Programación 2

1. Características generales

Nombre: Programación 2
Sigla: CI-0113
Créditos: 4
Horas lectivas: 5
Requisitos: CI-0112 Programación 1
Correquisitos: -
Clasificación: Curso propio
Ciclo de carrera: I ciclo, 2do año
Docente(s): Jeisson Hidalgo Céspedes
Datos de contacto: jeisson.hidalgo@ucr.ac.cr, oficina 3-21, casillero #4
Grupo: 04
Semestre y año: I ciclo 2020
Horario y lugar de clases: K 10 a 11:50 / V 9 a 11:50 203-IF
Horario y lugar de consulta: K 12 a 13 3-21 IF / V 12 a 13 102-IF
Asistente: Bryan Ulate Caballero <bryan.ulate@ucr.ac.cr>

2. Descripción del curso

En este curso el estudiante extiende su conocimiento en la programación de computadoras aprendiendo técnicas de abstracción que le ayuden a construir software de mayor complejidad y calidad. Las técnicas de abstracción son dependientes del lenguaje de programación, por ejemplo, las plantillas y el polimorfismo. El curso se concentra en técnicas de abstracción para construir software reutilizable y genérico. Se entiende por software reutilizable aquel que puede ser usado en diversos contextos, por ejemplo, las funciones de biblioteca. Se entiende por software genérico al software reutilizable que además puede utilizarse con tipos de datos arbitrarios, por ejemplo, un contenedor. En el curso el estudiante aprende tanto a reutilizar software como a crear software reutilizable y genérico.

3. Objetivos

Objetivo general

El objetivo general del curso es que el estudiante aprenda a resolver problemas de programación aplicando técnicas de abstracción para la creación y uso de software genérico y reutilizable.

Objetivos específicos

Durante este curso cada estudiante desarrollará habilidades para:

1. Explicar el modelo de ejecución del lenguaje de programación (máquina nocional) para implementar programas de forma correcta.
2. Aplicar diversas técnicas de abstracción para crear software genérico y reutilizable.
3. Utilizar software genérico para resolver problemas de forma eficiente.
4. Generar y aplicar pruebas unitarias a software genérico.
5. Aplicar buenas prácticas de programación (por ejemplo, documentación de código, programación defensiva y manejo de excepciones).

4. Contenidos

Objetivo específico	Eje temático	Desglose
1	Generalidades	a. Descripción del uso de la memoria estática, memoria automática y la memoria dinámica en el lenguaje de programación a usar en el curso. b. Descripción y comparación de los tipos de memoria disponibles en el lenguaje de programación del curso. c. Descripción de la estructura de proyecto en los ambientes de programación a usar en el curso. d. Descripción del depurador disponible en al menos uno de los ambientes de programación a usar en el curso. e. Descripción de los distintos mecanismos disponibles en el lenguaje de programación a usar en el curso para pasar a los métodos de una clase datos de tipos preconstruidos y objetos. f. Definición de métodos de instancia y métodos de clase en el lenguaje de programación a usar en el curso.

2	Especificación	a. Pautas para la especificación de clases concretas, clases abstractas y clases parametrizadas.
2	Herencia y polimorfismo	a. Definición de clases abstractas. b. Definición de clases que derivan de otras, ya sean abstractas o concretas. c. Definición de tipos polimórficos. d. Definición de métodos (y operadores, si el lenguaje lo permite) polimórficos.
3	Parametrización	a. Mecanismos de abstracción provistos por el lenguaje de programación para crear código genérico. b. Descripción general de una biblioteca de clases parametrizadas de uso estandarizado en el lenguaje de programación a usar en el curso. c. Descripción del uso de algunas clases parametrizadas de la biblioteca referida anteriormente, de acuerdo con las necesidades de los proyectos de programación y los ejemplos desarrollados en el curso.
2, 3	Estructuras de datos y algoritmos	a. Implementación de al menos una estructura de datos compleja, mediante asignación dinámica de memoria, y sus algoritmos.
5	Manejo de excepciones	a. Definición de clases de excepciones. b. Descripción del levantamiento de excepciones. c. Descripción de la captura y tratamiento de excepciones.
2, 3	Manejo de archivos planos	a. Lectura y escritura de archivos de texto y binarios. b. Manejo de archivos secuenciales y de acceso aleatorio.
4	Pruebas de programas	a. Pruebas de constructores y destructores. b. Pruebas de métodos modificadores. c. Pruebas de métodos observadores. d. Ordenamiento de los tipos de pruebas en un controlador de pruebas.

