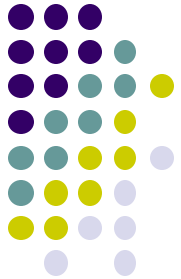
Conceptos



- Vinculación (o *binding*) es una asociación entre entidades de programación. Ej:
  - Una variable y su valor;
  - Un identificador y su tipo;
- Enfoque en la vinculación de identificadores a entidades;
- Importancia: la forma que un LP hace la vinculación define si el mismo es rígido o flexible;

Ejemplo: variables

¿Cuántas vinculaciones posee una variable?

Vinculación 4: la variable ocupa la dirección de memoria 0x…

```
// Código en C:
int var= 100;
```

Vinculación 2: la variable es de tipo int

Vinculación 1: la variable se llama var.

Vinculación 3: la variable posee valor 100.

Tiempos de Vinculación

Identifi-cador	Entidad	Tiempo de Vinculación
*	Operación de multiplicación	Diseño del LP
int	Intervalo de enteros	Diseño del LP (JAVA) implementación del compilador (C)
variable	Tipo de la variable	compilación (C) ejecución (polimorfismo en C++)
función	Código correspondiente de la función	linking
variable global	Posición de memoria ocupada	carga del programa
variable local	Posición de memoria ocupada	ejecución

- Vinculación Estática (hecha antes de la ejecución y no cambia).
- Vinculación Dinámica (cambia durante ejecución).

Identificadores

- Strings (términos) definidos por los programadores para servir de referencia a entidades de computación;
- Objetivo: aumentar la **legibilidad**, **facilidad de escritura y modificabilidad;**
- LPs pueden ser *case sensitive:*
  - C, C++, Java lo son, Pascal y Basic no;
  - Afecta legibilidad y facilidad de escritura
- LPs pueden limitar el número máximo de caracteres:
  - Pueden generar error o ignorar exceso;
  - Versiones iniciales de FORTRAN hacían eso;
- LPs pueden restringir caracteres especiales en los nombres.

Identificadores

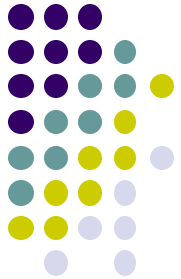
- Pueden tener significado especial:
  - Palabra reservada: no puede ser usada como identificador por el programador (ej.: goto en Java);
  - Palabra Clave: tiene significado pre-determinado en el lenguaje (ej.: goto en C, if en C y en Java);
  - Palabra pre-definida: tiene significado, pero el mismo puede ser redefinido (ej.: funciones de una API).

```
!Código válido en FORTRAN.
!Son palabras clave, pero no son palabras reservadas.
INTEGER REAL
REAL INTEGER
```

Ambientes de vinculación

- La interpretación de comandos y expresiones, tales como a = 5 o g(a + 1), dependen de lo que denotan los identificadores utilizados en esos comandos y expresiones
- Un ambiente (o *environment*) es un conjunto de vinculaciones.
- Cada vinculación posee un determinado **ámbito**, es decir, la región del programa donde la entidad es visible.

