Introducción a la Algorítmica y Programación (3300)

Prof. Ariel Ferreira Szpiniak - aferreira@exa.unrc.edu.ar
Departamento de Computación
Facultad de Cs. Exactas, Fco-Qcas y Naturales
Universidad Nacional de Río Cuarto

Teoría 3

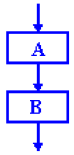
Estructras para componer Algoritmos Secuencial y Condicional

Noticias

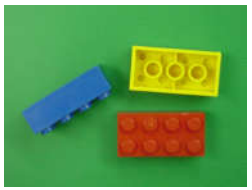
- **Hoja Aparte**
  - Publicación semanal de la UNRC con Noticias Universitarias en general. Sale todos los viernes y es gratuita. Se puede conseguir en el Comedor, Biblioteca, Facultad, Centro de Estudiantes, etc.

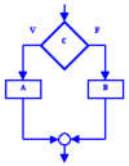
- **www.unrc.edu.ar**
  - Sitio institucional de la UNRC. Noticias diarias, información sobre aulas y horarios, becas, eventos, enlaces a sitios de interés, Biblioteca, Campus Virtual SIAT, Facultades, Carreras, etc.

Tipos de Estructuras para componer Algoritmos

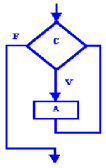


- Composición Secuencial



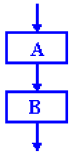


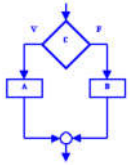
- Composición Condicional (Decisión, Selección)

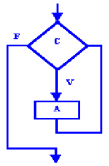


- Composición Iterativa (Repetición)

Tipos de Estructuras para componer Algoritmos





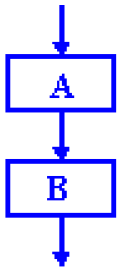


También son denominadas **Estructuras de Control**, sobre todo en los lenguajes de programación.

- Las estructuras de control permiten modificar el flujo de ejecución de las acciones de un algoritmo: tomar decisiones, realizar acciones repetitivas, etc, dependiendo de las condiciones que nosotros mismos propongamos en el algoritmo.
- Una estructura de control tiene un único punto de entrada y un único punto de salida.
- Una estructura de control se compone de acciones o de otras estructuras de control.

Composición Secuencial

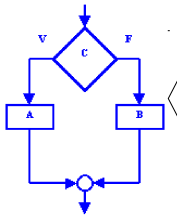
Es el tipo de composición más simple, está representada por una ***sucesión de acciones u operaciones*** (por ej. asignaciones), que se realizan una después de la otra, es decir, que el orden de ejecución coincide con el orden físico de aparición de las mismas, es decir, de arriba hacia abajo.



Las cajas A y B pueden ser definidas para ejecutar desde una simple acción hasta un módulo o algoritmo completo.

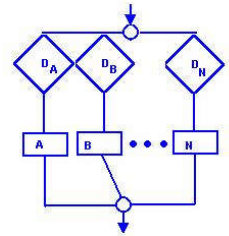
Composición Condicional

En un algoritmo representativo de un problema real es casi imposible que todo sea secuencial. Es necesario tomar ***decisiones*** en función de los datos del problema. La toma de decisión puede ser entre ***dos o más*** alternativas.



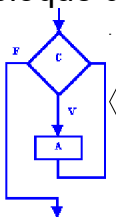
C es una condición que se evalúa. **A** es la acción que se ejecuta cuando la evaluación de este predicado resulta **verdadera** y **B** es la acción ejecutada cuando es **falsa**.

D_A, D_B,..., D_N son decisiones a tomar. **A** es la acción que se ejecuta cuando la **D_A** es verdadera. **B** es la acción ejecutada cuando **D_B** es verdadera. **N** es la acción ejecutada cuando **D_N** es verdadera.



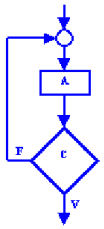
Composición Iterativa

En muchas ocasiones es necesario realizar una a varias acciones de manera repetida, ya sea una cantidad predeterminada de veces o no. La composición iterativa, en todas sus alternativas posibilita la ejecución repetida de un bloque de acciones. Veamos algunos ejemplos.



