# HW #05 – Grammars – (REV 1)
# CSCI-400 Spring 2013
# INSTRUCTOR SOLUTIONS

**Problem #1**

Consider the following BNF grammar:

```
<program>       -> main { <stmts> }
<stmts>         -> <stmt>;
                -> <stmt>; <stmts>
<stmt>          -> <var> = <expr>
<expr>          -> <term> + <term>
                -> <term> - <term>
                -> <term>
<term>          -> <var> | const
<var>           -> a | b | c | d
```

(a) Add rules to this grammar that will permit you to support C-style **`while()`** and **`for()`** loops. Add additional terminals, such as **`while`** and **`for`**, as needed.

*NOTE: There are many ways to accomplish this and the requirements are not tightly specified, so you have quite a bit of latitude. The grammar below supports relational test expressions, but does not support logical or bitwise expressions, nor does it not support assignment statements used as test expressions; as long as your grammar supports the test expression in the example that is sufficient (though hopefully you support at least the whole range of relational operators).*

*One complicating factor is that the instructions say to "add" rules and does not say that you can modify existing rules. This is a bit of an oversight because the simplest way to implement parts of the for() loop is to invoke the existing <stmt> rule, but this means that we can't just add the loops as productions under the <stmt> non-terminal. If you modified the original grammar in a reasonable way, this will be acceptable. The way it is done here is to add the loops as productions for the <stmts>non-terminal, although this is not particularly elegant.*

*However, your grammar should definitely permit nested loops of different types, such as a for() loop within a while() loop that contains two for() loops with some assignment statements before, after, and in between.*

```
<stmts>         -> <loop> | <loop> <stmts>
<loop>          -> while (<test>) { <stmts> }
<loop>          -> for (<stmt> ; <test> ; <stmt>) { <stmts> }
<test>          -> <expr> | <expr> <test_op> <expr>
<test_op>       -> == | != | \< | > | <= | >=
```

*The "\<" uses the excape character "\" to avoid confusion with the "<" used as a metasymbol.*

# HW #05 – Grammars – (REV 1)
# CSCI-400 Spring 2013
# INSTRUCTOR SOLUTIONS

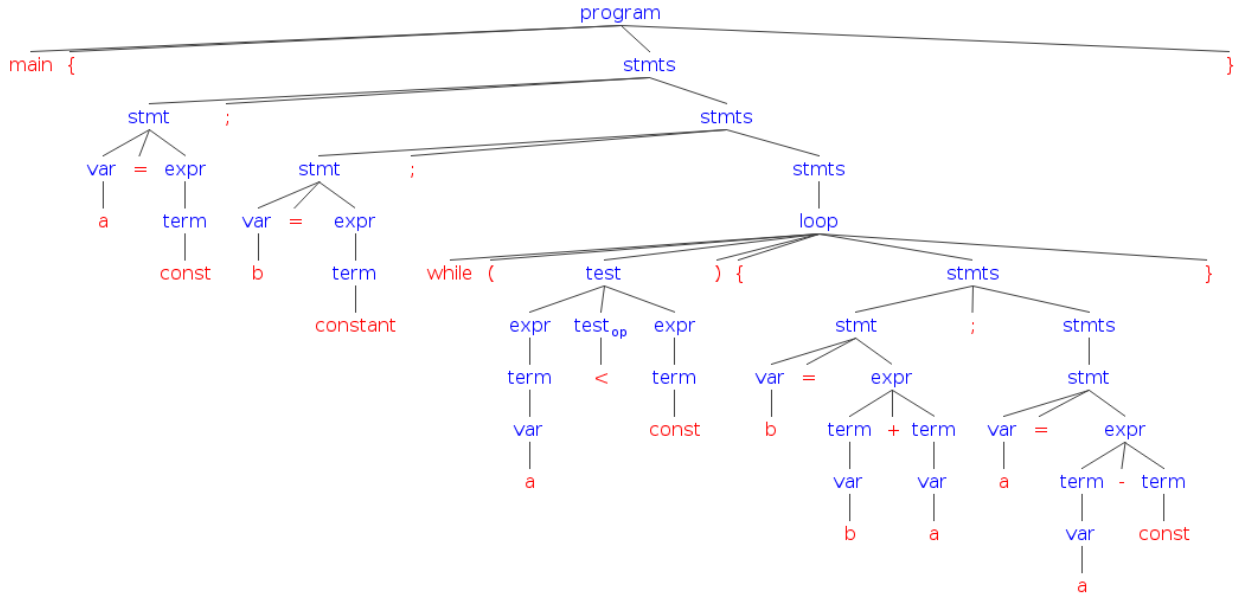
(b) Using your augmented grammar, prepare a leftmost derivation and a parse tree for the following programs:

```
main
{
   a = const;
   b = const;
   while (a < const)
   {
      b = b + a;
      a = a – const;
   }
}
```

```
<program>
main { <stmts> }
main { <stmt>; <stmts> }
main { <var> = <expr>; <stmts> }
main { a = <expr>; <stmts> }
main { a = <term>; <stmts> }
main { a = const; <stmts> }
main { a = const; <stmt>; <stmts> }
main { a = const; <var> = <expr>; <stmts> }
main { a = const; b = <expr>; <stmts> }
main { a = const; b = <term>; <stmts> }
main { a = const; b = const; <loop> }
main { a = const; b = const; while (<test>) { <stmts> } }
main { a = const; b = const; while (<expr> <test_op> <expr>) { <stmts> } }
main { a = const; b = const; while (<term> <test_op> <expr>) { <stmts> } }
main { a = const; b = const; while (<var> <test_op> <expr>) { <stmts> } }
main { a = const; b = const; while (a <test_op> <expr>) { <stmts> } }
main { a = const; b = const; while (a < <expr>) { <stmts> } }
main { a = const; b = const; while (a < <term>) { <stmts> } }
main { a = const; b = const; while (a < const) { <stmts> } }
main { a = const; b = const; while (a < const) { <stmt>; <stmts> } }
main { a = const; b = const; while (a < const) { <var> = <expr>; <stmts> } }
main { a = const; b = const; while (a < const) { b = <expr>; <stmts> } }
main { a = const; b = const; while (a < const) { b = <term> + <term>; <stmts> } }
main { a = const; b = const; while (a < const) { b = <var> + <term>; <stmts> } }
main { a = const; b = const; while (a < const) { b = b + <term>; <stmts> } }
main { a = const; b = const; while (a < const) { b = b + <var>; <stmts> } }
main { a = const; b = const; while (a < const) { b = b + a; <stmts> } }
main { a = const; b = const; while (a < const) { b = b + a; <stmt> } }
main { a = const; b = const; while (a < const) { b = b + a; <var> = <expr> } }
main { a = const; b = const; while (a < const) { b = b + a; a = <expr> } }
main { a = const; b = const; while (a < const) { b = b + a; a = <term> – <term> } }
main { a = const; b = const; while (a < const) { b = b + a; a = <var> – <term> } }
main { a = const; b = const; while (a < const) { b = b + a; a = a – <term> } }
main { a = const; b = const; while (a < const) { b = b + a; a = a – const } }
```

