UNIVERSIDAD NACIONAL GUILLERMO BROWN

Conceptos y Paradigmas de Lenguajes de Programación

Clase 5

SINTAXIS y SEMÁNTICA

SEMANTICA OPERACIONAL

- **REPASO CLASE ANTERIOR**

 Definición de Semántica

 Semántica Estática

 Formal: Gramática de Atributos

 Semántica Dinámica

 Formal: Semántica Axiomática – Semántica Denotacional

 No Formal: Semántica Operacional

 Procesamiento de los lenguajes

 Traductores

 Intérpretes

 Compiladores

 Proceso del compilador Análisis: Léxico, Sintáctico, Semántica Estática Síntesis: Optimización, Generación del código

- # VARIABLE

ENTIDAD	ATRIBUTO
Variable	nombre, tipo, área de memoria, etc
Rutina	nombre, parámetros formales, parámetros reales, etc
Sentencia	acción asociada

Entidad: “cosa” u “objeto” distinguible de otros objetos. – Atributo: propiedad de una entidad.

DESCRIPTOR: lugar donde se almacenan los atributos

- # CONCEPTO DE LIGADURA (BINDING)

Los programas trabajan con **entidades**

↓

Las entidades tienen **atributos**

↓

Estos atributos tienen que establecerse antes de poder usar la entidad

↓

LIGADURA: es la asociación entre la entidad y el atributo

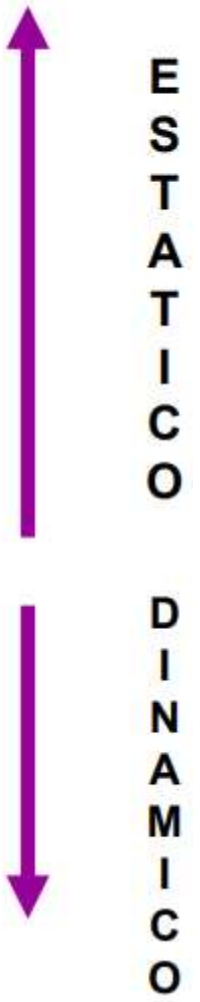
- # LIGADURA

Diferencias entre los lenguajes de programación

- El número de **entidades**
- El número de **atributos** que se les pueden ligar
- El **momento** en que se hacen las ligaduras (**binding time**).
- La **estabilidad** de la ligadura: una vez establecida se puede modificar?

Momento de Ligadura

- Definición del lenguaje
- Implementación del lenguaje
- Compilación (procesamiento)
- Ejecución



Momento y estabilidad

- Una **ligadura es estática** si se establece antes de la ejecución y no se puede cambiar. El termino estático referencia al momento del binding y a su estabilidad.

- Una **ligadura es dinámica** si se establece en el momento de la ejecución y puede cambiarse de acuerdo a alguna regla especifica del lenguaje.

  Excepción: constantes

MOMENTO Y ESTABILIDAD

Ejemplos:

- En **Definición**
  - Forma de las sentencias
  - Estructura del programa
  - Nombres de los tipos predefinidos
- En **Implementación**
  - Representación de los números y sus operaciones
- En **Compilación**
  - Asignación del tipo a las variables

En lenguaje C

int

Para denominar a los enteros

int

- Representación
- Operaciones que pueden realizarse sobre ellos

int a

- Se liga tipo a la variable

MOMENTO Y ESTABILIDAD

- En **Ejecución**
  - Variables con sus valores
  - Variables con su lugar de almacenamiento

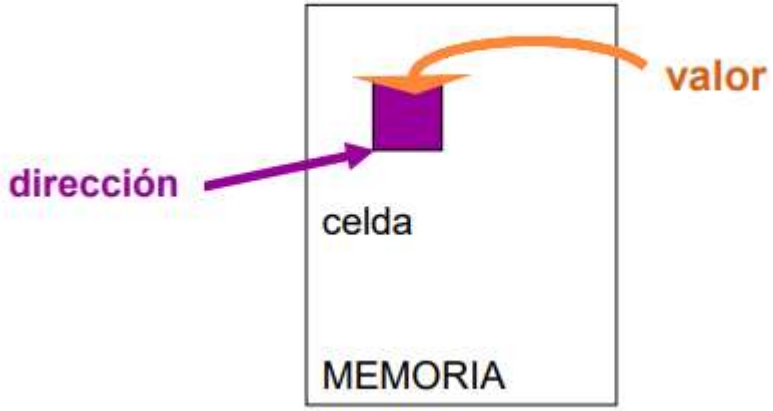
int a

- el valor de una variable entera se liga en ejecución y puede cambiarse muchas veces.

