PROGRAMACIÓN II (SISTEMAS) - CURSO 2002/03

EIFFEL ESTRUCTURADO (II)

6. TIPOS DE DATOS ESTRUCTURADOS

Los tipos de datos estructurados se representan mediante **clases** en Eiffel, lo que implica que existen muchas diferencias en su tratamiento respecto a Pascal:

- En Eiffel no existen apartados de *definición de tipos*, cada nuevo tipo estructurado que se desee crear se representa mediante una *definición de clase*. No vamos a necesitar crear nuevos tipos de datos, por lo que la única definición de clase que se va a realizar es la que representa al programa.
- Las variables de tipo estructurado **no** almacenan *valores*[1] de ese tipo, sino **referencias** a valores. Los valores deben ser **creados** utilizando la siguiente sintaxis:

  creación_objeto ::= **!!** *variable* **.** *método_creación*[**(** *parámetros* **)**]

- La construcción anterior crea un valor (objeto) del tipo de datos (clase) asociado a la variable, llama al método de creación elegido para que inicialice el objeto, y asigna una referencia a ese objeto a la variable.
- Cualquier variable de tipo estructurado (clase) puede almacenar el valor especial Void, que tiene el significado de **referencia nula** (la variable no almacena una referencia a un objeto válido). Este es el valor inicial de la variable antes de ser asignada.
- En el caso de arrays y strings existe una manera abreviada de crear objetos con valores determinados: los literales de tipo array y string. (ver ejemplo 7.3)

  literal_tipo_string ::= **"***texto***"**

  literal_tipo_array ::= **<<** *expresión* { **,** *expresión* } **>>**

- Para acceder o modificar el contenido de un objeto (acceso a un elemento de un array o a un carácter de un string, por ejemplo) se llama al método apropiado de la clase. No existe en

[1] En programación orientada a objetos, los valores cuyo tipo de datos es una clase se denominan **objetos**. Un objeto es una **instancia** (una materialización) de una clase. Al proceso de crear un objeto se le denomina instanciación de una clase.

Eiffel una construcción sintáctica equivalente a los corchetes **[ ]** de Pascal para acceder a los elementos de arrays y strings.

- En una asignación *a* := *b* donde *a* y *b* son variables de tipo estructurado (strings, por ejemplo), lo que se asignan son **referencias**, no valores (como sucede en Pascal). En este ejemplo, tras la asignación *a* y *b* almacenan la misma referencia a una única cadena de caracteres. En Pascal, al contrario, cada variable almacena directamente el valor, por lo que en el ejemplo anterior existirían dos cadenas de caracteres, ambas con el mismo contenido.
- Lo mismo sucede en las comparaciones con los operadores de **igualdad** (=) o **desigualdad** (/=) entre variables de tipo estructurado: Lo que se comparan son las *referencias*, no el *contenido* de los objetos. Al comparar dos variables el resultado será cierto si ambas apuntan al mismo objeto, y falso si apuntan a objetos distintos (aunque el contenido de ambos sea el mismo). Para comparar el contenido de objetos se deben utilizar métodos de la clase adecuada (ver ejemplo 7.1)
- Las funciones en Eiffel pueden devolver no sólo tipos simples, como en Pascal, sino cualquier tipo de datos. Esto significa que también pueden devolver tipos estructurados, es decir **referencias** a objetos.
- Por lo tanto, una tercera forma de crear un objeto es llamar a un método predefinido de tipo función que cree un objeto y devuelva una referencia a él, como los métodos substring de la clase STRING y subarray de la clase ARRAY (ver ejemplo 7.2).

  Cuidado! Un error muy común es suponer que el método last_string crea un nuevo objeto de clase STRING cada vez que se llama: En realidad reutiliza una única cadena, por lo que es responsabilidad nuestra crear nuevas cadenas para almacenar los datos (ver ejemplo 7.3)
- Cuando se pasa como parámetro a un subprograma un valor de tipo estructurado, en realidad se está pasando por valor una **referencia** a un objeto. Cualquier cambio que se produzca en el objeto dentro del subprograma se mantendrá una vez terminada la llamada (por lo tanto, a efectos prácticos, es como si se hubiera pasado *por variable*).
- Si los elementos de un array son de tipo estructurado (por ejemplo, un array de strings o un array cuyos componentes son a su vez arrays), debe quedar claro que estos elementos son **referencias** a objetos, por lo que el crear el array **no crea** objetos asociados a los elementos: El array almacena referencias, que serán nulas (valor Void) hasta que se creen (uno a uno) los elementos del array (ver ejemplo 7.3)
- En Eiffel no es necesario destruir los objetos cuando ya no se necesitan. El manejador de memoria detecta automáticamente cuando un objeto no es referenciado por ninguna variable (y por lo tanto es inalcanzable) y libera la memoria que ocupa. Esta estrategia se denomina *recolección automática de basura* (garbage collector).