# HW #05 – Grammars – (REV 1)
# CSCI-400 Spring 2013
# INSTRUCTOR SOLUTIONS

**Parse Tree for while() loop example code**




*NOTE*: The above tree was generated using the online tree generator located at

http://www.ironcreek.net/phpsyntaxtree/

The code used to generate it is on the next page. The formatting is not particularly visually pleasing, but reflects some of the idiosyncrasies of the online tree generator.

# HW #05 – Grammars – (REV 1)
# CSCI-400 Spring 2013
# INSTRUCTOR SOLUTIONS

### Parse Tree code for while() loop example code

```
[program [main]
         [{]
         [stmts [stmt [var [a]]
                      [=]
                      [expr [term [const]]]
                ]
                [;]
                [stmts [stmt [var [b]]
                             [=]
                             [expr [term [constant]]]
                       ]
                       [;]
                       [stmts [loop [while]
                                    [(]
                                    [test [expr [term [var [a]]]]
                                          [test_op [<]]
                                          [expr [term [const]]]
                                    ]
                                    [)]
                                    [{]
                                    [stmts [stmt [var [b]]
                                                 [=]
                                                 [expr [term [var [b]]]
                                                       [+]
                                                       [term [var [a]]]
                                                 ]
                                           ]
                                           [;]
                                           [stmts [stmt [var [a]]
                                                        [=]
                                                        [expr [term [var [a]]]
                                                              [-]
                                                              [term [const]]
                                                        ]
                                                  ]
                                           ]
                                    ]
                                    [}]
                             ]
                       ]
                ]
         ]
         [}]
]
```

