Universidad de Costa Rica
Escuela de Ciencias de la Computación e Informática
CI1201 - Programación II. Grupo 04. II-2014
Horario: K,V 13-15 303-IF. Consulta: K,V 16-17 104-IF

Profesor Jeisson Hidalgo-Céspedes
Casillero: 04. Correo: jeissonh@gmail.com
Asistente Óscar Esquivel Oviedo
Correo: oscar12esquivel@gmail.com

Carta al estudiante

Objetivos

Los estudiantes aprenderán a solucionar problemas mediante la programación de aplicaciones y bibliotecas reutilizables de software de mediana complejidad. Al finalizar el curso el estudiante será capaz de:

1. Escoger el tipo de sistema de software (o una combinación de ellos) para solucionar problemas (programa, biblioteca, comando, GUI, ...)
2. Comprender el proceso de construcción de código (preprocesador, compilador, ensamblador, enlazador, *makefiles*, etc.)
3. Comprender varios paradigmas de programación: imperativa, procedimental, orientada a objetos, genérica, metaprogramación y orientada a eventos.
4. Realizar tareas muy elementales de ingenería de software para la solución de problemas: análisis, diseño, implementación y pruebas de software.
5. Dominio básico del lenguaje de programación C++, sus paradigmas y mecanismos principales de abstracción: orientación a objetos, herencia, polimorfismo, programación genérica y manejo de excepciones.
6. Comprender en alto nivel la máquina computacional que se está programando y el estado de los programas.
7. Aplicar buenas prácticas de programación para el mantenimiento del código y trabajo colaborativo. Por ejemplo, documentación de código, consistencia en una convención de estilo, programación defensiva y uso de identificadores significativos.
8. Reutilizar código, a través del uso de la biblioteca estándar de C++ y al menos una biblioteca de terceros, para incrementar la funcionalidad de una solución: agregar persistencia, interfaces gráficas, videojuegos, etc.
9. Crear componentes genéricos de software, en especial clases contenedoras e iteradores, que sean reutilizables en una cantidad arbitraria de aplicaciones, a través de los mecanismos de "parametrización" de clases (plantillas de clases) o polimorfismo.

Contenidos

1. Generalidades
    a. Descripción del uso de la memoria estática y la memoria dinámica en el lenguaje de programación que se usará en el curso.
    b. 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 que se usarán en el curso.
    d. Descripción del depurador disponible en al menos uno de los ambientes de programación que se usará en el curso.
    e. Descripción de los distintos mecanismos disponibles en el lenguaje de programación que se usará 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 que se usará en el curso.
2. Especificación
    a. Pautas para la especificación de clases concretas y clases abstractas.
    b. Discusión de las pautas para la especificación de clases parametrizadas.
3. Herencia y polimorfismo
    a. Definición de clases abstractas.
    b. Definición de clases que derivan de otras, ya sea abstractas o concretas.
    c. Definición de tipos polimórficos.
    d. Definición de métodos (y operadores, si el lenguaje lo permite) polimórficos.
4. Parametrización de clases
    a. Descripción general de una biblioteca de clases parametrizadas de uso estándarizado en el lenguaje de programación que se usará en el curso.
    b. 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.
5. Árboles
    a. Algoritmos de inserción, búsqueda, eliminación y el recorrido en orden para árboles binarios ordenados no balanceados.
6. Validación de entrada de datos y manejo de excepciones

    a. Defínción de clases de excepciones.
    b. Descripción del levantamiento de excepciones.
    c. Descripción de la captura y tratamiento de excepciones.
7. Manejo de archivos planos
    a. Por medio de flujos de datos.
    b. Por medio de acceso aleatorio.
8. 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.

Los contenidos anteriores se organizan en 7 grandes temas:

1	Programación procedimental (C)	
2	Programación orientada a objetos (C++)	
3	Sobrecarga de operadores	Examen01
4	Programación genérica (plantillas)	
5	La biblioteca estándar de C++	Examen02
6	Herencia y polimorfismo	Examen03
7	Tema extra	Proyecto 2

Metodología y evaluación

Para cumplir los objetivos se seguirá una metodología tradicional. El profesor impartirá 32 lecciones presenciales. En cada una de ellas se introducirá un problema y el profesor lo resolverá en frente de la clase, junto con la teoría requerida. Al final de la misma se propondrán mejoras a la solución y se dejarán como práctica a los estudiantes para realizarlas de forma extraclase.

Cada semana se asignará una tarea corta relacionada con los temas vistos en clase. Dos tareas largas ("proyectos") se asignarán durante el semestre, las cuales agrupan conocimiento visto en el curso para resolver problemas de mediana complejidad. Las nociones aprendidas serán evaluadas con tres exámenes parciales en horas extraclase. Al inicio de toda clase podría realizarse un examen corto ("quiz") para dar seguimiento de los temas cubiertos. Evaluación:

- 15%. **Quices**. Ejercicios cortos que deberá el estudiante resolver en forma individual al iniciar la clase. No se repondrán por llegadas tardías.
- 20%. **Tareas cortas**. Ejercicios que deberá resolver el estudiante individualmente de forma extraclase. Se presentarán en el aula virtual disponible en Mediación Virtual [http://mediacionvirtual.ucr.ac.cr/course/view.php?id=3887].
- 25%. **Proyectos**. Dos proyectos de mediana complejidad. El profesor indicará si se pueden realizar en parejas o individualmente. El profesor también indicará si la ponderación será en proporción a su dificultad.
- 40%. **Exámenes parciales**. Tres exámenes de igual ponderación, cuyas fechas serán acordadas tras cubrir los temas correspondientes. El material cubierto en cada examen es acumulativo de los anteriores, por la naturaleza de los contenidos.

Observaciones

1. En toda asignación, sea en papel o digital, se evaluará la indentación, uso correcto de paréntesis (redondos, cuadrados y llaves), la eficiencia, la elección de identificadores significativos, y las buenas prácticas de programación.
2. La nota N de una tarea o proyecto entregado h horas tarde se calculará como $N = x - h^{3/2}$, donde x es la nota que habría obtenido si se hubiere entregado a tiempo.
3. Cualquier asignación donde se detecte plagio, su calificación será anulada por completo (no sólo el o los ejercicios donde se detecte el plagio).

Bibliografía

1. Chacon, Scott. *Pro Git*. Apress, 2009. Libro libre: disponible en inglés, español, o código fuente.
2. Deitel, H.M.; Deitel, P.J. *C++ How to Program*, 8th edition. Prentice-Hall, 2012.
3. Stroustrup, Bjarne. *The C++ Programming Language*, 4th edition. Addison-Wesley; 2013.
4. Josuttis, Nicolai. *The C++ Standard Library: A Tutorial and Reference*, 2nd edition. Addison-Wesley; 2012.
5. Kernighan, Brian; Ritchie, Dennis. *El lenguaje de programación C*, 2da edición. Pearson, México, 1991.