MOMENTO Y ESTABILIDAD

VARIABLES
CONCEPTO

- **Memoria principal: celdas** elementales, identificadas por una **dirección**.
- El contenido de una celda es una **representación codificada** de **un valor**



<NOMBRE, ALCANCE, TIPO, L-VALUE, R-VALUE>

- ***Nombre:*** string de caracteres que se usa para referenciar a la variable. (**identificador)**
- ***Alcance:*** es el rango de instrucciones en el que se conoce el nombre
- ***Tipo:*** valores y operaciones
- ***L-value:*** es el lugar de memoria asociado con la variable (**tiempo de vida**)
- ***R-value:*** es el valor codificado almacenado en la ubicación de la variable

<**NOMBRE**, ALCANCE ,TIPO, L-VALUE, R-VALUE>

Aspectos de diseño:

- Longitud máxima

  Algunos ejemplos: **Fortran**:6 **Python**: sin límite

  C: depende del compilador, suele ser de 32 y se ignora si hay más

  Pascal, Java, ADA: cualquier longuitud

- Caracteres aceptados (**conectores)**

  Ejemplo: Python, C, Pascal: _

  Ruby: solo letras minúsculas para variables locales

- Sensitivos

  Sum = sum = SUM ?

  Ejemplos: C y Python sensibles a mayúsculas y minúsculas

  Pascal no sensible a mayúsculas y minúsculas

palabra reservada - palabra clave

<NOMBRE, **ALCANCE** ,TIPO, L-VALUE, R-VALUE>

- El **alcance** de una variable es el rango de instrucciones en el que se conoce el nombre. **(visibilidad)**

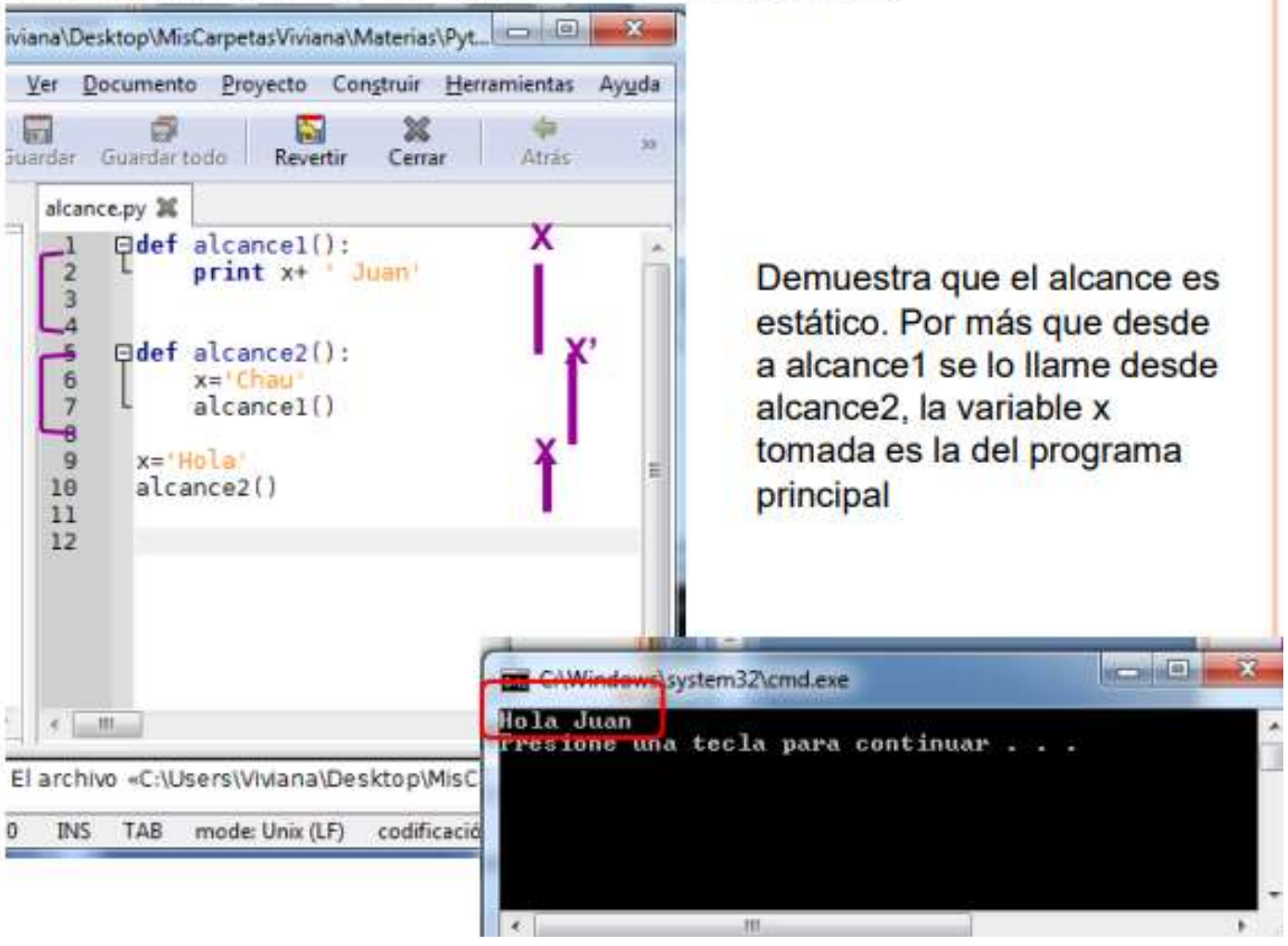
- Las instrucciones del programa pueden **manipular una variable** a través de su nombre **dentro de su alcance**

