TEMA 2: ESPECIFICACIÓN Y CORRECCIÓN DE ALGORITMOS

1. Corrección de un programa
2. Asertos
3. Especificación Pre/Post

1. Corrección de un programa

- **Uno de los grandes problemas de la programación es la escritura de programas totalmente correctos.**
  - Al implementar un programa **siempre** se introducen errores.
- **Técnicas de detección de errores:**
  - **Mediante pruebas:** Casos de prueba.
  - **Métodos formales**
    - Verificación formal
    - Derivación de programas

  Ej: Mult(a, b: Nat) dev m: Nat

1. Corrección de un programa

- **Problemas de la detección mediante pruebas:**
  - Elección de casos de prueba.
  - Escasa información cuando se detecta un error.
  - No es exhaustivo. No se garantiza que no hayan errores.
- **Problemas de la detección con métodos formales:**
  - Únicamente aplicable a programas de tamaño reducido.
  - No detecta errores de transcripción.

1. Corrección de un programa

-Def: Se define la corrección del programa como la coincidencia entre comportamiento deseado y comportamiento real.

- **Para definir el comportamiento esperado harán falta 3 elementos:**
  1. Datos suministrados (parámetros de entrada).
  2. Resultados esperados (parámetros de salida).
  3. Relación entre ellos.
- **Al conjunto de estos elementos se le denominará especificación.**

1.1. Semántica

- **Para definir el comportamiento real hará falta conocer el comportamiento exacto de cada instrucción: Semántica del lenguaje.**
- **Tipos de semánticas en informática:**
  - **Semántica operacional**: Expresa cómo se ejecuta cada instrucción (utiliza para expresarlo transiciones entre estados). Se utiliza sobre todo para la definición de lenguajes y construcción de compiladores.

1.1. Semántica

- **Semántica denotacional**: Expresa el resultado de ejecutar una o varias instrucciones mediante el uso de funciones matemáticas. Es útil para razonar sobre programas o analizarlos, con objeto por ejemplo de optimizarlos o detectar errores. Basado en el calculo Lambda.
- **Semántica axiomática**: Expresa propiedades específicas que se cumplen al ejecutar instrucciones mediante asertos lógicos. Muchos aspectos de la ejecución son ignorados, por lo que es útil para razonar a alto nivel de abstracción sobre el programa. Por ejemplo, si un programa es correcto y hace lo que debería de hacer.

TEMA 2: ESPECIFICACIÓN Y CORRECCIÓN DE ALGORITMOS

1. Corrección de un programa
2. Asertos
3. Especificación Pre/Post

2. Asertos

Def (Estado): Es una aplicación (σ) que asocia a cada identificador de variable un valor correspondiente a su dominio de datos.

σ: ID $\rightarrow$ D

Descripción completa de los valores asociados a todas las variables de un programa en un momento determinado de su ejecución.

- **Ejecutar una instrucción supone modificar el estado de un programa.**

Ejemplo:

```
Algoritmo Intercambia(x,y: Ent) dev x, y: Ent
Auxiliar: z: Ent
   z ← x
   x ← y
   y ← z
FIntercambia
```


2. Asertos

Ejemplo:

```
Intercambia(1,3)
```

Estado del programa

	x	y	z
	1	3	?
z ← x	1	3	1
x ← y	3	3	1
y ← z	3	1	1

```
FIntercambia
```


2. Asertos

Def (Aserto): Predicado que expresa un conjunto de estados de un programa.

- Cuando colocamos un aserto en un punto del programa, estamos indicando que en ese punto sólo son válidos los estados que hacen cierto el aserto.
- Cuando un estado hace cierto un aserto, decimos que satisface el aserto.

Ej.

```
si x >= 1 entonces
        {x >= 1}   Representa todos los estados del programa donde
                   x es mayor o igual que 1.
   x ← x - 1
        {x >= 0}
```


2.1. Propiedades de los asertos

Def (Equivalencia): Dos asertos P y Q son equivalentes ($P \equiv Q$) si los satisfacen los mismos estados, es decir, si los conjuntos de estados de P y Q son iguales:

estados(P) = estados(Q)

La equivalencia de asertos es similar a la doble implicación ($P \leftrightarrow Q$).

Ej:

c = (a + b) * (a – b) $\equiv$ (diferencia de cuadrados)

$c = a^2 - b^2$

2.1. Propiedades de los asertos

Def (Relación de fuerza): Dados dos asertos P y Q, se dice que P es más fuerte que Q ($P \Rightarrow Q$) si todo estado que satisface P también satisface Q. O dicho de otra forma, estados(P) $\subseteq$ estados(Q).

- El conjunto de asertos está parcialmente ordenado, ya que existen asertos entre los que no existe la relación.
- La relación de fuerza es similar a la implicación ($P \rightarrow Q$).
- ¿Elemento máximo? ¿Elemento mínimo?

