Universitat Autònoma de Barcelona

2019/2020

Compiladores

Código: 102782
Créditos ECTS: 6

Titulación	Tipo	Curso	Semestre
2502441 Ingeniería Informática	OB	3	2
2502441 Ingeniería Informática	OT	4	2

Contacto

Nombre: Francisco Javier Sánchez Pujadas
Correo electrónico: Javier.Sanchez.Pujadas@uab.cat

Uso de idiomas

Lengua vehicular mayoritaria: catalán (cat)
Algún grupo íntegramente en inglés: No
Algún grupo íntegramente en catalán: Sí
Algún grupo íntegramente en español: No

Equipo docente

Jorge Bernal del Nozal

Prerequisitos

Se aconseja haber cursado las asignaturas:

- Fundamentos de Informática
- Metodología de la Programación
- Laboratorio Programación

Conocimientos que podrán ser útiles para esta asignatura son:

- Los conocimientos sobre lenguajes de programación imperativos y orientados a objeto como C / C ++ facilitarán la comprensión de los ejemplos prácticos dados en teoría.
- Por la parte de generación de código serán útiles los conocimientos sobre assamblador.
- Los conocimientos sobre recursividad de la asignatura Análisis y Diseño de Algoritmos ayudarán a comprender el funcionamiento del análisis sintáctico.

Objetivos y contextualización

Conocimientos:

- Introducir los conceptos y estructuras fundamentales de un compilador de lenguajes imperativos.
- Mejorar el uso y comprensión de los compiladores (eficiencia, límites del lenguaje, etc.), y los lenguajes de programación.
- Implementación práctica de parsers simples.
  - Reconocedores de ficheros de configuración con parámetros complejos.
  - Flexibilización de diálogos con el usuario.
- Resolver problemas utilizando técnicas sintácticas y semánticas.
  - Definición e implementación de gramáticas para resolver problemas.

Decisión entre implementar un compilador entero o utilizar una herramienta para generar compiladores en función del tipo de problema a resolver.
- Implementar algoritmos complejos.
- Estrategias de test de programas complejos.
- Implementación de algoritmos recursivos.
- Uso de herramientas que generan código.
- Modularización en subproblemas fuertemente acoplados.
- Adaptación y utilización de un programa ya hecho.

Habilidades:

- Mejorar la calidad de los programas creados gracias al mayor conocimiento del funcionamiento de los compiladores.
- Utilización correcta de las estructuras de los lenguajes de programación para obtener programas más óptimos.
- Resolución de problemas de una fuerte complejidad aplicando los principios de modularidad y recursividad.
- Utilización de herramientas de programación automática.

Competencias

Ingeniería Informática
- Adquirir hábitos de pensamiento.
- Adquirir hábitos de trabajo personal.
- Capacidad para concebir, desarrollar y mantener sistemas, servicios y aplicaciones informáticas empleando los métodos de la ingeniería del software como instrumento para el aseguramiento de su calidad.
- Capacidad para conocer los fundamentos teóricos de los lenguajes de programación y las técnicas de procesamiento léxico, sintáctico y semántico asociadas, y saber aplicarlas para la creación, diseño y procesamiento de lenguajes.

Resultados de aprendizaje

1. Adaptar paradigmas existentes a problemas concretos y resolverlos computacionalmente.
2. Aplicar los conocimientos de análisis léxico, sintáctico y semántico a la generación de código en un compilador básico.
3. Conocer y comprender los procesos de análisis léxico, sintáctico y semántico de los lenguajes de programación, y analizar las diferentes alternativas en cada uno de ellos.
4. Desarrollar un pensamiento y un razonamiento crítico.
5. Gestionar el tiempo y los recursos disponibles. Trabajar de forma organizada.
6. Prevenir y solucionar problemas.
7. Tomar decisiones propias.

Contenido

I. Introducción. Conceptos básicos.

1. Introducción a los compiladores.
2. Estrategias de construcción de un compilador.
3. Definición de compilador y sus partes.
4. Presentación de los siguientes temas del curso.

II. Análisis lexicográfica: SCANNER.

1. Definición de análisis lexicográfico
2. expresiones regulares
3. Implementación de un analizador lexicográfico (Scanner)

4. Análisis lexicográfica.

III. Gramáticas y análisis sintáctico: PARSER.

- Definición de gramáticas libres de contexto.
- Notaciones para definir gramáticas.
- Características de las gramáticas y sus Lenguajes de Programación asociados.
- Gramáticas LL (1).
- Parsers LL (1).
- Recuperación de errores en parsers LL (1).
- Parsers LR, SLR, LALR.
- Consideraciones prácticas.

