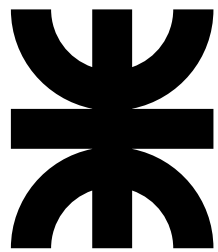
Informática II

Repaso del lenguaje C

Gonzalo F. Pérez Paina



Universidad Tecnológica Nacional
Facultad Regional Córdoba
UTN-FRC

– Marzo 2016 –

Contenido

1. Introducción
   - Un poco de historia
   - Algunas características
   - Algunos inconvenientes
2. Introducción al lenguaje C
3. Programación estructurada
4. Estructuras de selección
5. Estructura de repetición
6. Funciones (para más adelante)

Introducción

Un poco de historia

- Ideas provenientes de los lenguajes BCPL[1] (*Martin Richard, 1967*) y del lenguaje B (*Ken Thompson, 1970*) [lenguajes "sin tipo"].

[1] Basic Combined Programming Language

Introducción

Un poco de historia

- Ideas provenientes de los lenguajes BCPL[1] (*Martin Richard, 1967*) y del lenguaje B (*Ken Thompson, 1970*) [lenguajes "sin tipo"].
- C fue diseñado originalmente en 1972 para el sistema operativo UNIX en el DEC PDP-11 por *Dennis Ritchie* en los laboratorios Bell.

[1] Basic Combined Programming Language

Introducción

Un poco de historia

- Ideas provenientes de los lenguajes BCPL[1] (*Martin Richard, 1967*) y del lenguaje B (*Ken Thompson, 1970*) [lenguajes "sin tipo"].
- C fue diseñado originalmente en 1972 para el sistema operativo UNIX en el DEC PDP-11 por *Dennis Ritchie* en los laboratorios Bell.
- El primer libro de referencia fue *C Programming Language* (1978) de Brian Kernighan y Dennis Ritchie.

[1] Basic Combined Programming Language

Introducción

Un poco de historia

- Ideas provenientes de los lenguajes BCPL[1] (*Martin Richard, 1967*) y del lenguaje B (*Ken Thompson, 1970*) [lenguajes "sin tipo"].
- C fue diseñado originalmente en 1972 para el sistema operativo UNIX en el DEC PDP-11 por *Dennis Ritchie* en los laboratorios Bell.
- El primer libro de referencia fue *C Programming Language* (1978) de Brian Kernighan y Dennis Ritchie.
- En 1989 aparece el estándar ANSI C (ANSI[2]).

[1] Basic Combined Programming Language
[2] American National Standards Institute

Introducción

Un poco de historia

- Ideas provenientes de los lenguajes BCPL[1] (*Martin Richard, 1967*) y del lenguaje B (*Ken Thompson, 1970*) [lenguajes "sin tipo"].
- C fue diseñado originalmente en 1972 para el sistema operativo UNIX en el DEC PDP-11 por *Dennis Ritchie* en los laboratorios Bell.
- El primer libro de referencia fue *C Programming Language* (1978) de Brian Kernighan y Dennis Ritchie.
- En 1989 aparece el estándar ANSI C (ANSI[2]).
- En 1990 aparece el estándar ISO C (ISO[3]).

[1] Basic Combined Programming Language
[2] American National Standards Institute
[3] International Standards Organization

Introducción

Un poco de historia

- Ideas provenientes de los lenguajes BCPL[1] (*Martin Richard, 1967*) y del lenguaje B (*Ken Thompson, 1970*) [lenguajes "sin tipo"].
- C fue diseñado originalmente en 1972 para el sistema operativo UNIX en el DEC PDP-11 por *Dennis Ritchie* en los laboratorios Bell.
- El primer libro de referencia fue *C Programming Language* (1978) de Brian Kernighan y Dennis Ritchie.
- En 1989 aparece el estándar ANSI C (ANSI[2]).
- En 1990 aparece el estándar ISO C (ISO[3]).
- En 1999 aparece el estándar C99. El último estándar publicado de C es el C11 (ISO/IEC 9899:2011) (IEC[4]).

[1] Basic Combined Programming Language
[2] American National Standards Institute
[3] International Standards Organization
[4] International Electrotechnical Commission

- Lenguaje de propósitos generales ampliamente utilizado.

- Lenguaje de propósitos generales ampliamente utilizado.
- Tiene características de lenguajes de **bajo nivel**

- Lenguaje de propósitos generales ampliamente utilizado.
- Tiene características de lenguajes de **bajo nivel**
- Lenguaje relativamente pequeño: ofrece sentencias de control sencillas y funciones.

- Lenguaje de propósitos generales ampliamente utilizado.
- Tiene características de lenguajes de **bajo nivel**
- Lenguaje relativamente pequeño: ofrece sentencias de control sencillas y funciones.
- Permite **programación estructurada** y diseño modular.

- Lenguaje de propósitos generales ampliamente utilizado.
- Tiene características de lenguajes de **bajo nivel**
- Lenguaje relativamente pequeño: ofrece sentencias de control sencillas y funciones.
- Permite **programación estructurada** y diseño modular.
- Si los programas siguen el estándar ISO el código es portátil entre plataformas y/o arquitecturas.

- No es un lenguaje **fuertemente tipado**. (ventaja o desventaja?)

[5] IOCCC: The International Obfuscated C Code Contest

- No es un lenguaje **fuertemente tipado**. (ventaja o desventaja?)
- Bastante permisivo con la conversión de datos.

[5] IOCCC: The International Obfuscated C Code Contest

- No es un lenguaje **fuertemente tipado**. (ventaja o desventaja?)
- Bastante permisivo con la conversión de datos.
- Su versatilidad permite crear programas difíciles de leer (código ofuscado)[5].

[5] IOCCC: The International Obfuscated C Code Contest

Contenido

1. Introducción
2. Introducción al lenguaje C
   - Programas de ejemplo
   - Variables
   - Operadores
3. Programación estructurada
4. Estructuras de selección
5. Estructura de repetición
6. Funciones (para más adelante)

Programas en C

Los programas C consisten de módulos o piezas que se denominan funciones.
Esto facilita evitar volver a inventar la rueda: *reutilización de software*.

Programas en C

Los programas C consisten de módulos o piezas que se denominan funciones.
Esto facilita evitar volver a inventar la rueda: *reutilización de software*.

