Cap. 4 - Control de Programas

Esquema

4.1 Introducción

- Debería sentirse cómodo escribiendo programas simples de C

- En este capítulo
  - La repetición, en mayor detalle
  - Operadores lógicos para combinar condiciones
  - Principios de la programación estructurada

4.2 Lo esencial de la Repetición

- Bucle
  - Grupo de instrucciones que la computadora ejecuta repetidamente mientras alguna condición sigue siendo cierta
- Repetición Controlado por Contador
  - Repetición definida - se sabe cuántas veces se ejecutará el bucle
  - Variable de control utilizada para contar las repeticiones
- Repetición controlada por Centinela
  - Repetición Indefinida
  - Se utiliza cuando se desconoce el número de repeticiones
  - El valor del centinela indica "fin de los datos".

4.3 Esencia de la Repetición Controlada por Contador

- La repetición controlada por contador requiere
  - *nombre* de una variable de control (o contador de bucle).
  - *valor inicial* de la variable de control.
  - condición que testa para el *valor final* de la variable de control (esto es, si el bucle debe continuar).
  - *incrementar* (o *decrementar*) por el cual la variable de control se modifica cada vez a través del bucle.

4.3 Esencia de la Repetición Controlada por Contador (II)

- Ejemplo:

```
int counter =1;                 //inicialización
while (counter <= 10){          //condición de repetición
   printf( "%d\n", counter );
   ++counter;                   //incremento
}
```

- `int counter = 1;`
  - nombra `counter`,
  - declara como un número entero,
  - reserva espacio para él en la memoria,
  - y establece a un valor inicial de `1`

4.4 La estructura de Repetición para (`for`)

- El formato cuando se utiliza bucle **`Para`** (**`for`**)

  `for (`*inicialización*`;`*Test de verificación de bucle*`;` *incremento*`) {`

  declaraciones

  `}`

Ejemplo:

```
for( int counter = 1; counter <= 10; counter++){
    printf( "%d\n", counter );
}
```

  – Imprime los enteros de 1 a 10.

4.4 La estructura de Repetición para (`for`) (II)

- Bucle **`for`** puede usualmente ser re escrito como bucle **`while`**:

inicialización`;`

`while` **`(`**Prueba de continuación de bucle**`) {`**
    declaración;
    incremento;
`}`

- Inicialización e incremento
  - Pueden ser listas separadas por comas

```
for (int i = 0, j = 0;  j + i <= 10; j++, i++){
   printf( "%d\n", j + i );
}
```


4.5 La estructura "For": Notas y observaciones

- Expresiones Aritméticas
  - La inicialización, la continuidad del bucle y el incremento pueden contener expresiones aritméticas.

Si **`x = 2`** e **`y = 10`**

```
for ( j = x; j <= 4 * x * y; j += y / x )
```

es equivalente a

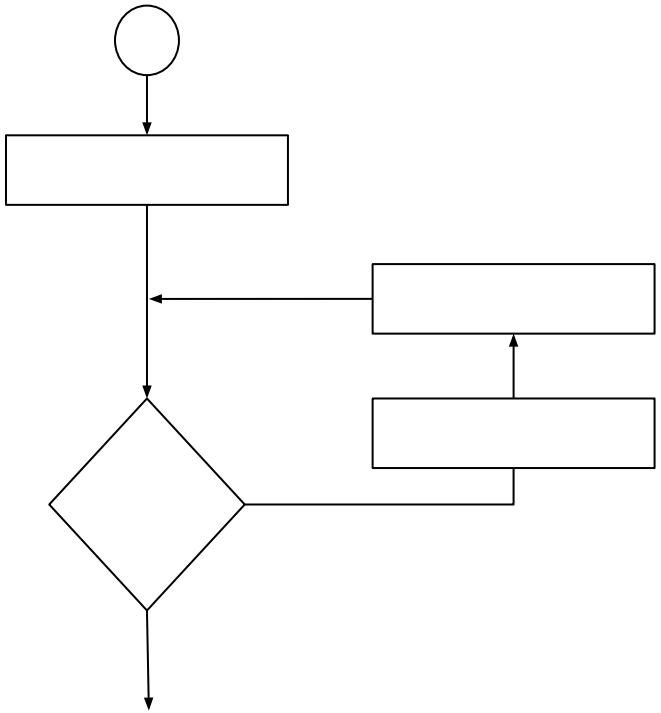
```
for ( j = 2; j <= 80; j += 5 )
```

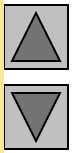
- "Incremento" puede ser negativo (decremento)
- Si *la prueba de continuidad de bucle* inicialmente es **`falso`**
  - El cuerpo de la estructura **`for`** no se realiza
  - El control procede con la declaración después de la estructura **`for`**

