COP-4020
PROGRAMMING LANGUAGE CONCEPTS
Instructor: Manuel Bermúdez
Spring 2018

MIDTERM EXAM II
Closed Books, Closed Notes, 50 minutes

Problem 1 20 (20p.)
Problem 2 10 (10p.)
Problem 3 30 (30p.)
Problem 4 20 (20p.)
Problem 5 20 (20p.)

SCORE 100 (100p.)

Name: SOLUTION

UFID: ______________________________

Note: Turn in your work on this exam only
The following string-to-tree transduction grammar generates regular expressions, over the vocabulary \( \Sigma = \{a,b\} \).

Expression  
\[ \rightarrow \text{Expression} \mid \]  
\[ \rightarrow \text{Term} \]

Term  
\[ \rightarrow \text{List} \]  
\[ \rightarrow \text{Term} \]  
\[ \rightarrow \text{List} \]

List  
\[ \rightarrow \text{List} \text{'}list\text{'} \text{Factor} \]  
\[ \rightarrow \text{Factor} \]

Factor  
\[ \rightarrow \text{Factor} \text{'}*\text{'} \]  
\[ \rightarrow \text{Factor} \text{'}+\text{'} \]  
\[ \rightarrow \text{Factor} \text{'}?\text{'} \]  
\[ \rightarrow \text{Primary} \]

Primary  
\[ \rightarrow \text{'}(\text{Expression} \text{'}\text{')} \]  
\[ \rightarrow \text{'}a\text{'} \]  
\[ \rightarrow \text{'}b\text{'} \]

**Problem 1 (20 pts.)** Draw the Derivation Tree for the following input string. You may abbreviate Expression as E, Term as T, etc.

\[(a*ba+) \text{ list } (ab) + a?\]
Problem 2 (10 pts.) Draw the Abstract Syntax Tree for the same input as in part 1.

[Diagram of an abstract syntax tree with nodes labeled 'list', 'cat', 'a', 'b', and 'a'.]
Problem 3 (30 pts.) Write (pseudo) code for a Recursive Descent parser for this grammar. Include "Build_Tree()" statements to build the Abstract Syntax Tree, in bottom-up fashion, for the original grammar.

```plaintext
proc E()
  T;
  while (NT = '11')
    Read ('11'); T; BT ('11', 2)
  end
end

proc T()
  L;
  if (NT in ['1', '1a', '1b'])
    T; BT ('cat', 2)
  end
end

proc L()
  F;
  while (NT = '11F')
    Read ('11F'); F; BT ('11F', 2)
  end
end

proc F()
  P;
  while (NT in ['14', '1+', '12'])
    Token = NT; Read (NT); BT (Token, 2)
  end
end
```
Problem 4 (20 pts.) Show the sequence of calls to Build_Tree() that are carried out by your recursive descent parser, for the same input string as in Part 2.

```
BT ('a', 0)  BT ('cat', 2)
BT ('*', 1)  BT ('+', 1)
BT ('b', 0)  BT ('list', 2)
BT ('a', 0)  BT ('a', 0)
BT ('+', 1)  BT ('?', 1)
BT ('cat', 2) BT ('cat', 2)
BT ('a', 0)  
BT ('b', 0)
```

Problem 5 (20 pts.) In the above grammar, the " (empty) concatenation operator is right associative. Rewrite the production rules for Term so that the concatenation operator becomes non-associative (neither right nor left). The AST for a Term should consist of either a) the AST for a single List, or b) a 'cat' node with two or more List trees under it. Write the new version of the procedure Term for your recursive descent parser.

```
Term → List List + =⇒ 'cat'
        → List
```

**proc T();**
```
L();  N=1;
while (NT in ['(', 'a', 'b'])
L();  N+=1;
end;
if (N>1) BT ('cat', N);
end;
```