Programación para Ingeniería Eléctrica

Gregory Randall

Instituto de Ingeniería Eléctrica, Facultad de Ingeniería,
Universidad de la República. Montevideo, Uruguay.

Montevideo, 2024

Agenda

1. Introducción
   - Presentación de la asignatura
   - Objetivos
   - Motivación
   - Modalidad de trabajo
2. Programación y Lenguajes
   - Historia de la programación

Presentación de la asignatura

- Plantel
  - Responsable del curso: Gregory Randall.
  - Teórico: Gregory Randall
  - Práctico: Gregory Randall
  - Consultas: Graciana Castro, Santiago Fernández
- EVA (https://eva.fing.edu.uy/course/view.php?id=637)
  - Foros: consultas, novedades
  - Material: prácticos, teóricos, obligatorios, software

Objetivos

- Programación en Ingeniería Eléctrica (IE)
  - Desarrollar e implementar algoritmos
  - Entender especificaciones de diseño
  - Saber utilizar bibliotecas externas
  - Lograr eficiencia, modularidad, portabilidad
- Lenguaje de programación C
  - Muy utilizado en todas las áreas de IE
  - Simple, portable, eficiente y poderoso
  - Base de otros lenguajes modernos (C++, C#, Obj. C, Java)

Motivación: por qué?

Necesaria en todas las áreas de la Ingeniería Eléctrica:

Potencia Localización de fallas, monitoreo

Electrónica Sistemas "embebidos", PICs, PLCs, DSPs, Android

Sistemas y Control Control de plantas, procesos

Telecomunicaciones Manejo de tramas, compresión y transmisión de datos, optimización de tráfico

Procesamiento de señales Procesar video de seguridad, reconocimiento de voz, biometría

Motivación: cómo?

"Tipos" de Programación:

- gestión (registros, bases de datos, lógica de empresa)
- científica (algoritmos, cálculo)
- interfaz de usuario (GUI)
- sistemas (drivers, archivos)
- scripting (Web)

Motivación: para qué?

Roles de un ingeniero eléctrico como desarrollador:

- Implementar software de acuerdo a especificación de diseño dada
- Comprender software realizado por otros (bibliotecas, APIs)
- Interactuar con ingenieros en computación dentro de equipos multidisciplinarios

Práctico

Filosofía

- A programar se aprende programando
- Los ejercicios prácticos son para programarlos y ejecutarlos,

Forma de trabajo

- Repartido de ejercicios en el EVA
- Clase práctica:
  - Ejemplos prácticos
  - Trabajo grupal en un problema
  - Consultas de ejercicios

Teórico

Filosofía

Se sigue un libro (casi) al pie de la letra:

- A. Kernighan, D. Ritchie. "The C Programming Language". Segunda edición. Prentice Hall. ISBN 0-13-110362-8. 1988
- (Por supuesto que se pueden consultar otras fuentes)

Forma de trabajo

- El alumno lee el material de antemano
- En clase teórica se recalca lo esencial
- Usamos ejemplos para entender los conceptos, que luego ustedes pueden probar.

Obligatorio

Filosofía

- Problema aplicado a IE
- Foco en algoritmos, no estructura
- Trabajo individual
- Evaluación del producto final, no de estructura interna

Forma de trabajo

- Aplicación en C de acuerdo a especificaciones dadas
- Dos obligatorios vinculados

Ganancia de curso

Ganancia (detalles en programa del curso)

- En base a dos obligatorios y un parcial (segundo periodo). No hay examen
- Deben alcanzarse mínimos en el total (60 puntos sobre 100) y en cada una de las partes (Obligatorio 1: 8/30, Obligatorio 2: 8/30, Parcial 10/40)

Historia de la programación (1)

1780 revolución industrial

1801 J-M Jacquard, "Jacquard Loom" (telar), brazo mecánico, tarjetas perforadas

1822 C. Babbage, "Difference Engine", logaritmos, trigonométricas, aprox. polinomial

1842 A. Lovelace, "Primer algoritmo", traducción, cálculo de números de Bernoulli

1890 H. Hollerith, "Primer almacenamiento", tarjetas perforadas para censo, trenes

Historia de la programación (2)

1937 A. Turing. "Turing machine". dispositivo teórico, programa en cinta, complejidad de algoritmos. Base del almacenamiento.

1940 Primeras computadoras eléctricas

1950 Primeros lenguajes de alto nivel (FORTRAN, LISP)

1970 Programación estructurada, portabilidad (C, Pascal)

1980 Programación modular, objetos (SmallTalk, C++)

1990 Internet (C#, Java)

2000 Interoperabilidad, web services

Computadora (máquina)