IV. Análisis semántico.

- Traducción dirigida por la Sintaxis.
- Tabla de Símbolos.
- Análisis semántico de declaraciones.
- Análisis semántico de expresiones.

V. Organización de la memoria en tiempo de ejecución.

- Ambiente de ejecución y asignación estática de memoria.
- Asignación de la pila.
- Heap.

VI. Generación de código.

- Máquina abstracta.
- Generación de código de expresiones.
- Proceso de referencias a estructuras de datos.
- Generación de código de funciones.
- Generación de código de estructuras de control.

Metodología

Esta asignatura tiene un vínculo muy fuerte entre lo que es la teoría, donde se hará una explicación de cómo funcionan y se pueden crear compiladores, y la práctica o los problemas, donde se explica cómo se crea realmente un compilador. Durante las clases de teoría se explican conceptos y técnicas que son necesarios para poder resolver la práctica. Los conceptos de práctica y problemas están relacionados, dado que el alumno diseñará una solución para resolver un problema y la implementará en la práctica. De esta forma, el proceso de aprendizaje del alumno se basará en estos tres tipos de actividades: clases de teoría, seminarios o explicaciones de cómo aplicar los conceptos teóricos a la práctica y las clases prácticas o de problemas en que los alumnos presentarán y discutirán sus soluciones.

Sesiones de clase unificadas

No se hace diferencia entre clases de teoría, problemas y prácticas. De esta forma se empieza la sesión explicando los conceptos teóricos como en una clase de teoría. Luego se aplican estos conceptos en problemas o prácticas. Así la sesión acaba como una clase de problemas o prácticas.

Clases de teoría

Las clases de teoría servirán para introducir los conceptos que se detallan en el temario de la asignatura. Se basarán en las explicaciones del profesor, que se ayudará de material complementario en forma de fotocopias o transparencias. El alumno deberá completar las clases de teoría con las horas de estudio, que deben servir para acabar de entender los contenidos de la asignatura y poder preparar correctamente las sesiones de

prácticas. Hay quetener presente que las clases de teoría presentan una continuidad a lo largo del curso, por lo que para poder seguir correctamente una clase se tiene que haber asimilado lo explicado en las clases anteriores.

Clases de problemas

Las clases de problemas se dedicarán básicamente a la explicación detallada de cómo aplicar los conceptos teóricos de manera práctica, y servirán de puente entre las clases teóricas y las prácticas. Los profesores plantearán problemas e indicarán cómo se resuelven preparando a los alumnos para resolver las prácticas. Es muy importante tener asimilados los conceptos teóricos previos para poder seguir correctamente las clases de problemas.

Antes de cada sesión de problemas el profesor propondrá una lista de ejercicios que los alumnos deberán intentar resolver previamente para poder presentar dudas a la hora de clase. En algunas sesiones se prepararán o discutirán problemas comunes surgidos en las prácticas.

Práctica

La práctica consistirá en hacer modificaciones sobre el compilador explicado a teoría. La práctica se realizará en grupos de 1, 2 o 3 personas. Las sesiones de prácticas estarán divididas en 3 bloques: Análisis sintáctico, Análisis semántico y Generación de código. Cada bloque tendrá uno o más entregas. El alumno podrá probar cada entrega con un autotest formado por las pruebas que debe pasar la práctica para aprobar. Después el profesor pasará un test más completo que será la base para obtener la nota de la práctica de grupo. La práctica se podrá probar con un corrector web todas las veces que el alumno necesite antes de la fecha de entrega. Además, se podrá evaluar la entrega, y los conocimientos obtenidos por cada alumno del grupo con una pequeña prueba oral o escrita por la nota individual.