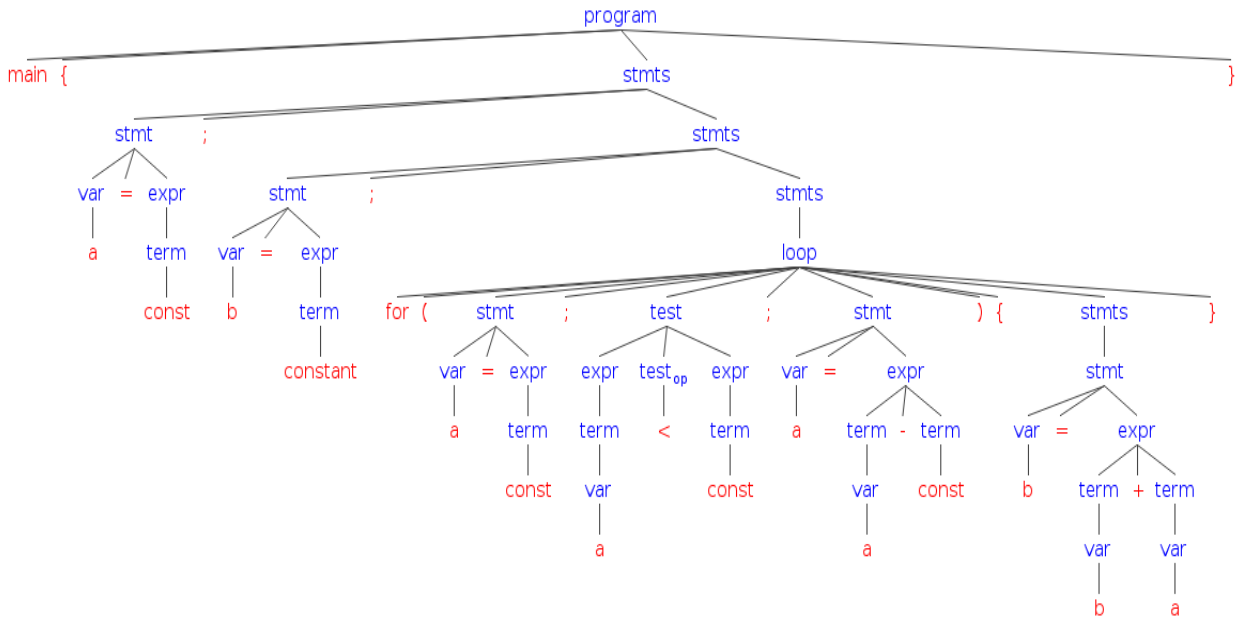
# HW #05 – Grammars – (REV 1)
# CSCI-400 Spring 2013
# INSTRUCTOR SOLUTIONS

```
main
{
   a = const;
   b = const;
   for (a = const; a < const; a = a - const)
   {
      b = b + a;
   }
}
```

```
<program>
main{<stmts>}
main{<stmt>;<stmts>}
main{<var>=<expr>;<stmts>}
main{a=<expr>;<stmts>}
main{a=<term>;<stmts>}
main{a=const;<stmts>}
main{a=const;<stmt>;<stmts>}
main{a=const;<var>=<expr>;<stmts>}
main{a=const;b=<expr>;<stmts>}
main{a=const;b=<term>;<stmts>}
main{a=const;b=const;<stmts>}
main{a=const;b=const;<loop>}
main{a=const;b=const;for(<stmt>;<test>;<stmt>){<stmts>}}
main{a=const;b=const;for((<var>=<expr>;<test>;<stmt>){<stmts>}}
main{a=const;b=const;for(a=<expr>;<test>;<stmt>){<stmts>}}
main{a=const;b=const;for(a=<term>;<test>;<stmt>){<stmts>}}
main{a=const;b=const;for(a=const;<test>;<stmt>){<stmts>}}
main{a=const;b=const;for(a=const;<expr><test_op><expr>;<stmt>){<stmts>}}
main{a=const;b=const;for(a=const;<term><test_op><expr>;<stmt>){<stmts>}}
main{a=const;b=const;for(a=const;<var><test_op><expr>;<stmt>){<stmts>}}
main{a=const;b=const;for(a=const;a<test_op><expr>;<stmt>){<stmts>}}
main{a=const;b=const;for(a=const;a<<expr>;<stmt>){<stmts>}}
main{a=const;b=const;for(a=const;a<<term>;<stmt>){<stmts>}}
main{a=const;b=const;for(a=const;a<const;<stmt>){<stmts>}}
main{a=const;b=const;for(a=const;a<const;(<var>=<expr>){<stmts>}}
main{a=const;b=const;for(a=const;a<const;(a=<expr>){<stmts>}}
main{a=const;b=const;for(a=const;a<const;(a=<term>-<term){<stmts>}}
main{a=const;b=const;for(a=const;a<const;(a=<var>-<term>){<stmts>}}
main{a=const;b=const;for(a=const;a<const;(a=a-<term>){<stmts>}}
main{a=const;b=const;for(a=const;a<const;(a=a-const){<stmts>}}
main{a=const;b=const;for(a=const;a<const;(a=a-const){<stmt>}}
main{a=const;b=const;for(a=const;a<const;(a=a-const){<var>=<expr>}}
main{a=const;b=const;for(a=const;a<const;(a=a-const){b=<expr>}}
main{a=const;b=const;for(a=const;a<const;(a=a-const){b=<term>+<term>}}
main{a=const;b=const;for(a=const;a<const;(a=a-const){b=<var>+<term>}}
main{a=const;b=const;for(a=const;a<const;(a=a-const){b=b+<term>}}
main{a=const;b=const;for(a=const;a<const;(a=a-const){b=b+<var>}}
main{a=const;b=const;for(a=const;a<const;(a=a-const){b=b+a}}
```

# HW #05 – Grammars – (REV 1)
# CSCI-400 Spring 2013
# INSTRUCTOR SOLUTIONS

**Parse Tree for for() loop example code**




*NOTE*: The above tree was generated using the online tree generator located at

http://www.ironcreek.net/phpsyntaxtree/

The code used to generate it is on the next page. The formatting is not particularly visually pleasing, but reflects some of the idiosyncrasies of the online tree generator.

