



Tema 2. Diseño Modular.

Diseño de Algoritmos.



E.U. Politécnica
Curso 2004-2005
Departamento Lenguajes y Ciencias de la Computación.
Universidad de Málaga
José Luis Leiva Olivencia.
Despacho: I-326D (Edificio E.U.P)/ 3.2.41 (Teatinos-E.T.S.I.I.)



Introducción.

¿Por dónde empezaste a escribir tu programa? Seguramente, después de una lectura rápida de las especificaciones y de los requisitos del programa la siguiente tarea fue la escritura del código en algún lenguaje de programación. Posteriormente comenzaría la ardua tarea de corregir errores, depurar el código, cambiar la lógica del programa, etc.

Realizar un proyecto grande de software requiere la participación de un equipo de programadores, que necesitan una **buena planificación, organización y comunicación** entre ellos para conseguir su meta. La **ingeniería del software**, afortunadamente, proporciona técnicas para facilitar el desarrollo de programas.



Introducción (II)

El término **resolver un problema** significa, a grandes rasgos, llevar a cabo todo el proceso de **definición** del problema y desarrollo de una **solución**, a modo de programa, que lo resuelva.

Una **solución** consta de dos componentes: **algoritmos** y estructuras de **datos**.



Introducción (III)

INCONVENIENTES DE LA PROGRAMACIÓN TRADICIONAL.

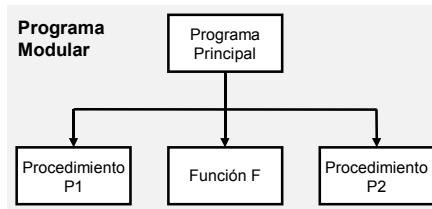
- rigidez e inflexibilidad de los programas,
- pérdida excesiva de tiempo en la corrección de errores
- documentación deficiente e ineficiente, incluso mala,
- imposibilidad de reutilizar el código o fragmentos del mismo en proyectos futuros



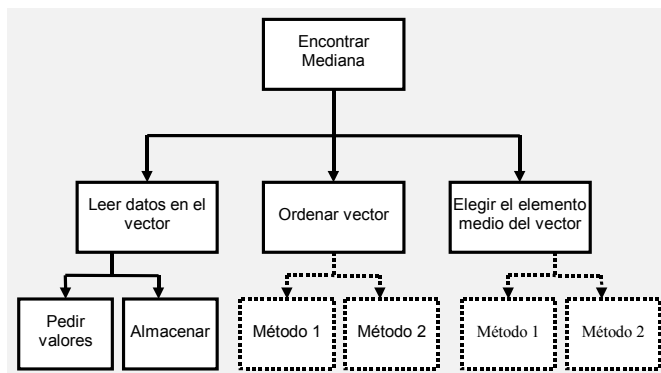
Introducción (IV)

DISEÑO TOP-DOWN

El diseño top-down es una de las metodologías más empleadas en programación. Está basada en la técnica de resolución humana de problemas: **divide y vencerás**. Consiste en dividir el algoritmo en unidades más pequeñas sucesivamente hasta que sean directamente ejecutables en el ordenador.



Introducción (V)





Introducción (VI)

- **Abstracción procedural:**
 - Cada algoritmo es como una CAJA NEGRA

Cada caja negra especifica **qué** se hace, no **como** se hace. A la vez, ninguna caja negra debe saber cómo otra caja negra realiza una tarea, sino sólo qué acción realiza. Los distintos componentes de una solución deben mantenerse aislados unos de otros.

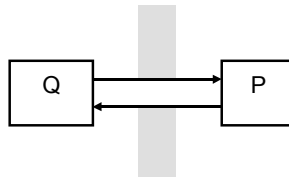
La **abstracción procedural** separa el propósito de un programa de su implementación.

Una vez que el programa ha sido escrito es posible usarlo sin necesidad de conocer las particularidades de su algoritmo, con sólo tener una definición de la acción que realiza y una descripción de los parámetros que maneja.



