Corrección de programas iterativos

Necesitamos herramientas para determinar si un programa es correcto sin tener que ejecutarlo varias veces.

Lo que haremos será pedir una serie de propiedades basadas en cómo va cambiando **el estado** de las variables implicadas.

Corrección de programas iterativos

Ejemplo: ordenacion de vectores por **selección**

(ver fichero `selecc.cc`)

Cómo sabemos que el vector queda realmente ordenado?

Corrección de programas iterativos

Consideramos bucles de la siguiente forma y sin **returns** internos

```
/* Pre */
while (B) {
    S
}
/* Post */
```

Principio de inducción

Enunciado básico: si *P(x)* es una propiedad sobre los números naturales, decimos que *P* es cierta si

- *P(0*) es cierta: **base de inducción**
- *P(x-1)* **=>** *P(x)* para cualquier *x>0*: **paso de inducción**; *P(x-1)* se llama Hipótesis de Inducción)

La corrección de un bucle se puede probar usando inducción.

Principio de inducción

Ejemplo: para todo natural n demostrar que

$$1 + 2 + 3 + \ldots + n = n(n+1)/2 \quad P(n)$$

Base: $P(0)$: $0 = 0$ cierto

Paso: si $n>0$: $P(n-1) \Rightarrow P(n)$

$$(1 + 2 + 3 + \ldots + n\text{-}1)+n \overset{HI}{=} n(n\text{-}1)/2+n =$$

$$(n^2\text{-}n+2n)/2 = (n^2+n)/2 = n(n+1)/2 \quad q.e.d.$$

Justificación de programas iterativos

Invariante: propiedad de las variables implicadas en el bucle, que representa el hecho de que el proceso iterativo va cumpliendo lo que queremos, es decir, nos va acercando a la postcondición

I.1) Se ha de cumplir al principio: inicialización

$$Pre \Rightarrow Inv$$

I.2) Ha de mantenerse tras cada iteración: cuerpo

$$\{Inv \text{ y } B\}\ S\ \{Inv\}$$

I.3) Al final, ha de implicar la postcondición:

$$Inv \text{ y no } B \Rightarrow Post$$

Justificación de programas iterativos

EL invariante también puede usarse para documentar la estrategia a seguir por el programa.

• Un recorrido hacia adelante de un vector:

Inv: v[0..i-1] (los elementos anteriores a v[i]) han sido visitados y tratados

• Lo mismo, marcha atrás:

Inv: v[i+1..v.size()-1] (los elementos posteriores a v[i]) han sido visitados y tratados

(Otro ejemplo: ver primer problema de `ex1105.pdf`)

Justificación de programas iterativos

Función de cota (t): magnitud que representa un máximo del número de vueltas que le quedan al bucle

C.1) Ha de ser un número natural al menos hasta el comienzo de la última vuelta:

Inv y B => t es un natural

C.2) Ha de decrecer estrictamente tras cada vuelta:

{I y B y t=T}S{t<T}

Justificación de programas iterativos

La corrección de un programa iterativo se establece así:

Teorema: el bucle acaba y cumple su especificación.

Lema (inductivo): el invariante se cumple antes y después de cada vuelta

Demostración lema:

- I.1 es la base de la inducción
- I.2 es el paso de la inducción

Justificación de programas iterativos

Demostración teorema :

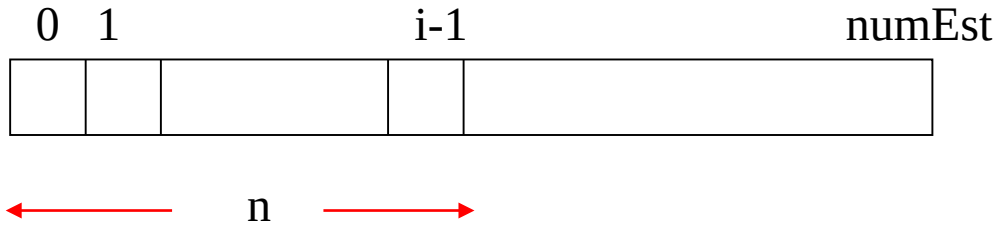
- el lema y I.3 implican que al acabar el bucle se obtiene la post
- C.1 y C.2 implican que el número de vueltas es acotado

