TEMA 5: DISEÑO DE ALGORITMOS ITERATIVOS

1. Instrucción Iterativa
2. Verificación de bucles
3. Transformación recursivo-iterativa
4. Derivación de bucles

1. Instrucción Iterativa

- **Instrucción iterativa:**
  - Permite ejecutar una instrucción repetidas veces. Similar a la instrucción alternativa.
  - Sintaxis:

```
while(B1) → I1    ó    *[B1 → I1
                         ]
```

  - Ejecución de la instrucción:
    a) Se evalúa la protección (B1).
    b) Si la protección está abierta (es cierta), se ejecuta I1 y se vuelve al paso *a*.
    c) Si la protección está cerrada (es falsa) se acaba la ejecución de la instrucción iterativa.

1. Instrucción Iterativa

- **Ejemplo:** Sumatorio de 1 a N

```
f = 0; i = 1;
while(i <= N)
{
    f = f + i;
    i = i + 1;
}
```

  - En lenguaje while:

```
f := 0; i := 1;
while(i <= N) → f := f + i;
                i := i + 1;
```


2. Verificación de bucles

- **Invariante de bucle:** Aserto que siempre es cierto al principio y final de cada iteración. Se usa para verificar el bucle.
  - Para el ejemplo anterior:

```
{ i >= 1}  ó  {f = Σα:1 ≤ α ≤ i-1:α∧(i<=n+1)}
```

  - Se intenta que el invariante sea lo más informativo posible. Normalmente indicará qué lleva calculado el bucle en una iteración cualquiera.
- **Función de cota:** Función que devuelve un natural y permite definir un preorden sobre los parámetros del bucle. Permite la aplicación del principio de inducción y garantiza que el bucle acaba.

```
Para bucle anterior: N-i+1
```


2. Verificación de bucles

- **Verificación de la instrucción iterativa:**

```
{Pre}
{Inv: A; Dec: t(x)}
   while(B1) → I1
{Post}
```

1. El invariante se cumple a la entrada del bucle: Pre $\Rightarrow$ A
2. El invariante se cumple al final de una iteración cualquiera: $\{A \wedge B1\}\ I1\ \{A\}$
3. La postcondición se cumple al acabar el bucle: $A \wedge \neg B1 \Rightarrow Post$
4. Función de cota definida en los naturales: $t(x) \in \mathcal{N}$
5. Función de cota estrictamente decreciente: $t(s(x)) < t(x)$

2. Verificación de bucles

- **Ejemplo:** Verificar la función sumatorio:

```
func sumatorio(n: Natural) dev f: Natural
{Pre: cierto}
    f := 0; i := 1;
{Inv: f = Σα:1 ≤ α ≤ i-1:α∧(i<=n+1); Dec: n-i+1}
    while(i <= n) → f := f + i;
                    i := i + 1;
{Post: f = Σα:1 ≤ α ≤ n:α}
```

3. Transformación recursivo-iterativa

- El proceso completo para pasar una función de recursiva a iterativa es:

3. Transformación recursivo-iterativa

- **Función recursiva final con postcondición constante:**

```
func f(X: T1; y: T2) dev r: T3
{Pre: Q(X,y);  Dec: t(X,y)}

    [ d(X,y)→ r := h(X,y)
     ~d(X,y)→ r := f(X, s(X, y))
    ]
{Post: R(X,r)}
```


3. Transformación recursivo-iterativa

- **Bucle asociado:**

```
func f(X: T1; y: T2) dev r: T3

{Inv: Q(X,y);  Dec: t(X,y)}
    while(~d(X,y)) → y := s(X, y)

    r := h(X, y)
{Post: R(X,r)}
```

  - Demostración en "Programación metódica", Balcazar, pp 206-208

3. Transformación recursivo-iterativa

- **Ejemplo:** Función factorial

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

Función auxiliar:

```
func fact(n:Nat) dev f: Nat
{Pre: cierto}
  f := i_fact(n, 1, n)
{Post: f = n!}
```


3. Transformación recursivo-iterativa

- **Ejemplo:** Suma de una pila

```
func g_sum2(p: Pila; w:Ent; P: Pila) dev s: Ent
{Pre: suma(p) + w = suma(P); Dec: altura(p)}
[nula(p) -> s := w
 ¬nula(p) -> s := g_sum2(desapilar(p), cima(p) + w, P)
]
{Post: s = suma(P)}
```

Función auxiliar:

```
func sum2(p: Pila) dev s: Ent
{Pre: cierto}
   s := g_sum2(p,0,p)
{Post: s = suma(p)}
```


4. Derivación de bucles

- El mayor problema al derivar una instrucción alternativa es encontrar el **invariante** adecuado.
- Un método para obtener un invariante es debilitar la postcondición.

```
func suma(a:vector) dev s: Ent
{Pre: cierto}
{Post: s = Σα:1 ≤ α ≤ N:a[α]}
```

Post. debilitada:

$s = \Sigma\alpha{:}1 \le \alpha \le i{:}a[\alpha] \wedge i \le N$ → **Invariante**

Ejercicio: Completar la derivación.

4. Derivación de bucles

- En ocasiones la postcondición no se puede debilitar sustituyendo constantes por variables.
- En este caso hay que intentar reproducir el proceso de obtención de la precondición en las funciones recursivas lineales con post. cte. → Relacionar resultado parcial con resultado final.

Ejemplo: Derivar la función altura de una pila.

```
func altura_i(p:pila) dev h: Nat
{Pre: cierto}
{Post: h = altura(p)}
```