Bloque 1. Conceptos y técnicas básicas en programación

UNIVERSIDAD DE CANTABRIA

1. Introducción
2. Datos y expresiones. Especificación de algoritmos
3. Estructuras algorítmicas básicas
4. Iteración y recursión
5. Iteración y recursión sobre secuencias
6. Iteración y recursión sobre tablas

Notas:

UNIVERSIDAD DE CANTABRIA

1. Introducción

2. Datos y expresiones. Especificación de algoritmos

- Tipos primitivos. Variables y constantes. Operadores y expresiones. Noción de especificación. Noción de predicado. Reglas de consecuencia de una especificación. Encapsulamiento: Concepto de clase y objeto. Métodos y paso de parámetros. Especificación de clases y objetos

3. Estructuras algorítmicas básicas

4. Iteración y recursión

5. Iteración y recursión sobre secuencias

6. Iteración y recursión sobre tablas

Tipos primitivos

UNIVERSIDAD DE CANTABRIA

Entero

- conjunto de los números enteros
- en la práctica limitado a un rango
- dispone de aritmética exacta en ese rango

Real

- conjunto de los números reales
- en la práctica limitado a un subconjunto de números racionales
- aritmética de punto flotante (no exacta)

Tipos primitivos (cont.)

UNIVERSIDAD DE CANTABRIA

Carácter

- un único carácter alfanumérico, incluyendo
  - letras y cifras de 0-9
  - signos de puntuación
  - espacios en blanco y tabuladores
  - otros caracteres sin representación gráfica (ej., salto de línea)

Booleano

- valor lógico (verdad o falso)

***Texto*:**

- secuencia de caracteres

Variables y constantes

Las variables definen una zona de memoria donde cabe un dato de un tipo determinado, cuyo valor podemos cambiar

Se identifican mediante un nombre

Existen diversos tipos

- *atributos* (variables de clase)
- *variables locales* (variables de método)
- *argumentos* o *parámetros* (datos que pasamos a métodos)

Las constantes son iguales, pero el valor no puede cambiar

Sintaxis para describir variables y constantes

La notación habitual en lenguajes de la familia del C/Java es:

```
tipo nombre;
tipo nombre=valorInicial;
tipo nombre1, nombre2, nombre3;
```

En otros lenguajes (Pascal/Ada) se define al revés:

```
nombre: tipo;
nombre: tipo:= valorInicial;
nombre1,nombre2,nombre3 : tipo;
```

Para la especificación pueden usarse indistintamente

Para definir constantes pondremos delante la palabra `const`

Ejemplos de definición de variables y constantes

```
entero i,j;
entero k=3;

real tempDeseada=22.0;
real tempAmbiente;

caracter c1='a';

booleano estaEncendido=verdad;

texto miNombre="Pedro González";
texto miDireccion;

const real tempMaxima=100.0;
```


Operadores y expresiones

Los cálculos y operaciones de transformación de datos se llaman *expresiones*

Una expresión permite obtener un resultado y se compone de

- uno o más *operandos*: datos que vamos a usar
  - *unarios*: operan con un solo dato
  - *binarios*: operan con dos datos
  - *ternarios*: operan con tres datos
- cero o más *operadores*: indican las operaciones que vamos a realizar

Reglas de precedencia

Los operandos tienen reglas de *precedencia*, para saber en qué orden se aplican

```
a+b/c-d
```

La precedencia puede modificarse mediante *paréntesis*

```
(a+b)/(c-d)
```


Operaciones con enteros

Operador	Ejemplo	Operación
`+`	`a+b`	suma
`-`	`a-b`	resta
`- (unario)`	`-a`	cambio de signo
`*`	`a*b`	multiplicación
`div` `/`	`a div b` `a/b`	división entera de a entre b
`mod` `%`	`a mod b` `a%b`	resto de la división entera de a entre b

Precedencia:

- los multiplicativos mayor predecencia que los aditivos

Operaciones con reales

Operador	Ejemplo	Operación
`+`	`a+b`	suma
`-`	`a-b`	resta
`- (unario)`	`-a`	cambio de signo
`a*b`	`a*b`	multiplicación
`a/b`	`a/b`	division real de a entre b
`a mod b` `a%b`	`a mod b` `a%b`	parte fraccionaria de la división de a entre b

Precedencia:

- los multiplicativos mayor predecencia que los aditivos

Ejemplos de expresiones

Enteras

```
entero i,j,k;

i
-25
(i*25+3) mod 2
(i-j)*k+(35 div i)
```

Reales

```
real x,y,z;

x*1.0e-6
x-z/y
-y
```

Conversiones de tipo

UNIVERSIDAD DE CANTABRIA

Generalmente intentaremos no mezclar datos de tipos distintos en las expresiones

Cuando sea imprescindible, usaremos el operador de cambio de tipo (cast)

```
(tipo) expresion
```