Ambientes de vinculación



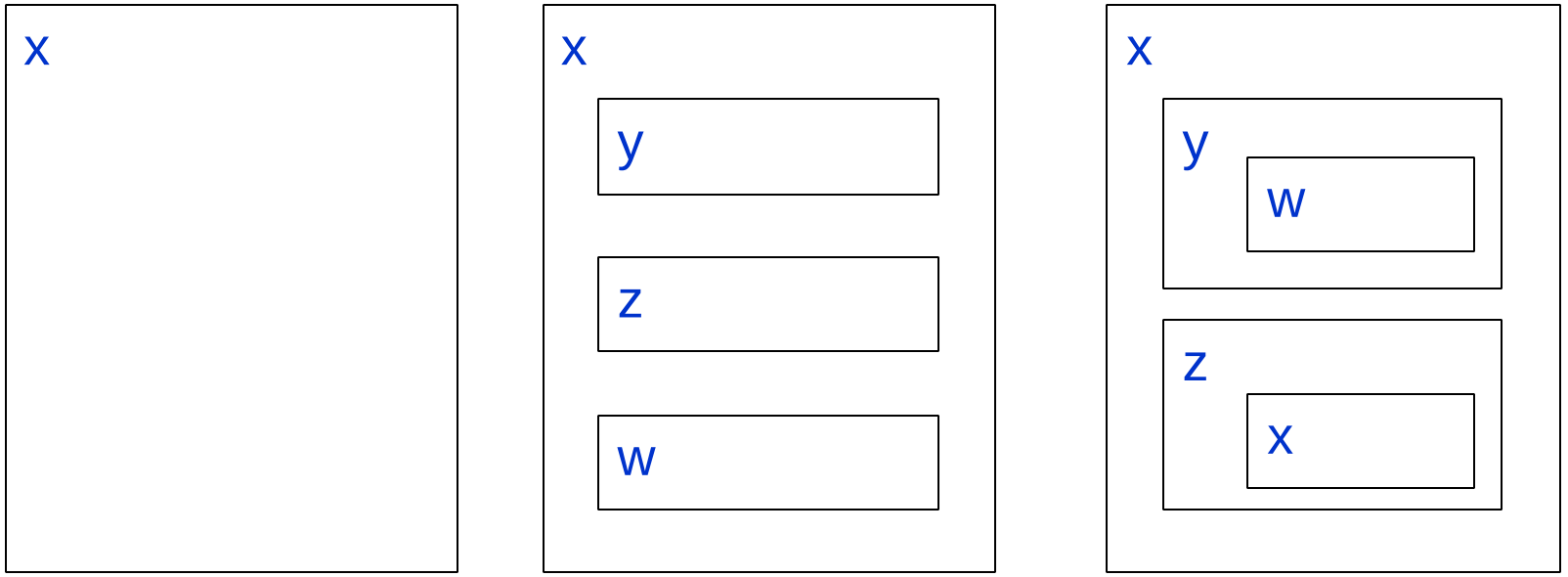
- Un Identificador puede estar vinculado a dos entidades distintas en el mismo Ambiente
- Ejemplo en C:

```
int a = 13;
void f( ) {
   int b = a;
   int a = 2;
   b = b + a;
} // Cual es el valor de b al final de f()?
```

Ámbito

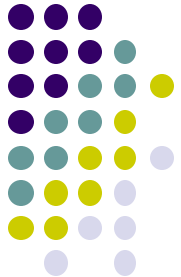
- Estático
  - definición del subprograma ;
  - tiempo de compilación;
  - texto del programa.
- Dinámico
  - llamada del subprograma
  - tiempo de ejecución
  - flujo de control del programa

Ámbito Estático



Bloque Monolítico Bloques No Anidados Bloques Anidados

Ámbito Estático



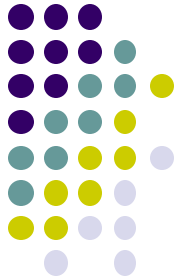
- Estructura Monolítica. Todo programa es compuesto por un único bloque. Las vinculaciones tienen como ámbito de visibilidad el programa entero. Es la más elemental posible y no es apropiada para programas grandes
- Estructura no anidada. El ámbito de visibilidad de los identificadores es el bloque donde fueron creados por los otros. Ej. Versiones antiguas de BASIC y COBOL. Identificadores que no pueden ser locales son forzados a ser globales
- Estructura Anidadas. Cualquier bloque puede ser anidado dentro de outro bloque y localizado en cualquier lugar que sea conveniente.

Ámbito Estático

- Ocultamiento de Entidad en Bloques Anidados

```
void main() {
    int i = 0,  x = 10;
    while (i++ < 100) {
          float x = 3.231;
          printf("x = %f\n", x*i);
    }
}
```

Ámbito Estático



- Referencia Selectiva en ADA

```
procedure A is
        x : INTEGER;
        procedure B is
        y : INTEGER;
        procedure C is
                        x : INTEGER;
                begin
                        x := A.x;
        end C;
        begin
                null;
        end B;
begin
        null;
end A;
```

Ámbito Estático

- Problemas con Estructura Anidada. Puede requerir que uma variable sea declarada globalmente a pesar que sea usada por pocos bloques.