false

4.5 La estructura "For": Notas y observaciones (II)

- Variable de Control
  - A menudo se imprime o se utiliza en el interior del cuerpo de **`for`**, pero no necesariamente
- El diagrama de flujo de **`For`** es similar al del **`while`**





Outline

4.6 Ejemplo de Uso de la estructura `for`

Programa para sumar los números pares del 2 al 100

```
1   /* Fig. 4.5: fig04_05.c
2      Sumatoria con for */
3   #include <stdio.h>
4
5   int main()
6   {
7      int sum = 0, number;
8
9      for ( number = 2; number <= 100; number += 2 )
10        sum += number;
11
12     printf( "Suma es %d\n", sum );
13
14     return 0;
15  }
```

Salida de Programa

```
Suma es 2550
```

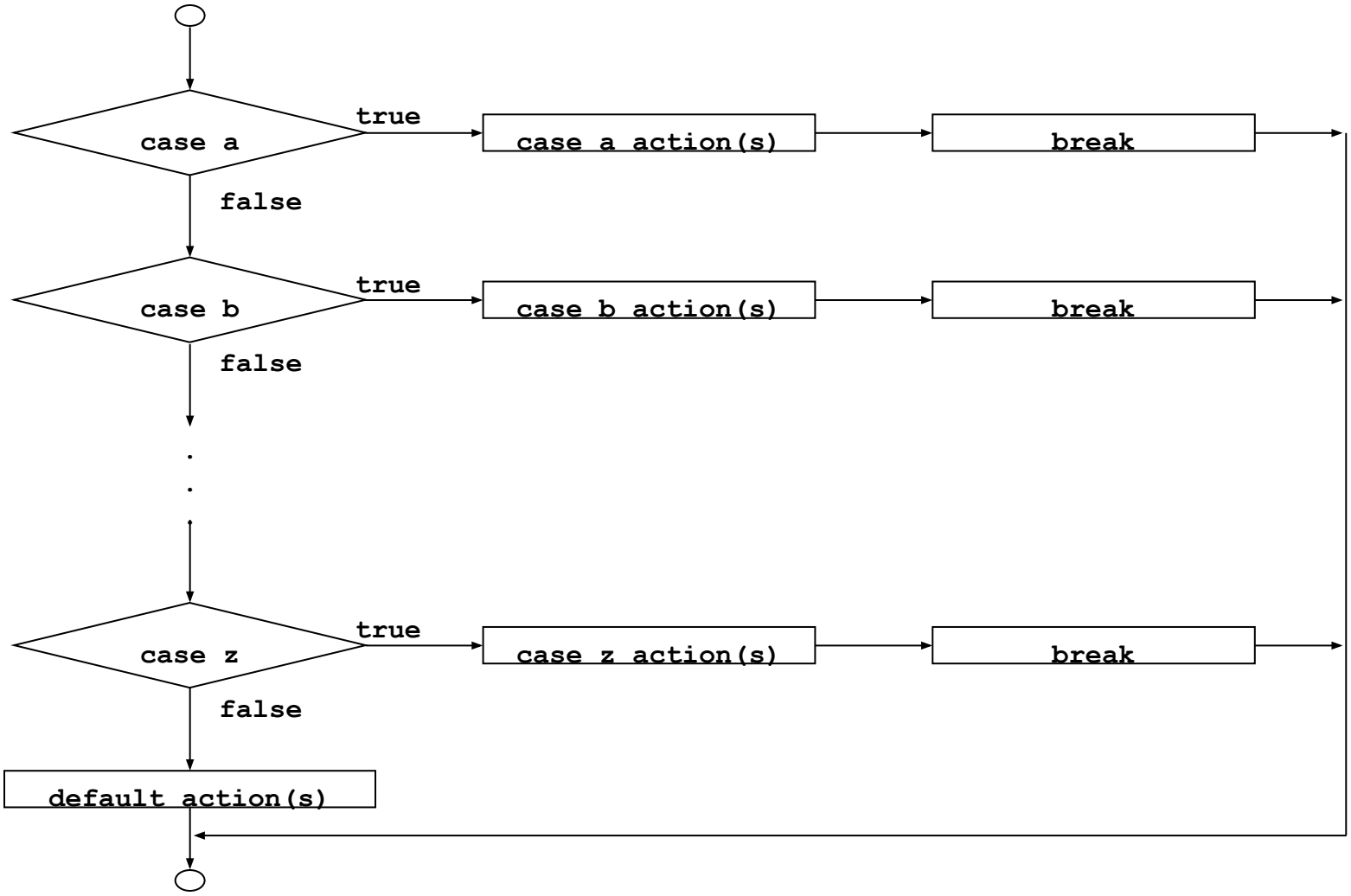
4.7 La estructura de selección múltiple `switch`