7. CADENAS DE CARACTERES (STRING)

Las cadenas de caracteres se representan mediante la clase STRING. Esta clase representa una secuencia de *n* caracteres indexados de 1 a *n*. Existen multitud de métodos definidos en la clase, en la tabla siguiente se muestran aquellos que pueden ser útiles para la realización de la práctica. (Como es habitual, se muestran ejemplos de uso. Las variables que comienzan por *cad* son de tipo STRING, las que comienzan por *c* de tipo CHARACTER y las que comienzan por *i* de tipo INTEGER. *n* es de tipo INTEGER y representa la longitud de la cadena)

Creación	!!cad.make_empty	Crea una cadena de cadena de caracteres vacía.
	!!cad.make_filled(c,n)	Crea una cadena de caracteres que consiste en *n* repeticiones del carácter *c*.
	cad := "texto"	Crea un objeto cadena de caracteres con la secuencia *texto* y asigna su referencia a *cad*
	cad := cad1 + cad2	Crea un objeto cadena de caracteres con la secuencia de *cad1* seguida de la secuencia de *cad2* y asigna su referencia a *cad*.
Acceso y modificación de caracteres	n := cad.count	Longitud (nº de caracteres) de la cadena
	c := cad.item(i)	Carácter en la posición *i*-ésima (se comienza a contar por 1)
	cad1.copy(cad2)	Hace que la secuencia de caracteres de *cad1* sea idéntica a la de *cad2*.
	cad.put(c,i)	Reemplaza el carácter en posición *i* por *c*
	cad.add_last(c)	Añade el carácter *c* al final
	cad.insert_character(c,i)	Inserta el carácter *c* de manera que pase a estar en la posición *i*. $1 \le i \le n+1$
	cad.remove(i)	Borra el carácter en posición *i* de la cadena, desplazando a la izquierda los caracteres siguientes.
Comparación	**if** cad1.is_equal(cad2) **then**	Cierto si *cad1* y *cad2* **contienen** la misma secuencia de caracteres (distingue casos)
	if cad1.same_as(cad2) **then**	Igual que la anterior pero sin distinguir entre mayúsculas y minúsculas
	i := cad1.compare(cad2)	Devuelve 0 si son iguales, -1 si *cad1* menor que *cad2*, +1 si *cad1* mayor que *cad2*. El orden lexicográfico es el del idioma inglés.
Subcadenas	cad2 := cad1.substring(i1,i2)	**Crea** una nueva cadena conteniendo la secuencia de *cad1* entre las posiciones *i1* e *i2*
	cad1.insert_string(cad2,i)	Modifica *cad1* insertando entre las posiciones *i* e *i*+1 la cadena *cad2*. $1 \le i \le n+1$ (*n* es la longitud de *cad1*).
	cad1.replace_substring(cad2,i1,i2)	Reemplaza los caracteres en posiciónes *i1..i2* (inclusive) de *cad1* por *cad2*. $1 \le s1 < s2 \le n$.
	cad.remove_between(i1,i2)	Elimina los caracteres entre *i1* e *i2* (inclusive), desplazando a la izquierda los caracteres siguientes.

Los métodos de acceso (item), modificación de caracteres (put) y longitud de la cadena (count) tienen el mismo nombre que en la clase ARRAY, para resaltar que estamos ante un caso especial de array.

Existen operadores especiales para tratamiento de cadenas, como el operador de concatenación (+), que crea un nuevo objeto de clase STRING conteniendo la secuencia resultante de concatenar dos objetos de clase STRING. Aunque los operadores <, <=, > y >= están redefinidos para realizar la comparación entre el **contenido** de dos objetos de clase STRING (no así los operadores = y /=), se aconseja en su lugar utilizar el método compare.

```
prueba is
local
  c1,c2,c3 : STRING;
do
  c1 := "Hola";
  c2 := C1;
  c3 := "Hola";

  if c1 = c2 then
    io.put_string("C1 y C2 son iguales.%N");
  else
    io.put_string("C1 y C2 son distintas.%N");
  end
  if c1 = c3 then
    io.put_string("C1 y C3 son iguales.%N");
  else
    io.put_string("C1 y C3 NO APUNTAN AL MISMO OBJETO.%N");
  end
  if c1.compare(c3) = 0 then
    io.put_string("Los objetos apuntados por C1 y C3 son iguales%N");
  else
    io.put_string("Los objetos apuntados por C1 y C3 son distintos%N")
  end

  -- Cambia el segundo caracter por 'a'
  c2.put('a',2);
  if c1 = c2 then
    io.put_string("C1 y C2 siguen siendo iguales a "+C1+"%N");
  else
    io.put_string("C1 y C2 ahora son distintas.");
  end

end -- prueba
```