Aprender a programar "C"

Consta de dos partes:

- Lenguaje C en sí mismo
- Funciones de la biblioteca estándar C.

Programas en C

Los programas C consisten de módulos o piezas que se denominan funciones.
Esto facilita evitar volver a inventar la rueda: *reutilización de software*.

Aprender a programar "C"

Consta de dos partes:

- Lenguaje C en sí mismo
- Funciones de la biblioteca estándar C.

Todos los sistemas C consisten, en general, en tres partes:

- el entorno
- el lenguaje y
- la biblioteca estándar C

Programas en C

Los programas C consisten de módulos o piezas que se denominan funciones.
Esto facilita evitar volver a inventar la rueda: *reutilización de software*.

Aprender a programar "C"

Consta de dos partes:

- Lenguaje C en sí mismo
- Funciones de la biblioteca estándar C.

Todos los sistemas C consisten, en general, en tres partes:

- el entorno
- el lenguaje y
- la biblioteca estándar C

Los programas C casi siempre pasan a través de **seis fases** para su ejecución: *editar*, *preprocesar*, *compilar*, *enlazar*, *cargar*, *ejecutar*.

Introducción al lenguaje C

Programas de ejemplo

```
1 /* Primer programa en C */
2 #include <stdio.h>
3
4 main()
5 {
6   printf("Hola␣mundo.\n");
7 }
```

Introducción al lenguaje C

Programas de ejemplo

```
1 /* Primer programa en C */
2 #include <stdio.h>
3
4 main()
5 {
6   printf("Hola mundo.\n");
7 }
```

```
Hola mundo.
```

- Comentarios (¿para qué?)

```
1 /* Primer programa en C */
2 #include <stdio.h>
3
4 main()
5 {
6   printf("Hola mundo.\n");
7 }
```

```
Hola mundo.
```

Introducción al lenguaje C

Programas de ejemplo

```
1 /* Primer programa en C */
2 #include <stdio.h>
3
4 main()
5 {
6   printf("Hola mundo.\n");
7 }
```

```
Hola mundo.
```

- Comentarios (¿para qué?)
- Directriz del preprocesador (#)

Introducción al lenguaje C

Programas de ejemplo

```c
/* Primer programa en C */
#include <stdio.h>

main()
{
   printf("Hola mundo.\n");
}
```

```
Hola mundo.
```

- Comentarios (¿para qué?)
- Directriz del preprocesador (#)
- Archivos de cabecera/header (.h)

Introducción al lenguaje C

Programas de ejemplo

```c
/* Primer programa en C */
#include <stdio.h>

main()
{
   printf("Hola mundo.\n");
}
```

```
Hola mundo.
```

- Comentarios (¿para qué?)
- Directriz del preprocesador (#)
- Archivos de cabecera/header (.h)
- Librería estándar

Introducción al lenguaje C

Programas de ejemplo

```c
/* Primer programa en C */
#include <stdio.h>

main()
{
   printf("Hola mundo.\n");
}
```

```
Hola mundo.
```

- Comentarios (¿para qué?)
- Directriz del preprocesador (`#`)
- Archivos de cabecera/header (.h)
- Librería estándar
- `stdin`, `stdout`

Introducción al lenguaje C

Programas de ejemplo

```c
/* Primer programa en C */
#include <stdio.h>

main()
{
   printf("Hola mundo.\n");
}
```

```
Hola mundo.
```

- Comentarios (¿para qué?)
- Directriz del preprocesador (`#`)
- Archivos de cabecera/header (.h)
- Librería estándar
- `stdin`, `stdout`
- Función main (paréntesis)

Introducción al lenguaje C

Programas de ejemplo

```c
/* Primer programa en C */
#include <stdio.h>

main()
{
   printf("Hola mundo.\n");
}
```

```
Hola mundo.
```

- Comentarios (¿para qué?)
- Directriz del preprocesador (**#**)
- Archivos de cabecera/header (.h)
- Librería estándar
- **stdin**, **stdout**
- Función main (paréntesis)
- Bloque (llaves)- Cuerpo de la función

Introducción al lenguaje C

Programas de ejemplo

```c
/* Primer programa en C */
#include <stdio.h>

main()
{
   printf("Hola mundo.\n");
}
```

```
Hola mundo.
```

- Comentarios (¿para qué?)
- Directriz del preprocesador (**#**)
- Archivos de cabecera/header (.h)
- Librería estándar
- **stdin**, **stdout**
- Función main (paréntesis)
- Bloque (llaves)- Cuerpo de la función
- Parámetros y valor de devolución

Introducción al lenguaje C

Programas de ejemplo

```c
/* Primer programa en C */
#include <stdio.h>

main()
{
   printf("Hola mundo.\n");
}
```

```
Hola mundo.
```

- Comentarios (¿para qué?)
- Directriz del preprocesador (**#**)
- Archivos de cabecera/header (.h)
- Librería estándar
- **stdin**, **stdout**
- Función main (paréntesis)
- Bloque (llaves)- Cuerpo de la función
- Parámetros y valor de devolución
- Enunciados (finaliza con ';')

Introducción al lenguaje C

Programas de ejemplo

```
1 /* Primer programa en C */
2 #include <stdio.h>
3
4 main()
5 {
6   printf("Hola mundo.\n");
7 }
```

```
Hola mundo.
```

- Comentarios (¿para qué?)
- Directriz del preprocesador (**#**)
- Archivos de cabecera/header (.h)
- Librería estándar
- **stdin**, **stdout**
- Función main (paréntesis)
- Bloque (llaves)- Cuerpo de la función
- Parámetros y valor de devolución
- Enunciados (finaliza con ';')
- Caracter de escape ('\')

Introducción al lenguaje C

Programas de ejemplo

```c
/* Primer programa en C */
#include <stdio.h>

main()
{
   printf("Hola mundo.\n");
}
```

```
Hola mundo.
```

- Comentarios (¿para qué?)
- Directriz del preprocesador (**#**)
- Archivos de cabecera/header (.h)
- Librería estándar
- **stdin**, **stdout**
- Función main (paréntesis)
- Bloque (llaves)- Cuerpo de la función
- Parámetros y valor de devolución
- Enunciados (finaliza con ';')
- Caracter de escape ('\')
- Secuencia de escape ('\n')

```c
/* Primer programa en C con
   comentario de dos líneas :) */
#include <stdio.h>

int main(void)
{
  printf("Hola mundo.\n");
  return 0;
}
```

```
Hola mundo.
```

