TEMA 4: DISEÑO DE ALGORITMOS RECURSIVOS

1. Principio de inducción
2. Diseño recursivo
3. Coste temporal de un algoritmo recursivo
4. Coste espacial de un algoritmo recursivo
5. Inmersión de programas
6. Inmersión de especificaciones

5. Inmersión de programas

- En ciertas ocasiones es interesante transformar un programa en otro programa que calcule el mismo resultado pero que posea diferentes propiedades.
- Una de las técnicas más usadas para conseguir esto se denomina **inmersión de programas**, que consiste en crear una nueva función más general que incluya como caso particular a la original (la original está *inmersa* en la actual).
- La función inmersora tendrá parámetros extra que podrán ser tanto de entrada como de salida (resultados).

5. Inmersión de programas

- Tipo de inmersión atendiendo a los parámetros extra:
  - **inmersión de parámetros:** Cuando la inmersión añada parámetros de entrada.
  - **inmersión de resultados:** Cuando la inmersión añada parámetros de salida.
- Tipo de inmersión atendiendo al objetivo de la inmersión:
  - **de eficiencia:** Para mejorar el coste temporal del algoritmo.
  - **obtención de recursividad final:** Para que la función sea recursiva final.

A los parámetros nuevos se les denomina parámetros de inmersión.

5. Inmersión de programas

- Ejemplo inmersión función factorial:
  - Función original:

```
int fact(int n)
{
  int f;
  if(n == 0)
    f = 1;
  else
    f = fact(n - 1) * n;          // f = fact(n-1)
  return f;                       // f = f * n
}
```

5. Inmersión de programas

- Función inmersora:

```
int i_fact(int n, int w)
{
  int f;
  if(n == 0)
    f = w;
  else
    f = i_fact(n - 1, n * w);
  return f;
}


int fact(int n)
{
  return i_fact(n,1);
}
```

5.1. Inmersión de resultados

- Dada la siguiente función recursiva:

```
func f(x: T1) dev r: T2
{Pre: Q(x)}
Var v:T2;
 [ d(x) -> r := h(x)
 ¬d(x) -> v := f(s(x));
  {Q(x) ∧ ¬d(x) ∧ R(s(x),v)}
   r := c(x, v)
]
{Post: R(x,r)}
```

Suponemos que existe una expresión $\phi(x)$ que nos interesa devolver como resultado para que otras llamadas recursivas la utilicen en sus cálculos. La devolveremos utilizando un nuevo resultado que llamaremos *w*.

5.1. Inmersión de resultados

- Función con inmersión de resultados:

```
func fr(x: T1) dev r: T2, w:T3
{Pre: Q(x)}
Var v:T2; u:T3;
[ d(x) ->  <r,w> := h'(x)
 ¬d(x) ->  <v,u> := fr(s(x));
   {Q(x) ∧ ¬d(x) ∧ R(s(x),v) ∧ u = ϕ(s(x))}
   <r,w> := c'(x, v, u)
]
{Post: R(x,r) ∧ w = ϕ(x)}
```

Se utiliza sobre todo para inmersiones de eficiencia.

5.1. Inmersión de resultados

- Ejemplo de inmersión de eficiencia mediante una inmersión de resultados: Función de Fibonacci.

```
1) fib(0) ≡ 0
2) fib(1) ≡ 1
3) fib(n+2) ≡ fib(n+1) + fib(n)

func fibo(n: Natural) dev f:Natural
{Pre: cierto; Dec: n}
[n = 0 -> f := 0
 n = 1 -> f := 1
 n > 1 -> f1 := fibo(n-1)
          f2 := fibo(n-2)
          f := f1 + f2
]
{Post: f = fib(n)}
```


5.2. Inmersión de parámetros

- Dada la siguiente función recursiva:

```
func f(x: T1) dev r: T2
{Pre: Q(x)}
Var v:T2;
 [ d(x) -> r := h(x)
  ¬d(x) -> v := f(s(x));
   {Q(x) ∧ ¬d(x) ∧ R(s(x),v)}
    r := c(x, v)
]
{Post: R(x,r)}
```

Suponemos que en nuestra función calculamos expresiones $\phi(x)$ que pueden ser útiles para las siguientes llamadas recursivas y por tanto se lo pasaremos como parámetro de entrada. La función inmersora tendrá un nuevo parámetro de entrada $w = \phi(x)$.