Competencias transversales: Las competencias transversales serán trabajadas y evaluadas en varios momentos a lo largo del curso. Algunos ejemplos de cómo se trabajan son los siguientes:

- T01.01 - Desarrollar un modo depensamiento y razonamiento críticos:
  En las clases de teoría se explican ejemplos de lenguajes y compiladores existentes de los que se valora los pros, contras y errores de diseño.
  En las clases de problemas, los alumnos han de crear un modelo abstracto del problema para poder resolver.
  En la práctica el alumnado debe deducir qué restricciones semánticas debe comprobar el compilador.
- T02.03 - Gestionar el tiempo y los recursos disponibles. Trabajar de manera organizada:
  En las prácticas, el alumnado debe decidir en qué orden resuelven los diversos problemas de la práctica y cuál es su estrategia para encararlos.
- T02.04 - Prevenir y solucionar problemas:
  En la práctica existen fuertes dependencias entre las diferentes etapas de la compilación. El alumnado debe tomar decisiones sobre qué hace a las primeras etapas para asegurar que las siguientes no se compliquen demasiado.
- T02.05 - Tomar decisiones propias:
  Tan a problemas como las prácticas, el alumnado debe tomar decisiones sobre cómo resuelve los diversos problemas que apareceré

Actividades

Título	Horas	ECTS	Resultados de aprendizaje
Tipo: Dirigidas			
Clases unificadas (teoría, problemas y prácticaas)	46	1,84	1, 2, 3, 4, 5, 7, 6
Tipo: Autónomas			
Estudio de la materia impartida en clases teóricas	36	1,44	1, 2, 3, 4

Preparación e implementación de la práctica	47	1,88	1, 2, 3, 4, 5, 7, 6
Preparación y resolución de problemas	12	0,48	1, 2, 3, 4, 5, 7, 6

Evaluación

Criterios e indicadores de evaluación:

- Comprensión de los conceptos teóricos de la asignatura.
- Utilización correcta de las principales herramientas utilizadas para construir un compilador.
- Aplicación de los principios generales de diseño en la resolución de problemas complejos.
- Testear problemas complejos.

Actividades e instrumentos de evaluación:

Nota Prácticas: En este apartado hay una nota de grupo y una individual:
Nota de grupo: Corresponde a la nota obtenida a las entregas por grupos de la práctica.
Nota individual: Es la puntuación obtenida en el examen de prácticas.
Nota Teoría: corresponde al examen final o examen / es parcial / es de teoría sobre la asignatura. Se ha/n de aprobar los examen/es parcial/es para eliminar materia en el examen final.
La nota final de la asignatura se obtiene combinando la evaluación de estas dos actividades de la siguiente manera:

Nota Final = (0.5 * Nota Prácticas) + (0.5 * Nota Teoría)

Nota Prácticas = (0.2 * Nota Individual) + (0.8 * Nota Grupo)
Nota Individual = Examen de prácticas.
Nota Grupo = (0.2 * Grupo A. Sintáctico) + (0.3 * Grupo A. Semántica) + (0.5 * Grupo Generación de Código)

Las notas de grupo de análisis sintáctico, semántica y generación de código pueden corresponder a una media ponderada de varias entregas.

Convalidación de prácticas:

No se convalidan prácticas de años anteriores.

Recuperación de prácticas: En el caso de haber suspendido una entrega de grupo, se podrá recuperar en las siguientes entregas de la práctica. La nota será 0.75 * (max (nota recuperación, nota entrega suspendido) -Nota entrega suspendido) + nota entrega suspendido. En el caso de suspender el examen de prácticas, el alumno deberá presentarse a un examen de recuperación de prácticas el mismo día del examen final.

Condiciones para aprobar la asignatura:

Nota Final> = 5
Nota práctica> = 5 y haber aprobado cada uno de las entregas individuales y de grupo.
Nota teoría> = 5 y haber aprobado por separado las partes que tenga el examen de teoría.

Condiciones no evaluable:

No tener ninguna parte de la asignatura suspendida.

Condiciones suspenso:

No alcanzar una nota media superior o igual a 5.
Suspender alguna de las actividades de evaluación de la asignatura aunque la media supere el 5. En este caso, la nota será la nota mínima obtenida de alguna de las partes (exámenes o prácticas).

Condiciones para la matrícula de honor:

La matrícula de honor se puede conseguir con una nota media superior o igual a 9,0.
Debido a que hay un número limitado de matrículas de honor que se pueden dar por grupo, se otorgarán por orden de nota de mayor a menor.

