Bengal Engineering and Science University, Shibpur
ME CST First Semester Examination, December 2011
Department of Computer Science and Technology
Principles of Programming Languages (CST - 905)

F.M.: 70 TIME : 3 hrs

- **Attempt any five questions.**
- **All questions carry equal marks.**
- **Answers should be in your own words as far as practicable.**
- **Make your own assumptions as and when necessary and state them at proper places.**

1. (a) State with example(s) the properties a programming language should have.
   (b) What do you mean by Hierarchical Data and how is it supported in Scheme Lisp?
   (c) In the context of data abstraction what is abstraction barrier and why is it important? [4+6+4]

2. (a) Explain with a complete case study the general motivation behind supporting generic operations.
   (b) Explain with example(s) the concepts of **data-directed programming** and its advantages.
   (c) How can generic operations be supported in Scheme Lisp programs? [6+4+4]

3. (a) Why is assignment operator needed for modeling the real world in terms of objects? What is the cost of introducing assignment in a programing language?
   (b) Write a recursive procedure in Scheme Lisp to concatenate two lists.
   (c) Construct a Pushdown automaton to accept the language $\{\omega \in \{a,b\}^* | \omega = \omega^R\}$, ω^R is the reverse of ω. [6+4+4]

4. (a) What are higher order procedures and explain with examples the role of higher order procedures in a programming environment.
   (b) Explain the usage of *Lambda* in Scheme Lisp in building up abstractions with procedures.
   (c) Explain with examples the effect of the absence of similar construct in *C* language.
   (d) Explain the concept of **Turing computable** functions from natural numbers to natural numbers. [4+3+4+3]

5. (a) Let (*operation* $operand_1$ $operand_2 \ldots operand_n$) be the syntax of a combination in Scheme Lisp representing pure arithmatic expressions. That is, *operation* is an arithmatic operator (+ or - or / or *) and $operand_i$, $1 \leq i \leq n$, is either a numeric constant or another arithmatic expression. Propose a context-free grammar for such combinations of Scheme Lisp.
   (b) Propose a comprehensive scheme using LEX and YACC or similar tools for constructing a flow-graph for any program in a programming language like C. [4+10]

6. (a) Construct a grammar $G = (V, \Sigma, R, S)$ to compute the function $f : \Sigma^* \rightarrow \Sigma^*$ defined as $f(\omega) = \omega\omega$. That is, $x_1\omega y_1 \Rightarrow_G^* x_2\omega\omega y_2$ (under the grammar G the string $x_1\omega y_1$ derives $x_2\omega\omega y_2$), $x_1, y_1, x_2, y_2 \in V^*$ and $\omega \in \Sigma^*$. Assume $\Sigma = \{a, b\}$. Justify that G indeed computes f.

   (b) Construct unrestricted grammars that generate each of the following languages.

   i. $\{a^{i*j} | i, j > 1\}$

   ii. $\{a^{3^i} | i \geq 0\}$

[5+ (4 + 5)]

7. Justify the following.

   (a) Programs can be written in Scheme Lisp to compute Primitive Recursive Functions.

   (b) Not all languages can have Finite Representations.

   (c) Prefix Representation is intutively more expressive than infix representation.

   (d) Using Lex in addition to Yacc generates more efficient parser than using YACC alone.

[5+2+3+4]