Introducción al lenguaje C

Programas de ejemplo

```c
/* Suma de dos números enteros */
#include <stdio.h>

main()
{
  /* Declaración de variables */
  int entero1, entero2, suma;

  printf("Ingrese el primer entero: ");
  scanf("%d", &entero1);
  printf("Ingrese el segundo entero: ");
  scanf("%d", &entero2);

  /* Asignación de la variable suma */
  suma = entero1 + entero2;
  printf("La suma es: %d\n", suma);

  return 0;      /* finaliza sin error */
}
```

```c
/* Suma de dos números enteros */
#include <stdio.h>

main()
{
  /* Declaración de variables */
  int entero1, entero2, suma;

  printf("Ingrese el primer entero: ");
  scanf("%d", &entero1);
  printf("Ingrese el segundo entero: ");
  scanf("%d", &entero2);

  /* Asignación de la variable suma */
  suma = entero1 + entero2;
  printf("La suma es: %d\n", suma);

  return 0;      /* finaliza sin error */
}
```

```
Ingrese el primer entero: 34
Ingrese el segundo entero: 67
La suma es: 101
```

```c
/* Suma de dos números enteros */
#include <stdio.h>

main()
{
  /* Declaración de variables */
  int entero1, entero2, suma;

  printf("Ingrese el primer entero: ");
  scanf("%d", &entero1);
  printf("Ingrese el segundo entero: ");
  scanf("%d", &entero2);

  /* Asignación de la variable suma */
  suma = entero1 + entero2;
  printf("La suma es: %d\n", suma);

  return 0;      /* finaliza sin error */
}
```

- Declaración de variables (¿Dónde?, ¿Cuáles son los tipos de datos?)
- Primer argumento de **printf**: cadena de control de formato
- %d: especificador de conversión
- **scanf**: cadena de control de formato

Introducción al lenguaje C

Programas de ejemplo

```c
/* Suma de dos números enteros */
#include <stdio.h>

main()
{
  /* Declaración de variables */
  int entero1, entero2;

  printf("Ingrese el primer entero: ");
  scanf("%d", &entero1);
  printf("Ingrese el segundo entero: ");
  scanf("%d", &entero2);

  /* Imprime el resultado */
  printf("La suma es: %d\n", entero1 + entero2);

  return 0;      /* finaliza sin error */
}
```

Variables

Cada variable tiene un *tipo*, un *nombre*, y un *valor*

Variables

Cada variable tiene un *tipo*, un *nombre*, y un *valor*

Nombre de variables

Un nombre de variable en C es cualquier identificador válido.

Un identificador es una serie de caracteres formados de letras, dígitos y subrayados (_) que no se inicie con un dígito.

C es sensible a las minúsculas y mayúsculas.

Variables

Cada variable tiene un *tipo*, un *nombre*, y un *valor*

Nombre de variables

Un nombre de variable en C es cualquier identificador válido.

Un identificador es una serie de caracteres formados de letras, dígitos y subrayados (_) que no se inicie con un dígito.

C es sensible a las minúsculas y mayúsculas.

Los nombres de variables significativos ayudan a autodocumentar el código.

Operadores aritméticos (binarios)

`+`, `-`, `/`, `*`, `=`, `%` (módulo - solo con operandos enteros)

Operadores aritméticos (binarios)

`+`, `-`, `/`, `*`, `=`, `%` (módulo - solo con operandos enteros)

Algunos comentarios:

- ¿Qué pasa con la división de enteros? (ej. 17/5, y 17 %5)

Operadores aritméticos (binarios)

`+`, `-`, `/`, `*`, `=`, `%` (módulo - solo con operandos enteros)

Algunos comentarios:

- ¿Qué pasa con la división de enteros? (ej. 17/5, y 17 %5)
- Precedencia de operadores

Operadores aritméticos (binarios)

`+, -, /, *, =, %` (módulo - solo con operandos enteros)

Algunos comentarios:

- ¿Qué pasa con la división de enteros? (ej. 17/5, y 17 %5)
- Precedencia de operadores
  - `a * (b + c)`

Operadores aritméticos (binarios)

`+, -, /, *, =, %` (módulo - solo con operandos enteros)

Algunos comentarios:

- ¿Qué pasa con la división de enteros? (ej. 17/5, y 17 %5)
- Precedencia de operadores
  - `a * (b + c)`
  - `a1 * b1 + a2 * b2`

Operadores aritméticos (binarios)

`+, -, /, *, =, %` (módulo - solo con operandos enteros)

Algunos comentarios:

- ¿Qué pasa con la división de enteros? (ej. 17/5, y 17 %5)
- Precedencia de operadores
  - `a * (b + c)`
  - `a1 * b1 + a2 * b2`
  - `a0 + a1 * x + a2 * x * x`

Operadores aritméticos (binarios)

`+, -, /, *, =, %` (módulo - solo con operandos enteros)

Algunos comentarios:

- ¿Qué pasa con la división de enteros? (ej. 17/5, y 17 %5)
- Precedencia de operadores
  - `a * (b + c)`
  - `a1 * b1 + a2 * b2`
  - `a0 + a1 * x + a2 * x * x`

Operadores aritméticos (binarios)

`+, -, /, *, =, %` (módulo - solo con operandos enteros)

Algunos comentarios:

- ¿Qué pasa con la división de enteros? (ej. 17/5, y 17 %5)
- Precedencia de operadores
  - `a * (b + c)`
  - `a1 * b1 + a2 * b2`
  - `a0 + a1 * x + a2 * x * x`

Regla de precedencia: primero `()`, luego `*, /, %`, y finalmente `+, -`.

Operadores aritméticos (binarios)

`+, -, /, *, =, %` (módulo - solo con operandos enteros)

Algunos comentarios:

- ¿Qué pasa con la división de enteros? (ej. 17/5, y 17 %5)
- Precedencia de operadores
  - `a * (b + c)`
  - `a1 * b1 + a2 * b2`
  - `a0 + a1 * x + a2 * x * x`

Regla de precedencia: primero `()`, luego `*, /, %`, y finalmente `+, -`.