- **`switch`**
  - Es útil cuando una variable o expresión se prueba para todos los valores que puede asumir y se toman diferentes acciones.
- Formato
  - Series de etiquetas case (**`case`**) y un caso optional **`default`**

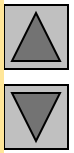
```
switch ( value ){
    case '1':
        actions;
        break;
    case '2':
        actions;
        break;
    default:
        actions;
    }
```

  - **`break;`** causes exit from structure

4.7 La estructura de selección múltiple `switch` (II)



```c
/* Fig. 4.7: fig04_07.c
   Contar las notas de las letras */
#include <stdio.h>

int main()
{
   int grade;
   int aCount = 0, bCount = 0, cCount = 0,
       dCount = 0, fCount = 0;

   printf(  "Introduce las calificaciones.\n"  );
   printf(  "Ingrese el caracter EOF para terminar entrada.\n"

   while ( ( grade = getchar() ) != EOF ) {

      switch ( grade ) {    /* switch anidado en mientras */

         case 'A': case 'a':  /* la nota era la A mayúscula */
            ++aCount;         /* o a minuscula */
            break;

         case 'B': case 'b':  /* la nota era la B mayúscula */
            ++bCount;         /* o b minuscula */
            break;

         case 'C': case 'c':  /* la nota era la C mayúscula*/
            ++cCount;         /* o c minuscula */
            break;

         case 'D': case 'd':  /* la nota era la D mayúscula */
            ++dCount;         /* o d minuscula */
            break;
```



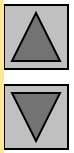
Outline

1. Variables Inicializadas

2. Entrada de Datos

2.1 Use el bucle del interruptor para actualizar el recuento

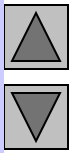
```
33
34            case 'F': case 'f':  /* la nota era la F mayúscula */
35               ++fCount;         /* o minuscula f */
36               break;
37
38            case '\n': case' ':  /* ignora en la entrada */
39               break;
40
41            default:     /* atrapa todos los otros caracteres */
42               printf( "Incorrecta letra-nota ingresada." );
43               printf( " Ingrese una nueva nota.\n" );
44               break;
45         }
46      }
47
48      printf( "\nTotal para cada nota son:\n" );
49      printf( "A: %d\n", aCount );
50      printf( "B: %d\n", bCount );
51      printf( "C: %d\n", cCount );
52      printf( "D: %d\n", dCount );
53      printf( "F: %d\n", fCount );
54
55      return 0;
56   }
```



Outline

2.1 Use el bucle del interruptor para actualizar el recuento

3. Imprime resultados



Outline

Salida del Programa

```
Ingrese la nota.
Ingrese el caracter EOF para finalizar entrada.
A
B
C
C
A
D
F
C
E
Incorrecta letra-nota ingresad. Ingrese nueva nota.
D
A
B

Total para cada letra-nota es:
A: 3
B: 2
C: 3
D: 2
F: 1
```

4.8 La estructura de repetición hacer/mientras (`do/while`)

- La estructura de repetición **`do/while`**
  - Similar a la estructura **`while`**
  - Condition para repetición se testa *after* del cuerpo bucle es realizado
    - Todas las acciones se realizan al menos una vez
- Format:

```
do {
      statement
} while ( condition );
```

  - Una buena práctica es poner los paréntesis, aunque no sea necesario.