2	Bibliotecas	Creación y utilización de bibliotecas estáticas (ej: .lib, .a) y dinámicas (ej: .dll, .so) como medios de distribución de código reutilizable.
	Temas opcionales (a convenir entre el profesor y los estudiantes)	a. Programación de interfaces gráficas de usuario (GUI). b. Programación de videojuegos. c. Programación orientada a eventos. d. Programación de bases de datos (ej: SQLite). e. Programación procedimental (ej: C) f. Sobrecarga de operadores (ej: clase Fracción, String) g. Control de versiones (ej: Git, Subversion) h. Herramientas de generación de código (ej: compilador, linker, makefiles) i. Programación de expresiones regulares para procesamiento de texto

5. Metodología

Para alcanzar los objetivos del curso se seguirá una metodología híbrida constructivista-tradicional. El estudiante dedicará 5 horas a la semana a actividades presenciales y 7 horas a actividades extra-clase. Las 5 horas presenciales pueden combinarse entre lecciones magistrales en aula y acompañamiento del docente en un laboratorio de computadoras. En las 7 horas extra-clase el estudiante resolverá problemas de programación individuales y proyectos en parejas.

Problemas

Los estudiantes resolverán problemas de programación planteados por el profesor. Los estudiantes opcionalmente pueden proponer nuevos problemas (inventados). Los problemas tendrán el formato usado en los concursos de programación de la ACM. Se utilizará el juez en línea HackerRank. Los estudiantes proveerán su correo electrónico al profesor y recibirán invitaciones para acceder a los problemas a resolver.

Los problemas están organizados en secciones temáticas. Aproximadamente cada semana del ciclo lectivo se habilitarán los problemas de una sección y se enviarán invitaciones a los estudiantes. Durante las lecciones de las semanas correspondientes, el profesor explicará, al menos parcialmente, los conceptos de programación requeridos por los problemas. Algunos problemas requerirán más detalles o conceptos de los vistos en clase, con el fin de que el estudiante desarrolle habilidades

autodidácticas requeridas en el ejercicio de la profesión. Cada vez que el estudiante resuelve un problema acumulará los puntos respectivos en el juez en línea HackerRank. El total de puntos que acumule serán convertidos al 25% de la nota del curso por regla de tres.

Los estudiantes también pueden aportar problemas de programación y recibirán crédito adicional por ello. Un problema inventado se debe ubicar en una de las secciones temáticas y recibirá crédito proporcional al peso de la sección donde se ubique. Los estudiantes plantearán problemas por un máximo de 2% adicional de la nota del curso. Cada problema inventado debe tener el mismo formato usado en los problemas a resolver. El profesor proveerá recursos para facilitar la elaboración de estos. La fecha para proponer problemas para una sección vence dos semanas después de que se haya cubierto dicha sección.

La creación de problemas correlaciona con una mayor comprensión de los conceptos de programación de acuerdo con la literatura científica. Para crearlos, el estudiante debe prestar atención a los conceptos de programación involucrados en el tema, y puede tomar como ejemplo los problemas en la plataforma HackerRank. Los problemas inventados deben ser únicos entre estudiantes. Es decir, si dos o más estudiantes proponen el mismo problema, se dará crédito sólo al primero en someterlo. El profesor servirá como ente centralizador de los problemas inventados por estudiantes.

Proyectos

Los estudiantes resolverán a lo largo del curso dos problemas de programación de mayor complejidad que los problemas en HackerRank, a los cuales se les llamará proyectos. Ambos proyectos deben resolverse en equipos de dos estudiantes. Los integrantes de los equipos pueden variar entre el Proyecto 1 y el Proyecto 2. En ambos proyectos realizarán las fases de un proceso de desarrollo básico: análisis, diseño, implementación y pruebas.

El problema por resolver en el primer proyecto será planteado por el profesor. Para el segundo proyecto los estudiantes plantearán opciones de problemas que sean de su interés. Los estudiantes deben indagar y comprender el problema (análisis), derivar requerimientos, acordar con el profesor requerimientos para cada entregable, implementar los requerimientos y probarlos. Las revisiones de los proyectos serán a través de entregables, aproximadamente dos o tres por proyecto.