Introducción (VII)

- **Ocultación:** El principio de ocultación de la información no sólo oculta los detalles en cajas negras, sino que asegura que ninguna otra caja negra puede acceder a esos datos



- **Q** no debe saber cómo **P** realiza la ordenación pero si que debe “pasarle” un array de enteros y de longitud **MaxArray**



Introducción (VIII)

- Las **especificaciones** suponen un conjunto de condiciones que deben cumplirse para que se ejecute un módulo. Puede verse como un contrato entre los módulos implicados que ayuda a descomponer el problema en tareas más pequeñas y, en caso de proyectos en equipo, a **depurar responsabilidades**. El contrato debe especificar: cómo usar la rutina o tarea; qué hace la rutina (resultado de su ejecución).



Introducción (IX)

- Aspectos favorables de la modularidad:
 - Construcción del programa (son + pequeños)
 - Depuración del programa (la modularidad aísla errores).
 - Facilidad para la legibilidad.
 - Modularidad aísla las modificaciones.
 - Eliminación de redundancia.



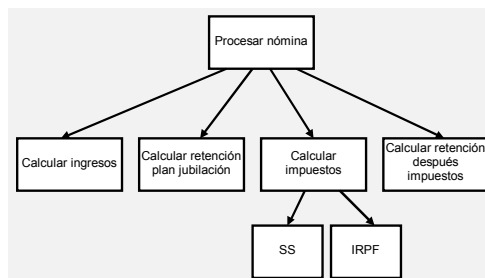
Introducción (X)

- Criterios de modularización:
 - NO HAY.
 - El objetivo de la modularidad es obtener software que sea manejable, de modo que una modificación futura afectare a unos pocos módulos.
 - Por **acoplamiento** se entiende el grado de interconexión entre los módulos. El objetivo será, por tanto, maximizar la independencia de los módulos, es decir, minimizar el acoplamiento.



Introducción (XI)

- El acoplamiento entre módulos se da de varias maneras: de **control** y de **datos**. El **acoplamiento de control** implica la transferencia del control de un módulo a otro (llam/ret de subprogramas). El **acoplamiento de datos** se refiere al compartir de datos entre los mod.



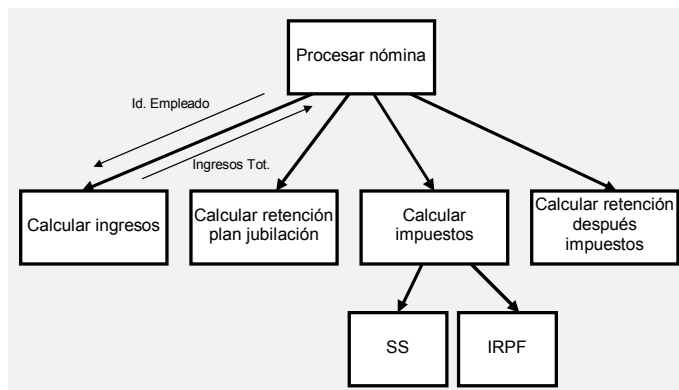


Introducción (XII)

- Los módulos se representan en la figura mediante rectángulos, mientras que las relaciones entre ellos se representan mediante flechas (**diagrama de estructura**) y constituyen el acoplamiento de control. El acoplamiento de datos se representa en los diagramas de estructura mediante flechas adicionales que indican los elementos de información (datos o parámetros) que se pasan a un módulo cuando otro módulo solicita sus servicios, así como los datos que devuelve el módulo al módulos que realiza la llamada.



Introducción (XIII)



OJO: EFECTOS LATERALES Y VBLES. GLOBALES.



Ventajas de la modularización.

- Crear grandes programas de forma sencilla.
- Ayuda a que un programa sea más fácil de modificar.
- Compilación separada.
- Reutilización.
- Mayor comprensión.



Módulos de biblioteca (I)

- Modulo de definición (.h) y módulo de implementación(.c o .cpp).
- En los archivos de cabecera “.h” incluiremos (1) las definiciones de constantes (la cláusula #define del preprocesador); (2) variables globales; (3) la descripción del programa; y (4) los prototipos de las funciones que aparecen en el programa.