Ejemplos

- convertir la expresión real a entera

```
(entero) (5.0*x+y)
```

- Usar un número entero en una expresión real

```
5.0*(y+(real)i)
```


Operaciones relacionales

UNIVERSIDAD DE CANTABRIA

Permiten comparar dos valores del mismo tipo para obtener un resultado booleano

Operador	Ejemplo	Operación
==	a==b	igual a
!=	a!=b	distinto de
>	a>b	mayor que
>=	a>=b	mayor o igual a
<	a<b	menor que
<=	a<=b	menor o igual que

Tienen menos precedencia que los operadores aritméticos

Operaciones lógicas

Permiten operar con valores booleanos para obtener un resultado también booleano

Operador	Ejemplo	Operación
`&`	`a & b`	y (a veces se pone $\wedge$)
`|`	`a | b`	o (a veces se pone $\vee$)
`!`	`! a`	no (a veces se pone $\neg$)
`&&`	`a && b`	y entonces (`b` sólo se evalúa si es necesario)
`||`	`a || b`	o si no (`b` sólo se evalúa si es necesario)

Precedencia

`!, &, |`

Ejemplos de expresiones relacionales y lógicas

Expresiones simples

```
x>5.0
i==j
```

Comprobación: `x` en el intervalo `[-5,3)`

```
x>=-5 && x<3
```

Comprobación: `x` fuera del intervalo `(-2,2)`

```
x<=-2 || x>=2
```

Observar que los operadores relacionales no pueden concatenarse

```
-5<x<3   //¡incorrecto!
```

Noción de especificación

Para desarrollar un algoritmo seguiremos dos pasos

- ***especificación*: describe lo que hace el algoritmo, sin detallar cómo lo hace**
- ***diseño*: describe con detalles lo que hace el algoritmo, es decir, su secuencia de instrucciones**

Elementos de una especificación

- ***declaración* de variables**
- ***precondición*: condiciones sobre el estado inicial de las variables**
- ***nombre* del algoritmo**
- ***postcondición*: condiciones sobre el estado final de las variables**

Noción de predicado

Un *predicado* es una relación entre objetos con un resultado booleano

- **La precondición y postcondición son predicados**

Pueden describirse de modo *informal* o *formal*

Ejemplos de predicados

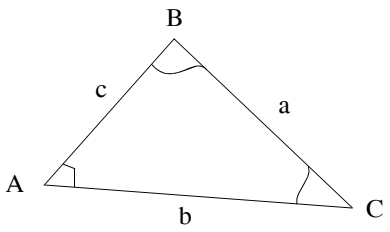
$x^2 + y^2 + z^2 = R^2 \wedge z = 0 \wedge x > 0 \wedge y > 0$
Un arco de 90º de radio R en el plano x-y

$z \geq x \wedge z \geq y \wedge z \in \{x, y\}$
z es el máximo de x e y z=max(x,y)

$\exists n \in N: m = k \cdot n$
m es múltiplo de k

Ejemplo de una especificación informal

Algoritmo que calcula la superficie de un triángulo, dados sus lados




```
var
   real a,b,c
   real s, area
fvar
{Pre: a,b,c contienen los lados del triángulo}
   Cálculo del área
{Post: s= semiperímetro del triángulo,
```

$$area = \sqrt{s(s-a)(s-b)(s-c)}$$ }

Ejemplo de una especificación formal

```
var
   real a,b,c
   real s, area
fvar
{Pre: a=A,b=B,c=C contienen los lados
       del triángulo}

   Cálculo del área

{Post: a=A, b=B, c=C, s=(A+B+C)/2,
```

$$area = \sqrt{s(s-A)(s-B)(s-C)}$$ }

Ejercicios de especificación

Especificar un algoritmo que no haga nada sobre dos variables enteras

Especificar un algoritmo que intercambie el valor de dos variables

Especificar un algoritmo que ordene tres variables enteras en orden creciente

Especificar un algoritmo que convierta una cantidad positiva de segundos en su correspondiente en horas, minutos y segundos

Especificar un algoritmo que indique si un entero dado es múltiplo de otro entero dado

Corrección de un algoritmo

Un algoritmo es *correcto* cuando satisface la especificación en todos los casos:

- comenzando con variables que satisfacen la *precondición*
- y después de *ejecutarse* el algoritmo
- el estado resultante satisface la *postcondición*

Expresaremos esta relación como $\{P\}\ S\ \{Q\}$, siendo P la precondición, S el algoritmo, y Q la postcondición

Pueden existir especificaciones para las que no exista ningún algoritmo que las satisfaga

