CI-0113, Programación 2, I-2018 Grupo 05

1. Características generales

Sigla	CI-0113	**Grupo**	05
Curso	Programación 2	**Profesor**	Jeisson Hidalgo Céspedes
Créditos	4		Jeisson.hidalgo@ucr.ac.cr
Horas	5 presenciales, 7 extraclase	**Horario**	K 10-13 V 11-13 304-IF
Requisitos	Programación 1	**Consulta**	KV 8-10 201-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

Obj	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

6. Metodología y evaluación

Tareas	30%	30%	Ejercicios resueltos
		2%	Ejercicios inventados (extra)
Proyectos	20%	10%	Proyecto 1
		10%	Proyecto 2
Exámenes	50%	15%	Examen 1
		17%	Examen 2
		18%	Examen 3

Para alcanzar los objetivos del curso se seguirá una metodología híbrida constructivista-tradicional, compuesta de las siguientes actividades: ejercicios de programación, proyectos, exámenes y coloquios. Se explican a continuación.

Ejercicios de programación

Los estudiantes resolverán ejercicios de programación. Opcionalmente pueden proponer ejercicios propios hasta un máximo de un 2% extra en la nota del curso. Los ejercicios tendrán el formato usado en los concursos de programación de la ACM. Se utilizará como juez automático la plataforma HackerRank [https://www.hackerrank.com/]. Los estudiantes proveerán su correo electrónico al profesor y recibirán invitaciones para acceder a los ejercicios.

Los ejercicios están organizados en temas por los conceptos de programación que se requieren para resolverlos. Aproximadamente cada semana del curso el profesor habilitará los ejercicios de un tema del curso y enviará invitaciones a los estudiantes. Los estudiantes resolverán los ejercicios en horas extraclase. Durante las lecciones de la semana correspondiente, el profesor explicará, al menos parcialmente, los conceptos de programación requeridos por los ejercicios. Algunos ejercicios requerirán más detalles o conceptos de los vistos en clase, con el fin de que el estudiante investigue 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 ejercicio de programación obtendrá puntos que irá acumulando en la plataforma HackerRank. El estudiante recibirá crédito por sus puntos de HackerRank hasta un máximo de 25% de la nota del curso. La cantidad de puntos de HackerRank correspondiente al 25% del curso equivaldrá a los puntos de todos los ejercicios.

Los ejercicios de programación están agrupados en aproximadamente 12 temas. Cada estudiante podrá inventar ejercicios durante el semestre y recibirá crédito por un 5% de la nota del curso. Cada **ejercicio inventado** debe tener el formato usado en la plataforma HackerRank y debe pertenecer a un tema distinto. La creación de ejercicios correlaciona con una mayor comprensión de los conceptos de programación, de acuerdo a 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 ejercicios resueltos. Los ejercicios inventados deben ser únicos entre estudiantes. Es decir, si dos o más estudiantes proponen el mismo ejercicio, se dará crédito sólo al primero en someterlo. El profesor servirá como el ente centralizador de los ejercicios inventados por estudiantes, y podrá ayudarles en el proceso de creación de los mismos durante las horas de consulta.

Proyectos

Los estudiantes resolverán a lo largo del curso dos problemas de programación de mayor complejidad que los ejercicios de programación, 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.

Para el primer proyecto el profesor será el cliente. Para el segundo proyecto los estudiantes definirán el cliente, que puede ser ellos mismos u otra persona. Los estudiantes deben indagar y comprender el problema del cliente (análisis), derivar requerimientos, acordar con el cliente (o 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. La revisión de los entregables se hará mediante reuniones de avance. La completitud se podrá evaluar por la cantidad de requerimientos implementados y se podrá usar el provisto por la herramienta de control de versiones. Algunas revisiones podrán realizarse mediante reuniones, 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 tanto, de la calidad de la documentación y buenas prácticas de programación empleadas por los miembros del equipo.

Los estudiantes podrán iniciar el trabajo en el primer proyecto desde la primera semana de clases. El segundo proyecto requiere la implementación de interfaces gráficas, pero los estudiantes pueden empezar a definir sus requerimientos cerca de la mitad del curso. Las fechas de entrega de los proyectos serán acordadas entre los estudiantes y el profesor.

Exámenes

Se realizarán tres **exámenes** parciales cuyas fechas serán acordadas entre los estudiantes y el profesor tras cubrir los temas correspondientes. 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 día sábado, y durante un período de tres horas.

El profesor convendrá con los estudiantes las horas y el lugar de consulta. En las horas de consulta el profesor podrá apoyar varias de las actividades descritas anteriormente:

1. Ayudar a los estudiantes en la resolución de los ejercicios de programación planteados en HackerRank.
2. Ayudar a los estudiantes a plantear sus propios ejercicios de programación.
3. Ayudar a los estudiantes en la solución de sus proyectos de programación.
4. Evaluar el avance de los proyectos de programación.
5. Aclarar dudas o consultas.

En cualquiera de los tres tipos de evaluaciones (tareas, 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, será anulada por completo.

Bibliografía

1. Kernighan, Brian; Ritchie, Dennis. *El lenguaje de programación C*, 2da edición. Pearson, México, 1991.
2. Stroustrup, Bjarne. *The C++ Programming Language (Fourth Edition)*. May 2013. Addison Wesley. USA. May 2013. ISBN 0-321-56384-0. 1360 pages
3. Deitel, Harvey M.; Deitel, Paul J.; *C++ How to Program, 9th edition*; Prentice-Hall; 2015. (o posterior)
4. Josuttis, Nicolai. *The C++ Standard Library: A Tutorial and Reference*, 2nd edition. Addison-Wesley; 2012
5. Osherove, Roy. *The art of Unit Testing*; 2nd edition; Manning Pub; 2014.
6. 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.
7. Chacon, Scott. *Pro Git*. Apress, 2014. Libro libre [https://git-scm.com/book/en/v2]