Diseño Modular Top-Down

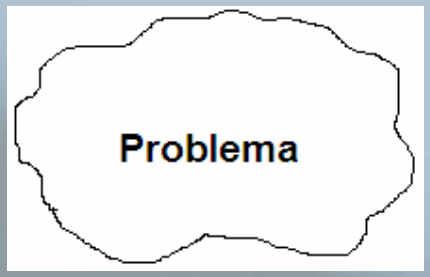
Diseño Tradicional

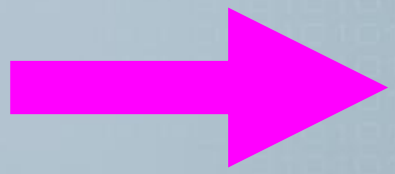
- • Pérdida excesiva de tiempo en la corrección de errores.
- • Documentación deficiente e ineficiente.
- • Imposibilidad de reutilizar el pseudocódigo o fragmentos del mismo en proyectos futuros.

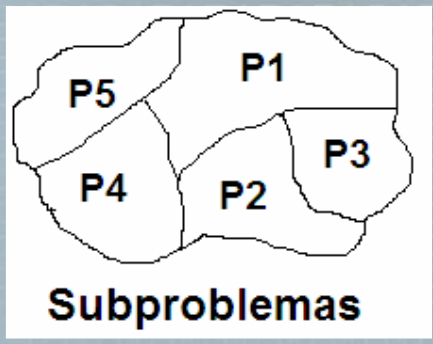
Top Down

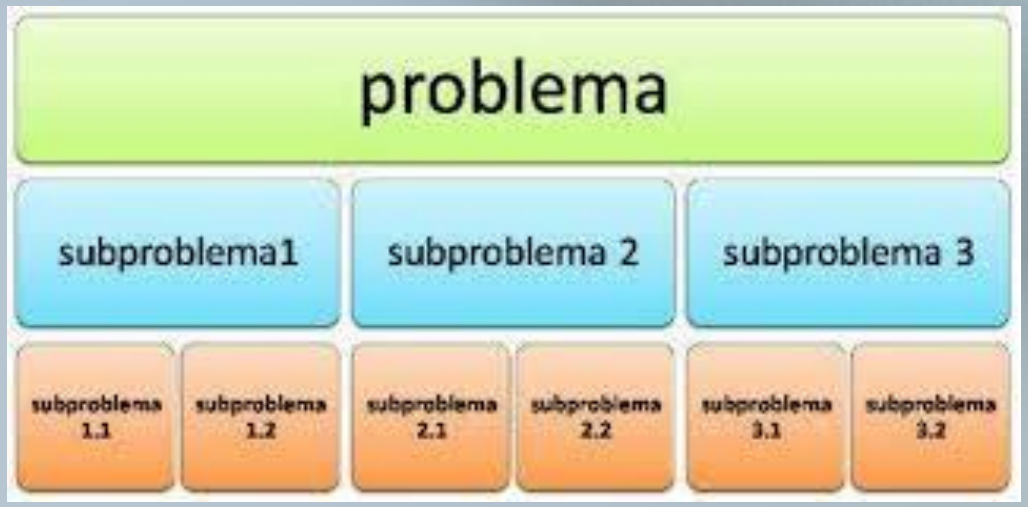
- El problema complejo es dividido en subproblemas más pequeños.
- Cada problema se resuelve por un modulo.
- Si algún subproblema, es demasiado complejo, éste se divide en subproblemas, y así sucesivamente

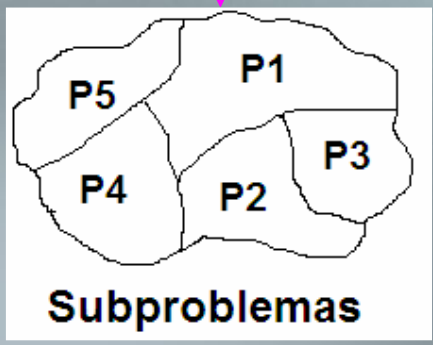
Ejemplo


Problema




P5
P1
P3
P4
P2
Subproblemas


problema
subproblema1
subproblema 2
subproblema 3
subproblema 1.1
subproblema 1.2
subproblema 2.1
subproblema 2.2
subproblema 3.1
subproblema 3.2


P5
P1
P3
P4
P2
Subproblemas

Ventajas Top-Down

- Construcción del algoritmos más pequeños.
- Depuración de los algoritmos es más fácil pues se aíslan los errores.
- Los algoritmos son más legibles.

