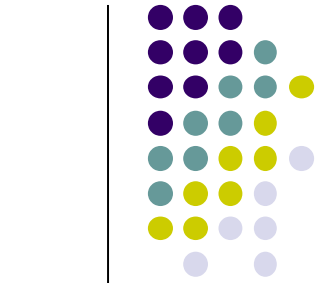
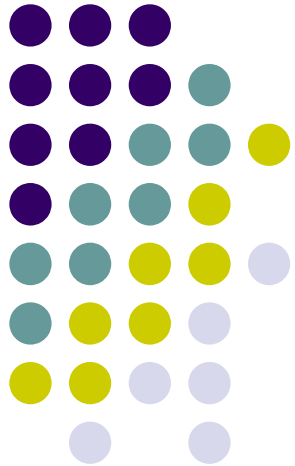
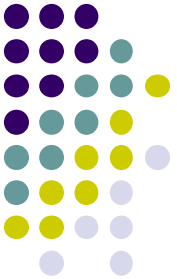
Comandos



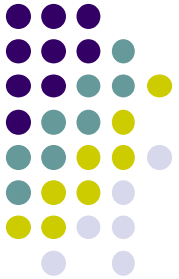


Comandos



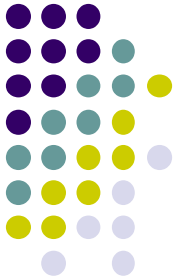
- Objetivo es actualizar variables o controlar el flujo de control
- Característicos de LPs imperativas
- Pueden ser primitivos o compuestos
- Bastan asignación, selección y desvio, sin embargo LP queda poco expresivo

Tipos de Comandos



- De acuerdo con [Watt, 1990]:
  - Asignaciones;
  - Comandos secuenciales;
  - Comandos colaterales;
  - Comandos condicionales;
  - Comandos iterativos;
  - Comandos de procedimientos;
  - Comandos de desvio incondicional.

Tipos de Comandos- Asignación



- = versus :=
  - Ada, Pascal, Modula2 y APL (ejemplo abajo) usan := para no confundir con = (matemática);

```
i := !i + 1
```

Dereferenciamiento explícito: i vinculado a 2 conceptos diferentes

```
if (a = 10) a += 3;
```

- Asignación Simple

```
a = b + 3 * c;
```

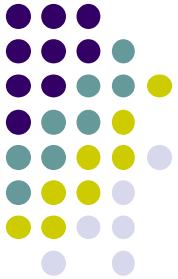
- Asignación Múltiple

```
a = b = 0;
```

- Asignación Condicional

```
(if a < b then a else b) := 2;
```

Tipos de Comandos - Asignación



- Asignación compuesta

```
a += 3;
a *= 3;
a &= 3;
```

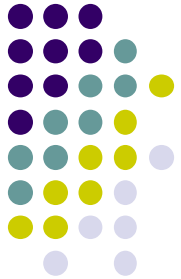
- Asignación Unaria

```
++a;
a++;
--a;
a--;
```

- Asignación como Expresión

```
while (( ch = getchar ( ) ) != EOF ) { printf("%c", ch); }
```

Tipos de Comandos- Secuenciales



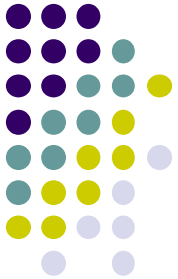
- Considerar un conjunto de comandos como uno sólo.

```
if (x > y) printf("x > y\n"); // Un comando.

if (x > y) {          // Bloque de comandos
      n = 1;
    n += 3;
      if (n < 5) {
            n = 10;
            m = n * 2;
      }
}
```

- Delimitadores de bloque: { / }, begin/end, identación, etc

Tipos de Comandos- Colaterales



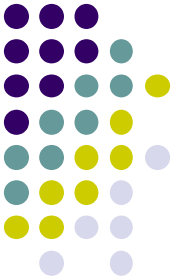
- Comandos que permiten procesamiento paralelo;
- Muy raros en LPs (excepción en LP reciente: Go);
- Ejemplo en ML:

```
a = 0;
a = 3, a = a + 1;

val altura = 2
and largo = 3
and ancho= 5
and volume = altura * largo * ancho
```

La última parte genera error, pues usa identificadores presentes en el mismo comando colateral, lo que es vetado por el lenguaje.