C es una condición que se evalúa. **A** es la acción o secuencia de acciones que se ejecuta cuando la evaluación de este predicado resulta **verdadera**. En caso de que la evaluación resulte **falsa**, **A** no se ejecuta más y **se continúa con la estructura siguiente**.

A es la acción o secuencia de acciones. **C** es una condición que se evalúa. Cuando la evaluación de este predicado resulta **falsa, A vuelve a ejecutarse**. Si la evaluación resulta **verdadera**, A no se ejecuta más y **se continúa con la estructura siguiente**.



Composición Secuencial

Debido a la sencillez de este tipo de composición, veremos su utilización a través de un ejemplo.

Más adelante veremos que es posible componer secuencialmente las demás estructuras, otros módulos, acciones y funciones.

"¿Cuánto es el precio final de un producto para la venta en un comercio?"

Composición Secuencial
Análisis del Problema

Datos de entrada: ¿Cuáles y cuántos son los valores de entrada? ¿Qué nombre significativo puedo darle a esos datos?
Dibujo o esquema que permita entender mejor el problema
Resultados (salida): ¿Cuáles y cuántos son los valores del resultado? ¿Qué nombre significativo puedo darle a esos resultados?

Relaciones o subproblemas: en caso de existir, describir las relaciones existentes entre los datos, los resultados u otra información adicional que sea necesaria para la resolución del problema. O suproblemas en caso de ser un problema más complejo.

- **Dato/s**: precio bruto es un número (precioBruto) e IVA un número (iva)
- **Resultado/s**: precio final es un número (precioFinal)
- **Relaciones o subproblemas**:

precioFinal = precioBruto + (precioBruto*iva)/100

Composición Secuencial
Diseño del Problema

Las acciones a ejecutar son: obtener el precio bruto y el iva, calcular el precio final, e informar el precio final. El algoritmo debe ejecutar estas acciones de manera secuencial y en este orden.

```
Algoritmo CalcularIva
Lexico
  precioBruto ∈ R       //variable dato
  iva ∈ R               //variable (o constante?) dato.
  precioFinal ∈ R       //variable resultado
Inicio
  Entrada:precioBruto iva
  precioFinal ← precioBruto + (precioBruto*iva)/100
  Salida:precioFinal
Fin
```


Composición Secuencial
Implementación del Problema

```
#include <stdio.h> //CalculaIva

/* léxico */
float precioBruto;
float iva;
float precioFinal;

/* función principal (main) en todo programa C */
void main(){
  scanf("%f",&precioBruto);
  scanf("%f",&iva);
  precioFinal = precioBruto+(precioBruto*iva)/100;
  printf("%f",precioFinal);
}
```


Composición Condicional

Existen diversas formas de composición condicional. **Una de las tareas en la etapa de diseño es la elección de la forma más adecuada** para el problema en cuestión.

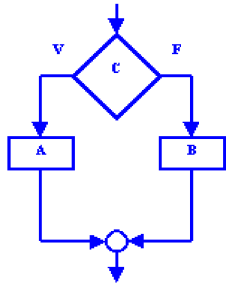
- Dos casos:
  - `si...entonces...sino`
- Un caso (condición excepcional):
  - `si...entonces`…
- Múltiples casos :
  - `segun ...`
  - `segun ... otros`

Composición Condicional

si ... entonces ... sino

Sintaxis:

```
si <condicion> entonces //comentario1
  <acciones1>
sino //comentario2
  <acciones2>
fsi
```



Semántica: Se evalúa la **condición**. Cuando la evaluación de la **condición** es **verdadera**, se ejecutan las **acciones** del **entonces** (acciones$_1$), únicamente. Cuando la evaluación de la **condición** es **falsa**, se ejecutan las **acciones** del **sino** (acciones$_2$), únicamente.

Composición Condicional

si ... entonces ... sino - Ejemplo

¿Cuándo un número es positivo y cuándo negativo?