Prácticas, trabajos o exámenes copiados:

Sin perjuicio de otras medidas disciplinarias que se estimen oportunas, y de acuerdo con la normativa académica vigente, las irregularidades cometidas por unestudiante que puedan conducir a una variación de la calificación se calificarán con un cero (0). Las actividades de evaluación calificadas de esta forma y por este procedimiento no serán recuperables. Si es necesario superar cualquiera de aquestesactivitats de evaluación para aprobar la asignatura, esta asignatura quedará suspendida directamente, sin oportunidad de recuperarla en el mismo curso. Estas irregularidades incluyen, entre otros:
- la copia total o parcial de una práctica, informe, o qualsevolaltra actividad de evaluación;
- dejar copiar;
- presentar un trabajo de grupo no hecho íntegramente por los miembros del grupo;
- presentar como propios materiales elaborados por un tercero, aunque sean traducciones o adaptaciones, y en general trabajos con elementos no originales y exclusivos del estudiante;
- tener dispositivos de comunicación (como teléfonos móviles, smart watches, etc.) accesibles durante las pruebas de evaluación teórico-prácticas individuales (exámenes).

En caso de que el estudiante haya cometido irregularidades en un acto de evaluación, la nota numérica del expediente será el valor menor entre 3.0 y la nota que le correspondería según el método de evaluación de la asignatura (y por tanto no será posible el aprobado por compensación).

En resumen: copiar, dejar copiar o plagiar en cualquiera de las actividades de evaluación equivale a un SUSPENSO con nota inferior a 3.0.

Publicación notas, fechas de exámenes, etc:

Las fechas de evaluación continua y entrega de trabajos se publicarán en el campus virtual y pueden estar sujetos a cambios de programación por motivos de adaptación a posibles incidencias. Siempre se informará en el campus virtual sobre estos cambios ya que se entiende que es el mecanismo habitual de intercambio de información entre profesor y estudiantes.

Procedimiento de revisión de las calificaciones

Para cada actividad de evaluación, se indicará un lugar, fecha y horade revisión enla que el estudiante podrá revisar la actividad con el profesor. En este contexto, se podrán hacer reclamaciones sobre la nota de la actividad, que serán evaluadas por el profesorado responsable de la asignatura. Si el estudiante no se presenta en esta revisión, no se revisará posteriormente esta actividad.

Actividades de evaluación

Título	Peso	Horas	ECTS	Resultados de aprendizaje
Análisis semántico de la práctica	Ver actividades e instrumentos de evaluación	1	0,04	1, 3, 4, 5, 7, 6
Análisis sintáctico de la práctica	Ver actividades e instrumentos de evaluación	1	0,04	1, 3, 4, 5, 7, 6
Examen de Prácticas	Ver actividades e instrumentos de evaluación	1	0,04	1, 2, 4
Examen final de teoría	Ver actividades e instrumentos de evaluación	3	0,12	1, 2, 3, 4
Examen/s parcial/es	Ver actividades e instrumentos de	2	0,08	1, 2, 3, 4

	evaluación			
Generación de código de la práctica	Ver actividades e instrumentos de evaluación	1	0,04	2, 4, 5, 7, 6

Bibliografía

Bibliografía básica

- *Charles N. Fischer & Richard J. Leblanc jr.*: Crafting a Compiler with C. The Benjamin / Cummings Publishing Company inc., 1991.
- *Jean-Paul Tremblay & Paul G. Sorenson*: The Theory and Practice of Compiler Writing. MCGRAW-HILL, 1985.
- *David A. Watt*: Programming Language Concepts and Paradigms. Prentice Hall, 1990.
- Alicia Garrido & all: Diseño de compiladores. Publicaciones Universidad de Alicante, 2002.

Bibliografía de consulta

- *David A. Watt.*: Programming Language Processors. Prentice Hall, 1993.
- *Anthony J. Field & Peter G. Harrison*: Functional Programming. Addison-Wesley, 1988.
- *Terrence W. Pratt*: Programming Languages: Design and Implementation. Prentice Hall International Editions, 1984.
- *Edited by G. Goos and J. Hartmanis*: Compiler Construction. An Advanced Course. Springer-Verlag, 1974
- *Jean-Paul Tremblay & Paul G. Sorenson*: An Implementation Guide to Compiler Writing. MCGRAW-HILL, 1982.
- *Ronald Mak*: Writing Compilers & Interpreters: An Applied Approach. Wiley, 1991.
- *J. P. Brown*: Writing Interactive Compilers and Interpreters. Wiley, 1979.
- *Allen Y. Holub*: Compiler Design in C. Prentice Hall, 1990.