- Los diferentes lenguajes adoptan diferentes reglas para ligar un nombre a su alcance.

<NOMBRE, **ALCANCE** ,TIPO, L-VALUE, R-VALUE>

- **Alcance estático**
  - Llamado **alcance léxico**.
  - Define el alcance en términos de la estructura léxica del programa.
  - Puede ligarse estáticamente a una declaración (explícita o implícita) examinando el texto del programa, sin necesidad de ejecutarlo.
  - La mayoría de los lenguajes adoptan reglas de ligadura de alcance estático.
- **Alcance dinámico**
  - Define el alcance del nombre de la variable en términos de la ejecución del programa.
  - Cada declaración de variable extiende su efecto sobre todas las instrucciones ejecutadas posteriormente, hasta que una nueva declaración para una variable con el mismo nombre es encontrado durante la ejecución.
  - **APL**, **Lisp** (original), **Afnix** (llamado *Aleph* hasta el 2003), **Tcl** (Tool Command Language), **Perl**

ALCANCE EN PYTHON - ESTÁTICO



Demuestra que el alcance es estático. Por más que desde a alcance1 se lo llame desde alcance2, la variable x tomada es la del programa principal

Estático vs Dinámico

- Las reglas dinámicas son mas fáciles de implementar
- Son menos claras en cuanto a disciplina de programación
- El código se hacen mas difícil de leer

CONCEPTOS ASOCIADOS CON EL ALCANCE

- **Local:** Son todas la referencias que se han creado dentro del programa o subprograma.
- **No Local:** Son todas las referencias que se utilizan dentro del subprograma pero que no han sido creadas en él.
- **Global:** Son todas las referencias creadas en el programa principal

ESPACIOS DE NOMBRES

- Definición:
  - Un espacio de nombre es una zona separada donde se pueden declarar y definir objetos, funciones y en general, cualquier identificador de tipo, clase, estructura, etc.; al que se asigna un nombre o identificador propio.
- Utilidad:
  - Ayudan a evitar problemas con identificadores con el mismo nombre en grandes proyectos o cuando se usan bibliotecas externas.

<NOMBRE, ALCANCE, **TIPO**, L-VALUE, R-VALUE>

- **Definición:**
  - **Conjunto** de **valores**
  - **Conjunto** de las **operaciones**
- Antes de que una variable pueda ser referenciada debe **ligársele un tipo**
- **Protege a las variables de operaciones no permitidas**

Chequeo de tipos: verifica el uso correcto de las variables

- Predefinidos
  - Tipos base
- Definidos por el usuario
  - Constructores
- TADs

- **Tipos predefinidos**:
  - Son los tipos base que están descriptos en la definición

    Tipo boolean

    valores: ***true, false***

    operaciones: ***and, or , not***

  - Los valores se ligan en la implementación a representación de maquina

true	string	*000000.....1*
false	string	*0000.....000*

<NOMBRE, ALCANCE, **TIPO**, L-VALUE, R-VALUE>

- **Tipos definidos por el usuario**:
  - Los lenguajes permiten al programador mediante la declaración de tipos definir nuevos tipos a partir de los predefinidos y los constructores

<NOMBRE, ALCANCE, **TIPO**, L-VALUE, R-VALUE>

- **Tipos de Datos Abstractos:**
  - No hay ligadura por defecto, el programador debe **especificar** la **representación** y las **operaciones**

TAD

  - Estructura de datos que representan al nuevo tipo
  - Rutinas usadas para manipular los objetos de este nuevo tipo