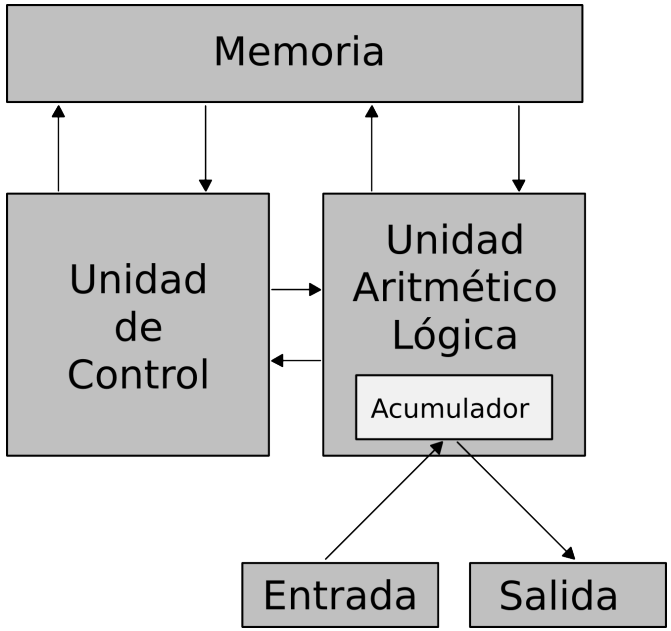
Arquitectura: von Neumann

Procesador/ALU aritmética (suma, resta., etc),

Procesador/PCU ejecución secuencial, bifurcación, interrupciones

Memoria Estado de la computadora, almacenamiento

Dispositivos Conexión con el mundo externo, entrada y salida



Programa de computadora

Programa

programa	Ordenes a la computadora
datos	Información manipulada por el programa
entorno	Interfaz con sistema operativo
Lenguaje	Traducción a código máquina

Ejemplo: BASIC

```
10 REM Hello World BASIC
20 PRINT "Hello World!"
```

Programación y Lenguajes

- Definición
- Tipos de Lenguajes
- Datos y Estructuras
- Instrucciones y Control de flujo
- Filosofía de diseño

Programación y Lenguajes

Definición

- Componentes:

léxico	qué palabras claves hay
sintaxis	cómo se forma una frase X válida
semántica	qué pasa cuando uno escribe X

- Aleatoriedad: recorrido determinístico o aleatorio?
- Propósito: para qué es?

Programación y Lenguajes

Clasificación

Según mecanismo de traducción

- compilados (C, Java)
- interpretados (Python, Perl, Ruby)
- scripts interactivos (shell, Matlab)

Según nivel de abstracción

- bajo: ensamblador (atado a arquitectura)
- alto: C, C++, java (independiente de arq.)
- metaprogramación: genexus (generador de programas)

Programación y Lenguajes

Datos

- Cómo se codifica en memoria la información
- Tipos básicos: numéricos, textual, lógicos
- Nombre: cómo se identifica un dato en el programa
- Operaciones: qué se puede hacer con un dato de tal tipo

Ejemplo

dato	tipo	valor	codificación binaria
nombre	texto (ASCII)	PIE	01010000 01001001 01000101
edad	entero (8 bits)	25	00011001
peso	real (pto. fijo)	65.5	01000001 1000000

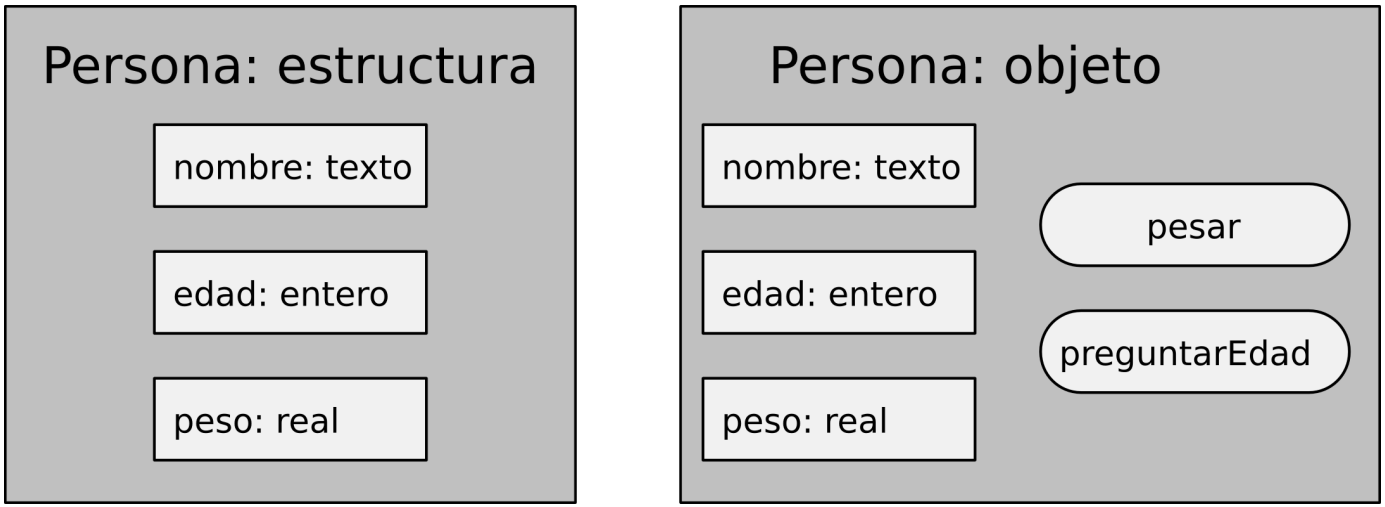
Programación y Lenguajes

Tipos de datos y estructuras

Tipos complejos a partir de tipos básicos:

estructuras conjunto de atributos, cada uno de un tipo particular

objetos conjunto de atributos, y comportamiento asociado



Programación y Lenguajes

Instrucciones y control de flujo

sentencias operaciones aritméticas, lógicas, e/s

control de flujo bifurcaciones, repeticiones, alternativas

interrupciones atención de eventos

excepciones manejo de errores

multi-tarea atender varias cosas simultáneamente

Programación y Lenguajes

Filosofía de diseño

Se consideran muchos aspectos, en general para favorecer cierto tipo de tareas. Por ejemplo:

- prolijidad vs. agilidad (ej., Python vs. Perl)
- generalidad vs. eficiencia (ej., Java vs C)
- eficiencia vs. portabilidad (ej., assembler vs. C)
- plasticidad vs. seguridad (ej., VisualBasic vs. Java)