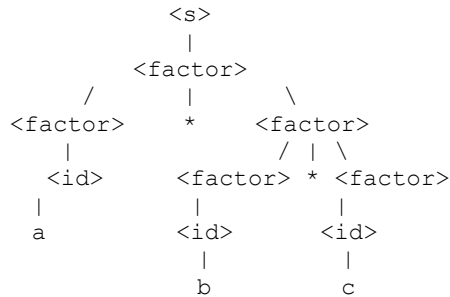
Prove (with an example) that the grammar below is ambiguous.

```
<s>      -> <factor>
<factor> -> <factor> * <factor> | <id>
<id>    -> a | b | c
```

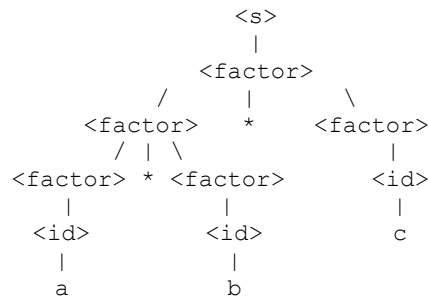
### Answer

Example ambiguous sentence:  $a * b * c$

Parse tree 1:



Parse tree 2:



It is seen that there are two different parse trees for the same sentence and thus, the grammar is ambiguous.

## Question 3: Linear Algebra Again (45 pts)

Consider the following grammar taken from your book:

```
<expr>  -> <expr> + <term>
         | <term>

<term>  -> <term> * <factor>
         | <factor>

<factor> -> ( <expr> )
         | <id>
```

Suppose that the elements in an expression are two-dimensional matrices instead of simple variables. For instance, the expression " $m_1 * m_2 + m_3$ " indicates that  $m_1, m_2$  and  $m_3$  are matrices (of appropriate sizes) and the result is also a matrix. Write an attribute grammar that evaluates an expression. You may assume that each nonterminal in the grammar has the attributes  $m$  (number of rows),  $n$  (number of columns) and  $value$  (contents of the matrix; a particular element in a matrix can be accessed using subscripts, e.g.  $value(i, j)$ ). The attribute grammar will have only semantic functions and there will be no predicate function. Semantic functions will be used to calculate the result of matrix operation and our goal is to get the result of a given expression. For instance, given the expression " $m_1 * m_2 + m_3$ " and knowing (via intrinsic attributes) the values of  $m, n, value$  for the expression (i.e.  $\langle expr \rangle.m, \langle expr \rangle.n, \langle expr \rangle.value$ ).

## Answer

---

The underlined parts are the syntaxes and the remaining parts denote the semantic rules of the attribute grammar.

```
<expr1> -> <expr2> + <term>  
<expr1>.m <- <expr2>.m  
<expr1>.n <- <expr2>.n  
for (i = 1 to <expr2>.m)  
  for (j = 1 to <expr2>.n)  
    <expr1>.value(i,j) <- <expr2>.value(i,j) + <term>.value(i,j)  
  endfor  
endfor
```

```
<expr> -> <term>  
<expr>.m <- <term>.m  
<expr>.n <- <term>.n  
for (i = 1 to <term>.m)  
  for (j = 1 to <term>.n)  
    <expr>.value(i,j) <- <term>.value(i,j)  
  endfor  
endfor
```

```
<term1> -> <term2> * <factor>  
<term1>.m <- <term2>.m  
<term1>.n <- <factor>.n  
for (i = 1 to <term2>.m)  
  for (j = 1 to <factor>.n)  
    <term1>.value(i,j) <- 0  
    for (k = 1 to <term2>.n)  
      <term1>.value(i,j) <- <term1>.value(i,j) + <term2>.value(i,k) * <factor>.value(k,j)  
    endfor  
  endfor  
endfor
```

```
<term> -> <factor>  
<term>.m <- <factor>.m  
<term>.n <- <factor>.n  
for (i = 1 to <factor>.m)  
  for (j = 1 to <factor>.n)  
    <term>.value(i,j) <- <factor>.value(i,j)  
  endfor  
endfor
```

```
<factor> -> ( <expr> )  
<factor>.m <- <expr>.m  
<factor>.n <- <expr>.n  
for (i = 1 to <expr>.m)  
  for (j = 1 to <expr>.n)  
    <factor>.value(i,j) <- <expr>.value(i,j)  
  endfor  
endfor
```

```
<factor> -> <id>  
<factor>.m <- <id>.m  
<factor>.n <- <id>.n  
for (i = 1 to <id>.m)  
  for (j = 1 to <id>.n)  
    <factor>.value(i,j) <- <id>.value(i,j)  
  endfor  
endfor
```

---

## Notes:

- See <http://www.cmpe.boun.edu.tr/~gungort/informationstudents.htm> for general information about the assignments and the course policy.
- The sole purpose of this assignment is to familiarize you with the processes involved in answering questions related to programming languages. Thus, please work on them by your own.