Arreglos y Subrangos

Programación 1

InCo - FING

Section 1

Tipos de datos definidos por el programador

Tipos de datos definidos por el programador

- Hasta ahora se vio que existen tipos de datos predefinidos (por ejemplo: `integer`, `char`, etc.).
- Pascal permite que el programador defina nuevos tipos de datos en un programa.
- Para definir un nuevo tipo de datos, se utiliza la palabra reservada **type**
- Existen diversas categorías de tipos de datos que pueden ser definidos por el programador (por ej.: subrangos, enumerados y estructurados).
- En la clase de hoy veremos los subrangos y los arreglos (estos últimos constituyen un caso particular de tipo estructurado).

Section 2

Tipo subrango

Subrangos de enteros

Se puede restringir el tipo de los enteros de forma de acotar los valores en un rango especificado:

```
type
    (* tipos definidos por el programador *)
    natural   = 0..MaxInt;
    decimal   = 0..9;
    mes       = 1..12;
    dia       = 1..31;
    otro      = 1000..13890;
```

Subrangos de caracteres

```
type
    letra  = 'A'..'Z';
    digito = '0'..'9';
    decimal   = 0..9;
```

- Los subrangos de caracteres se basan en el orden predefinido en el tipo `char` (en Free Pascal la tabla **ASCII**).
- Notar la diferencia entre los tipos `digito` y `decimal`.

Operaciones con subrangos

- Constantes, expresiones, y variables de un tipo subrango se comportan como el tipo *base* que lo contiene.
- Si una variable de un tipo subrango toma un valor fuera del rango especificado se produce un **error en tiempo de ejecución**.
- El compilador Free Pascal **requiere ser configurado** (activación de la flag `-Cr`) para realizar la verificación de rango.
- Puede dar un *error de compilación* si es posible advertir una violación del rango.

Subrangos en general

La declaración de un tipo subrango tiene esta forma:

```
type identificador = C1 .. C2;
```

donde `C1` y `C2` deben ser constantes del mismo tipo **ordinal**.

Es posible definir subrangos de cualquier tipo ordinal:

- `integer`, `char`, `boolean`, **enumerado**

(Tipos enumerados se ven más adelante)

(A los tipos ordinales también se les llama **escalares**)

Section 3

Arreglos

Tipos estructurados

Cada elemento del tipo está formado por un conjunto de valores (una *estructura*)

Tipos estructurados de Pascal:

- Arreglos (array)
- Registros (record)
- Conjuntos (set)

Motivación

- Almacenar un conjunto de valores bajo un mismo nombre de variable.
- Cada uno de los valores se puede acceder independientemente utilizando un índice (generalmente un subrango)
- Está inspirado en la notación habitual en matemáticas para secuencias finitas:
  $a_1, a_2, ..., a_n$

  donde a sería el nombre genérico de todos los valores y el subíndice es utilizado para identificar un valor en particular.

Declaración de un arreglo

```
type
    RangoArreglo = 1..9;
    Arreglo      = array [RangoArreglo] of integer;
var
    A : Arreglo;
```

Declaración de un arreglo

```
type
    RangoArreglo = 1..9;
    Arreglo      = array [RangoArreglo] of integer;
var
    A : Arreglo;
```

En nuestro ejemplo el arreglo ocupa 9 celdas:

```
*------*------*------*------*-----*-----*-----*-----*-----*
|  23  |  34  |  0   | -12  |  6  |  9  |  11 |  -2 | 34  |
*------*------*------*------*-----*-----*-----*-----*-----*
   1      2      3      4      5     6      7     8     9
```

Espacio de memoria: Un arreglo ocupa tantas celdas de memoria como el cardinal (número de elementos) de su tipo índice.

Declaración de un arreglo (2)

Los tipos arreglo se declaran en la sección `type` del programa:

```
type nombre = array [tipo_indice] of tipo_base;
```

donde:

- `tipo_indice` debe ser un tipo *ordinal*, generalmente es un subrango de enteros.
- `tipo_base` es cualquier tipo de Pascal.