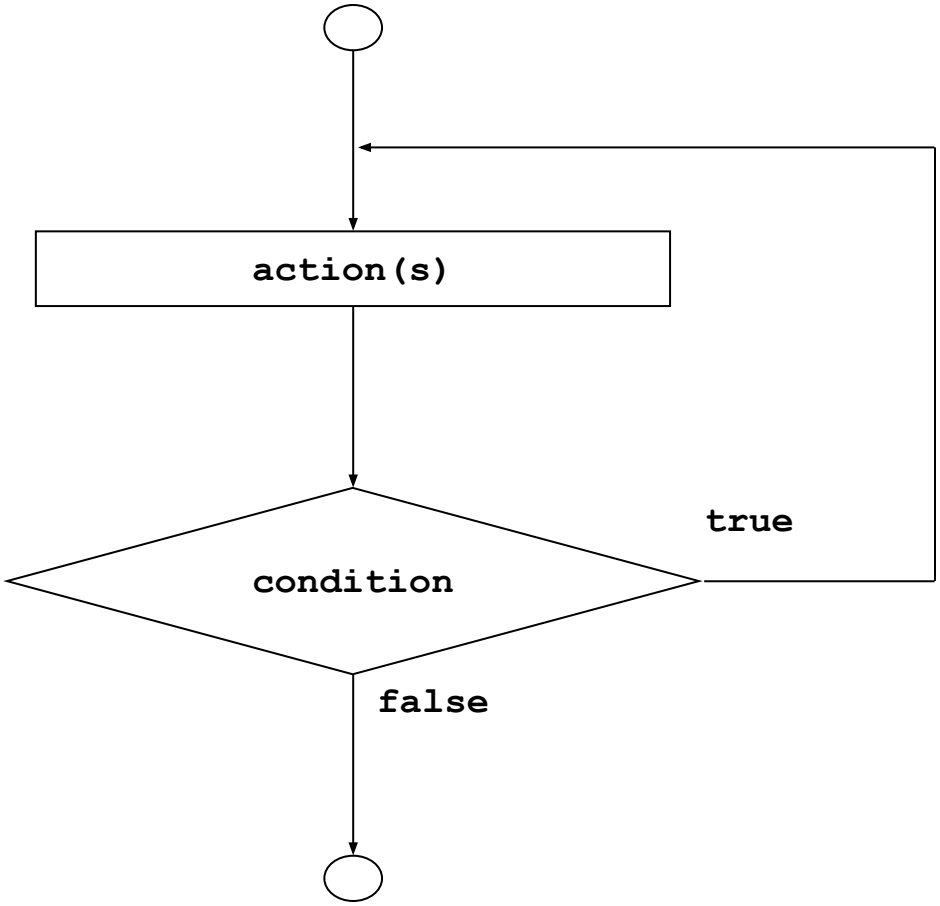
4.8 La estructura de repetición hacer/mientras (`do/while`) (II)

- Ejemplo (dejando **`counter = 1`**)

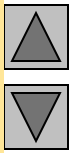
```
do {
   printf( "%d  ", counter );
} while (++counter <= 10);
```

Imprime los enteros desde 1 to 10.

4.8 La estructura de repetición hacer/mientras (`do/while`) (III)



```c
/* Fig. 4.9: fig04_09.c
   Usando la estructura de repetición do/while  */
#include <stdio.h>

int main()
{
   int counter = 1;

   do {
      printf( "%d  ", counter );
   } while ( ++counter <= 10 );

   return 0;
}
```



Outline

1. Inicialización de variables

2. Bucle

3. Impresión

Salida del Programa

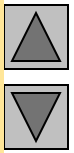
```
1  2  3  4  5  6  7  8  9  10
```

4.9 El quiebre y la continuidad

- **`quiebre (break)`**
  - Causa inmediata salida de una estructura **`while`**, **`for`**, **`do`**/**`while`** o **`switch`**
  - La ejecución del programa continúa con la primera declaración después de la estructura
  - Usos comunes de la declaración del **`break`**
    - Escapar temprano de un bucle
    - Saltar los sobrante en la estructura **`switch`**