- son especificaciones incorrectas

Reglas de consecuencia de una especificación

Existen reglas que nos ayudan a relacionar especificaciones entre sí

Primera regla de consecuencia

- Si $\{P\}\ S\ \{Q\}$ y $R \Rightarrow P$ entonces $\{R\}\ S\ \{Q\}$
  - si hay un predicado R que implica que se cumple P, entonces si sustituimos P por R, la especificación se sigue cumpliendo

Segunda regla de consecuencia

- Si $\{P\}\ S\ \{Q\}$ y $Q \Rightarrow R$ entonces $\{P\}\ S\ \{R\}$
  - si Q implica a otro predicado R, entonces si sustituimos Q por R, la especificación se sigue cumpliendo

Ejemplos de las reglas de consecuencia

Primera regla

entero x {x<=2} S {x<=10}	⇒	entero x {x<=-10} S {x<=10}

Segunda regla

entero x {x<=5} S {x<=12}	⇒	entero x {x<=5} S {x<=45}

Encapsulamiento: Concepto de clase y objeto

Cuando se usan técnicas de programación orientada a objetos

- **un *programa* se compone de módulos interrelacionados llamados *clases***

Una *clase* es un patrón de un determinado tipo de datos con operaciones definidas para manipular esos datos

- **cada dato que sigue ese patrón se llama *objeto* de la clase**

Objetos

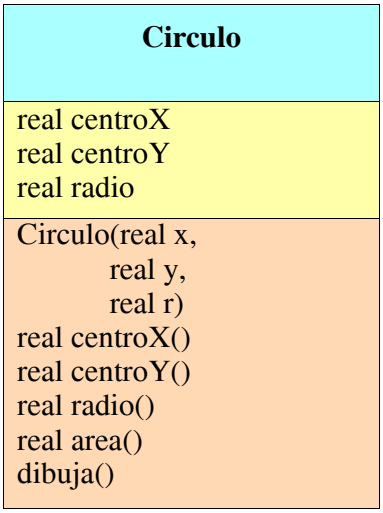
Un objeto se caracteriza por

- sus *atributos*: datos cuyo valor define el estado del objeto
  - habitualmente se ocultan (se hacen privados) para que nadie pueda cambiar su valor incontroladamente
- sus *métodos*: operaciones que se pueden efectuar
  - implementan un *algoritmo*
  - inicializan, consultan o modifican el estado del objeto
  - se invocan desde otro método
  - pueden necesitar *parámetros*: datos que se pasan desde el método que lo invoca
  - pueden *retornar* un valor al método desde el que se invocan

Diagramas de clases

Definen

- *nombre* de la clase
- *atributos*: nombres y tipos
- *métodos*: nombre, parámetros (con nombre y tipo de cada uno), y valor retornado (tipo)
  - pueden ser constructores, que inicializan el objeto



Declaración de atributos

Alternativamente al diagrama de clases, puede declararse los atributos igual que las variables, de la forma

```
atributos
    real centroX;
    real centroY;
    real radio;
fatributos
```


Declaración y especificación de métodos

Formal

```
método Circulo(real x, real y, real r)
    {Pre: x=X & y=Y & r=R>=0}
       constructor
    {Post: centroX=X & centroY=Y & radio=R}
fmétodo
```

Informal (se supone que x, y, r no cambian)

```
método Circulo(real x, real y, real r)
    {Pre: r>=0}
       constructor
    {Post: centroX=x & centroY=y & radio=r}
fmétodo
```

Declaración y especificación de métodos

Formal

```
método area() retorna a:real
  {Pre: centroX=X & centroY=Y & radio=R}
      cálculo del área
  {Post:centroX=X & centroY=Y & radio=R & a=π·R^2}
fmétodo
```

Informal (si no se dice nada, se supone que un dato no cambia)

```
método area() retorna real
   {Pre:}
      cálculo del área
   {Post: valor retornado=π·radio^2}
fmétodo
```


Uso de clases, objetos y métodos

Declaración de objetos. Se hace al declarar variables

```
Clase objeto =nuevo Constructor(parámetros)
```

Ejemplo

```
var
   Circulo c1=nuevo Circulo(10.0,12.0,100.0)
fvar
```

Asociación de parámetros

- cada valor concreto usado en la invocación se asocia, por su orden de aparición, a un argumento del método

```
x => 10.0
y => 12.0
r => 100.0
```

Uso de clases, objetos y métodos

Uso de los métodos de un objeto, en expresiones e instrucciones

```
objeto.metodo(parámetros)
```

Ejemplos

```
z=c1.area(); // asigna el valor retornado
             // por área a la variable z
c1.dibuja(); // invoca al método dibuja
```


UNIVERSIDAD DE CANTABRIA