

**Homework 1: BNF, EBNF, and Syntax Diagram**  
**Due: Jan 17, 2006 (in class)**  
**References allowed: Textbook and handout**

1. (15 pts) Consider the following BNF grammar rules:

```
<pop> ::= [<bop>, <pop>] | <bop>
<bop> ::= <mop> | (<pop>; <mop>)
<mop> ::= {<mop>} | <hop>
<hop> ::= x | y | z
```

Note that in this grammar, [ , ], ' , ' , ( , ) , ; , { , } , x , y , and z are terminals, whereas <pop> , <bop> , <mop> and <hop> are non-terminals. For each of the strings listed below, indicate **all** the non-terminals that can generate it.

- (a) (x; y)
- (b) [y, (x; z)]
- (c) [[x, y], z], x]
- (d) [(x; [y, z]), [(x; [z, y])]]

2. (5+5 pts) Given the following BNF:

```
<integer> ::= <unsigned> | <sign> <unsigned>
<unsigned> ::= <digits> | <unsigned><digits>
<digits> ::= <digits><digit> | <digit>
<digit> ::= 0 | 1 | ... | 9
<sign> ::= + | -
```

- (i) Convert the grammar into EBNF
- (ii) Show the syntax diagram of the above EBNF.

3.(5+5+5 pts) What kinds of languages are generated by each of the BNF grammars below:

(a).

```
<s> ::= a<s> b b | empty
```

(b).

```
<s> ::= <x> | <y> | empty
<x> ::= a <x> b | empty
<y> ::= a <y> b b | empty
```

(c).

```
<s> ::= <x> | <y>
<x> ::= a <x> b | <x1>
<x1> ::= a <x1> | a
<y> ::= a <y> b b | <y1>
<y1> ::= <y1> b | b
```

4. (5+5 pts) Given the following grammar:

```
<stmt> ::= if <exp> then <stmt> else <stmt>
          | begin <stmt> <end> | print <exp>
<end> ::= end | ; <stmt> <end>
<exp> ::= 0 | 1
```

Show the parse tree for the following string:

(a).

```

if 1 then
  if 0 then
    if 1 then
      if 0 then
        if 1 then
          print 1
        else
          print 0
        end
      else
        print 1
      end
    else
      print 0
    end
  else
    begin
      print 0
    end
  end
else
  print 0
end

```

(b)

```

begin
  print 1;
  if 1 then
    begin
      print 1;
      begin
        print 0;
      end;
      print 0;
    end
  else
    print 1;
  end
end

```

5. (10+5) Given the following EBNF grammar:

```

S ::=  [{}empty{}]
      | id := [expr] expr (+|-) [{}[expr]]
      | if expr then S [S] {;S} {elseif expr then {S;S} } [ else {S;S} S] end
      | loop [S[{}S]]S{;S} end
      | while expr do S {;S} {;S} [S] end

```

Token `id` represents a variable and token `expr` represents an expression. Note that `empty` stands for the empty string.

(i) Rewrite the grammar by removing redundant symbols and EBNF expressions.

(ii) Write a syntax diagram for the EBNF grammar.

(iii) Write a BNF version of this grammar.

6. (10+5) A grammar symbol  $X$  is useless if  $X$  can never appear in a derivation of a string. Write an algorithm for eliminating all productions containing useless symbols from a grammar. Apply your algorithm to the grammar:

```

<S> ::= 0 | <A>
<A> ::= <A> <B>
<B> ::= 1

```

7. (10+5) A single production grammar is one with a single non-terminal on its right side. Write an algorithm to convert a grammar into an equivalent grammar with no single production. apply your algorithm to the following grammar:

```

<statement> ::= declare id <option_list>
<option_list> ::= <option_list> <option> | empty string
<option> ::= <mode> | <scale> | <precision> | <base>
<mode> ::= real | complex
<scale> ::= fixed | floating
<precision> ::= single | double
<base> ::= binary | decimal

```