TEC | Tecnológico de Costa Rica

Programa del curso CE-3104

Lenguajes, Compiladores e intérpretes

Escuela de Ingeniería en Computación
Área de Ingeniería en Computadores.

[Última revisión de la plantilla: 05 mayo de 2015]

TEC | Tecnológico de Costa Rica

I parte: Aspectos relativos al plan de estudios

1 Datos generales

Nombre del curso:	CE-3104 Lenguajes, compiladores e intérpretes
Código:	CE-3104
Tipo de curso:	Teórico / Práctico
Electivo o no:	No
Nº de créditos:	4
Nº horas de clase por semana:	4
Nº horas extraclase por semana:	12
% de las áreas curriculares:	-
Ubicación en el plan de estudios:	Curso del 6er semestre de la carrera “Ingeniería en Computadores”
Requisitos:	CE-2103 Algoritmos y Estructuras de Datos II
Correquisitos:	Ninguno
El curso es requisito de:	CE-3102 Análisis numérico para ingeniería CE-3101 Bases de Datos
Asistencia:	Obligatoria
Suficiencia:	Sí
Posibilidad de reconocimiento:	No
Vigencia del programa:	2 Semestre 2022

2 Descripción general

Este curso introduce los cuatro paradigmas de programación principales y sus lenguajes más representativos. Además introduce los principios y métodos utilizados en el diseño e implementación compiladores e intérpretes para dichos lenguajes.

3 Objetivos

Objetivo(s) del curso	Atributo(s) correspondiente(s)	Nivel de desarrollo de cada atributo que se planea alcanzar: Inicial - I, intermedio - M o avanzado - A
1. Presentar principios y métodos para implementar lenguajes de programación.	CB/AP/DI	Inicial
2. Escribir de forma sistemática, compiladores e intérpretes para lenguajes de programación.	CB/AP/DI	Inicial
3. Aplicar los principios estudiados, mediante: 3.1. La construcción de partes de un compilador para un lenguaje de programación imperativo no trivial. 3.2. Una implementación prototipo de un lenguaje (funcional o imperativo) simple vía un intérprete.	CB/AP/DI	Inicial
4. Estudiar los conceptos fundamentales de los lenguajes de programación.	CB/AP/DI	Inicial
5. Estudiar los cuatro paradigmas principales de programación.	CB/AP/DI	Inicial
6. Programar en cada uno de los cuatro paradigmas principales de programación, así como conocer otros lenguajes para representar conceptos que no tengan los programas ejemplos.	CB/AP/DI	Inicial
7. Comprender algunos principios generales de diseño de lenguajes.	CB/AP/DI	Inicial
8. Ofrecer criterios para comparar, evaluar y seleccionar lenguajes de programación.	CB/AP/DI	Inicial
9. Dar criterios para discernir cual es el lenguaje de programación más adecuado para un problema dado.	CB/AP/DI	Inicial

4 Contenidos

1. Introducción

1.1. Niveles de lenguajes.
1.2. Procesadores de lenguajes.
1.3. Especificación de lenguajes.
1.4. Compiladores.
1.5. Intérpretes.
1.6. Ensambladores.

1.7. Máquinas reales y abstractas.
1.8. Portabilidad.
1.9. Bootstrapping.

2. Compilación
2.1. Estructura del proceso de compilación.
2.2. Pasadas.
2.3. Opciones de diseño de un compilador.
2.4. Ejemplo completo de un compilador.

3. Etapas de análisis
3.1. Análisis léxico:
3.1.1. Generalidades del análisis léxico.
3.1.2. Manipulación del texto fuente.
3.1.3. Autómatas.
3.1.4. Implementación manual de análisis léxico.
3.1.5. Tratamiento de errores léxicos.
3.2. Análisis sintáctico:
3.2.1. Reconocimiento sintáctico.
3.2.2. Gramáticas libres de contexto.
3.2.3. Reconocimiento descendente: métodos manuales y por tablas (reconocimiento predictivo).
3.2.4. Construcción de árboles sintácticos.
3.2.5. Reconocimiento ascendente: métodos por tablas y manuales.
3.2.6. Tratamiento de errores sintácticos.
3.3. Análisis de contexto:
3.3.1. Identificación.
3.3.2. Tablas de símbolos.
3.3.3. Validación de tipos.
3.3.4. Diseño de analizadores de contexto.
3.3.5. Tratamiento de errores contextuales.
3.3.6. Etapas de síntesis
3.3.7. Estructuras en tiempo de ejecución:
3.3.8. Representación de datos.
3.3.9. Evaluación de expresiones.
3.3.10. Asignación de memoria.
3.3.11. Acceso a objetos no locales.
3.3.12. Rutinas: paso de parámetros, ligas estáticas, argumentos, recursión.
3.3.13. Máquinas abstractas de pila.

4. Interpretación recursiva directa.
4.1. Interpretación de un lenguaje imperativo.
4.2. Interpretación de un lenguaje funcional.

5. Programación orientada a objetos. [Lenguaje Smalltalk. Otros lenguajes ilustrativos: Eiffel, BETA, C++, Object Pascal (Delphi)]
5.1. Objetos y Mensajes
5.2. Expresiones y aritmética
5.3. Clases y Métodos
5.4. Instancia y tipos de variables
5.5. Herencia y polimorfismo
5.6. Jerarquía de clases
5.7. Colecciones
5.8. Principios de diseño
5.9. Bloques de código y mensajes en cascada
5.10. Diferencias con lenguajes basados en OO [lenguaje ilustrativo: Visual Basic]

6. Programación funcional. [Lenguaje: Lisp o Scheme. Otros lenguajes ilustrativos: Standard ML, CaML o Haskell]
6.1. Principios de diseño y programación funcional.
6.2. Streams (evaluación perezosa).
6.3. Programación funcional con tipos.
6.4. Polimorfismo paramétrico.