Módulos de biblioteca (II)

Fichero "prueba.h":

```
#define CONST1 1
#define CONST2 2

int VariableGlobal;

/*
 Programa: Este programa muestra la utilidad de los ficheros de
 cabecera
 Desarrollado por: Periquito de Los Palotes
 Descripción: Se limita a escribir en pantalla los valores 1 y 2
 Variables:
 VariableGlobal: ilustra la inclusión de una variable global
 Constantes:
 CONST1: constante 1
 CONST2: constante 2
*/

/* Prototipos de funciones */
void funcion1 (int uno);
/*
 uno = variable para la salida por pantalla
 Imprime en pantalla el valor de la variable uno */
void funcion2 (int dos);
/*
 dos = variable para la salida por pantalla
 Imprime en pantalla el valor de la variable dos */
```

Departamento de Lenguajes y Ciencias de la Computación.

17



Módulos de biblioteca (III)

Fichero "prueba.cpp":

```
#include <stdio.h>
#include "prueba.h"

void funcion1(int uno){
    printf("Uno= %d\n", uno);
}

void funcion2(int dos){
    printf("Dos= %d\n", dos);
}

void main()
{
    funcion1(1);
    funcion2(2);
    printf("Variable Global= %d\n", VariableGlobal);
}
```

Departamento de Lenguajes y Ciencias de la Computación.

18



Módulos de biblioteca (IV)

```
#include <stdio.h>
#include <conio.h>
#include "prueba.cpp"
void main()
{
printf("Variable global %d", variableglobal);
variableglobal=10;
funcion1(1);
funcion2(2);
printf("Variable global %d", variableglobal);
getche();
}
```



Compilación separada (I)

Archivo: uno.cpp

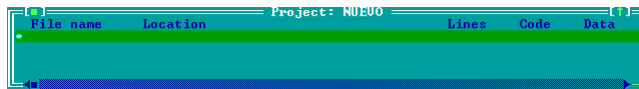
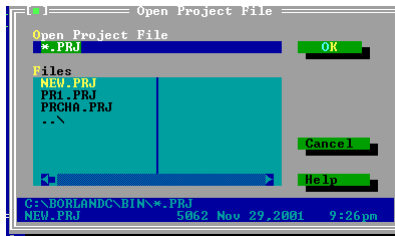
```
#include <stdio.h>
#include <conio.h>
void dos(int);
void main()
{
printf("Hola, es el programa uno\n");
dos(3);
getche();
}
```

Archivo: dos.cpp

```
#include <stdio.h>
void dos(int n)
{
printf("%d al cuadrado es %d\n",n,n*n);
}
```



Compilación separada(II)



Dev-C++(I)

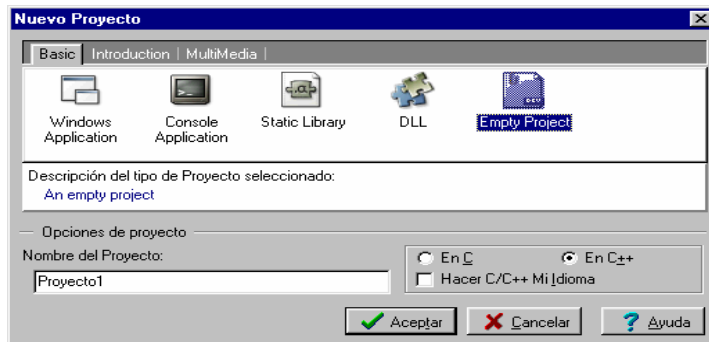


Figura 1



Dev-C++(II)

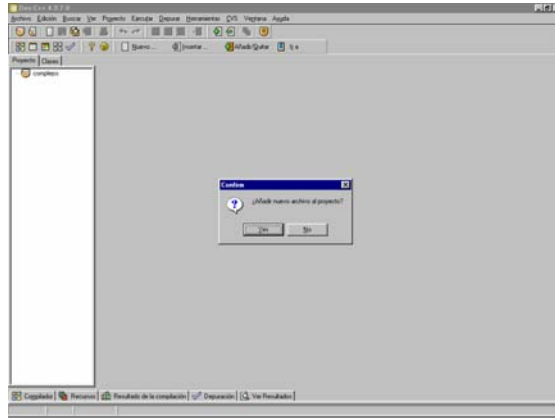


Figura 2