Los entregables se realizarán en un repositorio de control de versiones a convenir entre los estudiantes y el profesor. Algunas revisiones podrán realizarse mediante reuniones o presentaciones, en las cuales, el profesor pedirá a los dos integrantes explicar al menos una funcionalidad (implementación de un requerimiento). Una tercera funcionalidad será revisada por el profesor directamente a partir del código fuente. La calificación de esta tercera parte dependerá entonces de la

comprensión que el profesor pueda tener a partir del código fuente y, por lo tanto, de la calidad de la documentación y buenas prácticas de programación empleadas por los miembros del equipo.

Exámenes

Se realizarán dos exámenes parciales tras cubrir los temas correspondientes indicados en la evaluación. El material cubierto en el segundo examen es acumulativo del previo por la naturaleza de los contenidos. A menos de que se indique lo contrario, los exámenes serán realizados en papel, un sábado, y durante un período de tres horas.

En cualquiera de los tres tipos de evaluaciones (problemas, proyectos y exámenes), el profesor podría proponer ejercicios opcionales por crédito extra en la nota del curso. En cualquiera de las evaluaciones que se sospeche de plagio se aplicará el procedimiento establecido en el Reglamento de orden y disciplina de los estudiantes de la Universidad de Costa Rica.

6. Evaluación

%	Rubro	Descripción	Fecha*
25%	Problemas	Problemas en juez automático (25%), y problemas inventados opcionales (+2%). Todos debe realizarse de forma individual.	~cada semana
20%	Examen 1	Examen estrictamente individual en papel sobre el tema 1 del cronograma.	09-may 9 a.m
30%	Examen 2	Examen estrictamente individual en papel de los temas 1 a 5 del cronograma.	07-jul 9 a.m.
15%	Proyecto 1	Problema para resolver con el tema 1, en parejas, a presentar en control de versiones.	16-may 23:55
10%	Proyecto 2	Problema para resolver con los temas 1 a 5, en parejas, a presentar en control de versiones, máximo dos semanas después de cubierto el tema.	11-jul 23:55

***** Fechas tentativas en función de cuando se terminen de cubrir los temas respectivos.

7. Cronograma

Los temas del curso se presentan enmarcados a un paradigma de programación de acuerdo con siguiente cronograma. Las columnas de la tabla indican el número de tema, el paradigma y los conceptos más relevantes, el eje temático (sección 4 de contenidos) asociado, las actividades realizadas para cumplir con los objetivos, la duración en semanas (S), y las fechas tentativas. Se usarán como fechas efectivas de examen, problemas y proyectos, las fechas en que se terminen de cubrir los temas respectivos.

#	Paradigma y conceptos	Eje temático	A	S	Fechas
1	**Programación procedimental y metaprogramación**	Generalidades. Manejo de archivos planos. Bibliotecas. Pruebas de programas.	M J Y1 E1	7	10-mar a 02-may
	Resolución de problemas y algoritmos				
	E/S, expresiones y condicionales, repetición				
	Subrutinas, bibliotecas, metaprogramación, m. nocional				
	Indirección, arreglos, matrices, máquina nocional				
	Cadenas de caracteres, Unicode, máquina nocional				
	Registros, alineamiento de campos, máquina nocional				
2	**Programación orientada a objetos (OOP)**	Especificación. Bibliotecas. Manejo de excepciones. Pruebas de programas.	M J Y2 E2	3	04-may a 23-may
	Clases, objetos, miembros, atributos, métodos, encapsulación, composición, máquina nocional				
	Sobrecarga de operadores, lib-ecci, Fraction, String				
	Manejo de excepciones				
3	**Programación genérica**	Parametrización. Estructuras de datos y algoritmos.	M J Y2 E2	3	25-may a 13-jun
	Arreglo dinámico genérico. Plantilla de subrutina/clase				
	Lista enlazada genérica. Iterador				
	Árbol genérico. Arreglo asociativo (mapa)				
4	**Herencia y polimorfismo (OOP)**	Herencia y polimorfismo.	M J Y2 E2	2	15-jun a 27-jun
	Herencia múltiple de registros. Registro base/derivado. Constructores. Sobrescribir métodos.				
	Polimorfismo. Clase abstracta. Máquina nocional.				
5	**Programación orientada a eventos**	Temas opcionales.	Y2	1	29-jun a 04-jul
	Interfaces gráficas de usuario (GUI). Evento. Cola de eventos. Ciclo de eventos. Manejador de evento.				

