

**Nombre de la asignatura:** Lenguajes y Autómatas II

**Créditos:** 2 – 3 – 5

### **Aportación al perfil**

- Desarrollar, implementar y administrar software de sistemas o de aplicación que cumpla con los estándares de calidad con el fin de apoyar la productividad y competitividad de las organizaciones.
- Integrar soluciones computacionales con diferentes tecnologías, plataformas o dispositivos.
- Diseñar e implementar interfaces hombre – máquina y máquina – máquina para la automatización de sistemas
- Integrar soluciones computacionales con diferentes tecnologías, plataformas o dispositivos.
- Identificar y comprender las tecnologías de hardware para proponer, desarrollar y mantener aplicaciones eficientes.

### **Objetivo de aprendizaje**

- Aplicar las técnicas de árboles de expresiones, optimización y administración de memoria en la construcción de SW de base.

### **Competencias previas**

- Aplicar las expresiones regulares, autómatas y gramáticas (elementos de la teoría de la computación) en la construcción de las fases de léxico y sintaxis de un compilador.
- Analizar la complejidad de los algoritmos
- Manejar listas enlazadas
- Manejar de tablas Hash
- Manejar el lenguaje ensamblador
- Conocer la arquitectura de una computadora.

### **Temario**

- Análisis semántico
  - Árboles de expresiones.
  - Acciones semánticas en un analizador sintáctico.
  - Comprobación de tipos en expresiones.
  - Pila semántica en un analizador sintáctico.
  - Esquemas de traducción
  - Generación de la tabla de símbolos y de direcciones.
  - Manejo de errores semánticos.

- Generación de código intermedio.
  - Notaciones.
  - Representación de código intermedio.
  - Esquemas de generación.
- Optimización
  - Tipos de optimización.
  - Costos
- Generación de código objeto.
  - Registros
  - Lenguaje ensamblador.
  - Lenguaje máquina.
  - Administración de memoria.

### **Actividades de aprendizaje** (desarrollo de las competencias específicas)

- Conocer como se realiza la conversión de tipos.
- Establecer las reglas para la conversión de tipos.
- Agregar acciones semánticas a una gramática.
- Manipular la tabla de símbolos y de direcciones.
- Detectar y recuperar errores semánticos.
  
- Conocer las notaciones para la conversión de expresiones.
- Conocer como se representa el código intermedio.
- Generar notaciones para la conversión de expresiones.
- Representar el código intermedio utilizando un lenguaje propuesto.
- Utilizar un diagrama de sintaxis para representar acciones.
  
- Tener nociones algebraicas para estimar el número de veces que se realiza una instrucción dentro de un ciclo o ciclos anidadas.
- Conocer que recursos se consumen en invocación a funciones y expresiones simples.
- Conocer las técnicas de optimización de código sobre un código intermedio generado,
- Conocer los criterios de tiempo de ejecución o extensión de código generado.
- Poder optimizar un código intermedio existente.
  
- Conocer la estructura y funcionamiento del lenguaje ensamblador
- Conocer técnicas para administración de memoria para el almacenamiento de un programa en el momento de la ejecución.
- Conocer la arquitectura básica del procesador.

- Conocer las instrucciones del lenguaje ensamblador para el microprocesador en estudio y su equivalencia en código máquina.
- Solucionar problemas usando lenguaje ensamblador.
- Manipular registros de memoria

### **Sugerencias didácticas transversales para el desarrollo de competencias profesionales**

#### **Prácticas.**

- Realizar arboles de expresiones en casos de estudio.
- Realizar conversiones de tipos en expresiones.
- Construir la tabla de símbolos y de direcciones para la gramática propuesta
- Detectar errores de semántica en expresiones dadas.
- Modificar la GLC agregando las acciones semánticas correspondiente.
- Convertir expresiones mediante el uso de notaciones prefijas, infijas y postfijas.
- Definir e implementar la notación que más se ajuste a las estructuras de evaluación de expresiones de lenguaje.
- Proponer una estructura de código intermedio en base a las características propias de cada lenguaje.
- Desarrollar esquemas de generación de código intermedio
- Definir y construir el generador de código intermedio para su caso de estudio.
- Agregar acciones de representación intermedia al lenguaje de programación propuesto.
- Saber cuántos recursos y cuanto tiempo consume cada instrucción de código intermedio
- Evaluar el código intermedio generado para los programas escritos en el lenguaje de su caso de estudio y si aplica realizar la optimización correspondiente.
- Poder establecer una equivalencia entre las instrucciones del lenguaje intermedio y las instrucciones en ensamblador.
- Diseñar y construir el generador de código máquina u objeto para el lenguaje del caso de estudio.

#### **Criterios de evaluación:**

La evaluación de la asignatura se hará con base en siguiente desempeño:

Desarrollo de un proyecto final integrador en donde se aporte el desarrollo de software en alguna de las áreas de SW de base.