Tipos de Comandos- Condicionales



- Selección de camino condicionado

```
if (x < 0) { x = y + 2; x++; }
```

- Selección de camino doble

```
if (x < 0) { x = y + 2; x++; } else { x = y; x--; }
```

- Potencial problema con marcadores:

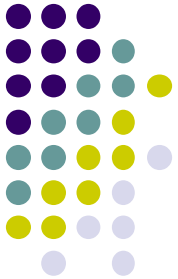
```
if ( x == 7 )
   if ( y == 11) {
      z = 13;
      w = 2;
   }
else z = 17;
```

```
if ( x == 7 ) {
   if ( y == 11 ) {
        z = 13;
        w = 2;
   }
} else z = 17;
```

ADA requiere marcador de final de comando

```
if x > 0 then
   if y > 0 then
          z := 0;
   end if;
else
   z := 1;
end if;
```

Tipos de Comandos- Condicionales

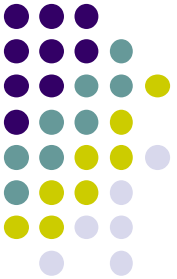


- Selección de Caminos Múltiples

```
switch (nota) {
   case 10:
   case 9: printf (“Muy Bien!!!”);
       break;
   case 8:
   case 7: printf (“Bien!”);
       break;
   case 6:
   case 5: printf (“Pasó…”);
       break;
   default: printf (“¡Estudiar más!”);
}
```

- Fortran: goto;
- Ada y Pascal: switch sin break;
- Python: no posee (purismo OO).

Tipos de Comandos- Condicionales

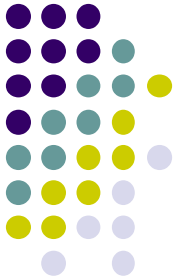


- Caminos Múltiples con ifs Anidados

```
if (rentaMes < 1000)
   iR = 0;
else if (rentaMes < 2000)
   iR = 0.15 * (2000 - rentaMes);
else
   iR = 0.275*(rentaMes-2000)+0.15*(2000-rentaMes);
```

  - Modula-2, Ada y Fortran-90 tiene elsif

Tipos de Comandos- Iterativos

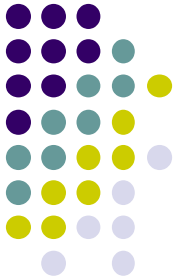


- Número Indefinido de Repeticiones
  - Pre-test y Pos-test

```
f = 1;
y = x;
while ( y > 0) {
    f = f * y;
    y--;
}
```

```
f = 1;
y = 1;
do {
     f = f * y;
     y++;
} while (y <= x);
```

Tipos de Comandos- Iterativos



- Problema con Pre-Test y Pos-Test : seria conveniente interrumpir la repetición después de algun comando interno del cuerpo de la repetición

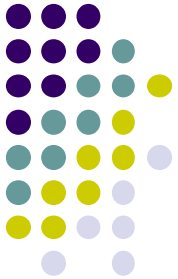
```
s = 0;
printf ("n: ");
scanf ("%d", &n);
while (n > 0) {
    s +=n;
    printf ("n: ");
    scanf ("%d", &n);
}
```

- Repetición de la lectura

```
s = 0;
do {
    printf ("n: ");
    scanf ("%d", &n);
    if (n > 0) s+=n;
} while (n > 0);
```

- Repetición de la verificación $n>0$
- Impacta en facilidad de escritura y eficiencia del código

Tipos de Comandos- Iterativos

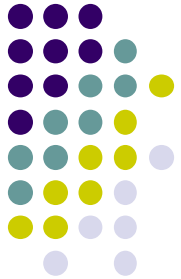


- Número definido de repeticiones
  - En Modula-2

```
s := 0;
FOR i := 10 * j TO 10 * (j + 1) BY j DO
    s := s + i;
END;
```

- Consenso
  - Valores de la variable de control conocidos antes del primer ciclo y fijos.
  - Realización del testeo antes de la ejecución del cuerpo

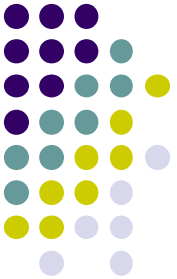
Tipos de Comandos- Iterativos



- Variación en el ámbito de la variable de Control
  - ADA y JAVA restringen al cuerpo
  - Ada no permite alteración de la variable de control en el cuerpo de la repetición.
  - FORTRAN, PASCAL y C tratan como variable ordinaria
  - C++ permite que ámbito comience en el comando

Algunos LPs no especifican si la variable de control puede ser usada como variable común después de la repetición, dejando a cargo del implementador del compilador (-portabilidad).

Tipos de Comandos- Iterativos



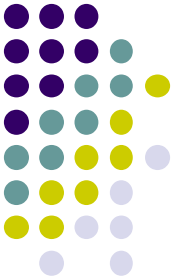
- Número definido de Repeticiones
  - Comando for de C

```
dif = 0;
for (i = 0; i < n; i++) {
    if (a[i] % 2 == 0) dif += a[i];
    if (a[i] % 3 == 0) dif -= a[i];
}


for (i = 10 * j, s = 0; i <= 10 * (j + 1); s += i++);

for (i = 0, s = 0; i <= n && s < 101 && a[i] > 0 ; )
     s += a[i++];
for (;;);
```

