**Tema 6: Metodología de Desarrollo de Programas II**

1. **Desarrollo con esquemas de selección e iteración.**

En este apartado se ilustra el empleo de la técnica de refinamientos sucesivos (convertir un problema complejo en problemas más simples) aplicadas a las estructuras de selección e iteración. Con ello se tienen tres posibilidades a la hora de refinar una acción compuesta:

* Organizarla como secuencia de acciones.
* Organizarla como selección entres acciones alternativas.
* Organizarla como iteración de acciones.

## ESQUEMA DE SELECCIÓN

Se trata de elegir una acción entre varias acciones alternativas. Habrá que **identificar** cada una de las **alternativas del esquema** y las acciones correspondientes e **Identificar** las **condiciones** para seleccionar una alternativa u otra.

Por ejemplo, calcular el número de días que tiene febrero en un determinado año. Las alternativas son que febrero tengo 28 o 29 días. Las acciones serán asignar uno de estos dos valores a una variable que almacene el numero de días. La condición para elegir una condición u otra es que el año sea bisiesto.

**Alternativas**: 28 ó 29 días.

**Condición**: año bisiesto o año no bisiesto.

## ESQUEMA DE ITERACIÓN

Una iteración o bucle consiste en la repetición de una acción o grupo de acciones hasta conseguir el resultado deseado. Para desarrollar un esquema de iteración deberemos identificar cada uno de los elementos componentes:

* Identificar acciones útiles a repetir y las variables necesarias. Precisar el significado de estas variables al comienzo y al final de cada repetición.
* Identificar como actualizar la información al pasar de cada iteración a la siguiente. Puede ser necesario introducir nuevas variables.
* Identificar la condición de terminación. Puede ser necesario introducir nuevas variables e acciones para mantenerlas actualizadas.
* Identificar valores iniciales de las variables y si es necesaria algunas acciones para asignárselos antes de entrar en el bucle.

Pongamos un ejemplo, como la serie de Fibonacci, Cada termino de esta serie se obtiene sumando los dos anteriores. La serie comienza con los términos 0 y 1, que se suponen ya impresos antes del bucle. Se trata de calcular e imprimir tantos términos como se posible. Procediendo paso a paso:

1. **Acciones útiles a repetir:** Imprimir un término.

|printf(“%d\n”,termino);

**Variables necesarias:** El término a Imprimir.

|**int** termino;

**Valor al empezar la repetición:** Ultimo término impreso hasta el momento.

1. **Actualizar las variables al pasar de una repetición a la siguiente:** Antes de imprimir, calcular el término actual a partir de los dos anteriores (se necesita tener almacenado el penúltimo término).

**|aux = termino + anterior;**

**|anterior = termino;**

**|termino = aux;**

**Variables adicionales:** el penúltimo termino y una variable temporal.

1. **Condición de terminación:** El término siguiente excedería del rango de los enteros. Hay que evaluar la condición sin calcular explícitamente el valor de dicho término, porque se produciría "overflow".

INT-MAX-termino < anterior;

1. **Valores iniciales de las variables:** Los dos primeros términos, O y 1I anterior = O; termino = 1;

El programa completo en C+- sería:

int termino;

int anterior;

int aux;

anterior = 0;

termino = 1;

**while** (int\_max-termino>=anterior){

aux = termino + anterior;

anterior =termino;

termino = aux;

printf(“%d\n”,termino);

}

1. **Verificación de programas.**

Uno de los objetivos de la programación es la corrección. Consideramos un programa correcto si produce siempre resultados de acuerdo con la especificación del programa. Un **ensayo** consiste en ejecutar el programa con unos datos preparados de antemano y para los cuales se sabe cuál ha de ser el resultado a obtener. Si el resultado obtenido no es el esperado, significa que el programa tiene un **error**, habrá que buscarlo y corregirlo. Este proceso se llama depuración.

La única manera de verificar con seguridad la corrección de un programa es demostrar formalmente que el programa cumple con sus especificaciones con toda precisión en forma de expresiones lógicas, y luego realizar una demostración lógico-matemática de que el programa las cumple. (Véase Apéndice C para ver este tipo de demostración).

# CORRECCIÓN PARCIAL Y TOTAL

Para programas que siguen el modelo imperativo el proceso de demostración se realiza en dos partes:

1. **Corrección parcial:** si el programa termina el resultado es correcto.

* Anotar el comienzo y final del programa con aserciones o asertos (afirmaciones, formalizadas como expresiones lógicas) . La condición del comienzo se denomina pre-condición y al final se denomina post-condición.
* Anotar en los puntos intermedios del programa con aserciones similares respecto al estado del programa en ese punto, o más bien, el estado que debería obtenerse en ese punto.

1. Corrección total: el programa termina y además para todo dato de entrada válido el programa termina.

* Esta se consigue realizando una corrección parcial del programa junto con la demostración de que los bucles finalizan tras un numero finito de repeticiones.

Para analizar el comportamiento de un fragmento de programa:

* Correspondiente a una sentencia de asignación: anotamos antes de dicho fragmento las condiciones que se cumplen antes de ejecutarla, estas condiciones no intervienen en la variable asignada. Posteriormente, pondremos las condiciones que se cumplen después de ejecución.

Debido a la complejidad de los siguiente epígrafes, según mi criterio, es mejor estudiarlos desde el libro.