Análisis

Datos: numero es un número (num)

Resultados: num es positivo o negativo (resultado).

Relaciones o subproblemas: num es positivo si es mayor o igual a cero. num es negativo si es menor a cero. El cero se considera positivo.

Composición Condicional

si ... entonces ... sino - Ejemplo

Diseño

```
Algoritmo PositivoNegativo
Lexico
  num ∈ Z //variable con el número a analizar
  resultado ∈ Logico // variable para informar
Inicio
  Entrada:num
  si num < 0 entonces //num es negativo
    resultado ← Falso
  sino //num es positivo
    resultado ← Verdadero
  fsi
  Salida:resultado
Fin
```


Composición Condicional

si ... entonces ... sino - Ejemplo

Diseño

```
Algoritmo PositivoNegativo2
Lexico
  num ∈ Z //variable con el número a analizar
  resultado ∈ Logico // variable para informar
Inicio
  Entrada:num
  si num >= 0 entonces //num es positivo
    resultado ← Verdadero
  sino //num es negativo
    resultado ← Falso
  fsi
  Salida:resultado
Fin
```

Composición Condicional
si ... entonces ... sino - Ejemplo

Diseño

```
Algoritmo OtraFormaDePositivoNegativo
Lexico
  num ∈ Z //variable con el número a analizar
  resultado ∈ Cadena // variable para informar
Inicio
  Entrada:num
  si num < 0 entonces //num es negativo
    resultado ← "es negativo"
  sino //num es positivo
    resultado ← "es positivo"
  fsi
  Salida:resultado
Fin
```


Composición Condicional
IF ... THEN ... ELSE - Ejemplo

```
#include <stdio.h> //PositivoNegativo
/* Lexico */
int  num; //variable con el numero a analizar
int resultado;// variable para informar
void main(){
  printf("Ingrese un numero entero: ");
  scanf("%d",&num);
  if (num<0) {
    //num es negativo
    resultado=0;
  }
  else {
    //num es positivo
    resultado=1;
  }
  // printf("El numero es %d \n", resultado);
}
```


Composición Condicional
IF ... THEN ... ELSE - Ejemplo

```
#include <stdio.h> //PositivoNegativo2
/* Lexico */
int  num; //variable con el numero a analizar
int resultado;// variable para informar
void main(){
  printf("Ingrese un numero entero: ");
  scanf("%d",&num);
  if (num>=0) {
    //num es positivo
    resultado=1;
  }
  else {
    //num es negativo
    resultado=0;
  }
  // printf("El numero es %d \n", resultado);
}
```


Composición Condicional
IF ... THEN ... ELSE - Ejemplo

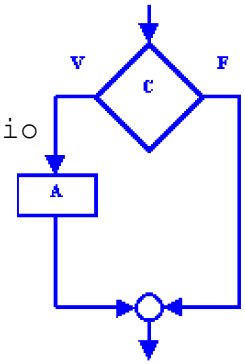
```
#include <stdio.h> <string.h> //OtraFormaDePositivoNegativo
/* Lexico */
int  num; //variable entera con el numero a analizar
char resultado[21]; //variable cadena para informar
void main(){
  printf("Ingrese un numero entero: ");
  scanf("%d",&num);
  if (num<0) {
    //num es negativo
    strcpy(resultado,"es negativo"); //Copia en resultado
  }
  else {
    //num es positivo
    strcpy(resultado,"espositivo"); //Copia en resultado
  }
  printf("El numero es %s \n", resultado);
}
```

Composición Condicional

si ... entonces

Sintaxis:

```
si <condicion> entonces //comentario
  <acciones>
fsi
```



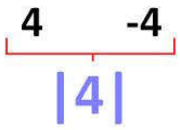
Semántica: Se evalúa la **condición**. Cuando la evaluación de la **condición** es **verdadera**, se ejecutan las **acciones** del **entonces**. Cuando la evaluación de la **condición** es **falsa**, no se hace **nada** y se continúa con la estructura siguiente.