Operadores de asignación

`+=, -=, *=, /=, %=`

Contenido

1. Introducción
2. Introducción al lenguaje C
3. Programación estructurada
   - Algoritmo, seudo-código y diagrama de flujo
   - Estructuras básicas
   - Estructuras de control
   - Algunos operadores
4. Estructuras de selección
5. Estructura de repetición
6. Funciones (para más adelante)

¿Qué es un algoritmo?

¿Qué es un algoritmo?

Un procedimiento para resolver un problema en términos de

- las acciones a ejecutarse, y
- el orden en el cual estas acciones deben de ejecutarse

se llama un algoritmo.

¿Qué es un algoritmo?

Un procedimiento para resolver un problema en términos de

- las acciones a ejecutarse, y
- el orden en el cual estas acciones deben de ejecutarse

se llama un algoritmo.

. . . ¿seudo-código?

¿Qué es un algoritmo?

Un procedimiento para resolver un problema en términos de

- las acciones a ejecutarse, y
- el orden en el cual estas acciones deben de ejecutarse

se llama un algoritmo.

. . . ¿seudo-código?

Es un lenguaje artificial e informal que auxilia a los programadores a desarrollar los algoritmos.
Ayudan al programa *"a pensar"* un programa antes de intentar escribirlo en un lenguaje de programación como C.
El seudo-código incluye solo enunciados ejecutables.

¿Qué es un algoritmo?

Un procedimiento para resolver un problema en términos de

- las acciones a ejecutarse, y
- el orden en el cual estas acciones deben de ejecutarse

se llama un algoritmo.

. . . ¿seudo-código?

Es un lenguaje artificial e informal que auxilia a los programadores a desarrollar los algoritmos.
Ayudan al programa *"a pensar"* un programa antes de intentar escribirlo en un lenguaje de programación como C.
El seudo-código incluye solo enunciados ejecutables.

. . . ¿y un diagrama de flujo?

¿Qué es un algoritmo?

Un procedimiento para resolver un problema en términos de

- las acciones a ejecutarse, y
- el orden en el cual estas acciones deben de ejecutarse

se llama un algoritmo.

. . . ¿seudo-código?

Es un lenguaje artificial e informal que auxilia a los programadores a desarrollar los algoritmos.
Ayudan al programa *"a pensar"* un programa antes de intentar escribirlo en un lenguaje de programación como C.
El seudo-código incluye solo enunciados ejecutables.

. . . ¿y un diagrama de flujo?

Un diagrama de flujo es una representación gráfica de un algoritmo o de una porción de un algoritmo.

Teorema del programa estructurado (Böhm-Jacopini)

Establece que toda función computable puede ser implementada en un lenguaje de programación que combine sólo tres estructuras lógicas. Esas tres formas (también llamadas estructuras de control) específicamente son:

Teorema del programa estructurado (Böhm-Jacopini)

Establece que toda función computable puede ser implementada en un lenguaje de programación que combine sólo tres estructuras lógicas. Esas tres formas (también llamadas estructuras de control) específicamente son:

- *Secuencia*: ejecución de una instrucción tras otra.

Teorema del programa estructurado (Böhm-Jacopini)

Establece que toda función computable puede ser implementada en un lenguaje de programación que combine sólo tres estructuras lógicas. Esas tres formas (también llamadas estructuras de control) específicamente son:

- *Secuencia*: ejecución de una instrucción tras otra.
- *Selección*: ejecución de una de dos instrucciones (o conjuntos), según el valor de una variable booleana.

Teorema del programa estructurado (Böhm-Jacopini)

Establece que toda función computable puede ser implementada en un lenguaje de programación que combine sólo tres estructuras lógicas. Esas tres formas (también llamadas estructuras de control) específicamente son:

- *Secuencia*: ejecución de una instrucción tras otra.
- *Selección*: ejecución de una de dos instrucciones (o conjuntos), según el valor de una variable booleana.
- *Iteración*: ejecución de una instrucción (o conjunto) mientras una variable booleana sea 'verdadera'. Esta estructura lógica también se conoce como ciclo o bucle.

En el lenguaje C

Teorema del programa estructurado (Böhm-Jacopini)

Establece que toda función computable puede ser implementada en un lenguaje de programación que combine sólo tres estructuras lógicas. Esas tres formas (también llamadas estructuras de control) específicamente son:

- *Secuencia*: ejecución de una instrucción tras otra.
- *Selección*: ejecución de una de dos instrucciones (o conjuntos), según el valor de una variable booleana.
- *Iteración*: ejecución de una instrucción (o conjunto) mientras una variable booleana sea 'verdadera'. Esta estructura lógica también se conoce como ciclo o bucle.

En el lenguaje C

- Secuencia?:

Teorema del programa estructurado (Böhm-Jacopini)

Establece que toda función computable puede ser implementada en un lenguaje de programación que combine sólo tres estructuras lógicas. Esas tres formas (también llamadas estructuras de control) específicamente son:

- *Secuencia*: ejecución de una instrucción tras otra.
- *Selección*: ejecución de una de dos instrucciones (o conjuntos), según el valor de una variable booleana.
- *Iteración*: ejecución de una instrucción (o conjunto) mientras una variable booleana sea 'verdadera'. Esta estructura lógica también se conoce como ciclo o bucle.

En el lenguaje C

- Secuencia?: Lenguaje secuencial.

Teorema del programa estructurado (Böhm-Jacopini)

Establece que toda función computable puede ser implementada en un lenguaje de programación que combine sólo tres estructuras lógicas. Esas tres formas (también llamadas estructuras de control) específicamente son:

- *Secuencia*: ejecución de una instrucción tras otra.
- *Selección*: ejecución de una de dos instrucciones (o conjuntos), según el valor de una variable booleana.
- *Iteración*: ejecución de una instrucción (o conjunto) mientras una variable booleana sea 'verdadera'. Esta estructura lógica también se conoce como ciclo o bucle.

