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

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.

(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;
   }
}
```

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

You do not need to support everything that C supports. For instance, you do not need to permit single-statement loops to omit the enclosing braces and you do not need to support comma-separated statements.

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

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.

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

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?

(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.

(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
```

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

Submission Procedures

Zip up all files into a .zip file named

`CS400_UserID_HW_05.zip`

You make do your work in a text file and use ASCII to draw your parse trees or you may use any other word processor and graphics tools to prepare your work and print them to a PDF file. Please try to avoid scanning hand drawn work as this often makes for large files that don't compress well. But if you need to do this, it is acceptable. Be sure that the scans are clear with good contrast.

You may put your work in a single file or use one file per problem. If the latter, be sure the file name clearly indicates which problem number. I would recommend just appending "_#1" to the zip file name for the first problem and so on.

GRADING RUBRIC – 80 pts

20 - Good Faith effort.

- Problem #1
5 - part (a) – while()
5 - part (a) – for()
5 - part (b) – while()
5 - part (a) – for()

- Problem #2
10 - part (a)
10 - part (b)

- Problem #3
6 - part (a)
6 - part (b)
8 - part (c)

-10 - Improper submission.