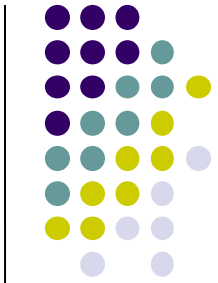
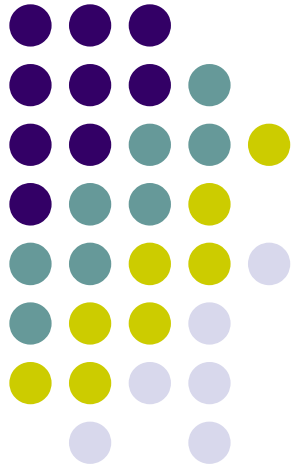
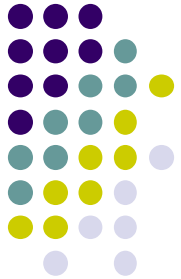


Expresiones



Expresiones

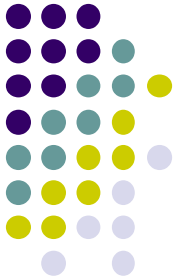


- Una expresión es una frase del programa que necesita ser evaluada y produce como resultado un valor

1 “aa” 1+4 f(1,g(x))

- **Elementos**
  - Operadores
  - Operandos
  - Resultado

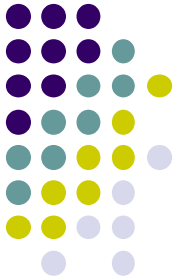
Expresiones



- Clasificación
  - Simples – apenas un operador
  - Compuestas – mas de un operador
- Notación
  - Prefijada: `a = !b; --b; -10`
  - Infijada: `a = a+b; c*d, 15-5`
  - Posfijada: `a = b++; b--;`
  - Otros: `x > y? x:y`
- Dialectos de Lisp permiten el uso de notación infijada o prefijada para operadores aritméticos:

  + a b

Operadores



- Aridad
  - unarios, binarios, ternarios, etc
  - enearios: aridad variable (varargs C/C++/Java)
  - Aridad elevada reduce legibilidad y facilidad de escritura

En Lisp:

(+ 1)

(+ 1 2)

(+ 1 2 4)

(+ 3 4 1 2)

...

el programador puede construir funciones con número de parámetros variable.

Operadores

En Java:

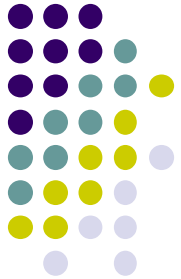
```
boolean positivo (int n) {
      return n > 0;
   }
```

- Origen
  - Pre-existentes
    - normalmente unarios y binarios
  - Definidos por el Programador
    - normalmente funciones con cualquier aridad
  - Composición de operadores – ML y APL

En ML:

```
val par = fn (n: int) => (n mod 2 = 0)
val negacion = fn (t:bool) => if t then false else true
val impar = negacion o par
```

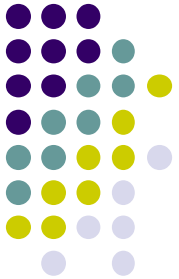
Tipos de Expresiones



- Podemos clasificar expresiones en diferentes tipos: Literales, agregaciones, aritméticas, relacionales, booleanas, binarias, condicionales, etc
- El tipo más simple de expresión son los **literales**;
- Ejemplos en C:

```
2,72 99 0143     'c'   0x43
```

Tipos de Expresiones



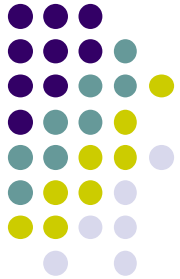
- Agregación
  - Subtipo de expresión compuesta;
  - Construye un valor a partir de sus componentes.

```
int c[ ] = {1, 2, 3};
struct fecha d = {1, 7, 1999};
char * x = {'a','b','c','\0'};
int b[6] = {0};  //inicializa todos los val. de b
char * y = "abc"; //equivale a la 3ra línea
```

  - Agregaciones pueden ser estáticas o dinámicas.

```
void f(int i) {
    int a[] = {3 + 5, 2, 16/4}; // Estática
    int b[] = {3*i, 4*i, 5*i}; // Dinámica
    int c[] = {i + 2, 3 + 4, 2*i}; // Mixta
}
```

Expresiones de Agregación

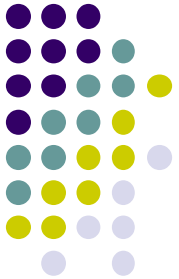


- En C, la agregación solo puede ser hecha en operaciones de inicialización y en el caso de strings constantes.
- Otros lenguajes (ej.: ADA) son más flexibles:

```
type fecha is record
    dia : integer range 1..31;
    mes : integer range 1..12;
    ano : integer range 1900..2100;
end record;

aniversario: fecha;
fecha_admision: fecha := (29, 9, 1989);
aniversario := (28, 1, 2001);
fecha_admision  := (dia => 5, ano => 1980, mes => 2);
```

