Programación 2

1. Características generales

Nombre:	Programación 2
Sigla:	CI-0113
Créditos:	4
Horas:	5
Requisitos:	CI-0112 Programación 1
Correquisitos:	-
Clasificación:	Curso propio
Ciclo carrera:	I ciclo, 2do año
Docentes:	Gr01: Jeisson Hidalgo Céspedes <jeisson.hidalgo@ucr.ac.cr>
	Gr02/03: Luis Gustavo Esquivel Quirós <luis.esquivel@ucr.ac.cr>
Asistentes:	Gr01: José David Vargas Artavia <josed1608@gmail.com>
	Gr02: María Chaves Salas <marijesuscs08@gmail.com>
	Gr03: Gabriel Galvez Fallas <gabo141298@gmail.com>
Grupo y semestre:	01, II ciclo 2019
Horario de clase:	Gr01: K 16 a 17:50 306-IF / V 15 a 17:50 106-IF
	Gr02: L 10 a 11:50 305-IF / J 9 a 11:50 305-IF
	Gr03: L 7 a 9:50 305-IF / J 7 a 8:50 305-IF
Horario de consulta:	Gr01: K 18 a 19 321-IF2 / K 18 a 19 106-IF
	Gr02/03: K 8-12 222-IF

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. En las 7 horas extra-clase, el estudiante resolverá problemas y ejercicios de programación individuales y proyectos en parejas.

La metodología de las horas presenciales podría alternarse parcial o totalmente entre lecciones magistrales y aula invertida. En las semanas cuando se aplique aula invertida, en las horas presenciales el docente acompañará a los estudiantes a superar dificultades de programación en un laboratorio de computadoras de la Escuela. En las horas extra-clase de aula invertida los estudiantes consultarán materiales preparados o asignados por el docente.

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 como juez automático la plataforma 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 indague otras fuentes de información para poder resolver el problema y desarrollar habilidades autodidácticas requeridas en el ejercicio de la profesión. Cada vez que el estudiante resuelve un problema acumulará los puntos respectivos en la plataforma 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. Los profesores servirán como entes centralizadores 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 tres o cuatro por proyecto, distanciados por dos semanas.

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 tres exámenes parciales. Cada uno se realizará dos semanas después de cubrir los temas correspondientes indicados en la evaluación. El material cubierto en cada examen es acumulativo de los anteriores 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 detecte plagio, en la primera vez la evaluación obtendrá una calificación de cero, y en caso de reincidencia se aplicará lo establecido en el Reglamento de orden y disciplina de los estudiantes de la Universidad de Costa Rica.

6. Evaluación

%	Rubro	Descripción
30%	Ejercicios	Problemas en juez automático (25%), ejercicios del material del curso (5%), e inventados opcionales (+2%). Todos son estrictamente individuales.
15%	Examen 1	Examen estrictamente individual en papel sobre el tema 1 del cronograma, a realizar dos semanas después de cubiertos el tema 1.
17%	Examen 2	Examen estrictamente individual en papel de los temas 1 a 3 del cronograma, a realizar dos semanas después de cubiertos los temas.
18%	Examen 3	Examen estrictamente individual en papel de los temas 1 a 5 del cronograma, a realizar dos semanas después de cubiertos los temas.
10%	Proyecto 1	Problema para resolver con el tema 1, en parejas, a presentar en control de versiones, máximo tres semanas después de cubierto el tema.

10%	Proyecto 2	Problema para resolver con el tema 5 (requiere de los previos), en parejas, a presentar en control de versiones, máximo dos semanas después de cubierto el tema.

7. Cronograma

#	Temática	Tecnología	Sem.	Fechas*
1	Programación procedimental y metaprogramación	C	5	12-ago a 14-set
2	Programación orientada a objetos (OOP)	C++	3	16-set a 05-oct
3	Programación genérica	C++	3	07-oct a 26-oct
4	Herencia y polimorfismo (OOP)	C++	3	28-oct a 16-nov
5	Programación orientada a eventos	Qt	2	18-oct a 30-nov

***** Fechas tentativas. Se usarán como fechas efectivas de examen, ejercicios y proyectos, las fechas en que se terminen de cubrir los temas respectivos.

8. Bibliografía

Libro de texto recomendado:

1. Stroustrup, Bjarne. The C++ Programming Language (Fourth Edition). May 2013. Addison Wesley. USA. May 2013. ISBN 0-321-56384-0. 1360 pages
2. Deitel, Harvey M.; Deitel, Paul J.; "C++ How to Program, 9/e"; Prentice-Hall; 2015.

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 2nd ed"; Revised and updated by Tom Badgett and Todd Thomas, with Corey Sandler; ISBN-0-471-46912-2; John Wiley & Sons, Inc.; 2004.
3. Kernighan, Brian; Ritchie, Dennis. El lenguaje de programación C, 2da edición. Pearson, México, 1991.
4. Josuttis, Nicolai. The C++ Standard Library: A Tutorial and Reference, 2nd edition. Addison-Wesley; 2012