# HW #05 – Grammars – (REV 1)
# CSCI-400 Spring 2013
# INSTRUCTOR SOLUTIONS

### Parse Tree code for for() loop example code

```
[program [main]
         [{]
         [stmts [stmt [var [a]]
                      [=]
                      [expr [term [const]]]
                ]
                [;]
                [stmts [stmt [var [b]]
                             [=]
                             [expr [term [constant]]]
                       ]
                       [;]
                       [stmts [loop [for]
                                    [(]
                                    [stmt [var [a]]
                                          [=]
                                          [expr [term [const]]]
                                    ]
                                    [;]
                                    [test [expr [term [var [a]]]]
                                          [test_op [<]]
                                          [expr [term [const]]]
                                    ]
                                    [;]
                                    [stmt [var [a]]
                                          [=]
                                          [expr [term [var [a]]]
                                                [-]
                                                [term [const]]
                                          ]
                                    ]
                                    [)]
                                    [{]
                                    [stmts [stmt [var [b]]
                                                 [=]
                                                 [expr [term [var [b]]]
                                                       [+]
                                                       [term [var [a]]]
                                                 ]
                                           ]
                                    ]
                                    [}]
                              ]
                       ]
                ]
         ]
         [}]
]
```

# HW #05 – Grammars – (REV 1)
# CSCI-400 Spring 2013
# INSTRUCTOR SOLUTIONS

## Problem #2

Consider the following if-else grammar

```
<stmt>    -> if <expr> then <stmt>
          -> if <expr> then <stmt> else <stmt>
```

(a) Show that this grammar is ambiguous by showing two different leftmost derivations, with corresponding parse trees, for the following statement:

```
if Expr1 then if Expr2 then Stmt1 else Stmt2
```

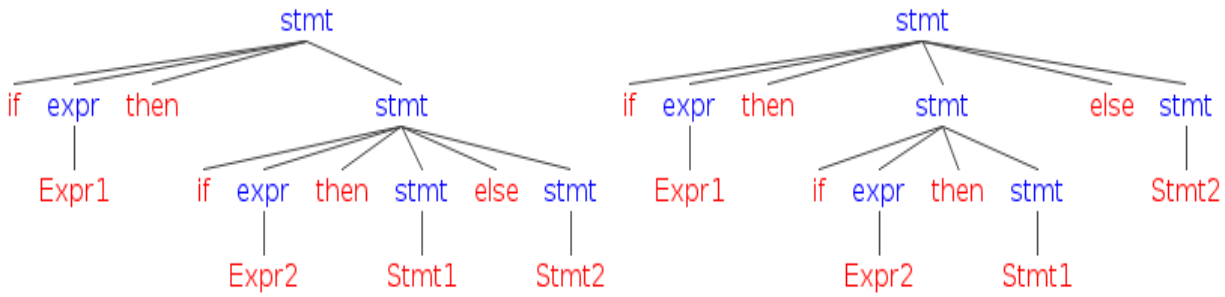
For convenience, you may interpret terminals such as **`Expr1`** and **`Stmt2`** as representing a valid string of tokens that reduce to the corresponding non-terminal.

Derivation #1 (tree on left below)

```
<stmt>    -> if <expr> then <stmt>
<stmt>    -> if Expr1 then <stmt>
<stmt>    -> if Expr1 then if <expr> then <stmt> else <stmt>
<stmt>    -> if Expr1 then if Expr2 then <stmt> else <stmt>
<stmt>    -> if Expr1 then if Expr2 then Stmt1 else <stmt>
<stmt>    -> if Expr1 then if Expr2 then Stmt1 else Stmt2
```

Derivation #2 (tree on right below)

```
<stmt>    -> if <expr> then <stmt> else <stmt>
<stmt>    -> if Expr1 then <stmt> else <stmt>
<stmt>    -> if Expr1 then if <expr> then <stmt> else <stmt>
<stmt>    -> if Expr1 then if Expr2 then <stmt> else <stmt>
<stmt>    -> if Expr1 then if Expr2 then Stmt1 else <stmt>
<stmt>    -> if Expr1 then if Expr2 then Stmt1 else Stmt2
```