Ejemplos de justificación con vectores de estudiantes

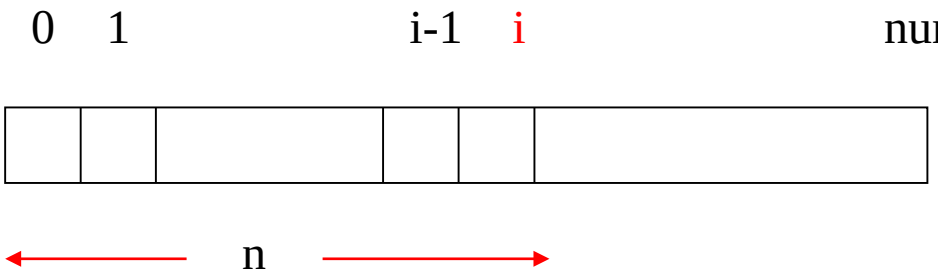
- % de estudiantes presentados `presentados.cc`
- redondear notas `arrodonir.cc`
- buscar un dni `busquedalin.cc`

Bucle de Presentados

ANTES del **if**: n es el número de estudiantes en vest[0..i-1] que tienen nota; i<numEst



DESPUÉS del **if**: n es el número de estudiantes en v[0..i] que tienen nota; i<numEst



Si incrementamos i, recuperamos el invariante

Justificación presentados

• **Inicio**: si i=0, el intervalo [0..-1] es vacío y no hay ningún estudiante => n=0

• **Salida del bucle**: si i=numEst, entonces Inv = Post

• **Cuerpo del bucle**: Para acercarnos al final incrementamos la i en 1. Al entrar en el bucle sabemos que

n es el número de elementos en vest[0..i-1] que tienen nota

Al salir queremos lo mismo hasta i. Sólo queda por saber si v[i] tiene nota y, en su caso, incrementar n.

• **Decrecimiento**: en cada vuelta i aumenta => la función numEst-i decrece

Justificación redondear

• **Inicio**: si i=0 => el intervalo [0..-1] es vacío y no hay ningún estudiante => no hay que hacer nada

• **Salida del bucle**: si i=numEst, entonces Inv = Post

• **Cuerpo del bucle**: Para acercarnos al final incrementamos la i en 1. Al entrar en el bucle sabemos que

los estudiantes en vest[0..i-1] tienen su nota redondeada

Al salir queremos lo mismo hasta i. Sólo queda por redondear la nota de vest[i], si la tiene.

• **Decrecimiento**: en cada vuelta i aumenta => la función numEst-i decrece

Justificación buscar versión 1

- **Inicio**: si *i=0*, el intervalo *[0..-1]* es vacío y no hay ningún estudiante que cumpla la propiedad, por tanto *b* ha de ser falso

- **Salida del bucle**:
  - si se sale por *i=numEst*, entonces Inv = Post
  - si no, se sale por *b=cierto;* entonces el elemento buscado está en *v[0..i]*, y por tanto también en el vector completo, luego *b* ya vale lo que tiene que valer: Inv =>Post

Justificación buscar v1 (cont.)

• **Cuerpo del bucle**: Supongamos que antes de entrar en el bucle se cumple Inv, además b=false:

por ser b = false

"b = hi ha un estudiant a vest[0..i-1] amb DNI = dni" ⇔

"no hi ha cap estudiant a vest[0..i-1] amb DNI = dni" ⇔ A

Al salir, como se ha incrementado i, queremos lo mismo hasta i

"b = hi ha un estudiant a vest[0..i] amb DNI = dni" ⇔ por A

"b = el DNI de vest[i] és dni" (ya que en los anteriores no lo hay)

Esta igualdad se consigue con la correspondiente asignación

Cota: en cada vuelta i aumenta => la función numEst-i decrece

Justificación buscar versión 2

- **Inicio**: si *i=0*, el intervalo *[0..-1]* es vacío y no hay ningún estudiante que cumpla la propiedad, por tanto b ha de ser falso

- **Salida del bucle**:
  - si *i=numEst*, entonces Inv = Post y *b=falso* es correcto
  - si no, significa que *b=cierto*; entonces el elemento buscado está en *vest[i]*, y por tanto también en el vector completo, luego *b* vale lo que tiene que valer: Inv =>Post

Justificación buscar v2 (cont.)