Tipos de Expresiones

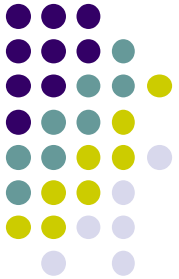


- Aritméticas

```
float f;
int num = 9;
f = num/6;
f = num/6.0;
```

  - + y – unários y binários
- Relacionales
  - Usadas para comparar los valores de los operandos
- Booleanas
  - Realizan las operaciones de negación, conjunción y disyunción del álgebra de Boole

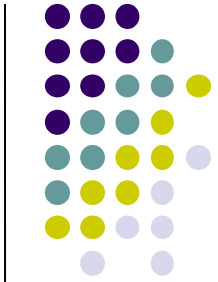
Tipos de Expresiones



- Binarias.- operaciones lógicas bit a bit:

```
void main() {
    int j = 10;
    char c = 2;
    printf("%d\n", ~0);     /* imprime  -1 */
    printf("%d\n", j & c); /* imprime   2 */
    printf("%d\n", j | c); /* imprime  10 */
    printf("%d\n", j ^ c); /* imprime   8 */
    printf("%d\n", j << c);/* imprime  40 */
    printf("%d\n", j >> c);/* imprime   2 */
}
```

Tipos de Expresiones



- **Condicionales**
  - En ML

```
val c = if a > b  then a - 3 else b + 5
val k = case i of
          1 =>  if  j > 5  then  j - 8  else  j + 5
        | 2 => 2*j
        | 3 => 3*j
        | _ => j
```

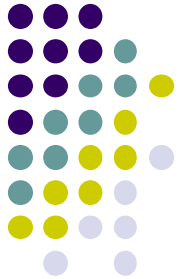
  - En JAVA

```
max = x > y ? x : y;
```

  - Algunos LPs (tal como ADA) no ofrecen Expresiones Condicionales – forzan el uso de comandos condicionales (if):

```
if x > y then max := x; else max := y; end if;
```

Tipos de Expresiones



- **Llamadas de Funciones**: también son consideradas expresiones:
  - Operador = nombre de la función;
  - Operandos = parámetros;
  - Resultado = retorno de la función;

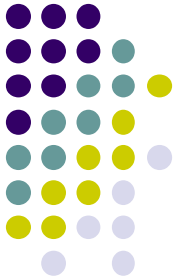
- Llamada Condicional de Función en ML

```
val taza = (if difPgVenc > 0 then descuento else
multa) (difPgVenc)
```

- Simulando esa misma construcción en C

```
//puntero para función:
double (*p)(double);
p = difPgVenc < 0 ? descuento: multa;
taza = (*p) (difPgVenc);
```

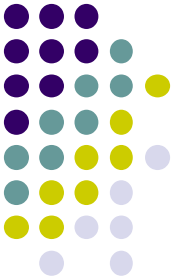
Tipos de Expresiones



- Operadores de los LPs denotan funciones

Expresión	Representación Prefijada
a * b	*(a , b)
c / d	/(c , d)
a * b + c / d	+ (*(a,b) , /(c,d))

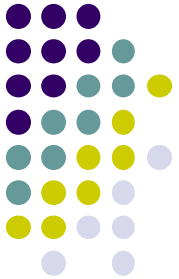
Tipos de Expresiones



- Algunas signaturas de Operadores en JAVA

Operador	Signatura de la Función
!	[boolean → boolean]
&&	[boolean x boolean → boolean]
*	[int x int → int] [float x float → float]

Tipos de Expresiones



- Con Efectos Colaterales

```
x = 3.2 * ++c;
```

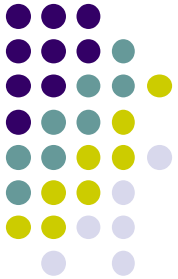
  - Pueden generar indeterminismo

```
x = 2;
y = 4;
z = (y = 2 * x + 1) + y;
printf("%d, %d, %d\n", x, y, z);
```

  - Funciones posibilitan la ocurrencia de efectos colaterales
  - Expresiones cuyo único objetivo es producir efectos colaterales

```
delete p;
```

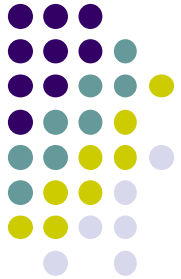
Tipos de Expresiones



- Referenciamiento
  - Usadas para accesar el contenido o retornar referencia para variables o constantes

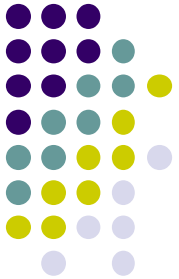
```
*q = *q + 3;
const float pi = 3.1416;
int radio = 3;
float perimetro = 2*pi*radio;
p[i] = p[i + 1];
*q = *q + 3;
r.anho = r.anho + 1;
s->dia = s->dia +1;
t = &m;
```