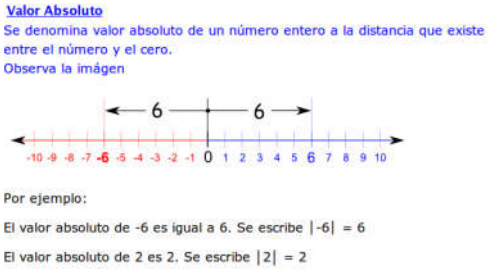
Composición Condicional

si ... entonces - Ejemplo

Encontrar el valor absoluto de un número entero.

Análisis





Datos: número es número entero (num)
Resultados: valor absoluto de num (num)
Relaciones o subproblemas: num no cambia si es mayor o igual a cero. Si num es negativo se le deba cambiar el signo (multiplicar el número por -1).

Composición Condicional

si ... entonces - Ejemplo

Diseño

```
Algoritmo ValorAbsoluto
Lexico
  num ∈ Z //var para almacenar el número a analizar
Inicio
  Entrada:num
  si num <= 0 entonces
     num ← num*(-1) // o num ← -num
  fsi
  Salida:num
Fin
```

Hacer otra solución usando si … entonces … sino

Composición Condicional

IF ... THEN - Ejemplo

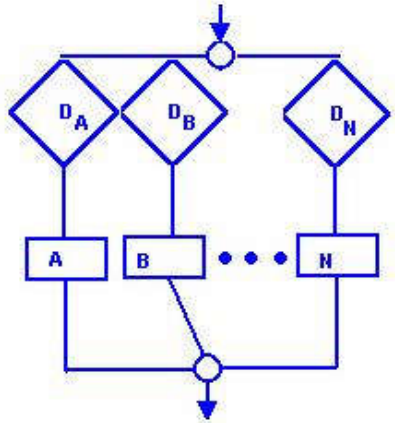
```
#include <stdio.h> // ValorAbsoluto

/* Variables */
int  num; //variable para almacenar el numero a analizar

void main(){
  printf("Ingrese un numero entero: ");
  scanf("%d",&num);
  if (num <=0) {
    //num es negativo
    num=num*(-1); // num=-num
  }
  printf ("Valor absoluto es: %d\n", num);
}
```

Composición Condicional

segun...

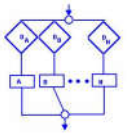
Sintaxis:

segun
<condicion$_1$>:<acciones$_1$>
<condicion$_2$>:<acciones$_2$>
⋮
<condicion$_n$>:<acciones$_n$>
fsegun



Composición Condicional

segun...



Semántica:

La toma de decisión involucra la evaluación de **una serie de condiciones**. Se evalúan cada una de las **condiciones**. Cuando la evaluación de una **condición** es **verdadera**, se ejecutan las **acciones** correspondientes a dicha **condición**.

- Las condiciones deben cubrir todo el dominio.
- Las condiciones deben ser mutuamente excluyentes, es decir, solo una es verdadera.

Hay otras semánticas para el según

Composición Condicional

segun ... - Ejemplo

Determinar si un ciudadano es niño, adolescente, adulto, o adulto mayor, de acuerdo a la edad que posee.

Análisis

Datos: edad es un número real (edad).

Resultados: madurez

Relaciones o subproblemas: edad entre 0 y 11 inclusive madurez es niño; edad entre 12 y 17 inclusive madurez es adolescente; edad entre 18 y 49 inclusive madurez es adulto; edad de 50 en adelante madurez es adulto mayor.