Ejemplo 7.1

El resultado de ejecutar el programa anterior debe ser:

- C1 y C2 son iguales.
- C1 y C3 NO APUNTAN AL MISMO OBJETO.
- Los objetos apuntados por C1 y C3 son iguales
- C1 y C2 siguen siendo iguales a Hala

8. ARRAYS

Los arrays se definen en Eiffel utilizando la palabra reservada ARRAY seguida, entre corchetes, del tipo de datos de los elementos que se van a almacenar.

tipo_array ::= **ARRAY** **[** *tipo_elementos* **]**

Las diferencias esenciales respecto a Pascal son las siguientes:

- El tamaño de los arrays se define cuando se crean (es decir, en tiempo de ejecución), no en la definición del tipo de datos.
- Los índices sólo pueden ser enteros.
- Los arrays son estructuras dinámicas: Pueden aumentar o disminuir de tamaño en tiempo de ejecución.

A continuación se muestra un ejemplo de uso de los métodos principales de la clase ARRAY. En lo que sigue *vec* representa un ARRAY y *x* una variable o valor del mismo tipo que el de los elementos de *vec*.

Creación	!!vec.make(i1,i2)	**Crea** un array indexado desde *i1* hasta *i2* (inclusive). Todos los elementos se inicializan al valor por defecto (0 si son numéricos, %U si son caracteres, Void si son estructurados)
	!!vec.with_capacity(n,i1)	**Crea** un array vacío (con 0 elementos) donde el índice inicial es *i1*. Al array se le pueden añadir elementos, *n* indica una *estimación* del máximo de elementos que va a contener.
	vec := << 4, 5, 6 >>;	Crea un array de enteros indexado de 1 a 3 y conteniendo los valores 4,5,6.
Acceso y modificación de elementos	i1 := vec.lower	Índice inferior del array
	i2 := vec.upper	Índice superior del array
	n := vec.count	Número de elementos del array (es igual a upper-lower+1). Puede ser cero.
	x := vec.item(i)	Acceso al elemento con índice *i*
	vec.put(x,i)	Se reemplaza el elemento con índice *i* por el valor *x*
	vec.swap(i1,i2)	Se intercambian los elementos en posiciones *i1* e *i2*
	vec.set_all_with(x)	Se asigna el valor *x* a todas las posiciones
Inserción y borrado	vec.add_first(x)	Inserta el valor *x* al principio del vector, desplazando el resto a la derecha.
	vec.add_last(x)	Inserta el valor *x* al final del vector.
	vec.add(x,i)	Inserta *x* entre los índices *i* e *i*+1, desplazando a la derecha los elementos siguientes.
	vec.remove_first	Elimina el primer elemento, desplazando el resto a la izquierda
	vec.remove_last	Elimina el último elemento
	vec.remove(i)	Elimina el elemento con índice *i*, desplazando a la izquierda los elementos siguientes.

Copiado	vec2 := vec1.subarray(i1,i2)	**Crea** un nuevo array conteniendo los elementos entre los índices *i1* e *i2* (ambos inclusive). El nuevo vector está indexado desde *i1* hasta *i2*
	vec1.copy(vec2)	El vector *vec1* pasa a ser una **copia** del vector *vec2* (con los mismos índices que *vec2*). **Atención**: *vec1* y *vec2* tienen referencias a objetos *distintos* (aunque con el mismo contenido)

```
ejemplo_creacion is
local
  c1,c2,c3 : STRING;
  v1,v2,v3 : ARRAY[STRING]
do
  -- creacion directa
  !!c1.make_empty;
  c1.add_last('H'); c1.add_last('o'); c1.add_last('l'); c1.add_last('a');
  !!v1.make(-3,-1);
  v1.put(c1,-3); v1.put("Hola",-2); v1.put(c1,-1);
  -- creacion mediante literales
  c2 := "Segunda cadena";
  v2 := <<"Primero","Segundo","Tercero">>
  -- creacion usando metodos que crean objetos
  c3 := c2.substring(3,10);
  v3 := v1.subarray(-3,-2);
  -- modificacion de objetos
  escribe_vector("v1",v1);
  escribe_vector("v3",v3);
  c1.put('a',2);
  escribe_vector("v1",v1);
  escribe_vector("v3",v3);
  v1.item(-3).put('P',1);
  escribe_vector("v1",v1);
  escribe_vector("v3",v3);
  io.put_string("c1 = "+c1+"%N");
  escribe_vector("v2",v2);
  v2.copy(v3);
  escribe_vector("v2",v2);
end -- ejemplo_creacion

escribe_vector(msg: STRING; v: ARRAY[STRING]) is
local i : INTEGER;
do
  from
    i := v.lower;
  until i > v.upper loop
    io.put_string(msg+"["+i.to_string+"] = "+v.item(i)+" ");
    i := i+1
  end
  io.put_new_line
end -- escribe_vector
```

