COMP 4040

Exam 2

Fall 2013

Name: Solutions

Rules:

- No potty breaks.
- Turn off cell phones/devices.
- Closed book, closed note, closed neighbor.

- WEIRD! Do not write on the backs of pages. If you need more pages, ask me for some.

Reminders:

- Verify that you have all pages.
- Don't forget to write your name.
- Read each question carefully.
- Don't forget to answer every question.

1. [10pts] Consider the following ANTLR grammar.

```
grammar BlockLang;

prog : block+ ;
block : '{' (stat | block)* '}' ;
stat : ID ';' ;

ID : [a-zA-Z_$][a-zA-Z0-9_$]* ;
WS : [ \t\r\n]+ -> skip ;
```

On the next page, fill in Java code to complete the parse-tree walker listener class so that it will "pretty print" the parse tree. To clarify, here are two example inputs ("Before") and their respective outputs ("After").

Before:
```
{ helloILoveYou; wont; { youTell; me; } your; name; }
```

After:
```
{
    helloILoveYou;
    wont;
    {
        youTell;
        me;
    }
    your;
    name;
}
```

Before:
```
{ { { foo; } { bar; } } }
```

After:
```
{
    {
        {
            foo;
        }
        {
            bar;
        }
    }
}
```

Print all output to standard output (i.e., using `System.out.print(...)`). Note that a method `getIndent()` is provided for you to use. Hint: Don't forget about the *X*`Context` method `getText()`.

```
public class MyBlockLangListener extends BlockLangBaseListener {
```

```
int indentLevel = 0;
```

```
    public void enterBlock(@NotNull BlockLangParser.BlockContext ctx) {
```

```
System.out.print(getIndent(indentLevel) + "{\n");
++indentLevel;
```

```
    }
    public void exitBlock(@NotNull BlockLangParser.BlockContext ctx) {
```

```
--indentLevel;
System.out.print(getIndent(indentLevel) + "}\n");
```

```
    }
    public void enterStat(@NotNull BlockLangParser.StatContext ctx) {
```

```
System.out.print(getIndent(indentLevel) +
                 ctx.getText() + "\n");
```

```
    }

    // Returns a string of spaces appropriate for the given indent level.
    public String getIndent(int indentLevel) { … elided … }
}
```

2. [10pts] Consider this fragment of a grammar:

expr ⟶ *term term_tail*
term_tail ⟶ *add_op term term_tail* | ϵ
term ⟶ *factor factor_tail*

For this question, you must write Java code that would implement the part of a recursive descent parser associated with this grammar fragment. Do so by filling in the blanks with the lettered items below. All items should be used at least once. Some items should be used more than once. Some lines should be left blank. Note that item f is different from the others in that it is a placeholder for code that I have not given you. Also, note that the pseudocode language below is the same one we used in lecture (keywords bolded).

```
d   h
  a
    f   k
    b   c
        |
d   l   |
  a     |
    f   g
    f   e
    b   c

d   j
  a
    f   i
    b   c
```

a. **case** input_token **of**

b. otherwise

c. parse_error

d. procedure

e. skip

f. *<predict set for this rule>* **:**

g. add_op; term; term_tail

h. expr

i. factor; factor_tail

j. term

k. term; term_tail

l. term_tail

3. [10pts] For this question, you must run the following algorithm for finding FIRST sets and EPS values on the grammar below.

```
-- EPS values and FIRST sets for all symbols:
    for all terminals c, EPS(c) := false; FIRST(c) := {c}
    for all nonterminals X, EPS(X) := if X ⟶ ϵ then true else false; FIRST(X) := ∅
    repeat
        ⟨outer⟩ for all productions X ⟶ Y1 Y2 ... Yk,
            ⟨inner⟩ for i in 1..k
                add FIRST(Yi) to FIRST(X)
                if not EPS(Yi) (yet) then continue outer loop
            EPS(X) := true
    until no further progress
```

This grammar captures an array-free version of the JSON language:

```
grammar JsonLang;

json
    : object
    ;
object
    : '{' pair_list '}'
    ;
pair_list
    : pair pair_tail
    | // empty
    ;
pair
    : STRING ':' value
    ;
pair_tail
    : ',' pair pair_tail
    | // empty
    ;
value
    : STRING
    | NUMBER
    | object
    | 'true'
    | 'false'
    | 'null'
    ;

STRING :  ... ;
NUMBER : ... ;
```

To answer this question, fill out the table on the next page. Record the result of the algorithm's initialization steps in the "Intial" columns, and then record the results of up to 3 iterations of the algorithm's "repeat" loop. Only write in cells that changed, but write the full resulting value of the changed cell. For example, if a given cell is unchanged during the first iteration, then you can leave that cell in the Iter. 1 column blank. If a value is added to a cell that had other values added previously, then write all the values currently associated with that cell.

Don't forget that the algorithm assumes BNF grammars, so in the above ANTLR grammar, you should treat the parts of a rule separated by bars (i.e., '|'s) as separate rules, each with the same left-hand side name.