Ej:

- A1 $\equiv$ n >=1 A2 $\equiv$ n >=0
- A1 $\equiv$ n >=1 A2 $\equiv$ n <=0

2.1. Propiedades de los asertos

- **Ejercicio:** Establecer la relación de orden (si la hay) entre cada uno de los asertos siguientes:
  - x > 0, (x > 0) $\wedge$ (y > 0)
  - x > 0, (x > 0) $\vee$ (y > 0)
  - x > 0, y >= 0
  - x > 0, (x >= 0) $\wedge$ (y >= 0)
  - (x > 0) $\wedge$ (y > 0), (x > 0) $\vee$ (y > 0)
  - (x > 0) $\wedge$ (y > 0), y >= 0
  - (x > 0) $\wedge$ (y > 0), (x >= 0) $\wedge$ (y >= 0)

2.2. Cuantificadores

- **Un cuantificador es una abreviatura para una secuencia de operaciones análogas.**
- **Los cuantificadores los escribiremos utilizando la siguiente sintáxis:**

  $\Sigma\alpha$: dominio: E(α)

  donde α es una variable ligada al cuantificador, que toma valores en el conjunto definido por *dominio*.

  Ejemplo:

  $\Sigma\alpha : 1 <= \alpha <= 4 : \alpha \equiv 10$

2.2. Cuantificadores

- **Los cuantificadores que vamos a utilizar son los siguientes:**

  $\Sigma\alpha$: dominio: E(α)

  $\Pi\alpha$: dominio: E(α)

  $\forall\alpha$: dominio: B(α)

  $\exists\alpha$: dominio: B(α)

  Nα : dominio: B(α)

  Max α : dominio: E(α)

  Min α : dominio: E(α)

E(x) es una expresión numérica y B(x) una expresión booleana.

2.2. Cuantificadores

- **Dominio nulo**
  - Un cuantificador con un dominio nulo es igual al elemento neutro de la operación asociada al cuantificador.
  - Los cuantificadores *Min* y *Max* no están definidos para un dominio nulo.

  Ejemplo:

  $\Sigma\alpha : 1 <= \alpha <= 0 : \alpha \equiv 0$

2.2. Cuantificadores

- **Separación de un elemento del cuantificador:**
  - Un cuantificador con un dominio no nulo, se puede separar uno de los elementos del cuantificador.
  - El elemento separado se combina con el cuantificador con la correspondiente operación asociada.

  Ejemplo: Para N > 0

  $\Sigma\alpha : 1 <= \alpha <= N : 2 * \alpha \equiv$

  $2 * 1 + \Sigma\alpha : 2 <= \alpha <= N : 2 * \alpha$

2.2. Cuantificadores

- **Separación de un elemento en el cuantificador de conteo:**

  $c = N\alpha : 1 <= \alpha <= N : B(\alpha) \equiv$ (para N > 0)

  $[c = 1 + N\alpha : 2 <= \alpha <= N : B(\alpha) \wedge B(1)] \vee$

  $[c = N\alpha : 2 <= \alpha <= N : B(\alpha) \wedge \text{-}B(1)]$

2.3. Variables

- **Las variables de los asertos pueden ser de los siguientes tipos:**
  - Ligadas (griegas)
  - Libres
    - de programa (Minúsculas)
    - iniciales (Mayúsculas)

Ejemplo 1:

```
{x = X ∧ y = Y}
P
{x = Y ∧ y = X}
```

Ejemplo 2:

$\Sigma\alpha : 1 <= \alpha <= n : v[\alpha] = A$

2.3. Sustituciones

- **Operación definida sobre asertos. Útil para razonar sobre asignaciones.**

Def (Sustitución simple): Sea A un aserto, x una variable y E una expresión del mismo tipo que x, entonces se define A[x ← E] como el aserto que se obtiene al sustituir todas las apariciones de x en A por la expresión E.

Ejemplo:

```
x = y + 2 [x ← x * 2]
```

- **NO deben existir variables ligadas que se llamen igual que la variable de programa a sustituir.**

2.3. Sustituciones

Def (Sustitución múltiple): Sea A un aserto, x1, ..., xn variables y E1, ..., En expresiones del mismo tipo que las respectivas variables, entonces se define el aserto A[x1,...,xn←E1, ..., En] como el aserto que se obtiene al sustituir todas las apariciones de xi en A por Ei para todo i.