Referenciamiento



Operador	Significado
[ ]	Acceso a valor o retorno de referencia de elemento de vector
*	Acceso a valor o retorno de referencia de variable o constante apuntada por puntero
.	Accesso a valor ou retorno de referencia de elemento de estructura
->	Acceso a valor o retorno de referencia de elemento de estructura apuntada por puntero
&	Retorno de referencia a cualquier tipo de variable o constante

Tipos de Expresiones



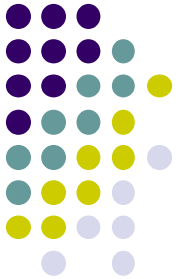
- Categóricas
  - Realizan operaciones sobre tipos de dados
  - Tamaño del Tipo

```
float * p = (float *) malloc (10 * sizeof (float));
int c [] = {1, 2, 3, 4, 5};
for (i = 0; i < sizeof c / sizeof *c; i++) c[i]++;
```

  - Conversión de Tipo

```
float f;
int num = 9, den = 5;
f = (float)num/den;
```

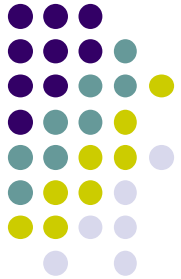
Expresiones Categóricas



- Identificación de Tipo

```
Profesion p = new Ingeniero ( );
if (p instanceof Medico)
        System.out.println  ("Registrese  en  el
CMP");
if (p instanceof Ingeniero )
        System.out.println  ("Registrese  en  el
CIP");
```

Evaluación de Expresiones Compuestas

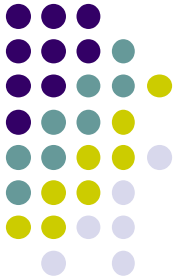


- Precedencia de Operadores
  - Elección inadecuada puede afectar a facilidad de escritura

```
/* if a > 5 and b < 10 then */
if (a > 5) and (b < 10) then a := a + 1;
```

  - Memorizar el orden dificulta el aprendizaje
  - Ausencia de precedencia (SMALLTALK y APL) baja a facilidad de escritura
  - Paréntesis aseguran el orden, pero reducen facilidad de escritura e impiden optimizaciones

Evaluación de Expresiones Compuestas



- Asociatividad de Operadores
  - Operadores de Misma Precedencia
  - Normalmente de izquierda para la derecha

```
x = a + b - c;
y = a < b < c;
```

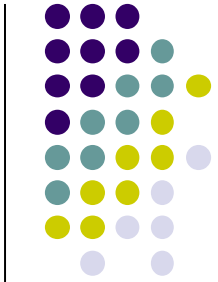
  - Pueden existir excepciones a esa regla

```
x = **p;
if (!!x) y = 3;
a = b = c;
```

  - APL no tiene precedencia y siempre asocia de la derecha para la izquierda

```
X = Y ÷ W - Z
```

Evaluación de Expresiones Compuestas



- Asociatividad de Operadores
  - En Fortran, exponenciación es de la derecha para la izquierda.
  - Compiladores pueden optimizar, pero eso puede causar problemas

```
x = f() + g() + h();
```

- Precedencia de Operandos

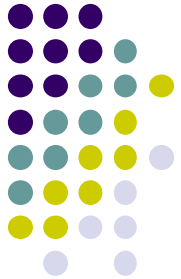
```
x = 2;
y = 4;
z = (y = 2 * x + 1) + y;
```

  - No determinismo en Expresiones

```
a[i] = i++;
```

  - JAVA resuelve adoptando precedencia de operandos de izquierda para derecha. Garantiza portabilidad, pero compromete eficiencia (impede optimizaciones especificas de plataforma)

Evaluación de Expresiones Compuestas



- Corto Circuito
  - Situación Potencial

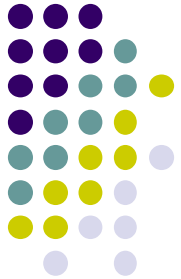
```
z = (x - y) * (a + b) * (c - d);
```

  - Generalemente usado en Expresiones Booleanas

```
int[] a = new int [n];
i = 0;
while (i < n && a[i] != v) i++;
```

  - JAVA y ADA tiene operadores específicos para evaluación con (&&, ||) y sin (&, |) corto circuito;
  - Pascal no posee corto circuito. Algunos compiladores permiten activarlo, pero entonces es usado siempre.

Evaluación de Expresiones Compuestas



- Corto Circuito
  - Operadores booleanos de C y C++ usan corto circuito
  - Se puede usar operadores binários *&* *y* | pues no usan corto circuito
  - Corto circuito con efectos colaterales reduce facilidad de escritura

```
if (b < 2*c || a[i++] > c ) {
    a[i]++;
}
```