# HW #05 – Grammars – (REV 1)
# CSCI-400 Spring 2013
# INSTRUCTOR SOLUTIONS

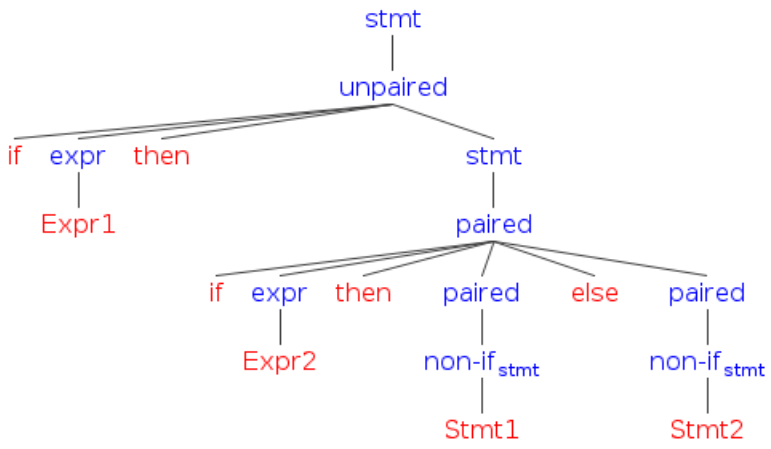
(b) Rewrite the grammar to remove this ambiguity and show the leftmost derivation and parse tree using the revised grammar.

*NOTE: It is tempting to require that statements controlled by the* **`if`** *and* **`else`** *keywords be enclosed in braces (or some other delimiter). However, doing this makes it impossible to generate the program string as given. One solution to this problem is presented in the text in section 3.3.1.10.*

```
<stmt>      -> <paired> | <unpaired>
<paired>    -> if <expr> then <paired> else <paired>
            -> <non-if_stmt>
<unpaired>  -> if <expr> then <stmt>
            -> if <expr> then <paired> else <unpaired>
```

Revised Derivation

```
<stmt>  -> <unpaired>
        -> if <expr> then <stmt>
        -> if Expr1 then <stmt>
        -> if Expr1 then <paired>
        -> if Expr1 then if <expr> then <paired> else <paired>
        -> if Expr1 then if Expr2 then <paired> else <paired>
        -> if Expr1 then if Expr2 then <non-if_stmt> else <paired>
        -> if Expr1 then if Expr2 then Stmt1 else <paired>
        -> if Expr1 then if Expr2 then Stmt1 else <non-if_stmt>
        -> if Expr1 then if Expr2 then Stmt1 else Stmt2
```



# HW #05 – Grammars – (REV 1)
# CSCI-400 Spring 2013
# INSTRUCTOR SOLUTIONS

**Problem #3**

```
S -> ASE |  CC | BCS
A -> bEC |  Da |  gB
B -> aDE |  cD | kAB
C -> dEa |  eS |  hf
D -> iB  | jkC | fSk
E -> CS  |  Aa |  cb
```

(a) What are the first sets for each non-terminal in this grammar? Which, if any, are not pairwise disjoint?

| | S | A | B | C | D | E |
|---|---|---|---|---|---|---|
| **Rule #1** | **b, f, g, i, j** | **b** | **a** | **d** | **i** | **d, e, h** |
| **Rule #2** | **d, e, h** | **f, i, j** | **c** | **e** | **j** | **b, f, g, i, j** |
| **Rule #3** | **a, c, k** | **g** | **k** | **h** | **f** | **c** |
| **Disjoint?** | **Y** | **Y** | **Y** | **Y** | **Y** | **Y** |

(b) Prepare the parse table for the above grammar with the non-terminals in the leftmost column and the terminals in the top row. Remember that the entry in each cell represents the production rule that should be used when in the state for that row and looking at the token for that column. If any of the rules are not pairwise disjoint, then there will be cells that contain multiple production rules and, when parsing, each must be tried in turn.

| | a | b | c | d | e | f | g | h | i | j | k |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **S** | **BCS** | **ASE** | **BCS** | **CC** | **CC** | **ASE** | **ASE** | **CC** | **ASE** | **ASE** | **BCS** |
| **A** | **--** | **bEC** | **--** | **--** | **--** | **Da** | **gB** | **--** | **Da** | **Da** | **--** |
| **B** | **aDE** | **--** | **cD** | **--** | **--** | **--** | **--** | **--** | **--** | **--** | **kAB** |
| **C** | **--** | **--** | **--** | **dEa** | **eS** | **--** | **--** | **hf** | **--** | **--** | **--** |
| **D** | **--** | **--** | **--** | **--** | **--** | **fSk** | **--** | **--** | **iB** | **jkC** | **--** |
| **E** | **--** | **Aa** | **cb** | **CS** | **CS** | **Aa** | **Aa** | **CS** | **Aa** | **Aa** | **--** |