Ejemplo:

```
x = y [x, y ← x + y, y + 1]
```


TEMA 2: ESPECIFICACIÓN Y CORRECCIÓN DE ALGORITMOS

1. **Corrección de un programa**
2. **Asertos**
3. **Especificación Pre/Post**

3. Especificación Pre/Post

- **Especificación de un conjunto de instrucciones:**
  - Consta de dos asertos: *Precondición* y *Postcondición*.
  - La precondición indica las condiciones que debe cumplir el estado inicial del programa.
  - La postcondición indica las condiciones que ha de cumplir el estado final del programa.
- **Significado de la especificación:** Si al empezar a ejecutarse el conjunto de instrucciones se cumple la precondición, al acabar necesariamente tiene que cumplirse la postcondición.

3. Especificación Pre/Post

- **Ejemplo de especificación:**

```
{x > 0}
P
{x > 2}
```

Programas que podrían cumplir esta especificación:

```
x := 3;          x := x + 2;          ...
```

3. Especificación Pre/Post

- **Verificar un programa consistirá en demostrar, a partir de la semántica de las instrucciones y de la precondición, que se cumple la postcondición.**
- **Derivar un programa consistirá en, a partir únicamente de la especificación (precondición y postcondición), calcular un programa que la cumple.**

3. Especificación Pre/Post

- **Especificación de funciones:**
  - **Cabecera**: Nombre de la función, nombre y tipo de los parámetros de entrada y nombre y tipo de los resultados (parámetros de salida).
  - **Precondición**: Aserto a cumplir para que la función se pueda ejecutar. Sólo puede hacer referencia a los parámetros de entrada.
  - **Postcondición**: Aserto que se cumplirá cuando la función acabe, y que relaciona los resultados con los parámetros de entrada.

Sintaxis de cabecera:

func Nombre(parámetros entrada) dev resultados

Especificación informal

Especificación informal

```
// Ejemplo de especificaciones

const int N = 10;

typedef int Vector[N];

/*************************************************************
* Funcion Inicializa
* Pone el valor inicial de todos los elementos del vector a 0
*
* Parametros:
*  v            E y S vector
*************************************************************/
void Inicializa(Vector v)
{
        int i;

        for(i = 0; i < N; i++)
                v[i] = 0;
}

/*************************************************************
* Funcion Suma
* Suma dos vectores
*
* Parametros:
*  v1             E vector a sumar
*  v2             E vector a sumar
*  vres           S vector resultado
*
*************************************************************/
void Suma(Vector v1, Vector v2, Vector vres)
{
        int i;

        for(i = 0; i < N; i++)
                vres[i] = v1[i] + v2[i];
}
```

3. Especificación Pre/Post

- **Especificación formal de las funciones:**

```
func Inicializa(v: Vector) dev v: Vector
{Pre: cierto}
{Post: ∀α: 0 <= α < N: v[α] = 0}

func Suma(v1, v2: Vector) dev vres: Vector
{Pre: cierto}
{Post: ∀α: 0 <= α < N: vres[α] = v1[α] + v2[α] }
```

- **Ejercicios:**
  1. Especificación de una función que calcule el cociente y resto de una división entera.
  2. Especificación de una función que devuelva el máximo de un vector.
  3. Especificación de una función que devuelva la posición del máximo de un vector.

3.1. Propiedades de especificaciones

- **Sean la siguiente especificación y programa correctos:**

```
{A1}
 P
{A2}
```

- Si A1' ⇒ A1 entonces `{A1'} P {A2}` es correcto. Esto se denomina reforzar la precondición.
- Si A2 ⇒ A2' entonces `{A1} P {A2'}` es correcto. Esto se denomina debilitar la postcondición.

3.2. Precondición más débil

- El concepto de precondición / postcondición se puede formalizar mediante lógica de predicados. Para ello definiremos el predicado **pmd(P, A)** (precondición más débil).
- Este predicado transforma el aserto A en otro aserto que es la pmd que garantiza que tras ejecutar P se cumplirá A.
- Una especificación se reescribe en lógica de predicados de la siguiente manera:

```
{A1}
 P          ⇒   A1 ⇒ pmd(P, A2)
{A2}
```

3.2. Precondición más débil

- **El predicado pmd debe cumplir los siguientes 4 axiomas:**
  1. pmd(P, falso) $\equiv$ falso
  2. pmd(P, A1) $\wedge$ pmd(P, A2) $\equiv$ pmd(P, A1 $\wedge$ A2)
  3. Si A1 $\Rightarrow$ A2 entonces pmd(P, A1) $\Rightarrow$ pmd(P, A2)
  4. [pmd(P, A1) $\vee$ pmd(P, A2)] $\Rightarrow$ pmd(P, A1 $\vee$ A2)
- **Mediante estos axiomas se pueden demostrar diferentes propiedades de la pmd.**
- **Para cada instrucción de nuestro lenguaje deberemos definir la pmd.**

TEMA 2: ESPECIFICACIÓN Y CORRECCIÓN DE ALGORITMOS

1. **Corrección de un programa**
2. **Asertos**
3. **Especificación Pre/Post**