Composición Condicional

segun ... - Ejemplo

Diseño

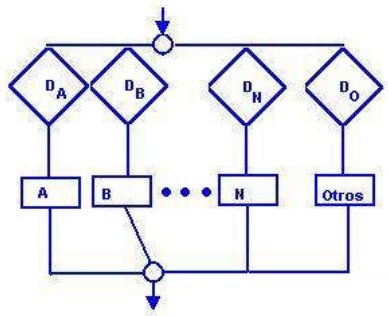
```
Algoritmo Edades
Lexico
  edad ∈ Z+ //var para almacenar la edad a analizar
  madurez resultado ∈ Cadena // variable para informar
Inicio
  Entrada:edad
  segun
    (0<=edad<=11): madurez ← "el ciudadano es niño"
    (12<=edad<=17):madurez ← "el ciudadano es adolescente"
    (18<=edad<=49):madurez ← "el ciudadano es adulto"
    (edad>=50):madurez ← "el ciudadano es adulto mayor"
  fsegun
  Salida:madurez
Fin
```

Composición Condicional

segun... otros

Sintaxis:

segun
<condicion$_1$>: <acciones$_1$>
<condicion$_2$>: <acciones$_2$>
<condicion$_1$>: <acciones$_1$>
⋮
<condicion$_n$>: <acciones$_n$>
otros: <acciones$_m$>
fsegun



Semántica: En su estructura es similar al anterior. Se utiliza cuando uno de los casos corresponde al complemento de la unión de los restantes. Si **ninguna condición** es **verdadera** si o si se ejecutan las **acciones$_m$**. Digamos que **otros** es una tautología, siempre es verdadera.
La evaluación de las condiciones es similar al según anterior, donde ***otros*** entra en juego solo si ninguna de las **n** condiciones fue verdadera.

Composición Condicional

segun ... - Ejemplo

Diseño

```
Algoritmo Edades
Lexico
  edad ∈ Z //var para almacenar la edad a analizar
  madurez resultado ∈ Cadena // variable para informar
Inicio
  Entrada:edad
  segun
    (0<=edad<=11): madurez ← "el ciudadano es niño"
    (12<=edad<=17):madurez ← "el ciudadano es adolescente"
    (18<=edad<=49):madurez ← "el ciudadano es adulto"
    (edad>=50):madurez ← "el ciudadano es adulto mayor"
  otros:madurez ← "edad incorrecta por ser negativa"
  fsegun
  Salida:madurez
Fin
```


Composición Condicional

segun ... otros - Ejemplo

Determinar si un número introducido por el usuario es mayor o igual que 50, menor o igual que 10 o está entre ambos.

Análisis

Datos: número es un número real (num)

Resultados: segmento

Relaciones o subproblemas: num>=50 segmento es mayor o igual que 50; num<=10 segmento es menor o igual que 10; 10<num<50 segmento está entre 10 y 50.

Composición Condicional

segun ... otros - Ejemplo

Diseño

```
Algoritmo EjemploOtros
Lexico
  num ∈ R //var para almacenar el número a analizar
  segmento ∈ Cadena // variable para informar
Inicio
  Entrada:num
  segun
    (num>=50):segmento ← "es mayor o igual que 50"
    (num<=10):segmento ← "es menor o igual que 10"
   otros:segmento ← "está entre 10 y 50"
  fsegun
  Salida:segmento
Fin
```

Composición Condicional

segun

- Es una de las estructuras condicionales más poderosas y utilizadas en la construcción de algoritmos.
- Permite expresar más claramente una solución condicional.
- Ahorra código.
- Hace más legibles los algoritmos.
- Posee un alto grado de abstracción.

Composición Condicional

segun

No existe una estructura condicional similar en los lenguajes de programación más conocidos.

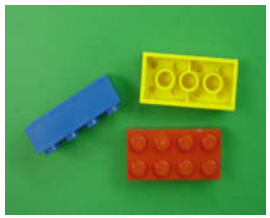
A la hora de implementar un algoritmo que utiliza el según debemos analizar como traducirlo a una estructura propia del lenguaje (generalmente similares al si ... entonces ... sino).

Atención!

El ***switch*** de C, y de la mayoría de los lenguajes, **no es equivalente** al ***segun***.

Composición Condicional

Equivalencias



Así como hay varias maneras de hacer una pared usando ladrillitos, también hay varias maneras de diseñar algoritmos. Entre los distintos tipos de **composiciones condicionales** hay ciertas **equivalencias** que posibilitan traducir unas en otras y viceversa.