4.9 El quiebre y la continuidad (II)

- **`continuar (continue)`**
  - Se salta el resto de las declaraciones en el cuerpo de una estructura **`while`**, **`for`** o **`do/while`**
    - Procede con la siguiente iteración del bucle
  - **`while`** y **`do/while`**
    - La prueba de continuidad del bucle se evalúa inmediatamente después de la ejecución de la declaración de continuidad
  - **`estructura for`**
    - Se ejecuta la expresión de incremento, luego se evalúa la prueba de continuidad de bucle



Outline

```
1    /* Fig. 4.12: fig04_12.c
2Usando la declaración de continuar en una estructura de for */
3    #include <stdio.h>
4
5    int main()
6    {
7       int x;
8
9       for ( x = 1; x <= 10; x++ ) {
10
11         if ( x == 5 )
12            continue;  /* saltar el código restante sólo en el
13                          si x == 5 */
14
15         printf( "%d ", x );
16      }
17
18      printf( "\nUsa continue para saltar la impresión de 5\n" );
19      return 0;
20   }
```

1. Inicializa variables

2. Bucle

3. Impime

Salida del Programa

```
1 2 3 4 6 7 8 9 10
Usa continue para saltar la impresión de 5
```

4.10 Operadores Logicos

- **`&&`** (AND lógico)
  - Retorna verdadero (**`true`**) si ambas condiciones son verdadera (**`true`**)
- **`||`** (OR lógico)
  - Retorna verdadero si cualquiera de sus condiciones es verdadero
- **`!`** (NOT lógico, negación lógica)
  - Invierte la verdad/falsedad de su condición
  - Operador único, tiene un operando
- Útiles como condiciones en los bucles

Expresión	Resultado
`true && false`	`false`
`true || false`	`true`
`!false`	`true`
`!true`	`false`

4.11 Confusión de operadores de igualdad (==) y asignación (=)

- Peligroso error
  - No suele causar errores de sintaxis
  - Cualquier expresión que produzca un valor puede ser utilizada en las estructuras de control
  - Los valores que no son cero son verdaderos, los valores cero son falsos
- Ejemplo:

```
if ( payCode == 4 )
   printf( "You get a bonus!\n" );
```

Checa paycode, si este es 4 entonces un bono es ganado.

4.11 Confusión de operadores de igualdad (==) y asignación (=) (II)

- Ejemplo, replazando **==** con **=**:

```
if ( payCode = 4 )
   printf( "You get a bonus!\n" );
```

  - Este establece **paycode** a 4
  - 4 no es cero, entonces la expresión es **true**, y el bono es ganado sin importar cuál era el código de pago
  - Un error lógico, no un error de sintaxis

4.11 Confusión de operadores de igualdad (==) y asignación (=) (III)

- lvalues
  - Las expresiones que pueden aparecer en el lado izquierdo de una ecuación
  - Sus valores pueden ser cambiados, como los nombres de las variables
    - `x = 4;`
- rvalues
  - Expresiones que sólo pueden aparecer en el lado derecho de una ecuación
  - Las constantes, como los números
    - No se puede escribir `4 = x;`
  - lvalues puede ser usadas como rvalues, pero viceversa no
    - `y = x;`