Rule	Initial		Iter. 1		Iter. 2		Iter. 3	
	FIRST	EPS	FIRST	EPS	FIRST	EPS	FIRST	EPS
json	∅	F			{			
object	∅	F	{					
pair_list	∅	T			STRING			
pair	∅	F	STRING					
pair_tail	∅	T	,					
value	∅	F	STRING, {, NUMBER, true, false, null					
STRING	STRING	F						
NUMBER	NUMBER	F						
{	{	F						
}	}	F						
,	,	F						
:	:	F						
true	true	F						
false	False	F						
null	null	F						

4. [2pts] Which of the following do you not need to compute a PREDICT set?

   a. FIRST set

   (b.) LAST set

   c. FOLLOW set

   d. EPS values

   e. None of the above

5. [2pts] Given a grammar *G* such that *G* has two productions with the same left-hand side, and those two rules' PREDICT sets are {`start`, `begin`} and {`foo`, `start`, `bar`}, respectively, then is *G* an LL(1) grammar?

   a. Yes

   (b.) No

   c. Can't tell. Not enough information

6. [2pts] Given a production A→B, which of the following must PREDICT(A→B) contain? Hint: Don't forget that there may be other rules with A as a left-hand side.

   a. FIRST(A)

   b. FOLLOW(A)

   (c.) FIRST(B)

   d. FOLLOW(B)

   e. None of the above

7. [2pts] What types of things go in a production's PREDICT set?

   a. Grammars

   b. Productions

   c. Non-terminals

   (d.) Terminals

   e. None of the above

8. [2pts] Which broad family of parsers does ANTLR belong to?

   (a.) LL

   b. LR

   c. RL

   d. RR

   e. None of the above

9. [2pts] T or F? Early binding generally makes languages more flexible than does late binding.

   a. True

   (b.) False

10. [2pts] Dynamic scope is determined at …

    a. … language-design time

    b. … language-implementation time

    c. … compile time

    (d.) … run time

    e. None of the above

Consider this code:

```
x : integer
procedure foo
      x := 55;
procedure bar
      x : integer
      foo()
x := 75
bar()
print(x)
```

11. [2pts] If this program is statically scoped, what value is printed? What if it's dynamically scoped?

If statically scoped, 55.
If dynamically scoped, 75.

Consider this Java code:

```
Foo f1 = new Foo();
Foo f2 = f1;
```

12. [2pts] The instance of Foo ...

    a. ... is overloaded

    b. ... has multiple aliases

    c. ... is overridden

    d. ... has multiple copies

    e. None of the above

13. [2pts] Java generics use what type of polymorphism?

    a. Generic polymorphism

    b. Subtype polymorphism

    c. Parametric polymorphism

    d. Duck polymorphism

    e. None of the above

14. [2pts] Which of these is an example of Cambridge Polish notation?

    a. (* (+ 1 3) 2)

    b. (1 + 3) * 2

    c. ((1 3 +) 2 *)

    d. * + 1 3 2

    e. None of the above

15. [2pts] The fact that Algol has no separate notion of statements and expressions (for example, an if-statement can be used as the right-hand side to an assignment statement) is an indication of …

    a. Polymorphism

    b. Functionality

    (c.) Orthogonality

    d. Recursion

    e. None of the above

16. [2pts] In general, does iteration or recursion tend to perform better?

    (a.) Iteration

    b. Recursion

    c. Neither performs better than the other

17. [2pts] What special type of recursion is easy for a compiler to optimize?

    a. Head recursion

    (b.) Tail recursion

    c. Foot recursion

    d. Body recursion

    e. None of the above

18. [2pts] T or F? Using go-to statements in high-level languages is generally considered helpful.

    a. True

    (b.) False

19. [2pts] Which of the following is <u>not</u> a set of rules that a type system typically has?

    a. Type inference

    b. Type compatibility

    (c.) Type defense

    d. Type equivalence

    e. None of the above

20. [2pts] T or F? In most languages, if a type is expected in a particular context, then the language requires that any value used in that context must be type equivalent.

    a. True

    (b.) False

Consider this snippet of code:

```
String s = "10";
int sum = 5 + 6 + s;
```

21. [2pts] If all of the above addition is considered integer addition by the language, and the code works the way you'd hope anyway, then it must be an example of ...

    a. ... dynamic binding

    b. ... static typing

    c. ... universal reference types

    (d.) ... type coercion

    e. None of the above

Consider this Java code in which Y is a subclass of X:

```
/* 1 */  X x = new Y();
/* 2 */  Y y = x;
/* 3 */  Object o = x;
```

22. [2pts] Does the above contain an error? If so, explain what it is.

Yes. In line 2, the X reference x cannot be a r-value in an assignment statement with a subclass Y reference as the l-value.

23. [2pts] Which one of the above lines requires dynamic typechecking? (Careful!!)

    a. Line 1

    b. Line 2

    c. Line 3

    (d.) None of the above