Ejemplo 7.2

9. FICHEROS DE TEXTO

El concepto de fichero se representa por varias clases predefinidas en Eiffel. En la práctica de la asignatura sólo va a ser necesario trabajar con ficheros de texto, que se representan mediante dos clases: TEXT_FILE_READ representa un fichero de texto abierto para **lectura** y TEXT_FILE_WRITE un fichero de texto abierto para **creación** y **escritura**.

El tratamiento de ficheros es muy parecido a como se realiza en Pascal: Se crea el fichero (make), se asigna el fichero físico al fichero lógico (connect_to), se comprueba si existe el fichero físico (is_connected), se leen o escriben datos, en el caso de lectura se comprueba que no se halla alcanzado el final del fichero (end_of_input), y cuando se termina el tratamiento se cierra el fichero (disconnect).

En la tabla siguiente se muestran los métodos principales. Los métodos de lectura y escritura de datos son idénticos a los de lectura por teclado y escritura por pantalla. La variable *fich_ent* representa un fichero de tipo TEXT_FILE_READ, la variable *fich_sal* un fichero de tipo TEXT_FILE_WRITE y la variable *fich* un fichero de cualquier tipo:

Creación, apertura, cierre	!!fich.make	Crea un fichero no asignado a ningún fichero físico
	fich.connect_to(cad)	Asigna el fichero al fichero físico descrito por la cadena *cad*. Este paso no produce errores.
	fich.disconnect	Cierra el fichero
	if fich.is_connected **then**	Cierto si el fichero ha sido conectado a un fichero físico existente.
	if fich_ent.end_of_input **then**	Cierto si se ha alcanzado el final del fichero
Lectura	fich_ent.read_character; c := fich_ent.last_character	Lectura de datos
	fich_ent.read_integer; n := fich_ent.last_integer	
	fich_ent.read_double; d := fich_ent.last_double	
	!!cad.make; … fich_ent.read_line; cad.copy(fich_ent.last_string)	En el caso de lectura de cadenas de texto, se debe siempre **copiar** a una cadena existente la cadena leída, ya que **last_string** no crea cadenas nuevas.
Escritura	fich_sal.put_character(c)	Escritura de datos
	fich_sal.put_integer(i)	
	fich_sal.put_double(d)	
	fich_sal.put_string(cad)	

En el ejemplo siguiente se ha creado un programa que lee un fichero de texto en un vector, y escribe un fichero con el nombre resultado.txt igual al primero excepto en que se han eliminado las lineas en blanco:

```
indexing "Copia un fichero de texto eliminando lineas vacías"
class COPIADOR
creation inicial
feature

  inicial is
  local
    nom_fich: STRING;
    fich_ent: TEXT_FILE_READ;
    fich_sal: TEXT_FILE_WRITE;
    vector: ARRAY[STRING];
  do
    !!nom_fich.make_empty;
    !!fich_ent.make;
    !!fich_sal.make;
    !!vector.with_capacity(10,1); -- capacidad inicial 10 lineas.
    io.put_string("Nombre del fichero: ");
    io.read_line;
    nom_fich.copy(io.last_string);
    fich_ent.connect_to(nom_fich);
    if fich_ent.is_connected then
      lee_fichero(fich_ent,vector);
      fich_ent.disconnect;
      fich_sal.connect_to("resultado.txt");
      escribe_fichero(fich_sal,vector);
      fich_sal.disconnect;
    else
      io.put_string("Error: fichero no existe o no puede abrirse!");
    end
  end -- inicial

  lee_fichero(fich: TEXT_FILE_READ; vec: ARRAY[STRING]) is
  local linea: STRING;
  do
    !!linea.make_empty;
    from
    until fich.end_of_input loop
      fich.read_line;
      linea.copy(fich.last_string);
      vec.add_last(linea);
    end
  end -- lee_fichero

  escribe_fichero(fich: TEXT_FILE_WRITE; vec: ARRAY[STRING]) is
  local i: INTEGER;
  do
    from
      i := vec.lower;
    until i > vec.upper loop
      if vec.item(i).count > 0 then
        fich.put_string(vec.item(i)+"%N");
      end
      i := i+1
    end
  end -- lee_fichero
end -- COPIADOR
```

Ejemplo 7.3