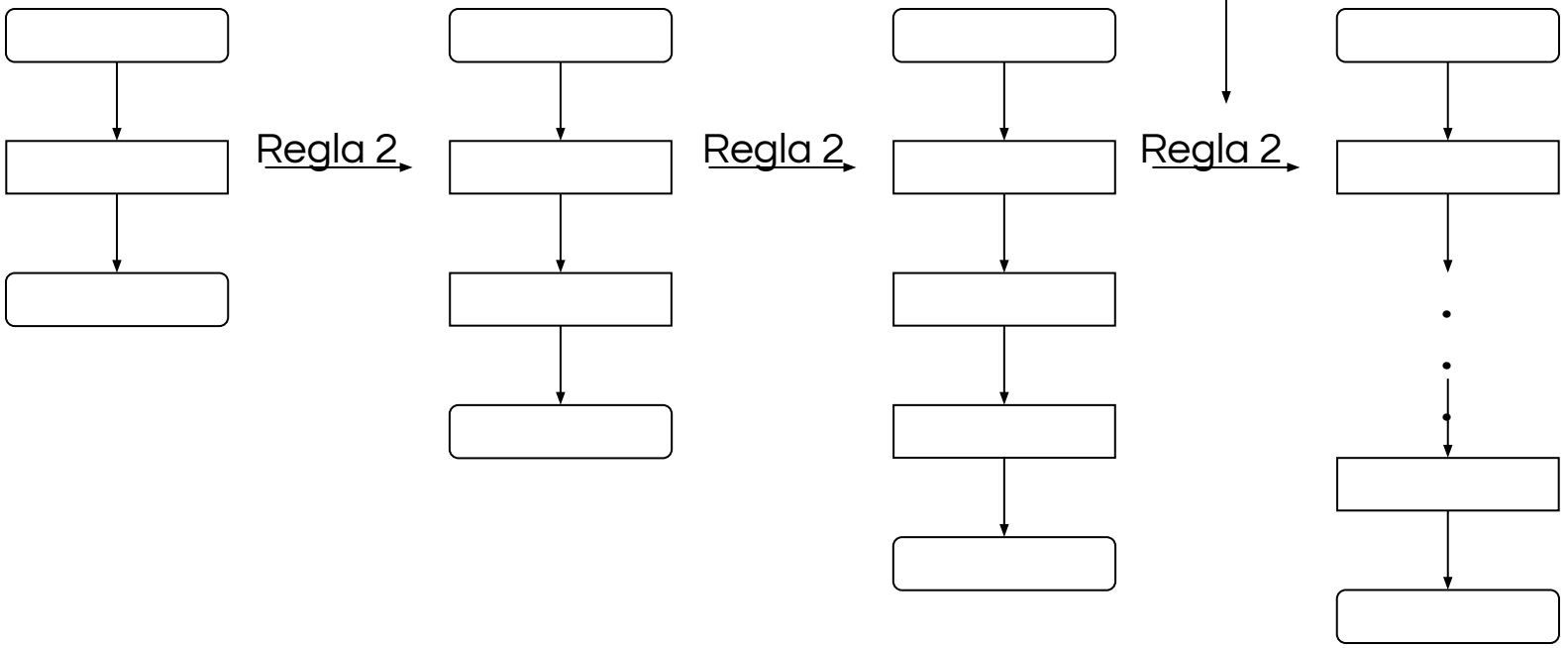
4.12 Resumen de Programación Estructurada

- Programación Estructurada
  - Más fácil que los programas no estructurados para entender, probar, depurar y modificar programas
- Reglas para la programación estructurada
  - Reglas desarrolladas por la comunidad de programación
  - Sólo se utilizan estructuras de control de una sola entrada/salida
  - Reglas:
    1) Iniciar con el más simple "diagrama de flujos"
    2) Cualquier rectángulo (acción) puede ser reemplazada por dos rectángulos (acciones) en secuencia.
    3) Cualquier rectángulo (acción) puede ser reemplazado por cualquier estructura de control (secuencia, **`if`**, **`if/else`**, **`switch`**, **`while`**, **`do/while`** o **`for`**).
    4) Reglas 2 y 3 pueden ser aplicadas en cualquier orden y múltiples veces.