# HW #05 – Grammars – (REV 1)
# CSCI-400 Spring 2013
# INSTRUCTOR SOLUTIONS

(c) Perform a recursive descent parse of the following sentence and prepare the corresponding leftmost derivation and parse tree.

```
fdgcj khfaa dbcbh faaka
kbhfa icjkd cbacb hfedc
bahfd cbahf cjkdc bahfd
cbahf cb
```

*NOTE: In the following parse sequence, the LHS is the left-most symbol from the prior right-hand side and, in parens, is the current terminal being viewed at the front of the input stream. If the symbol is a non-terminal, then the nonterminal at the front of the prior right-hand side is replaced with the production indicated by the parse table. If the symbol is a terminal and it matches the current terminal being viewed, then both are removed from their respective streams. If they do not match, then a syntax error has occurred. The $ character indicates end of input*

Recursive Descent Parse

```
S(f) -> ASE
A(f) -> DaSE
D(f) -> fSkaSE
f(f) -> SkaSE
S(d) -> CCkaSE
C(d) -> dEaCkaSE
d(d) -> EaCkaSE
E(g) -> AaaCkaSE
A(g) -> gBaaCkaSE
g(g) -> BaaCkaSE
B(c) -> cDaaCkaSE
c(c) -> DaaCkaSE
D(j) -> jkCaaCkaSE
j(j) -> kCaaCkaSE
k(k) -> CaaCkaSE
C(h) -> hfaaCkaSE
h(h) -> faaCkaSE
f(f) -> aaCkaSE
a(a) -> aCkaSE
a(a) -> CkaSE
C(d) -> dEakaSE
d(d) -> EakaSE
E(b) -> AaakaSE
A(b) -> bECaakaSE
b(b) -> ECaakaSE
E(c) -> cbCaakaSE
c(c) -> bCaakaSE
b(b) -> CaakaSE
C(h) -> hfaakaSE
h(h) -> hfaakaSE
f(f) -> aakaSE
a(a) -> akaSE
a(a) -> kaSE
k(k) -> aSE
a(a) -> SE
S(k) -> BCSE
B(k) -> kABCSE
k(k) -> ABCSE
A(b) -> bECBCSE
b(b) -> ECBCSE
E(h) -> CSCBCSE
C(h) -> hfSCBCSE
h(h) -> fSCBCSE
f(f) -> SCBCSE
S(a) -> BCSCBCSE
B(a) -> aDECSCBCSE
a(a) -> DECSCBCSE
D(i) -> iBECSCBCSE
i(i) -> BECSCBCSE
B(c) -> cDECSCBCSE
c(c) -> DECSCBCSE
D(j) -> jkCECSCBCSE
j(j) -> kCECSCBCSE
k(k) -> CECSCBCSE
C(d) -> dEaECSCBCSE
d(d) -> EaECSCBCSE
E(c) -> cbaECSCBCSE
c(c) -> baECSCBCSE
b(b) -> aECSCBCSE
a(a) -> ECSCBCSE
E(c) -> cbCSCBCSE
c(c) -> bCSCBCSE
b(b) -> CSCBCSE
C(h) -> hfSCBCSE
h(h) -> fSCBCSE
f(f) -> SCBCSE
S(e) -> CCCBCSE
C(e) -> eSCCBCSE
e(e) -> SCCBCSE
S(d) -> CCCCBCSE
C(d) -> dEaCCCBCSE
d(d) -> EaCCCBCSE
E(c) -> cbaCCCBCSE
c(c) -> baCCCBCSE
b(b) -> aCCCBCSE
C(h) -> hfCCBCSE
h(h) -> fCCBCSE
f(f) -> CCBCSE
C(d) -> dEaCBCSE
d(d) -> EaCBCSE
E(c) -> cbaCBCSE
c(c) -> baCBCSE
b(b) -> aCBCSE
a(a) -> CBCSE
C(h) -> hfBCSE
h(h) -> fBCSE
f(f) -> BCSE
B(c) -> cDCSE
c(c) -> DCSE
D(j) -> jkCCSE
j(j) -> kCCSE
k(k) -> CCSE
C(d) -> dEaCSE
d(d) -> EaCSE
E(c) -> cbaCSE
c(c) -> baCSE
b(b) -> aCSE
a(a) -> CSE
C(h) -> hfSE
h(h) -> fSE
f(f) -> SE
S(d) -> CCE
C(d) -> dEaCE
d(d) -> EaCE
E(c) -> cbaCE
c(c) -> baCE
b(b) -> aCE
a(a) -> CE
C(h) -> hfE
h(h) -> fE
f(f) -> E
E(c) -> cb
c(c) -> b
b(b) -> θ
$($) -> ACCEPTS
```

