University of Toronto
CSC324 – Principles of Programming Languages, Winter 2007

Assignment # 1

Due Date

This assignment is due on **Friday January 26, 2007** at **noon**.

This assignment is out of **60** marks and is work **5%** of your final grade.

Silent Policy

A silent policy will take effect 24 hours before this assignment is due. This means that no question will be answered, whether it is asked on the newsgroup, by email, or in person.

Handing in this Assignment

You must hand in this assignment on paper in the CSC324 drop box in the Bahen Computer Lab, BA2220, or electronically on CDF:

```
https://www.cdf.toronto.edu/students
```

We will accept your submission in plain ASCII (in which case your submission file must be called a1.txt), in PostScript (a1.ps), or in PDF (a1.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. It is your responsibility to ensure that your assignment is readable when printed from CDF. Unreadable portions of your submission will receive 0 marks. If you submit the assignment both on paper and electronically, only the electronic submission will be graded. Your electronic submission must contain your first name, last name, and student number, all clearly labeled.

Clarification Page

It is your responsibility to consult the AI Clarification Page

```
http://www.cs.toronto.edu/~sheila/assns/A1/a1_faq.html
```

and the newsgroup for any corrections or clarifications to the assignment.

Please include the following information with your submission.
(No signature is necessary if you submit electronically.)

Last Name:_____ **First Name:**_____

Student #:_____ **CDF Login:**_____

Email:_____ **Date & Time:**_____

Grace days used for this assignment: _____

*All answers are my own, written in isolation, without help from others. This submission is in accordance with the University of Toronto Code of Behaviour on Academic Matters (*http://www.artsandscience.utoronto.ca/ofr/calendar/rules.htm#behaviour).

Signature: _____

University of Toronto
CSC324 – Principles of Programming Languages, Winter 2007

Assignment # 1

1. [28 marks total, 4 marks each]

For each of the following languages,

- provide a context-free grammar in BNF that generates all strings in the language and no other strings, or say it cannot be done, and
- provide a regular expression that accepts all strings in the language, or say it cannot be done.

If you claim that a context-free grammar / regular expression cannot be provided, you do not have to explain why.

a) All binary strings that contain **101** as a substring.

b) All binary strings that do not contain **101** as a substring.

c) All palindromes over {**x**,**y**,**z**}.

d) All strings over {**a**,**b**} with twice as many **a**'s as **b**'s.

e) All strings over {**a**,**b**} in which every **a** is followed by a **b** (not necessarily immediately).

f) All strings over {**a**,**b**,**c**,**d**} of the form $\mathbf{a}^i\ \mathbf{b}^j\ \mathbf{c}^k\ \mathbf{d}^l$, for positive naturals i,j,k,l, such that $i=k$ and $j=l$.

g) All strings over {**a**,**b**,**c**} of the form $\mathbf{a}^i\ \mathbf{b}^j\ \mathbf{c}^k$, for positive naturals i,j,k,l, such that $j=2i+k$.

2. [6 marks total] Consider the following regular expression:

abb*(c + d*) + (a*bc*bc* + c*)*b

Specify a context-free grammar in BNF that generates all the strings accepted by this regular expression and no other strings. Include all components of the grammar. Simpler grammars will be given more marks.

3. [12 marks total] Consider the following grammar:

start symbol: <expr>

production rules:

<expr> ::= <pred> | <quant>

<quant> ::= exists <vars> . <conj> | <conj>

<conj> ::= <neg> and <conj> | <neg>

<neg> ::= not <pred> | <pred>

<pred> ::= P | Q | R

<vars> ::= <var> | <var> , <vars>

<var> ::= x | y | z

a) **[1 mark]** What are the terminals in this grammar?

b) **[1 mark]** What are the non-terminals in this grammar?

c) **[2 mark]** Give two different strings generated by this grammar. One of the strings must contain all terminals of the grammar.

d) **[4 mark]** Is there a string generated by the grammar using two different derivations? If your answer is yes, show the derivations. If your answer is no, explain why not.

e) **[4 mark]** Is the grammar ambiguous? If you think the grammar is ambiguous, prove it. If you think it is not, explain why.

4. [14 marks total]

Your task is to define an unambiguous context-free grammar in BNF that generates all valid program statements in the programming language SIMPLE and no other strings. A valid program statement in SIMPLE can be an assignment statement, an if-then-else statement or an **ok** statement. The syntax of these statements is described informally below.

- an if-then-else statements has the following syntax:
  if boolean_expression **then** valid_program_statement **else** valid_program_statement
- an assignment statement has syntax x**:=**E meaning "x is assigned the value E", where x is a variable and E is either a boolean expression or an arithmetic expression.
- program statement **ok**, represented as such, has meaning "do nothing".
- parentheses may be used for grouping of program statements and expressions.
- valid boolean expressions may be constructed using the following three boolean operators: negation (**not**), conjunction (**and**), and disjunction (**or**).
  1. an atomic boolean expression is a boolean expression.
  2. **not** E and **(**E**)** are boolean expressions for a boolean expression E.
  3. A **and** B and A **or** B are boolean expressions for boolean expressions A and B.
- Valid arithmetic expressions may be constructed using the following five arithmetic operators: the unary operator **–** and binary operators **+**, **–**, *****, and **/**
  1. a number is an arithmetic expression.
  2. **–** E and **(**E**)** are arithmetic expressions for an arithmetic expression E.
  3. A**+**B , A**–**B , A*****B , and A**/**B are arithmetic expressions for arithmetic expressions A and B.

Assume that there are production rules for generating atomic boolean expressions, numbers, and variables starting from non-terminals <**boolean**>, <**number**>, and <**variable**>, respectively. So you may refer to these non-terminals in your grammar, but you need not define them.

By way of illustration, suppose that <**number**> generates integers 1 to 5, <**boolean**> generates boolean expressions P, Q, and R , and <**variable**> generates variables x and y . Then

If P **then** (**if** Q **or** R **then** *x* := 1 + 5 * (2 + 3) **else** *y* := P) **else ok**

is a valid program statement in SIMPLE.

In addition, the following properties should hold:

- The precedence order of boolean operations is (from highest to lowest priority):
  1) **not**
  2) **and**
  3) **or**

  Parentheses override the precedence order.
- The precedence order of arithmetic operations is (from highest to lowest priority):
  1) unary –
  2) ***** and /
  3) + and –

  Parentheses override the precedence order.
- **and**, **or**, +, and – are left-associative, / and – are right-associative.