Ejemplos de arreglo

```
type
    rango = 33..90;
    arr1 = array [char] of integer; (* 256 celdas *)
    arr2 = array [33..90] of real;
    arr3 = array [integer] of char; (* demasiado grande! *)
    arr4 = array ['0'..'9'] of arr3;(* matriz *)
    arr5 = array [rango] of boolean;
```

Tipos Anónimos

Son tipos que se utilizan sin asignarles un nombre.

```
type
    (* indice anonimo *)
    arreglo = array [0..9] of real;
var
    (* arreglo anonimo *)
    arr : array ['A'..'Z'] of boolean;
```

Los tipos anónimos se utilizan comúnmente cuando un tipo sólo aparece una vez.

Accediendo un arreglo

Cada celda del arreglo se puede acceder como una variable independiente.

Suponemos `A` del tipo `array [1..10] of integer`

```
(* asignaciones *)
A[1]:= 12;
A[10]:= 90;
....
A[i]:= i+1;   (* i en el intervalo [1,10]   *)
A[i+4]:= 20;  (* i+4 en el intervalo [1,10] *)

A[200]:= 23;  (* error, fuera de rango *)

A:= 0;        (* error *)
```

Inicialización

Para inicializar un arreglo es necesario asignar cada celda por separado:

```
(* celdas de 1 a 10 en cero *)
procedure ArregloEnCero(var A: arreglo);
var i: integer;
begin
  for i:= 1 to 10 do
   A[i]:= 0;
end;

(* celdas de 1 a 10 en pares 2, 4, 6, ... *)
procedure ArregloEnPares(var A: arreglo);
var i: integer;
begin
for i:= 1 to 10 do
   A[i]:= 2 * i;
end;
```

Inicialización

```
(* celdas leidas desde la entrada *)
procedure LeerArreglo(var A: arreglo);
var i: integer;
begin
  for i:= 1 to 10 do
   ReadLn(A[i]);
end;
```

Recorridas de arreglos

La estructura de control for es la adecuada para recorrer **completamente** un arreglo

```
function SumarTodos(A: arreglo): integer;
var  i : integer;
begin
suma:= 0;      (* suma de todos los elementos *)
for i:= 1 to 10 do
 suma:= suma + A[i];
    SumarTodos := suma;
end;

 procedure SumarATodos(incremento: integer; var A: arreglo);
 var  i : integer;
 begin
  for i:= 1 to 10 do
   A[i]:= A[i] + incremento; (* suma incremento a todos los elemento
 end;
```

Recorridas de arreglos

```
function maximo(A: arreglo): integer;
var max,i : integer;
begin
   max:= A[1];  (* hallar el maximo del arreglo  *)
   for i:= 2 to 10 do
      if A[i] > max
       then max:= A[i];
   maximo := max;
end;
```

Recorridas de arreglos

```
procedure MostrarArreglo1(A: arreglo);
var  i : integer;
begin
  for i:= 1 to 10 do
      (* desplegar indice y elemento *)
      WriteLn(i,'-->',A[i]);
end;

procedure MostrarArreglo2(A: arreglo);
var i : integer;
begin
  (* desplegar separados por comas *)
  write(A[1]);
  for i:= 2 to M do
    write(',',A[i]);
end;
```

Búsqueda en un arreglo

- Para realizar una búsqueda **NO es correcto** utilizar `for` (aunque "anda").
- La estructura `for` es **ineficiente** porque continúa buscando aún después de haber encontrado un valor que cumpla con la condición deseada.

```
(* INCORRECTO! *)
encontre:= false;
for i:= PRIMERO to ULTIMO do
   if (cumple_condicion(A[i])) then
      encontre:= true;
```

Búsqueda en un arreglo (2)

- Para realizar una búsqueda, lo correcto es utilizar una estructura de **repetición condicional**.
- La búsqueda debe *detenerse* cuando se encuentre un valor del arreglo que cumple con la condición deseada o bien cuando se llegue al final del arreglo.
- El siguiente código ilustra el esquema genérico de una búsqueda.

```
(* CORRECTO! *)
i:= PRIMERO;
while (i <= ULTIMO) and not cumple_condicion(A[i]) do
      i:= siguiente(i);
```

Recordemos que Free Pascal realiza evaluación por **circuito corto**, por lo que este esquema de búsqueda funciona correctamente.