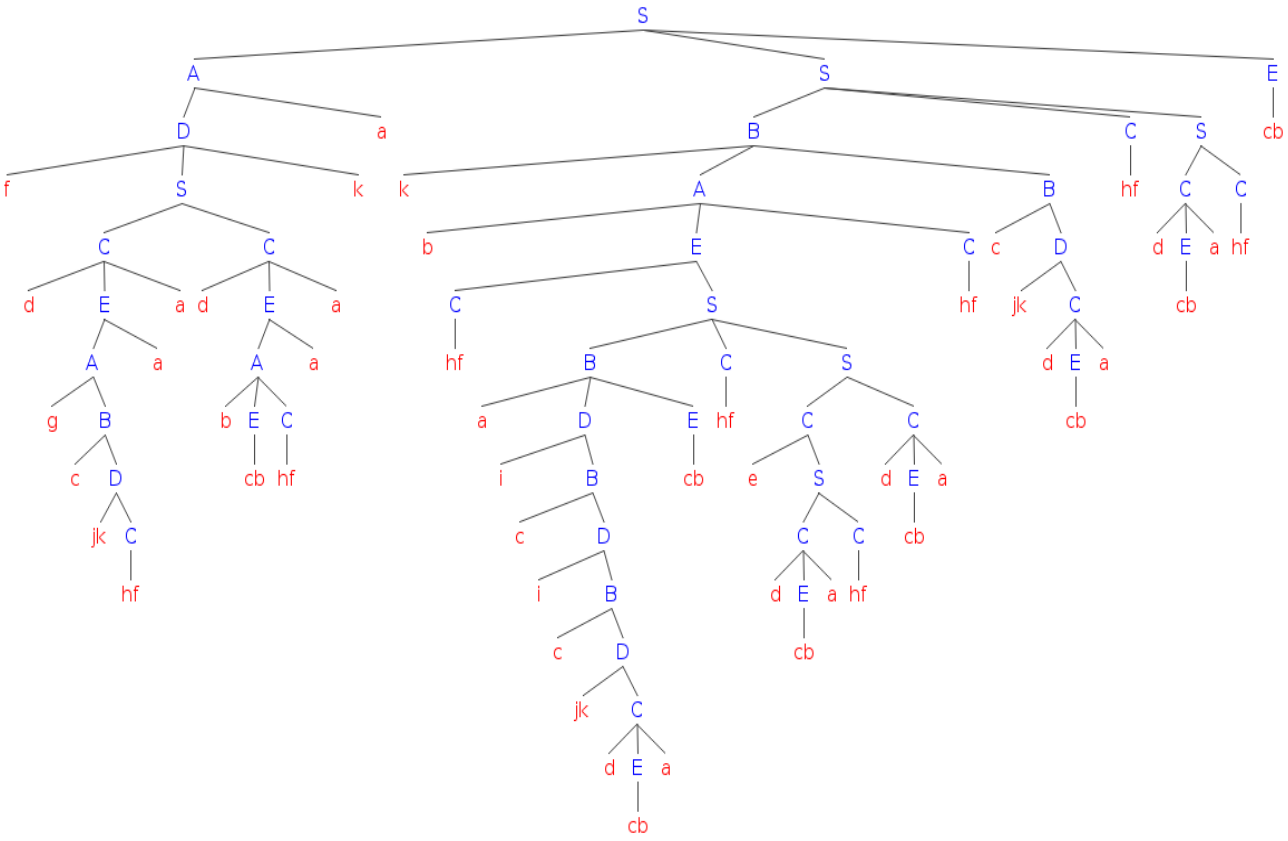
# HW #05 – Grammars – (REV 1)
# CSCI-400 Spring 2013
# INSTRUCTOR SOLUTIONS