5.2. Inmersión de parámetros

- Función con inmersión de parámetros:

```
func fp(x: T1; w: T3) dev r: T2
{Pre: Q(x) ∧ w = ϕ(x)}
Var v: T2;
[ d(x) ->  r := h'(x,w)
 ¬d(x) ->  v := fp(s(x), ϕ(s(x)) );
           {Q(x) ∧ ¬d(x) ∧ R(s(x),v)}
            r := c'(x, v, w)
]
{Post: R(x,r)}
```


5.2. Inmersión de parámetros

- Ejemplo de inmersión de eficiencia mediante inmersión de parámetros: Evaluación de un polinomio.

```
func eval(a: Vector; i: Natural; x: Real) dev v: Real
{Pre: i ≤ N; Dec: N-i}
[i = N -> v := a[i]*x^i
 i < N -> v := eval(a, i+1, x);
           {v = Σα: i+1 ≤ α ≤ N:a[α] * x^α}
           v := a[i]*x^i + v
]
{Post: v = Σα: i ≤ α ≤ N:a[α] * x^α}
```

eval(a, i, x) calcula x^i

eval(a, i+1, x) calcula x^{i+1}

Podemos utilizar x^i para calcular x^{i+1}. $\phi(a, i, x) = x^i$.

5.2. Inmersión de parámetros

- Función inmersora:

```
func eval2(a:Vector; i:Natural; x:Real; w:Real) dev v:Real
{Pre: i ≤ N ∧ w = x^i; Dec: N-i}
[i = N -> v := a[i] * w
 i < N -> v := eval(a, i+1, x, x*w);
           {v = Σα: i+1 ≤ α ≤ N:a[α] * x^α}
           v := a[i]*w + v
]
{Post: v = Σα: i ≤ α ≤ N:a[α] * x^α}
```

Función auxiliar:

```
func eval'(a:Vector; x:Real) dev v:Real
{Pre: cierto}
   v := eval2(a,0,x,1)
{Post: v = Σα: 0 ≤ α ≤ N:a[α] * x^α}
```

5.2. Inmersión y recursividad final.

- Vamos a realizar una **inmersión de parámetros** para transformar un programa **recursivo lineal no final** en **recursivo final**.
- El método que vamos a utilizar se denomina **desplegado y plegado**.
- En recursividad normal, el cálculo del resultado se realiza al volver de la recursión.
- Mediante la inmersión de parámetros, el cálculo del resultado lo realizaremos al ir en la recursión.

5.2. Inmersión y recursividad final.

- Ejemplo factorial:

```
int fact(int n)
{
  if(n == 0)
    f = 1;
  else
    f = fact(n - 1) * n;          // f = fact(n-1)
  return f;                       // f = f * n
}

int i_fact(int n, int w)
{
  if(n == 0)
    f = w;
  else
    f = i_fact(n - 1, n * w);
  return f;
}
```

5.2. Inmersión y recursividad final.

- Para buscar la función inmersora nos basamos en la expresión de la función para el caso recursivo:

  v := f(s(x));
  r := c(v, x); } --> f(x) = c(f(s(x)), x)

- La función inmersora (g) tendrá una forma similar a ésta, pero cambiando en la expresión c todo lo que no es la función f por un parámetro (o varios) de inmersión.

  g(y,w) = c'(f(y), w)

  donde c' será bastante similar a c, y en muchas ocasiones igual.