4.12 Resumen de Programación Estructurada (II)

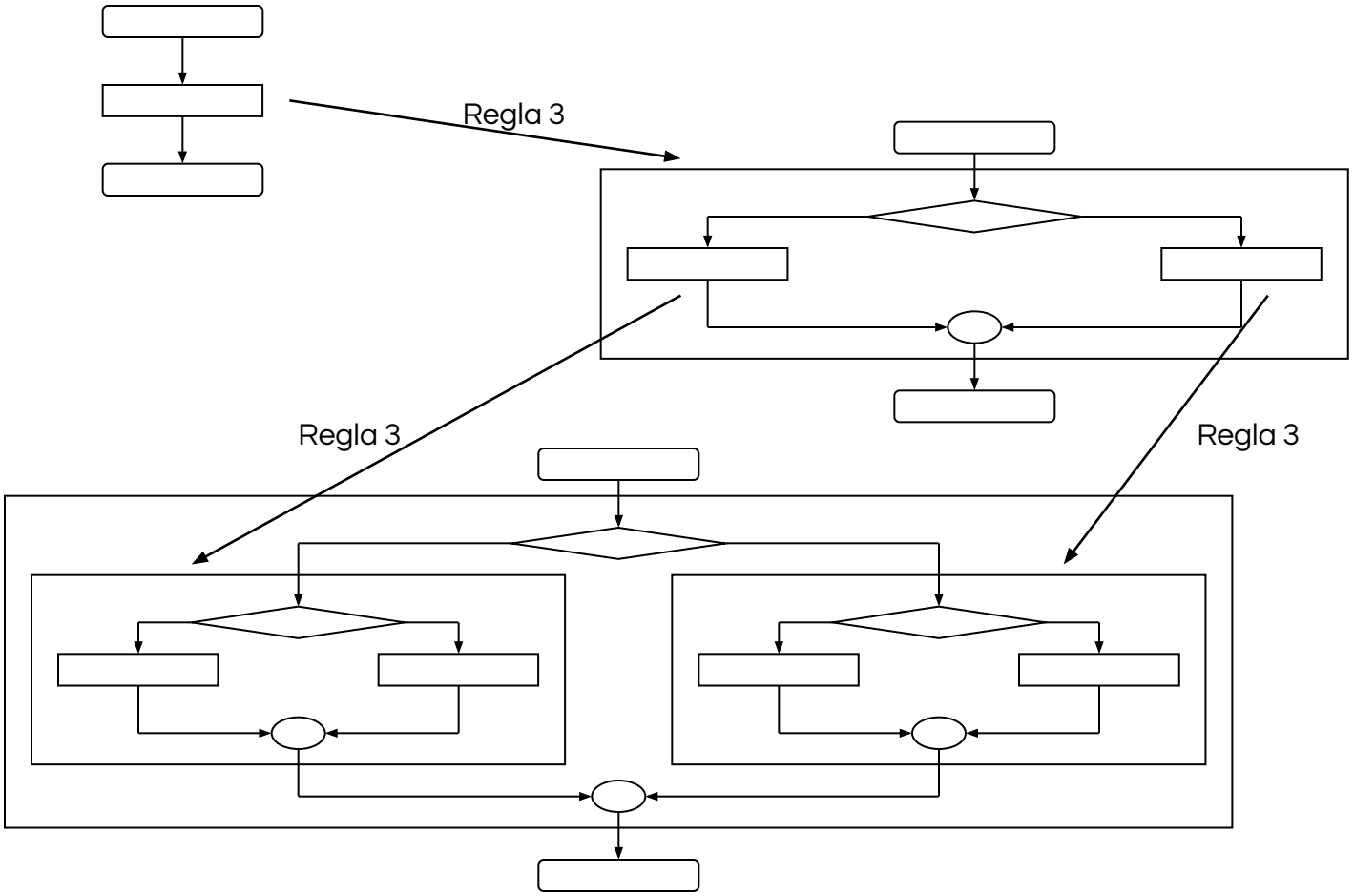
Regla 1 - Inicial con el más simple diagrama de flujo

Regla 2 - Cualquier rectángulo puede ser reemplazado por dos rectángulos en secuencia



4.12 Resumen de Programación Estructurada (III)

Regla 3 - Reemplazar cualquier rectángulo con una estructura de control



4.12 Resumen de Programación Estructurada (IV)

- Todos los programas se pueden dividir en 3 partes

  Secuencial - trivial

  Selección - **`if`**, **`if/else`**, o **`switch`**

  Repetición - **`while`**, **`do/while`**, o **`for`**

  – Cualquier selección puede ser reescrita como una declaración "if", y cualquier repetición puede ser reescrita como una declaración "while".

- Los programas se reducen a
  – Secuencia
  – Estructura **`if`** (selección)
  – Estructura **`while`** (repetición)
    - Las estructuras de control sólo pueden combinarse de dos maneras: anidamiento (regla 3) y apilamiento (regla 2)
  – Esto promueve la simplicidad

Cap. 4 - Control de Programas