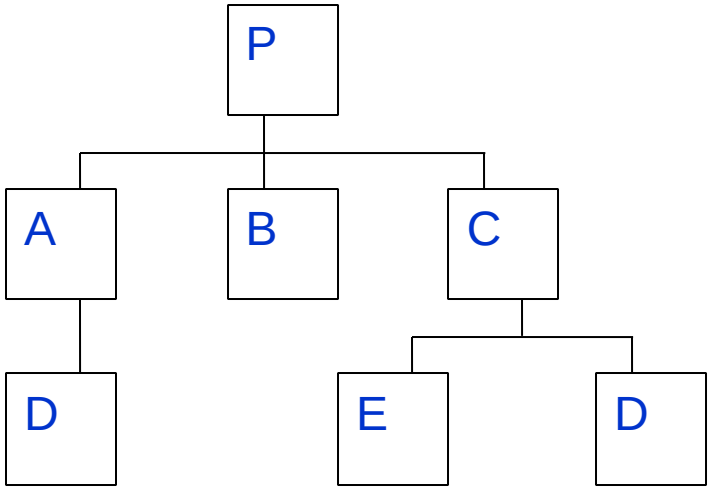


Figura a

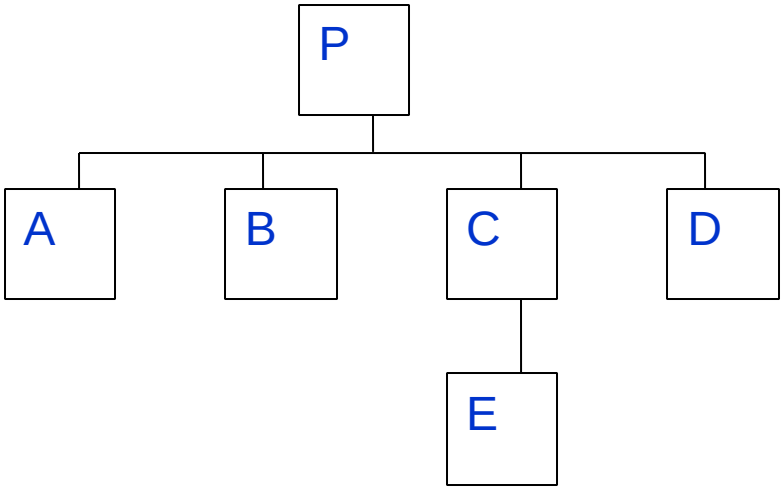
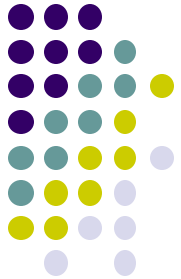


Figura b

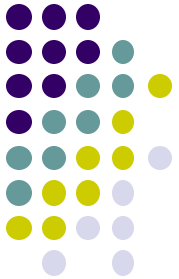
Ámbito Estático



- Estructura de Bloques de C.

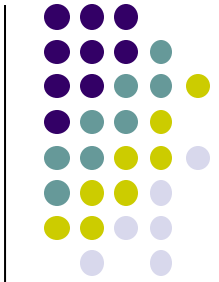
```
int x = 10;
int y = 15;
void f( ) {
   if (y – x) {
       int z = x + y;
   }
}
void g( ) {
    int w;
    w = x;
}
void main() {
    f();
    x = x + 3;
    g();
}
```

Ámbito Dinámico



```
procedimiento sub() {
          entero x = 1;
          procedimiento sub1() {
              escriba( x);
            }
          procedimento sub2() {
               entero x = 3;
              sub1();
            }
          sub2();
          sub1();
}
```

Ámbito Dinámico



- Problemas
  - Eficiencia: Verificación de tipos durante ejecución, acceso debe seguir secuencia de llamadas;
  - Legibilidad: se debe seguir la secuencia de llamadas para entender la vinculación
  - Confiabilidad: subprograma puede acessar variables locales del bloque que lo llama;
  - Propenso a errores del programador;
- Poco usado por LPs:
  - APL, Snobol4 y versiones iniciales de Lisp y Perl