Las actividades para cumplir con los objetivos (columna "A") son las siguientes:

- Clases magistrales (M) donde el estudiante aprovecha para solicitar explicaciones al profesor mientras éste expone los conceptos teóricos (contenidos) y su aplicación en la resolución de problemas de programación. Se complementa con actividades autodidácticas que el estudiante realiza en horas extra-clase. Objetivos 1 a 5.
- Problemas en el juez en línea (J) que el estudiante resuelve en horas extra-clase y cuenta con apoyo del profesor en potenciales horas de laboratorio o en horas de consulta. Objetivos 2, 3, y 5.
- Proyecto 1 (Y1) donde los estudiantes en parejas resuelven un problema de mayor complejidad aplicando prácticas de programación necesarias

en la industria. Además, aplican nociones y habilidades para crear y reutilizar código a través de bibliotecas de subrutinas. Objetivos 2 a 5.

- Proyecto 2 (Y2), también en parejas, donde los estudiantes resuelven un problema que requiera la creación y reutilización código a través de la programación genérica y polimórfica. Si el tiempo lo permite, se introducen interfaces gráficas de usuario y el paradigma de programación orientada a eventos para resolver problemas más realistas. Objetivos 2 a 5.
- Examen 1 (E1) donde los estudiantes resuelven un problema aplicando las nociones de programación procedimental para crear software reutilizable (bibliotecas de subrutinas) de forma estrictamente individual en un período de tres horas. Objetivos: 1, 2 y 5.
- Examen 2 (E2) donde los estudiantes resuelven un problema aplicando las nociones de programación orientada a objetos y genérica para crear software reutilizable (plantillas y polimorfismo) de forma estrictamente individual en un período de tres horas. Objetivos: 1 a 5.

8. Bibliografía

Textos recomendados:

1. Stroustrup, Bjarne; "Programming principles and practice, using C++"; 2nd edition; Addison-Wesley; 2014.
2. Deitel, Harvey M.; Deitel, Paul J.; "C++ How to Program, 9/e"; Prentice-Hall; 2015.
3. Forouzan, Behrouz A.; Gilberg, Richard F. "C++ Programming: an object oriented approach"; McGraw Hill; 2019.
4. Skiena, Steven S.; Revilla, Miguel A. "Programming challenges". Springer, 2003.

Otra bibliografía de apoyo:

1. Osherove, Roy. "The art of Unit Testing"; 2nda edición; Manning Pub; 2014.
2. Myers, Glenford J.; "The art of software testing 3rd ed"; John Wiley & Sons, Inc.; 2013.

9. Aspectos relacionados con el sistema de bibliotecas, acceso a los reglamentos estudiantiles, y evaluación por parte de los profesores

El Sistema de Bibliotecas, Documentación e Información (SIBDI) de la Universidad de Costa Rica (http://sibdi.ucr.ac.cr/) cuenta con una amplia gama de recursos de información bibliográfica en diferentes formatos como libros, folletos, publicaciones periódicas, trabajos finales de graduación, entre otros. Algunos de estos recursos se encuentran en Biblioteca Virtual, desde la cual se pueden acceder las publicaciones de conferencias y revistas de ACM, IEEE (http://sibdi.ucr.ac.cr/dbingenieria.jsp), o Springer, entre otras. La Biblioteca Luis Demetrio Tinoco ofrece cursos de capacitación a los estudiantes del área de

las Ingenierías y Computación.

El sitio web del Consejo Universitario de la UCR contiene las diferentes normativas estudiantiles, que pueden ser consultadas desde el enlace http://www.cu.ucr.ac.cr/normativa/estudiantil.html.

Los procedimientos de evaluación y orientación establecidos en el Reglamento de Régimen Académico Estudiantil se encuentran en http://www.cu.ucr.ac.cr/uploads/tx_ucruniversitycouncildatabases/normative/regimen_academico_estudiantil.pdf

Destacamos especialmente los siguientes artículos de dicho Reglamento:

- El Artículo 14 se refiere al contenido que debe tener un programa del curso, incluyendo "las normas de evaluación desglosadas y con las ponderaciones de cada aspecto a evaluar".
- El Artículo 17 indica que "las normas de evaluación conocidas por los estudiantes pueden ser variadas por el profesor con el consentimiento de la mayoría absoluta (más del 50% de los votos) de los estudiantes matriculados en el curso y grupo respectivo".
- El Artículo 22 indica que "el profesor debe entregar a los alumnos las evaluaciones calificadas y todo documento o material sujeto a evaluación, a más tardar diez días hábiles después de haberse efectuado las evaluaciones y haber recibido los documentos".
- El Artículo 24 establece el procedimiento y fechas para realizar la reposición de evaluaciones.