Momentos - **Estático**

- El tipo se **liga en compilación y no puede ser cambiado**
  - El chequeo de tipo también será estático

Pascal, Algol, Simula, ADA, C, C++, Java, etc

<NOMBRE, ALCANCE, **TIPO**, L-VALUE, R-VALUE>

- Momento – Estático - **Implícito**
  - La ligadura se deduce por **reglas**
  - Ej. Fortran:
    - Si el nombre comienza con I a N es entera
    - Si el nombre comienza con letra A-H ó O- Z es real

Semánticamente la explicita y la implícita son equivalentes, con respecto al tipado de las variables, ambos son estáticos. El momento en que se hace la ligadura y su estabilidad es el mismo en los dos lenguajes.

- Momento – Estático - **Inferido**
  - El tipo de una expresión se deduce de los tipos de sus componentes
  - Lenguaje funcional. Ej. Lisp

Si se tiene en un script

doble x = 2 * x

Si no está definido el tipo **se infiere**
doble : : num -> num

- Momento – **Dinámico**

  - **El tipo se liga en ejecución y puede cambiarse**
    - Mas flexible: programación genérica
    - Mas costoso en ejecución: mantenimiento de descriptores
    - Variables polimórficas.
    - Chequeo dinámico
    - Menor legibilidad

APL, Snobol, Smalltalk, Python, Ruby, etc

- **Área de memoria ligada a la variable**
- **Tiempo de vida** (lifetime) o extensión:

  Periodo de tiempo que existe la ligadura

- **Alocación**:

  Momento que se reservar la memoria

El tiempo de vida es el tiempo en que la variable esté alocada en memoria

Momentos - **Alocación**

- **Estática:** sensible a la historia
- **Dinámica**
  - Automática; cuando aparece la declaración
  - Explícita: a través de algún constructor

- **Persistente**: su tiempo de vida no depende de la ejecución:

  existe en el ambiente

  - Archivos - Bases de datos

- Los lenguajes de programación permiten que un programa este compuesto por **unidades**.

UNIDAD ⟶ **acción abstracta**

- En general se las llama **rutinas**

PROCEDIMIENTOS **FUNCIONES** ⟶ un valor

- Analizaremos las características sintácticas y semánticas de las rutinas y los mecanismos que controlan el flujo de ejecución entre rutinas con todas las ligaduras involucradas.

Hay lenguajes que SOLO tienen “funciones” y “simulan” los procedimientos con “funciones que devuelven void”. Ej.: C, C++, Python, etc

<NOMBRE, ALCANCE, TIPO, **L-VALUE, R-VALUE**>

- ***l-value:*** Es el lugar de memoria en el que se almacena el cuerpo de la rutina.

- ***r-value:*** La llamada a la rutina causa la ejecución su código, eso constituye su r-valor.
  - **estático**: el caso mas usual.
  - **dinámica**: variables de tipo rutina.

  Se implementan a través de punteros a rutinas

COMUNICACIÓN ENTRE RUTINAS

- **Ambiente no local**
- **Parámetros**

Diferentes datos en cada llamado

Mayor legibilidad y modificabilidad.

- **Parámetros formales**: los que aparecen en la definición de la rutina
- **Parámetro reales**: los que aparecen en la invocación de la rutina. (dato o rutina)

UNIVERSIDAD NACIONAL GUILLERMO BROWN

ESTRUCTURA DE EJECUCIÓN DE LOS LENGUAJES DE PROGRAMACIÓN

- **Estático**
- **Basado en pila**
- **Dinámico**

ESTATICO: ESPACIO FIJO

- El espacio necesario para la ejecución se deduce del código
- Todo los requerimientos de memoria necesarios se conocen antes de la ejecución
- La alocación puede hacerse estáticamente
- No puede haber recursión

BASADO EN PILA: ESPACIO PREDECIBLE

- El espacio se deduce del código. Algol-60
- Programas más potentes cuyos requerimientos de memoria no puede calcularse en traducción.
- La memoria a utilizarse es **predecible** y sigue una disciplina last-in-first-out.
- Las variables se alocan automáticamente y se desalocan cuando el alcance se termina
- Se utiliza una estructura de pila para modelizarlo.

DINAMICO: ESPACIO IMPREDECIBLE

- Lenguajes con impredecible uso de memoria.
- Los datos son alocados dinámicamente solo cuando se los necesita durante la ejecución.
- No pueden modelizarse con una pila, el programador puede crear objetos de dato en cualquier punto arbitrario durante la ejecución del programa.
- Los datos se alocan en la zona de memoria heap

- Dudas - consultas??