Por ejemplo, es aconsejable usar el **segun** para construir la **primer versión** de un **algoritmo**, porque así es más legible, y luego ir acercándonos a las estructuras de los lenguajes de programación mediante el uso de equivalencias.

Composición Condicional

Equivalencias

```
si <condicion> entonces
  <acciones1>
sino
  <acciones2>
fsi
```

```
segun
  <condicion>:<acciones1>
  no <condicion>:<acciones2>
fsegun
```

Composición Condicional

Equivalencias - Ejemplo

```
Algoritmo EjemploSi
Lexico
  num ∈ R
  resultado ∈ Cadena
Inicio
 Entrada:num
 si num<0 entonces
    resultado← "es negat"
 sino
    resultado← "es posit"
 fsi
Fin
```

```
Algoritmo EjemploSegun
Lexico
  num ∈ R
  resultado ∈ Cadena
Inicio
  Entrada:num
  segun
   (num<0):resultado← "es negat"
   no(num<0):resultado← "es posit"
  fsegun
Fin
```


Composición Condicional

Equivalencias

```
según
  <condicion1>:<acciones1>
  <condicion2>:<acciones2>
  <condicion3>:<acciones3>
fsegún
```

Nota: solo si <condicion$_1$>, <condicion$_2$> y <condicion$_3$> son mutuamente excluyentes.

Esta equivalencia puede ser extendida a todos los casos que sean necesarios siguiendo la misma lógica.

```
si <condicion1> entonces
  <acciones1>
sino
  si <condicion2> entonces
    <acciones2>
  sino
    <acciones3>
  fsi
fsi
```

*A esta forma se la conoce como **si anidado***

Composición Condicional

Equivalencias

```
según
  <condicion1>:<acciones1>
  <condicion2>:<acciones2>
  <condicion3>:<acciones3>
  otros: <acciones4>
fsegún
```

```
si <condicion1> entonces
  <acciones1>
sino
  si <condicion2> entonces
    <acciones2>
  sino
    si <condicion3> entonces
      <acciones3>
    sino
      <acciones4>
    fsi
  fsi
fsi
```


Composición Condicional

Equivalencias - Ejemplo

```
Algoritmo EjemploSegun
Lexico
  num ∈ R //var para almacenar el número a analizar
  resultado ∈ Cadena
Inicio
  Entrada:num
  segun
    (num>=50):resultado← "es mayor o igual que 50"
    (num<=10):resultado← "es menor o igual que 10"
    (10<num<50):resultado← "está entre 10 y 50"
  fsegun
Fin
```

Composición Condicional
Equivalencias - Ejemplo

```
Algoritmo EjemploSiAnidado
Lexico
  num ∈ R //var para almacenar el número a analizar}
  resultado ∈ Cadena
Inicio
  Entrada:num
  si num>=50 entonces //num es mayor o igual que 50
    resultado ← "es mayor o igual que 50"
  sino //num es menor que 50
    si num<=10 entonces //num es menor que 50 y menor o igual que 10
      resultado ← "es menor o igual que 10"
    sino //num es menor que 50 y mayor que 10
      resultado ← "está entre 10 y 50"
    fsi
  fsi
Fin
```


Implementación en Lenguaje C
Algoritmo EjemploSiAnidado

```
#include <stdio.h> <string.h> //EjemploSiAnidado
/* Lexico */
int  num; //variable con el numero a analizar
char resultado[21]; //variable para informar
void main(){
  printf("Ingrese un numero entero: ");
  scanf("%d",&num);
  if (num>=50) { //num es mayor o igual que 50
    strcpy(resultado,"es mayor o igual que 50 \n");
  }
  else { //num es menor que 50
    if (num<=10) {
      //num es menor que 50 y menor o igual que 10
      strcpy(resultado,"es menor o igual que 10 \n");
    }
    else { //num es menor que 50 y mayor que 10
      strcpy(resultado,"esta entre 10 y 50 \n");
    }
  }
  printf("%d %s", num,resultado);
}
```


Composición Condicional
SEGUN vs. SWITCH de C

La estructura SEGUN de la notación algorítmica es mucho más general y poderosa que el ***switch*** de C. La ventaja del SEGUN es que permite encontrar soluciones simples, cortas y fáciles de leer.