Tipos de Comandos- Iterativos

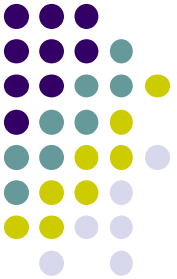


- Número definido de Repeticiones
  - Puede no restringirse a Tipos Primitivos Discretos

```
@dias = ("Dom", "Seg", "Ter", "Qua","Qui", "Sex", "Sab");
foreach $dia (@dias) {
      print $dia
}
```

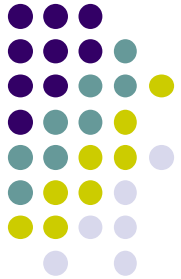
  - Mayoria de los LPs no ofrece
  - JAVA y C++ ofrecen iteradores asociados a colecciones
  - Java (>= 5) y C++ ofrecen for-each

Tipos de Comandos- Llamada de Procedimientos



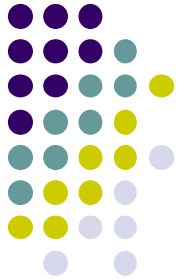
- Objetivo es actualizar variables
- En algunos LPs, substituido por función que retorna un valor vacio (void).

Tipos de Comandos- Desvios Incondicionales



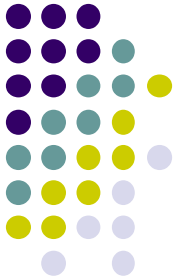
- Solamente comandos de Entrada y Salida única pueden ser restrictivos en algunas situaciones
- Entrada única y salidas múltiples es positivo
- Entradas Múltiples es negativo
- Tipos
  - Desvios Irrestrictos;
  - Escapes.

Desvio Irrestricto



- Conocido como comando goto
- Puede ser nocivo a la buena programación
- Algunos LPs lo eliminaron (MODULA-2, Java)
  - En Java, goto es palabra reservada;
  - Combinación de escape (rotulado) y excepciones han sido suficiente para no precisar de goto;
- Es importante en algunas situaciones
- También es usado para propagación de errores en LPs sin tratamiento de Excepciones
  - Visual Basic: On Error GoTo X

Necesidad de goto



- Es necesario en algunos casos

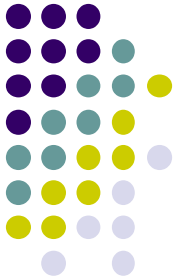
Sin goto

```
encontro = 0;
for (i = 0; i < n && !encontro; i++)
   for (j = 0; j < n && !encontro; j++)
      if ( a[i] == b[j] ) encontro = 1;
if (encontro) printf ("encontro!!!");
else printf ("no encontro!!!");
```

Con goto

```
for (i = 0; i < n; i++)
     for (j = 0; j < n; j++)
          if ( a[i] == b[j] )
              goto salida;
saida:
if (i < n) printf ("encontro!!!");
  else printf ("no encontro!!!");
```

Escapes



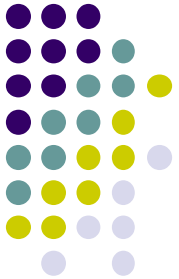
- Desvios incondicionales estructurados
- No pueden crear o entrar en repeticiones
  - break

```
s = 0;
for(;;) {
    printf ("n: ");
    scanf ("%d", &n);
    if (n <= 0) break;
    s+=n;
}
```

  - continue

```
i = 0;
s = 0;
while(i < 10) {
    printf ("n: ");
    scanf ("%d", &n);
    if (n < 0) continue;
    s+=n;
    i++;
}
```

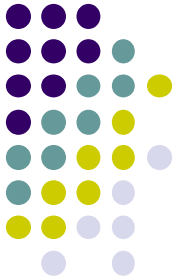
Escapes



- Asociación con iteraciones rotuladas puede ser útil
- ADA y JAVA ofrecen

```
salida:
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        if ( a[i] < b[j] ) continue salida;
        if ( a[i] == b[j] ) break salida;
    }
}
if (i < n) printf ("encontró!!!");
else printf ("no encontró!!!");
```

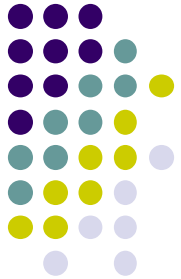
Escapes



- Pueden interrumpir la ejecución de Subprogramas y Programas

```
void trata (int error) {
     if (error == 0) {
    printf ("nada a tratar!!!");
    return;
     }
     if (error < 0) {
    printf ("error grave – nada para hacer!!!");
    exit (1);
     }
     printf("error tratado!!!");
}
```

Consideraciones



- LP queda empobrecido cuando no ofrece tipos de expresiones o los comandos listados
- Expresiones y comandos adicionales pueden no acrescentar nada
  - Entrada y salida en COBOL y FORTRAN versus llamadas de procedimiento
- Existen intersecciones entre Expresiones y Comandos
  - LPs Orientados a Expresión (ALGOL-68 y ML)
  - C es orientado a expresión si consideramos Flujo de Control retornando void