5.2. Inmersión y recursividad final.

- Ejemplo 1: Factorial.
- Ejemplo 2: Función elevar (en C++).

```
int elev(int a, int b)
{   assert(b >= 0);
    int e;

    if(b == 0)
      e = 1;
    else
      e = elev(a, b - 1) * a;
    return e;
}
```

5.2. Inmersión y recursividad final.

- Ejemplo 3: Suma de una pila de enteros.

```
1) suma(p_nula) ≡ 0
2) suma(apilar(x,p)) ≡ x + suma(p)


func sum(p:pila) dev s: Entero
{Pre: cierto; Dec: altura(p) }
[nula(p) -> s := 0
¬nula(p) -> s := sum(desapilar(p));
            s := s + cima(p)
]
{Post: s = suma(p)}
```

5.3. Obtención de postcondición constante.

- Una función recursiva final presenta postcondición constante si la postcondición no varía de una llamada recursiva a otra, es decir, si la postcondición sólo depende de parámetros que no varían.
  - Ejemplo función factorial recursiva final:

```
fact(n, w:Nat) dev f:Nat
{Pre: cierto}
{Post: f = n! * w}

fact(4,1)
{Post: f=4!*1}
```



```
fact(3,4)
{Post: f=3!*4}
```

La postcondición ha variado !!

5.3. Obtención de postcondición constante.

- Realizamos una inmersión de parámetros para duplicar los parámetros que varían. Así conservamos los valores originales de los parámetros.
  - Función factorial recursiva final con postcondición constante:

```
fact(n, w, N:Nat) dev f:Nat
{Pre: ??}
{Post: f = N! }

fact(4,1,4)
{Post: f=4!}
```

```
fact(3,4,4)
{Post: f=4!}
```


5.3. Obtención de postcondición constante.

- La obtención de postcondición constante sirve para transformar la función recursiva en iterativa. El proceso completo para pasar una función de recursiva a iterativa es:

Recursiva lineal

Recursiva final

Post. constante

Iterativa

5.3. Obtención de postcondición constante.

- Los pasos a realizar para obtener esta inmersión son los siguientes:
  - Duplicar los parámetros que varían.
  - Postcondición nueva será igual que la anterior pero referida a los parámetros que no varían.
  - Reforzar la precondición expresando que el resultado antiguo es igual al nuevo.
- Cuando el programa viene de una inmersión de parámetros, sabemos cuales son los valores iniciales, por lo que no hace falta duplicar los parámetros de inmersión.

5.3. Obtención de postcondición constante.

- Caso general:

```
func g(x: T1) dev r: T2
{Pre: Q(x)}

    [ d(x)→ r:= e(x)
     ¬d(x)→ r:= g(s(x))
    ]
{Post: R(x,r)}
```

Postcondición constante:

```
func g'(x,X: T1) dev r: T2
{Pre: Q(x) ∧ Q(X) ∧ (∀α: R(x,α) ⇒ R(X,α) )}
    [ d(x)→ r:= e(x)
     ¬d(x)→ r:= g'(s(x),X)
    ]
{Post: R(X,r)}

Si R(x,r) es de la forma r = h(x):
(∀α: R(x,α) ⇒ R(X,α) ) = (h(x) = h(X))
```


5.3. Obtención de postcondición constante.

- Ejemplo 1: Factorial.

```
func i_fact(n, w:Nat) dev f: Nat
{Pre: cierto; Dec: n}
[n = 0 -> f := w
 n > 0 -> f := i_fact(n-1, n * w)
]
{Post: f = n! * w}
```

- Ejemplo 2: Suma de una pila.

```
func g_sum(p: Pila; w:Ent) dev s: Ent
{Pre: cierto; Dec: altura(p)}
[ nula(p) -> s := w
 ¬nula(p) -> s := g_sum(desapilar(p), cima(p) + w)
]
{Post: s = suma(p) + w}
```