COMP 301 : PROGRAMMING LANGUAGES
PROBLEM SET 6

Due date: Dec 17, 2009 Thursday 09:30 am

Please note:

- In this course, there is a 1 day late policy: You can still submit your homework within 24 hours past the deadline, but will lose 25% of your grade. Anything submitted after that will not be graded.

- Name your file as homework#_novellname.doc (you can submit docx files as well)
  - Example: hw1_bdevrim.docx
- Submit your homework to: F:\COURSES\UGRADS\COMP\COMP301\HOMEWORK

- For each problem set, your grade will be converted to a plus, half a plus or a minus, based on the following scheme:
  - 100 ≥ grade ≥75 → plus
  - 75 > grade ≥ 50 → half a plus
  - 50> grade ≥ 0 → minus

- Following each problem set, there will be an in class quiz that covers the same material. You can bump up your homework grade if you manage to get a better score in the quiz. Grading for quizzes is the same as the problem sets.

- Any instance of cheating/plagiarism will be referred to the disciplinary committee. All involved parties (e.g., recipient AND the receiver of assistance) will receive an F as their final grade.

Problem 1\(^1\): Add to the defined language a facility that adds a cond expression. Use the grammar

\[
\text{Expression} ::= \text{cond \{Expression ==>} Expression\}}^*\text{end}
\]

In this expression, the expressions on the left-hand sides of the ==>'s are evaluated in order until one of them returns a true value. Then the value of the entire expression is the value of the corresponding right-hand expression. If none of the tests succeeds, the expression should report an error.

Problem 2\(^2\): Extend the language of this section to include procedures with multiple arguments and calls with multiple operands, as suggested by the grammar

\[
\text{Expression} ::= \text{proc \{(Identifier)^*(,)} Expression\\n\text{Expression} ::= (Expression \{Expression\}^*)
\]

---

\(^1\) See EOPL, p.73, Exercise 3.12
\(^2\) See EOPL, p.80-81, Exercise 3.21
**Problem 3**

The tricks of the previous exercises can be generalized to show that we can define any recursive procedure in PROC. Consider the following bit of code:

```plaintext
def makerec = proc (f)
def d = proc (x)
    proc (z) ((f (x x)) z)
in d proc (n) ((f (d d)) n)
in let maketimes4 = proc (f)
    proc (x)
    if zero?(x)
    then 0
    else -((f -(x,1)), -4)
in let times4 = (makerec maketimes4)
in (times4 3)
```

Show that it returns 12.

**Problem 4**

Extend the lexical address translator and interpreter to handle `cond` from problem 1.

---

3 See EOPL, p.81, Exercise 3.25
4 See EOPL, p.101, Exercise 3.38