Ventajas

- Pruebas separadas.
- Reutilización.
- Aísla las modificaciones.

Características

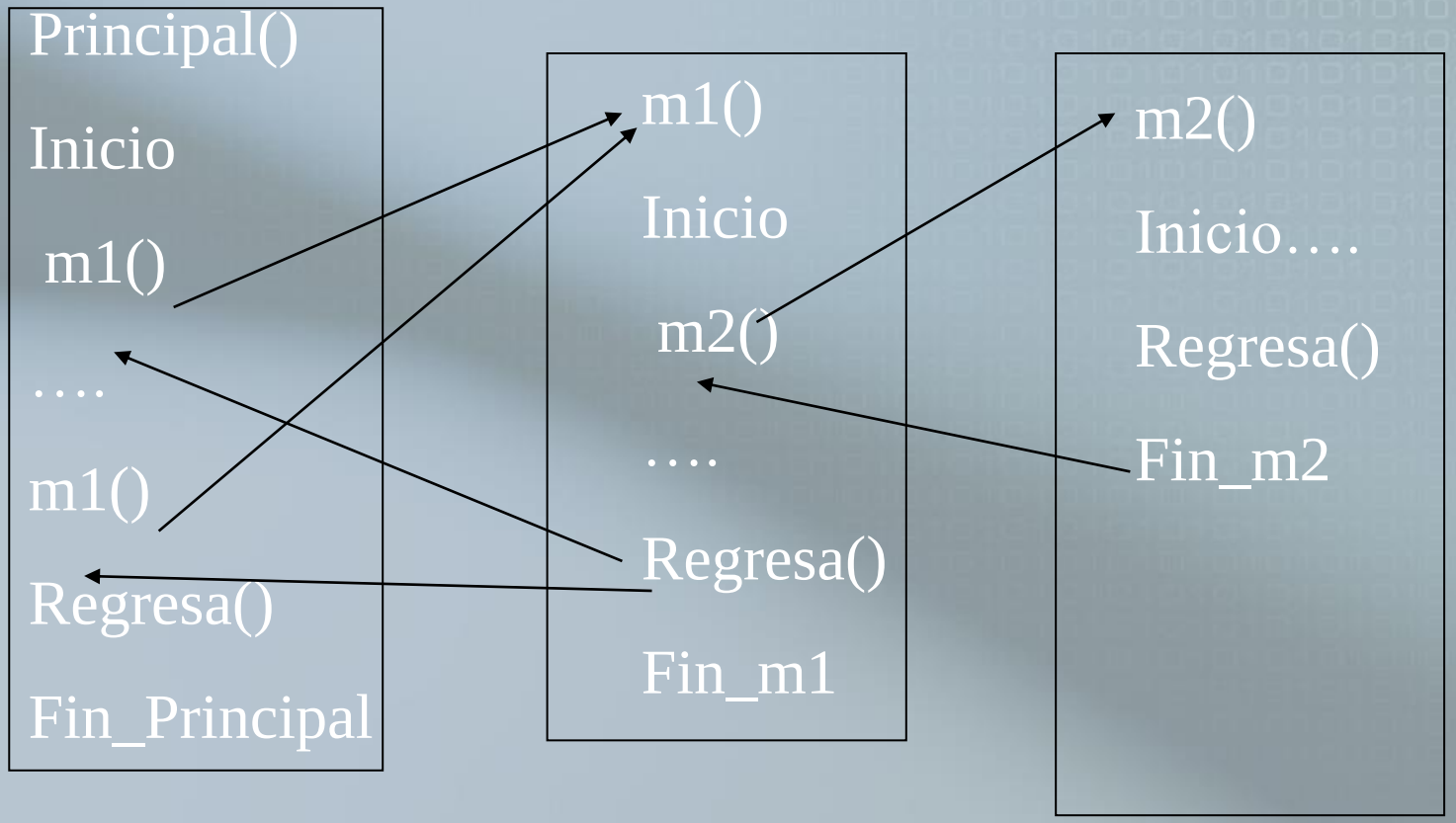
- Permiten dividir un problema en subproblemas mas fáciles de realizar y mantener.

- Todo algoritmo consta de al menos un módulo

Modulo

- Son bloques de sentencias con los que se construyen Algoritmos

Flujo de Control de llamado



Definición de módulo

```
tipo nombre([,argumentos])
   Inicio
      [declaraciones]
      sentencias
      Regresa([expresión])
   Fin_nombre
```

Encabezado

- Tipo: El tipo del valor devuelto
- Nombre: Identificador válido
- Argumentos: Declaraciones separadas por , de argumentos que recibe la función.