En el lenguaje C

- Secuencia?: Lenguaje secuencial.
- Selección?:

Teorema del programa estructurado (Böhm-Jacopini)

Establece que toda función computable puede ser implementada en un lenguaje de programación que combine sólo tres estructuras lógicas. Esas tres formas (también llamadas estructuras de control) específicamente son:

- *Secuencia*: ejecución de una instrucción tras otra.
- *Selección*: ejecución de una de dos instrucciones (o conjuntos), según el valor de una variable booleana.
- *Iteración*: ejecución de una instrucción (o conjunto) mientras una variable booleana sea 'verdadera'. Esta estructura lógica también se conoce como ciclo o bucle.

En el lenguaje C

- Secuencia?: Lenguaje secuencial.
- Selección?: **if, if/else, switch**.

Teorema del programa estructurado (Böhm-Jacopini)

Establece que toda función computable puede ser implementada en un lenguaje de programación que combine sólo tres estructuras lógicas. Esas tres formas (también llamadas estructuras de control) específicamente son:

- *Secuencia*: ejecución de una instrucción tras otra.
- *Selección*: ejecución de una de dos instrucciones (o conjuntos), según el valor de una variable booleana.
- *Iteración*: ejecución de una instrucción (o conjunto) mientras una variable booleana sea 'verdadera'. Esta estructura lógica también se conoce como ciclo o bucle.

En el lenguaje C

- Secuencia?: Lenguaje secuencial.
- Selección?: **if, if/else, switch**.
- Iteración/repetición?:

Teorema del programa estructurado (Böhm-Jacopini)

Establece que toda función computable puede ser implementada en un lenguaje de programación que combine sólo tres estructuras lógicas. Esas tres formas (también llamadas estructuras de control) específicamente son:

- *Secuencia*: ejecución de una instrucción tras otra.
- *Selección*: ejecución de una de dos instrucciones (o conjuntos), según el valor de una variable booleana.
- *Iteración*: ejecución de una instrucción (o conjunto) mientras una variable booleana sea 'verdadera'. Esta estructura lógica también se conoce como ciclo o bucle.

En el lenguaje C

- Secuencia?: Lenguaje secuencial.
- Selección?: **if, if/else, switch**.
- Iteración/repetición?: **while, do/while, for**.

Teorema del programa estructurado (Böhm-Jacopini)

Establece que toda función computable puede ser implementada en un lenguaje de programación que combine sólo tres estructuras lógicas. Esas tres formas (también llamadas estructuras de control) específicamente son:

- *Secuencia*: ejecución de una instrucción tras otra.
- *Selección*: ejecución de una de dos instrucciones (o conjuntos), según el valor de una variable booleana.
- *Iteración*: ejecución de una instrucción (o conjunto) mientras una variable booleana sea 'verdadera'. Esta estructura lógica también se conoce como ciclo o bucle.

En el lenguaje C

- Secuencia?: Lenguaje secuencial.
- Selección?: **if, if/else, switch**. ¿Qué hacen?
- Iteración/repetición?: **while, do/while, for**. ¿Qué hacen?

Teorema del programa estructurado (Böhm-Jacopini)

Establece que toda función computable puede ser implementada en un lenguaje de programación que combine sólo tres estructuras lógicas. Esas tres formas (también llamadas estructuras de control) específicamente son:

- *Secuencia*: ejecución de una instrucción tras otra.
- *Selección*: ejecución de una de dos instrucciones (o conjuntos), según el valor de una variable booleana.
- *Iteración*: ejecución de una instrucción (o conjunto) mientras una variable booleana sea 'verdadera'. Esta estructura lógica también se conoce como ciclo o bucle.

En el lenguaje C

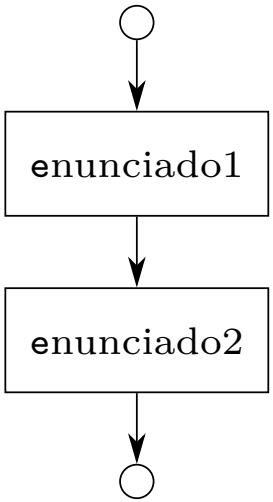
- Secuencia?: Lenguaje secuencial.
- Selección?: **if, if/else, switch**. ¿Qué hacen?
- Iteración/repetición?: **while, do/while, for**. ¿Qué hacen?