El ***switch*** de C determina cual bloque de instrucciones va a ejecutar mediante una **única expresión denominada selector**. No permite colocar una expresión por bloque. Cada bloque se etiqueta con una constante.

```
switch (selector){
  case <constante 1>:
       <bloque de instrucciones>
       break;
  case <constante 2>:
       <bloque de instrucciones>
       break;
       .
       .
       .
  case <constante n>:
       <bloque de instrucciones>
       break;
  default:
       <bloque de instrucciones>
}
```


Composición Condicional
SEGUN vs. SWITCH

Reglas del switch:

- Una sentencia ***switch*** contiene un selector cuyo tipo debe ser int, char o enumerado (ordenal).
- Cuando una sentencia ***switch*** se ejecuta, el valor del selector se compara con las etiquetas ***case*** *(constantes)*.
- Si alguna de ellas concuerda con ese valor se ejecutará la correspondiente secuencia de sentencias.

Composición Condicional
SEGUN vs. SWITCH

Reglas del switch (cont.):

- La palabra reservada ***break*** permite que el flujo de programa se detenga justo despu´es de la ejecuci´on de la sentencia anterior a ese ***break***, impidiendo que se ejecuten las sentencias correspondientes a las siguientes alternativas del ***switch***. Por tanto, debemos obligatoriamente acabar cada bloque de sentencias correspondiente a cada alternativa con una sentencia ***break***.
- La alternativa ***default*** es opcional y engloba un conjunto de sentencias (que puede ser vacío, contener una sola sentencia o varias) que se ejecutan en caso de que ninguna de las alternativas del ***switch*** tenga un valor.

Composición Condicional
SEGUN vs. SWITCH

```
#include <stdio.h>
int dia;
void main(){
   printf( "Introduzca un numero de dia de la semana: " );
   scanf( "%d", &dia);
   switch (dia) {
      case 1 : printf( "\n Lunes" );
               break;
      case 2 : printf( "\n Martes" );
               break;
      case 3 : printf( "\n Miercoles" );
               break;
      case 4 : printf( "\n Jueves" );
               break;
      case 5 : printf( "\n Viernes" );
               break;
      case 6 : printf( "\n Sabado" );
               break;
      case 7 : printf( "\n Domingo" );
               break;
      default : printf( "\n  Día incorrecto." );
   }
}
```


Datos y direcciones útiles

Convenciones para escribir algoritmos y programas

En fotocopiadora del CECEx y en el sitio de la materia:
ConvencionesPseudocódigo.pdf y ConvencionesC.pdf

- **Introducción a la Algorítmica y Programación**

http://www.ucm.es/info/dsip/clavel/courses/ip0203/ip0203.html
http://www.algoritmica.com.ar/

Bibliografía

– Scholl, P. y Peyrin, J.-P. “Esquemas Algorítmicos Fundamentales: Secuencias e iteración”:
- Composición secuencial (pags. 35 - 55)
- Composición condicional (pags. 57 - 69)

– Biondi, J. y Clavel, G. “Introducción a la Programación. Tomo 1: Algorítmica y Lenguajes”:
- Composición condicional (35 - 53)

- Quetglás, Toledo, Cerverón. "Fundamentos de Informática y Programación“
  - http://robotica.uv.es/Libro/Indice.html
  - Capítulo 3 (91 - 98, 125 - 126)

- Diagramas de Flujo – Pseudocódigo
  - www.programacion.com/tutorial/jap_data_alg/
  - es.wikipedia.org/wiki/Pseudocódigo
  - www.desarrolloweb.com/articulos/2198.php
  - www.itver.edu.mx/comunidad/material/algoritmos/U2-22.htm
  - fcqi.tij.uabc.mx/docentes/mgarduno/Program1/Unidad1/u1_1.htm

- C.Böhm, G.Jacopini, Comm. ACM vol.9, nº5, 366-371,1966