University of Toronto
csc324—Principles of Programming Languages, Fall 2002

Problem Set 2—CFGs and Scheme

Due: *Monday, 7 October at noon*
Weight: *4% of your course grade.*

1. Java expressions.

   (a) Write a CFG in BNF for a subset of expressions in Java. Your grammar should handle the basic arithmetic operators, the ternary if-else operator (? :), as well as the dot notation for access to member variables of an object, and method calls.

   The following table lists these operators, in order from highest precedence at the top to lowest precedence at the bottom. Make sure your grammar respects these precedence rules, and the specified associativity.

Description	Operators	Associativity
Member variable/method access	.	Left
Unary prefix operators	$+, -$	N/A
Multiplicative operators	$*, /, \%$	Left
Additive operators	$+, -$	Left
Ternary if-else operator	? :	Middle and right

   Assume that the non-terminals <literal> and <identifier> are already defined. Use <identifier> for object names, class names, member variable names, member method names—*i.e.*, any kind of identifier that might appear in a Java expression.

   Member variable access and member method calls differ from each other in that a method call requires parentheses and a list of zero or more arguments, separated by commas if there are two or more. The arguments to a method call can be any kind of expressions.

   The ternary if-else operator is described as having "middle and left" associativity. (Note that this does not conform to how Java really works.) Middle associativity means that an expression like `a ? b ? c : d : e` is allowed and should be interpreted to mean `a ? (b ? c : d) : e`

   Right associativity, for a ternary operator, means that an expression like `a ? b : c ? d : e` is allowed and should be interpreted to mean `a ? b : (c ? d : e)`

   Your grammar should also allow the use of parentheses to override the default grouping of arithmetic operations, but not within member variable and method access. Thus, `a * -(b + c)` and `a.b(3,4).c` are allowed, but `a.(b.c)` is not.

   (b) Draw a parse tree for each of the following expressions, using your grammar. In your parse trees, you should assume that you can go directly from <literal> to any number in the expressions, and directly from <identifier> to any identifer in the expressions.

   i. `a * -b.c(3 * 5, 1).d().e`
   ii. `(a ? b : c) ? d : e ? f : g`
   iii. `a.b(-3 * 4 + c.d)`

2. We can represent a complex number in Scheme as a list of two numbers, (r c), where r is the real part and c is the imaginary part. For example, $3 + 2i$ would be represented as (3 2). Write Scheme procedures to handle the following complex number operations:

   (a) (*sum* c_1 c_2) returns the sum of the two complex numbers.
   Precondition: c_1 and c_2 are complex numbers.

   (b) (*mult* c_1 c_2) returns the product of the two complex numbers.
   Precondition: c_1 and c_2 are complex numbers.

   (c) (*power* c n) returns the nth power of the complex number c.
   Precondition: c is a complex number and n is a non-negative integer.

   (d) (*abs* c) returns the magnitude of complex number c.
   Precondition: c is a complex number.

3. Write Scheme procedures to perform the following operations on lists.

   (a) (*replace* a b *lst*) returns *lst* with every occurrence of a in *lst* (at the top level only) replaced by b. (Use *eq?* to test for equality.)
   Precondition: *lst* is a list.
   Example: (*replace* 'x '3 '(f x (+ x x))) returns (f 3 (+ x x)).

   (b) (*deep-replace* a b c) returns c with every occurrence of a in c (at any level) replaced by b.
   Precondition: none.
   Example: (*replace* 'x '3 'x) returns 3.
   Example: (*replace* 'x '3 '(f x (+ x x))) returns (f 3 (+ 3 3)).

   (c) (*deep-properlist?* a) holds if all lists and sublists in a are proper lists.
   Precondition: none.
   Example: (*deep-properlist?* 'x) returns #t. (Since there are no improper lists in x.)
   Example: (*deep-properlist?* '(a b c (d . e) f)) returns #f. (MIT Scheme will display () if you test your code with it.)
   Example: (*deep-properlist?* '(a b c (d e) f)) returns #t.

   (d) (*deep-swap-first-third* a) swaps the first and third element of every list contained in a which has at least three elements.
   Precondition: none.
   Example: (*deep-swap-first-third* 'x) returns x.
   Example: (*deep-swap-first-third* '(a (b c) (d e f) g)) returns ((f e d) (b c) a g).
   Example: (*deep-swap-first-third* '((a b c) (d () f))) returns ((c b a) (f () d)).

Silent Policy

A silent policy will take effect 24 hours before this assignment is due. This means that no question asked after noon on Sunday, 6 October will be answered, whether it is asked on the newsgroup, by e-mail or in person.

Handing It In

You can hand in this assignment on paper in the drop box (Sandford Flemming, 2nd floor, by the overpass to Pratt) or give it to your TA **before** the beginning of the day section tutorial. You may also hand it in electronically on CDF using the following command:

```
submit -c csc324h -a PS2 ps2.txt
```

We will accept your submission in plain ASCII (in which case your submission file must be called `ps2.txt`), in PostScript (`ps2.ps`), or in PDF (`ps2.pdf`). We recommend ASCII—the electronic submission option is meant to be convenient and easy, not to create additional work for you, so don't waste your time on a fancy presentation. You may draw your parse trees using any reasonable representation, as long as it is clear.

Type "`man submit`" on CDF for more information about the submit command.