C tiene solo 7 estructuras de control

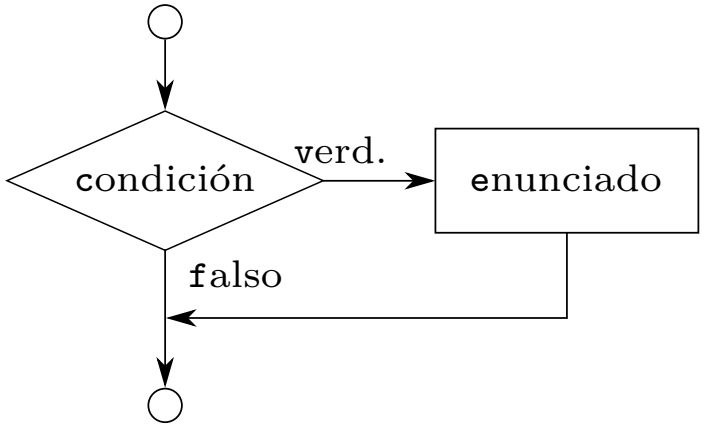
Programación estructurada

Estructuras de control

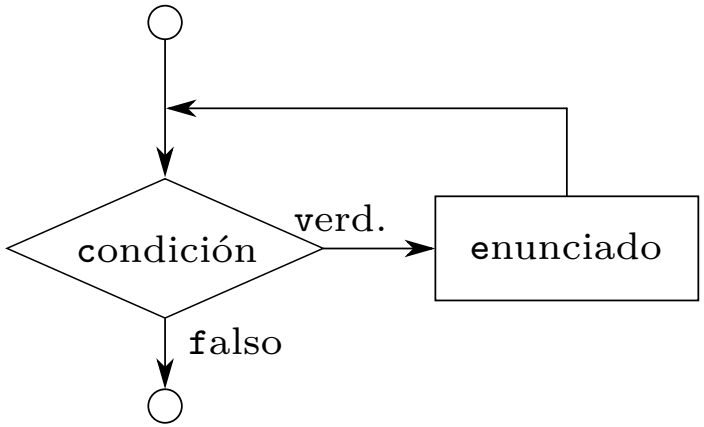
Estructura secuencial



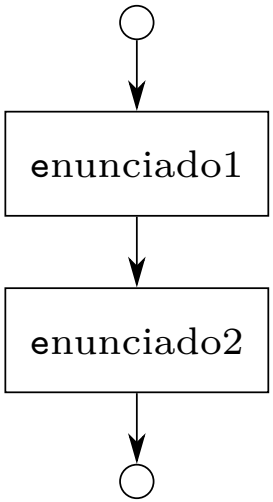
Estructura de selección



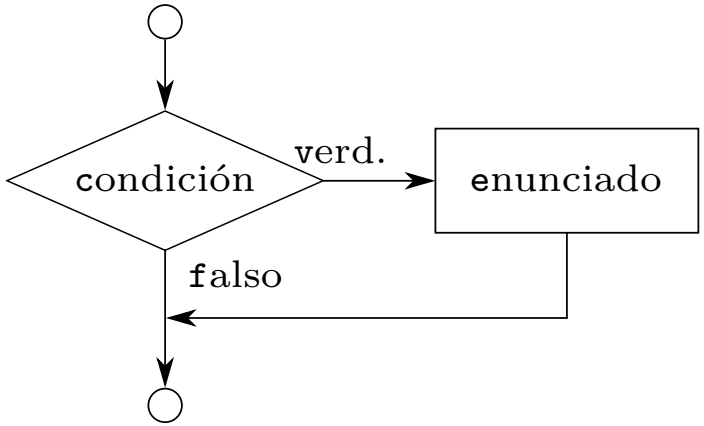
Estructura de repetición



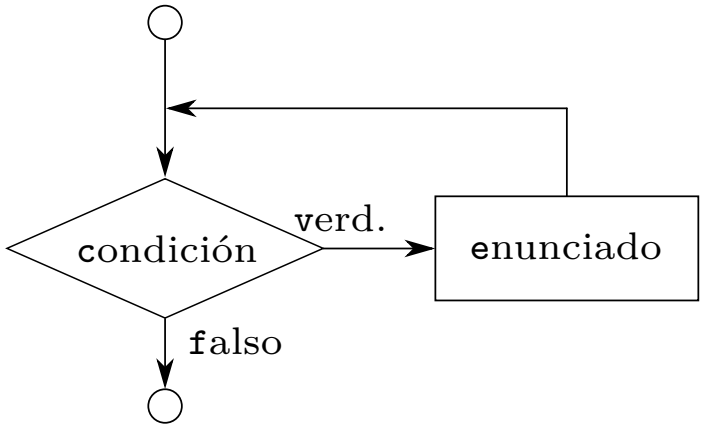
Estructura secuencial



Estructura de selección

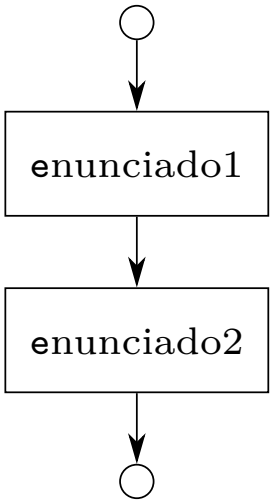


Estructura de repetición

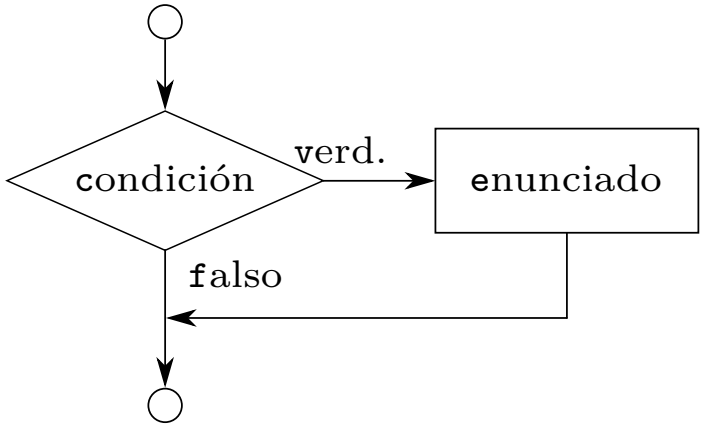


- Cada estructuras tienen una sola entrada y una sola salida.

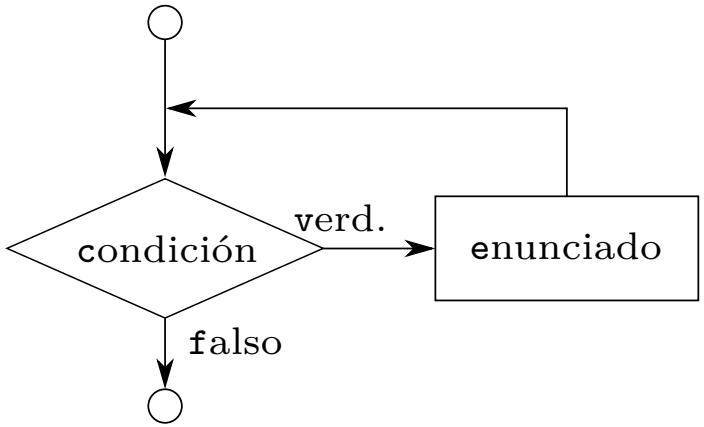
Estructura secuencial



Estructura de selección



Estructura de repetición



- Cada estructuras tienen una sola entrada y una sola salida.
- Ellas se puede conectar mediante:
  - *apilamiento*
  - *anidamiento*.