• **Cuerpo del bucle**: Antes de entrar en el bucle se cumple que *b=falso* e Inv, es decir

"no hay ningún elemento en *vest[0..i-1]* con DNI igual a *dni*"

Al salir, hemos de reducir la cota y los nuevos valores de *i* y *b* (llamémoslos *E* y *F)* han de cumplir

"no hay ningún elemento en *vest[0..E]* con DNI igual a *dni; F=>vest[E]* tiene DNI = *dni*"

El primer elemento por tratar es *vest[i]*. Si su DNI es igual a *dni* el objetivo se logra con *b* `= true`. Si no, hay que incrementar *i*

• **Cota**: en cada vuelta *i* aumenta => la función *numEst-i*-int*(b)* decrece

Ejercicios

Proponer un código iterativo y una justificación:

- nota máxima de un vector de estudiantes
- comprobar si un vector de enteros es capicúa

Ejemplos de justificación iterativa con bucles imbricados

- suma de matrices `suma_matriz.cc`
- producto de matrices `prod_matriz.cc`

Ejemplos de justificación iterativa con pilas, colas y listas

- altura de una pila `altura_pila.cc`
- buscar en una cola `buscar_cola.cc`
- sumar una cantidad a todos los elementos de una lista `sumar_k_lista.cc`

Justificación altura de una pila

• **Inicio**: no se ha tratado ningún elemento de *P* => *ret = 0*

• **Salida del bucle**: si *p* es vacía => se han tratado todos los elementos de *P* => Inv = Post

• **Cuerpo del bucle**: para acercarnos al final desapilamos *p*. Al acabar cada vuelta sabemos que

parte tratada de *P* = parte tratada de *P* antes de la vuelta + cima de *p* antes de la vuelta

Por Inv sabemos que

ret = núm de elems de *P* antes de la vuelta

Por tanto, el núm. de elems. de *P* tras la vuelta es ***ret+1***

• **Cota**: en cada vuelta, la parte **no visitada** de *P* es más pequeña

Justificación buscar en una cola

- **Inicio**: no se ha tratado ningún elemento de *c* => *ret=falso*
- **Salida del bucle**:
  - si *c* es vacía, por Inv hemos tratado toda *C* => el resultado es lo que valga *ret*
  - si *c* no es vacía => *ret = cierto* => por Inv, *i* está en *C* => el resultado es lo que valga *ret*

Justificación buscar en una cola (cont.)

• **Cuerpo del bucle**: Para acercarnos al final, avanzamos en *c* (eliminamos su primer elemento). Al acabar cada vuelta del bucle sabemos que

parte visitada de *C* = parte visitada de *C* antes de la vuelta + primero de *c* antes de la vuelta

Como hemos entrado con *ret=falso*, por el invariante sabemos que

i está en la parte visitada de *C* ⇔ ***i = c.front()***

y ese es el valor que ponemos en *ret*

• **Cota**: en cada vuelta, la parte **no** visitada de *c* decrece

Justificación sumar k a los elementos de una lista

- **Inicio**: si ***it = l.begin(),*** no hay que sumar *k* a ningún elemento
- **Salida del bucle**: si ***it = l.end(),*** ya está sumado *k* a todos los elementos => Inv = Post
- **Cuerpo del bucle**: para acercarnos al final incrementamos *it*.

Por Inv sabemos que ya se ha sumado *k* a todos los elementos anteriores a *it*, por tanto antes de incrementar *it* solo falta hacer lo mismo con la posición marcada por *it*.

Cota: en cada vuelta, el número de elementos entre ***it*** y ***l.end()*** es más pequeño

Ejemplos de justificación iterativa con pilas, colas y listas

Ejercicio

- comprobar si una lista de enteros es capicúa
  - usando size()
  - sin usar size

Diseño inductivo de programas iterativos

Usar la justificación como guía del diseño.
Dadas una pre y una post, proponer:

- un invariante basado en ellas
- una función de cota que asegure la terminación
- una inicialización que junto a la Pre asegure el Inv
- una condición del bucle que, si no se cumple, haga que el Inv implique la Post
- un cuerpo del bucle que mantenga el Inv

Diseño inductivo de programas iterativos

Ejemplos:

- contar solitarios en un vector (volvemos al primer problema de `ex1105.pdf`
- longitud del prefijo más corto de suma máxima de una lista de double