Leftmost Derivation

```
S -> ASE : ASE
A -> Da  : DaSE
D -> fSk : fSkaSE
S -> CC  : fCCkaSE
C -> dEa : fdEaCkaSE
E -> Aa  : fdAaaCkaSE
A -> gB  : fdgBaaCkaSE
B -> cD  : fdgcDaaCkaSE
D -> jkC : fdgcjkCaaCkaSE
C -> hf  : fdgcjkhfaaCkaSE
C -> dEa : fdgcjkhfaadEakaSE
E -> Aa  : fdgcjkhfaadAaakaSE
A -> bEC : fdgcjkhfaadbECaakaSE
E -> cb  : fdgcjkhfaadbcbCaakaSE
C -> hf  : fdgcjkhfaadbcbhfaakaSE
S -> BCS : fdgcjkhfaadbcbhfaakaBCSE
B -> kAB : fdgcjkhfaadbcbhfaakakABCSE
A -> bEC : fdgcjkhfaadbcbhfaakakbECBCSE
E -> CS  : fdgcjkhfaadbcbhfaakakbCSCBCSE
C -> hf  : fdgcjkhfaadbcbhfaakakbhfSCBCSE
S -> BCS : fdgcjkhfaadbcbhfaakakbhfBCSCBCSE
B -> aDE : fdgcjkhfaadbcbhfaakakbhfaDECSCBCSE
D -> iB  : fdgcjkhfaadbcbhfaakakbhfaiBECSCBCSE
B -> cD  : fdgcjkhfaadbcbhfaakakbhfaicDECSCBCSE
D -> iB  : fdgcjkhfaadbcbhfaakakbhfaiciBECSCBCSE
B -> cD  : fdgcjkhfaadbcbhfaakakbhfaicicDECSCBCSE
D -> jkC : fdgcjkhfaadbcbhfaakakbhfaicicjkCECSCBCSE
C -> dEa : fdgcjkhfaadbcbhfaakakbhfaicicjkdEaECSCBCSE
E -> cb  : fdgcjkhfaadbcbhfaakakbhfaicicjkdcbaECSCBCSE
E -> cb  : fdgcjkhfaadbcbhfaakakbhfaicicjkdcbacbCSCBCSE
C -> hf  : fdgcjkhfaadbcbhfaakakbhfaicicjkdcbacbhfSCBCSE
S -> CC  : fdgcjkhfaadbcbhfaakakbhfaicicjkdcbacbhfCCBCSE
C -> eS  : fdgcjkhfaadbcbhfaakakbhfaicicjkdcbacbhfeSCCBCSE
S -> CC  : fdgcjkhfaadbcbhfaakakbhfaicicjkdcbacbhfeCCCCBCSE
C -> dEa : fdgcjkhfaadbcbhfaakakbhfaicicjkdcbacbhfedEaCCCBCSE
E -> cb  : fdgcjkhfaadbcbhfaakakbhfaicicjkdcbacbhfedcbaCCCBCSE
C -> hf  : fdgcjkhfaadbcbhfaakakbhfaicicjkdcbacbhfedcbahfCCBCSE
C -> dEa : fdgcjkhfaadbcbhfaakakbhfaicicjkdcbacbhfedcbahfdEaCBCSE
E -> cb  : fdgcjkhfaadbcbhfaakakbhfaicicjkdcbacbhfedcbahfdcbaCBCSE
C -> hf  : fdgcjkhfaadbcbhfaakakbhfaicicjkdcbacbhfedcbahfdcbahfBCSE
B -> cD  : fdgcjkhfaadbcbhfaakakbhfaicicjkdcbacbhfedcbahfdcbahfcDCSE
D -> jkC : fdgcjkhfaadbcbhfaakakbhfaicicjkdcbacbhfedcbahfdcbahfcjkCCSE
C -> dEa : fdgcjkhfaadbcbhfaakakbhfaicicjkdcbacbhfedcbahfdcbahfcjkdEaCSE
E -> cb  : fdgcjkhfaadbcbhfaakakbhfaicicjkdcbacbhfedcbahfdcbahfcjkdcbaCSE
C -> hf  : fdgcjkhfaadbcbhfaakakbhfaicicjkdcbacbhfedcbahfdcbahfcjkdcbahfSE
S -> CC  : fdgcjkhfaadbcbhfaakakbhfaicicjkdcbacbhfedcbahfdcbahfcjkdcbahfCCE
C -> dEa : fdgcjkhfaadbcbhfaakakbhfaicicjkdcbacbhfedcbahfdcbahfcjkdcbahfdEaCE
E -> cb  : fdgcjkhfaadbcbhfaakakbhfaicicjkdcbacbhfedcbahfdcbahfcjkdcbahfdcbaCE
C -> hf  : fdgcjkhfaadbcbhfaakakbhfaicicjkdcbacbhfedcbahfdcbahfcjkdcbahfdcbahfE
E -> cb  : fdgcjkhfaadbcbhfaakakbhfaicicjkdcbacbhfedcbahfdcbahfcjkdcbahfdcbahfcb
```

# HW #05 – Grammars – (REV 1)
# CSCI-400 Spring 2013
# INSTRUCTOR SOLUTIONS

**Parse Tree for Problem #3**



# HW #05 – Grammars – (REV 1)
# CSCI-400 Spring 2013
# INSTRUCTOR SOLUTIONS

**Parse Tree code for Problem #3**

```
[S [A [D f
         [S [C d[E [A g[B c [D jk [C hf]]]]a]a]
            [C d[E [A b[E cb][C hf]]a]a]
         ]
         k
      ]
      a
   ]
   [S [B k[A b[E [C hf]
                 [S [B a[D i[B c[D i[B c[D jk
                                          [C d[E cb]a]
                                       ]
                                    ]
                                 ]
                              ]
                           ]
                           [E cb]
                     ]
                     [C hf]
                     [S [C e[S [C d[E cb]a]
                               [C hf]
                            ]
                         ]
                         [C d[E cb]a]
                     ]
                 ]
             ]
             [C hf]
         ]
         [B c
            [D jk
               [C d[E cb] a]
            ]
         ]
      ]
      [C hf]
      [S [C d
            [E cb]
            a
         ]
         [C hf]
      ]
   ]
   [E cb]
]
```