7. Programación lógica. [Lenguaje: Prolog o Turbo Prolog. Otros lenguajes ilustrativos: Prolog III, Gödel]
7.1. Relaciones vs funciones
7.2. Hechos y consultas
7.3. Cálculo de predicados
7.4. Dominios, datos compuestos y listas.
7.5. Unificación.
7.6. Control de flujo:
7.6.1. Backtracking y orden de descripción.
7.6.2. Corte y Fail.

8. Elementos avanzados de lenguajes de programación: [Lenguajes: occam, POOL, C-Linda, Orca].
8.1. Concurrencia, paralelismo y distribución.
8.2. Sistemas de tipos.

8.3. Ligas entre programas de distintos lenguajes.
8.4. Elementos del diseño de lenguajes de programación.
8.5. Evaluación y selección de lenguajes de programación.

II parte: Aspectos operativos

5 Metodología de enseñanza y aprendizaje

Se emplearán técnicas de clases magistrales por parte del profesor, en donde se desarrollarán los aspectos teóricos y prácticos más relevantes de los diferentes temas. Además se combinarán con una alta participación por parte de los estudiantes durante el transcurso de las lecciones, por medio de llamadas orales, respuestas a casos en la pizarra y de trabajos en grupo.

Se presupone que el alumno profundiza los temas abordados en la clase en las lecturas recomendadas por el profesor y que el estudiante será responsable de desarrollar los diferentes entregables que se asignen en el curso.

El profesor asumirá el papel de facilitador y el estudiante tendrá la mayor responsabilidad de su progreso.

6 Evaluación

Lenguajes	
Proyecto de investigación	5%
Pruebas cortas	5%
Examen	10%
Tareas cortas	5%
Proyectos programados	25%
Total	**50%**

Compiladores e Intérpretes	
Proyecto de investigación	5%
Pruebas cortas	5%
Examen	10%
Tareas cortas	5%
Proyectos programados	25%
Total	**50%**

7 Bibliografía

[1] Aho; Sethi; Ullman. Compilers: principles, techniques and tools. Reading: Addison-Wesley, 1986. [Existe en espa nol]
[2] Backhouse. Syntax of programming languages. Prentice-Hall, 1979.
[3] Borland International. Turbo Prolog Reference Guide V 2.0. Borland International, 1988.
[4] Bornat. Understanding and writing compilers. Macmillan, 1979.
[5] Digitalk Inc. Smalltalk V: Tutorial and programming Handbook. Digitalk Inc. 1986.
[6] Fischer; LeBlanc. Crafting a compiler. Benjamin Cummings, 1988.
[7] Friedman. From Babbage to Babel and beyond: A brief history of programming languages. Computer Language 17, 1992. pp. 1-17.
[8] Friedman, Daniel y Felleisen, Matthias. The Little LISPer. Massachusetts Institute of Tech nology, 1988.
[9] Hoare. Hints on programming language design. Originalmente publicado en 1973. En Hoare y Jones (ed.), Essays in Computing Science, Prentice-Hall, 1989. También en Horowitz (ed.), Programming languages, a grand tour. Computer Science Press / Springer-Verlag.
[10] Kernighan, B., Ritchie, D. El lenguaje de programación C. Prentice Hall Hispanoamericana, 1985.
[11] Leroy; Weis. Manuel de référence du lenguage Caml. París: InterEditions, 1993.
[12] Leroy. The Caml Light system, release 0.6, Documentation and user's manual. París: INRIA, 1993.
[13] Maclennan, Bruce. Principles of Programming Languages. Holt, Rinehart and Winston. 1983.
[14] Mauny. Functional programming with Caml Light. Parıs: INRIA, 1991.
[15] Sethi. Lenguajes de programación: conceptos y constructores. Addison-Wesley, 1992.
[16] Stansifer. The study of programming languages. Prentice-Hall, 1995.
[17] Tennent. Principles of programming languages. Prentice-Hall, 1981.
[18] Waite; Goos. Compiler construction. Nueva York: Springer-Verlag, 1984.
[19] Weis; Leroy. Le langage Caml. Par´ıs: InterEditions, 1993.
[20] Welsh; Elder. Structured system programming. Prentice-Hall, 1980.
[21] Watt. Programming language processors. Hemel Hempstead: Prentice-Hall, 1993.
[22] Watt. Programming language concepts and paradigms. Prentice-Hall, 1990.
[23] Wirth. On the design of programming languages. Originalmente publicado en 1974. En Horowitz (ed.), Programming languages, a grand tour. Computer Science Press / Springer-Verlag.

8 Profesores

Compiladores e intérpretes:
Ing. Marco Hernández Vásquez, MSc
Tecnológico de Costa Rica
Email: marco.hernandez@itcr.ac.cr / mhernandezvz@gmail.com

Horas de consultas a coordinar con el profesor en la primera clase.

Lenguajes:
Ing. Marco Rivera Meneses
Tecnológico de Costa Rica
marivera@itcr.ac.cr

Horas de consultas a coordinar con el profesor en la primera clase.