Búsqueda en un arreglo (3)

Repasemos cómo funciona la evaluación por circuito corto.

- Para evaluar (`E1 and E2`)
  1. Se evalúa `E1`. Sea `b1` su valor.
  2. Si `b1` es `false` entonces **no se evalúa E2** y el resultado es `false`.
  3. Si `b1` es `true` entonces se evalúa `E2`. Sea `b2` su valor.
  4. El resultado es `b2`.

La mayoría de los lenguajes de programación evalúan por circuito corto, por lo cual el esquema de búsqueda anterior funciona en todos ellos.

Búsqueda en un arreglo (4)

Veamos algunos ejemplos concretos de búsqueda con circuito corto.

En primer lugar, buscamos un valor concreto x dentro del arreglo A.

```
type arreglo = array [1..M] of integer;

function pertenece(x: integer; A: arreglo): boolean;
(* retorna true si x pertenece al arreglo *)
var i: integer;
begin
   i:= 1;
   (* evaluacion por circuito corto *)
   while (i <= M) and (A[i] <> x) do
        i:= i + 1;
   pertenece:= i <= M
end; {pertenece}
```

Cuando `i` toma el valor `M+1`, **no** se accede a `A[i]`

Búsqueda en un arreglo (5)

El siguiente código funciona aún sin circuito corto (¿por qué?):

```
function pertenece(x: integer; A: arreglo): boolean;
(* retorna true si x pertenece al arreglo *)
var i: integer;
begin
   i:=1;
   while (i < M) and (A[i] <> x) do
         i:= i+1;
   pertenece := A[i] = x
end;
```

Búsqueda en un arreglo (6)

El siguiente código **no funciona** aún con circuito corto (¿por qué?).

```
(* INCORRECTO! *)
function pertenece(x: integer; A: arreglo): boolean;
(* retorna true si x pertenece al arreglo *)
var i: integer;
begin
  i:= 1;
  while (a[i] <> x) AND (i<=M) do
        i:= i+1;
  pertenece := i <= M
end;
```

Búsqueda en un arreglo (7)

Veamos otro ejemplo de búsqueda. Encontrar un elemento tal que el **inmediato siguiente** sea un valor determinado. La función debe devolver el índice del elemento o -1 si no existe.

```
function ElSiguiente(x: integer; A: arreglo): integer;
(* devuelve i si A[i+1]=x, -1 si no existe *)
var i: integer;
begin
  i:=1;
  (* circuito corto *)
  while (i < M) and (A[i+1] <> x) do
     i:= i+1;
  if i < M then ElSiguiente := i
  else ElSiguiente := -1
end;
```

Búsqueda en un arreglo (8)

Verificar si todos los elementos son pares.

Este problema también constituye un ejemplo de búsqueda, ya que es equivalente a buscar un elemento que **no** sea par.

```
function TodosPares(A: arreglo): boolean;
(* devuelve true si todos son pares, false si no *)
var i: integer;
begin
  i:=1;
  (* circuito corto *)
  while (i <= M) and (A[i] mod 2 = 0) do
      i:= i+1;
  TodosPares := i >  M
end;
```

Búsqueda en un arreglo (9)

Existen diversos problemas que pueden resolverse mediante **búsquedas**. Por ejemplo, los siguientes problemas sobre un arreglo de enteros:

- Saber si el arreglo contiene algún valor que sea impar
- Saber si todos los valores del arreglo son positivos
- Saber si el arreglo está ordenado de menor a mayor
- ¿Otros?

Se considera un **error grave** realizar una **recorrida completa** para resolver un problema de búsqueda (por más que "ande")

Arreglos Multidimensionales

Pueden considerarse como **arreglos de arreglos** o como arreglos con un **índice múltiple**.

A los arreglos bidimensionales también se los llama **matrices**.

```
type  nombre = array [1..20, 1..15] of char;
      datos = array ['0'..'9', 1..10] of integer;
      tabla = array [-10..10, 1..15] of char;
```

Para acceder a una celda de un arreglo a bidimensional se utiliza alguna de las siguientes formas:

- `a[i][j]`
- `a[i,j]`

La segunda es la más utilizada.

Mas sobre arreglos multidimensionales: **Sección 9.3 de Konvalina**