[Condicionales] Operadores de igualdad

`==`, `!=`

(menor nivel de precedencia)

[Condicionales] Operadores relacionales

`>`, `<`, `>=`, `<=`

(mayor nivel de precedencia)

[Condicionales] Operadores de igualdad

`==`, `!=`

(menor nivel de precedencia)

[Condicionales] Operadores relacionales

`>`, `<`, `>=`, `<=`

(mayor nivel de precedencia)

Incremento y decremento

`++`, `--`

(pre/pos incremento/decremento)

[Condicionales] Operadores de igualdad

`==`, `!=`

(menor nivel de precedencia)

[Condicionales] Operadores relacionales

`>`, `<`, `>=`, `<=`

(mayor nivel de precedencia)

Incremento y decremento

`++`, `--`

(pre/pos incremento/decremento)

Operadores lógicos

- OR lógico: `||`
- AND lógico: `&&`
- NOT lógico: `!`

Contenido

1. Introducción
2. Introducción al lenguaje C
3. Programación estructurada
4. Estructuras de selección
   - La estructura if
   - La estructura if/else
   - La estructura switch
5. Estructura de repetición
6. Funciones (para más adelante)

Seudo-código

```
Si calificación es mayor o igual a 60
  Imprimir "Aprobó"
```

Código C

```c
if(calificacion >= 60)
   printf("Aprobó␣\n");
```

Estructuras de selección

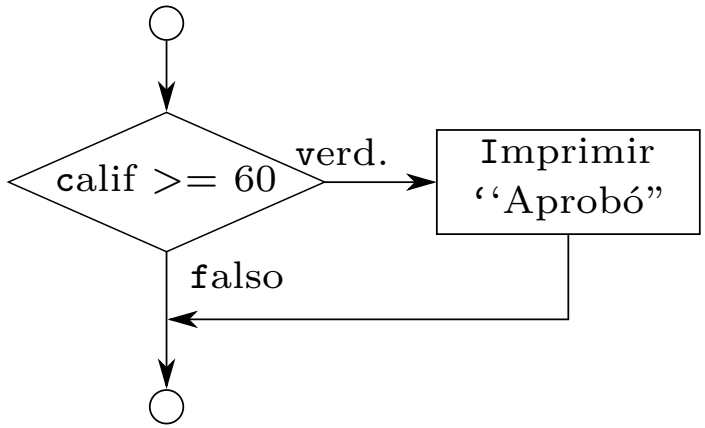
La estructura if

Seudo-código

```
Si calificación es mayor o igual a 60
  Imprimir "Aprobó"
```

Código C

```
if(calificacion >= 60)
   printf("Aprobó␣\n");
```



Estructuras de selección

La estructura if/else

Seudo-código

```
Si calificación es mayor o igual a 60
  Imprimir "Aprobó"
Si no
  Imprimir "No aprobó"
```

Código C

```c
if(calificacion >= 60)
   printf("Aprobó\n");
else
   printf("No aprobó\n");
```

Estructuras de selección

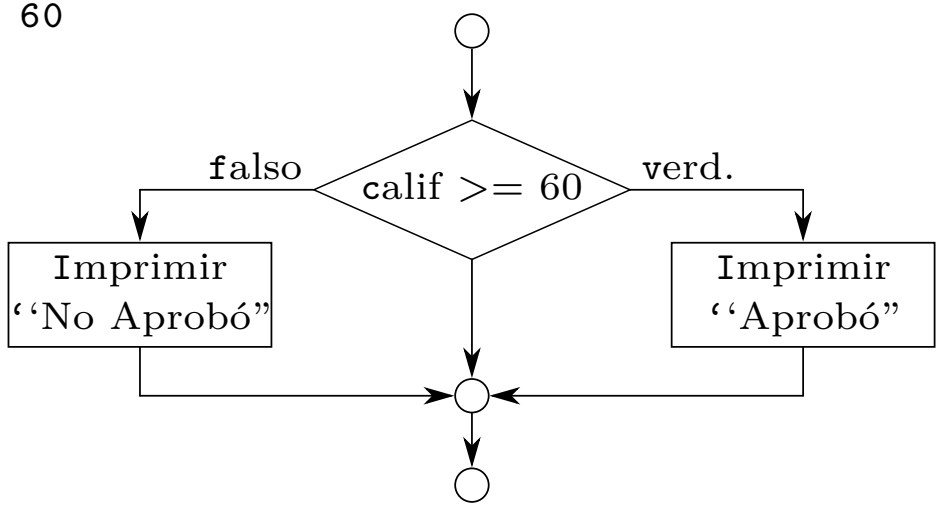
La estructura if/else

Seudo-código

```
Si calificación es mayor o igual a 60
  Imprimir "Aprobó"
Si no
  Imprimir "No aprobó"
```

Código C

```
if(calificacion >= 60)
   printf("Aprobó\n");
else
   printf("No aprobó\n");
```



Estructuras de selección

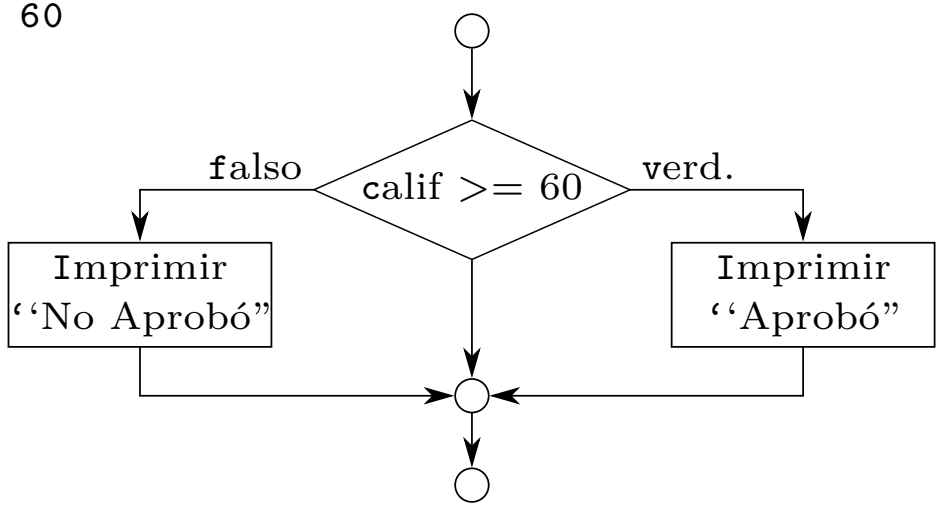
La estructura if/else

Seudo-código

```
Si calificación es mayor o igual a 60
  Imprimir "Aprobó"
Si no
  Imprimir "No aprobó"
```

Código C

```
if(calificacion >= 60)
   printf("Aprobó\n");
else
   printf("No aprobó\n");
```



C tiene el operador condicional `'?:'` que está relacionado con la estructura **if/else**.

