CSCE 330: Programming Language Structures

1. Course number and name: CSCE 330: Programming Language Structures

2. Credit: 3-hrs; Contact: 3 lectures of 50 minutes each or 2 lectures of 75 minutes each per week

3. Instructor: Fall 2010: Marco Valtorta

4. Text book: Michael L. Scott. *Programming Language Pragmatics*, third edition, Morgan-Kaufmann, 2009

   Graham Hutton. *Programming in Haskell*. Cambridge University Press, 2007

5. Specific course information
   a. Catalog description: Formal specification of syntax and semantics; structure of algorithms; list processing and string manipulation languages; statement types, control structures, and interfacing procedures.
   b. Prerequisites: CSCE 240 and MATH 374
   c. Required in CS curricula

6. Specific goals for the course
   a. Specific outcomes of instruction:
      - Categorize a language as imperative (procedural), functional (applicative) or declarative (logic).
      - Generate and use syntax descriptions in EBNF.
      - Write code in a functional language (e.g., Haskell).
      - Write code in a logic language (e.g., Prolog).
   b. Relation of course outcomes to Student Outcomes: CE: see page 2; CS & CIS: see page 3

7. Topics covered and approximate weight (14 weeks, 3 hours/week, 42 hours total)
   1. Evolution of major programming languages (2 hours)
   2. Formal description of programming language syntax (4 hours)
   3. Denotational semantics (1 hours)
   4. Interpreters, compilers, assemblers (2 hours)
   5. Data abstractions (3 hours)
   6. Control abstractions (2 hours)
   7. Run-time behavior of programs and procedural semantics (3 hours)
   8. Programming environments (3 hours)
   9. Functional languages (14 hours)
   10. Logic languages (8 hours)

c.

Computer Engineering

Relation of Course Outcomes to EAC Student Outcomes*

Course Outcomes (CE)	Student Outcomes											
	(a) apply knowledge of mathematics, science, and engineering	(b) design and conduct experiments, … interpret data	(c) design a system, component, or process to meet desired needs …	(d) function on multidisciplinary teams	(e) identify, formulate, and solve engineering problems	(f) an understanding of professional and ethical responsibility	(g) communicate effectively	(h) the broad education to understand the impact of engineering solutions …	(i) a recognition of the need for, and an ability to engage in life-long learning	(j) a knowledge of contemporary issues	(k) use the techniques, skills, and modern engineering tools ….	(CE) demonstrate knowledge of discrete mathematics [CE]
Criteria	a	b	c	d	e	f	g	h	i	j	k	CE
1. Categorize a language as imperative (procedural), functional (applicative) or declarative (logic).			1								1	
2. Generate and use syntax descriptions in EBNF.	1		1		1							1
3. Write code in a functional language (e.g., Haskell)	2		3		2				2	1	3	1
4. Write code in a logic language (e.g., Prolog).	2		3		3				2		3	1

* 3 = major contributor, 2 = moderate contributor, 1 = minor contributor; blank if not related

d.

Computer Science & Computer Information Systems

Relation of Course Outcomes to CAC Student Outcomes*

Course Outcomes (CS & CIS)	Student Outcomes											
	All									CS		CIS
	(a) apply knowledge of computing and mathematics appropriate to the discipline	(b) analyze a problem, and identify and define the computing requirements …	c) design, implement, and evaluate a computer-based system, …	(d) function effectively on teams to accomplish a common goal	(e) An understanding of professional, ethical, legal, … responsibilities	(f) communicate effectively with a range of audiences	(g) analyze the local and global impact of computing on … society	(h) Recognition of the need for … continuing professional development	(i) current techniques, skills, and tools necessary for computing practice	(j) apply mathematical foundations, algorithmic principles, and CS theory …	(k) apply design and development principles	(j) An understand processes that support the information systems environment.
Criteria	a	b	c	d	e	f	g	h	i	j	k	j
1. Categorize a language as imperative (procedural), functional (applicative) or declarative (logic).		1	1									
2. Generate and use syntax descriptions in EBNF.	1		1						2	2		
3. Write code in a functional language (e.g., Haskell)	3	1	3					2	3	2	1	1
4. Write code in a logic language (e.g., Prolog).	3	1	3					2	1	2	1	1

* 3 = major contributor, 2 = moderate contributor, 1 = minor contributor; blank if not related