- Nota: si no recibe ni devuelve no se coloca nada

Cuerpo del módulo

- Sentencia compuesta que define lo que hace el modulo.
- Puede contener declaraciones de variables utilizadas en dichas sentencias. Estas variables, por defecto son locales al modulo.
- **Regresa([expresión])** se utiliza para devolver el valor del modulo

Ámbito de las variables

- Desde el punto de un módulo las variables pueden ser **locales** o **globales**:

- **Variables locales** se declaran dentro de un módulo y sólo tienen utilidad dentro de ese módulo.

Variables globales

- Son declaradas de forma que puedan ser utilizadas (consultada y/o modificada) desde cualquiera de los módulos que forman el algoritmo.
- En este caso, no puede haber dos variables con el mismo nombre(local y global)
- Nota: tienen prioridad las locales

Ejemplo

Ejemplo:
variables: x ← 20 de tipo entero Coment: variable global
escribe_x()
Inicio
  Escribir(" El valor de x (Global) es = ",x) coment: x=20
  Regresa()
Fin_escribe_x
Modulo_Principal ()
Inicio
 variables: x ← 12 de tipo entero Coment: variable local
 escribe_x() coment: Escribe el Valor de x = 20
 Escribir("El valor de x (Local) es = ",x) coment: x=12
 Regresa()
Fin_Principal

Ejemplo

letrero() ← No recibe

Inicio

coment: no variable locales

Escribir(" Este es un módulo simple")

Regresa() coment: no devuelve

Fin_letrero

No devuelve

Ejemplo

```
checa_num(entero i)        Si recibe
 Inicio    coment: No variables locales
   si (i>0) entonces
      Escribir(i, "es positivo")
  si_no
     Escribir(" i es negativo")
  Regresa()    coment: no devuelve
 Fin_checa_num
```

No devuelve

Ejemplo

```
entero suma(entero a, entero b)
Inicio
  entero valor coment: variable local
  valor ← a+b
  Regresa(valor) coment: devuelve
Fin_suma
```

Recibe enteros

Devuelve entero

Ejemplo

```
Entero GeneraFibo(N: Entero)
    Variables: Fn-1, Fn-2, Fn, i: Entero
    Inicio
      Fn-1 ← 0, Fn-2 ← 1
      Para (i←1 hasta N, incremento 1)
          Fn← Fn-1 + Fn-2
          Fn-1 ← Fn-2
          Fn-2 ← Fn
      Fin_Para
      Regresa(Fn)
Fin_GeneraFibo
```

$f_n = f_{n-1} + f_{n-2}$:

$f_0 = 0$

$f_1 = 1$

$f_2 = f_1 + f_0 = 1$

$f_3 = f_2 + f_1 = 2$

...

Ejemplo

```
Entero SumaDig(num: Entero)
    Variables: r,dig: Entero
    Inicio
      dig ←0
      Repetir
        r ←num mod 10
        dig ← dig + r
        num ←  num / 10
      Hasta(num<=0)
      Regresa(dig)
Fin_SumaDig
```

10

Ejemplo

```
Entero OrdSeln()
Inicio
    Variables: min, tem, i: Entero
                A[] ← {'d', 'i', 'c', 'e'}
    Inicio
      min ← 1
      Para i ← 1 Hasta 3 Inc 1
        Si (A[i+1] < A[min])
           min ← i+1;
         Fin_Si
        Fin_Para
        tem ← A[1];
        A[1] ← A[min];
        A[min] ← tem;
        Regresa()
Fin_ OrdSeln
```

```
Entero OrdSel()
Inicio
    Variables: min, tem, lugar: Entero
                A[] ← {'d', 'i', 'c', 'e'}
    Inicio
      Para lugar ← 1 Hasta 4 Inc 1
          min ← lugar
            Para i ← 1 Hasta 3 Inc 1
              Si (A[i+1] < A[min])
                 min ← i+1;
               Fin_Si
              Fin_Para
              tem ← A[lugar];
              A[lugar] ← A[min];
              A[min] ← tem;
        Fin_Para
        Para lugar ← 1 Hasta 4 Inc 1
           Escribir ("El arreglo es:" A[i])
        Fin_Para
              Regresa()
Fin_ OrdSel
```