Estructuras de selección

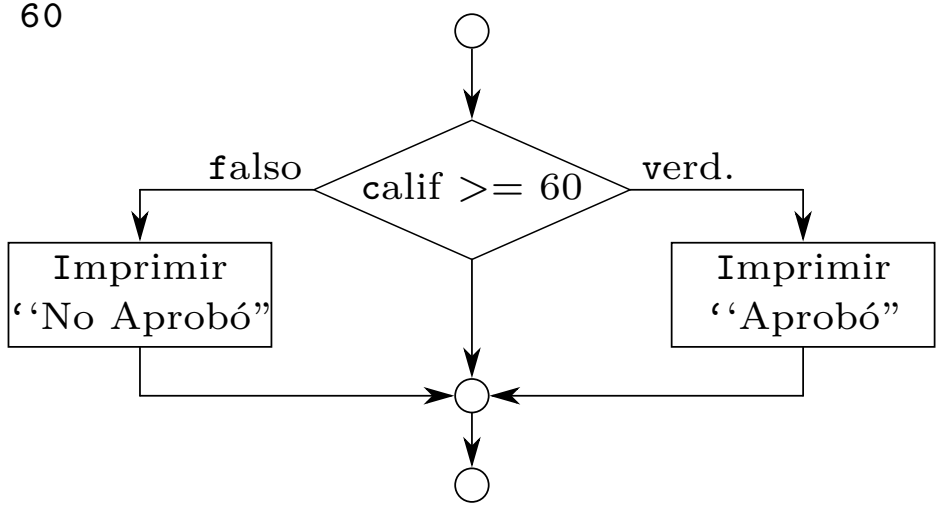
La estructura if/else

Seudo-código

```
Si calificación es mayor o igual a 60
  Imprimir "Aprobó"
Si no
  Imprimir "No aprobó"
```

Código C

```
if(calificacion >= 60)
   printf("Aprobó\n");
else
   printf("No aprobó\n");
```



C tiene el operador condicional **’?:’** que está relacionado con la estructura **if/else**.

```
printf("%s\n", calificacion >= 60 ? "Aprobó" : "No aprobó");
```

Estructuras de selección

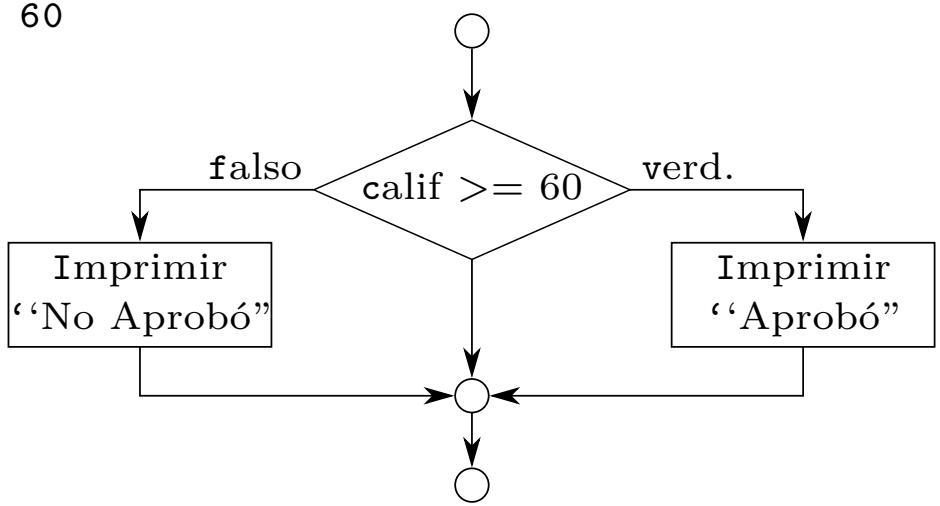
La estructura if/else

Seudo-código

```
Si calificación es mayor o igual a 60
  Imprimir "Aprobó"
Si no
  Imprimir "No aprobó"
```

Código C

```
if(calificacion >= 60)
   printf("Aprobó\n");
else
   printf("No aprobó\n");
```



C tiene el operador condicional **’?:’** que está relacionado con la estructura **if/else**.

```
printf("%s\n", calificacion >= 60 ? "Aprobó" : "No aprobó");
```

```
calificacion >= 60 ? printf("Aprobó\n") : printf("No aprobó\n");
```

Estructuras de selección

La estructura if/else

Enunciados compuestos:

- para incluir varios enunciados en el cuerpo de un **if** o **if/else**

Estructuras de selección

La estructura if/else

Enunciados compuestos:

- para incluir varios enunciados en el cuerpo de un **if** o **if/else**

```
if(calificacion >= 60)
{
    printf("Aprobó\n");
    printf("Felicitaciones!! :) \n");
}
else
{
    printf("No aprobó\n");
    printf("Debes recursar :( \n");
}
```

Estructuras de selección

La estructura if/else

Enunciados compuestos:

- para incluir varios enunciados en el cuerpo de un **if** o **if/else**

```
if(calificacion >= 60)
{
    printf("Aprobó\n");
    printf("Felicitaciones!! :) \n");
}
else
{
    printf("No aprobó\n");
    printf("Debes recursar :( \n");
}
```

Algunas preguntas:

- ¿Qué sucede si no estuvieran las llaves en el **else**?

Estructuras de selección

La estructura if/else

Enunciados compuestos:

- para incluir varios enunciados en el cuerpo de un **if** o **if/else**

```
if(calificacion >= 60)
{
    printf("Aprobó\n");
    printf("Felicitaciones!! :) \n");
}
else
{
    printf("No aprobó\n");
    printf("Debes recursar :( \n");
}
```

Algunas preguntas:

- ¿Qué sucede si no estuvieran las llaves en el **else**?
- ¿Qué sucede si se coloca un punto y coma luego de un **if**? . . . y el **if/else**?

Estructuras de selección

La estructura if/else

Enunciados compuestos:

- para incluir varios enunciados en el cuerpo de un **if** o **if/else**

```
if(calificacion >= 60)
{
    printf("Aprobó\n");
    printf("Felicitaciones!! :) \n");
}
else
{
    printf("No aprobó\n");
    printf("Debes recursar :( \n");
}
```

Algunas preguntas:

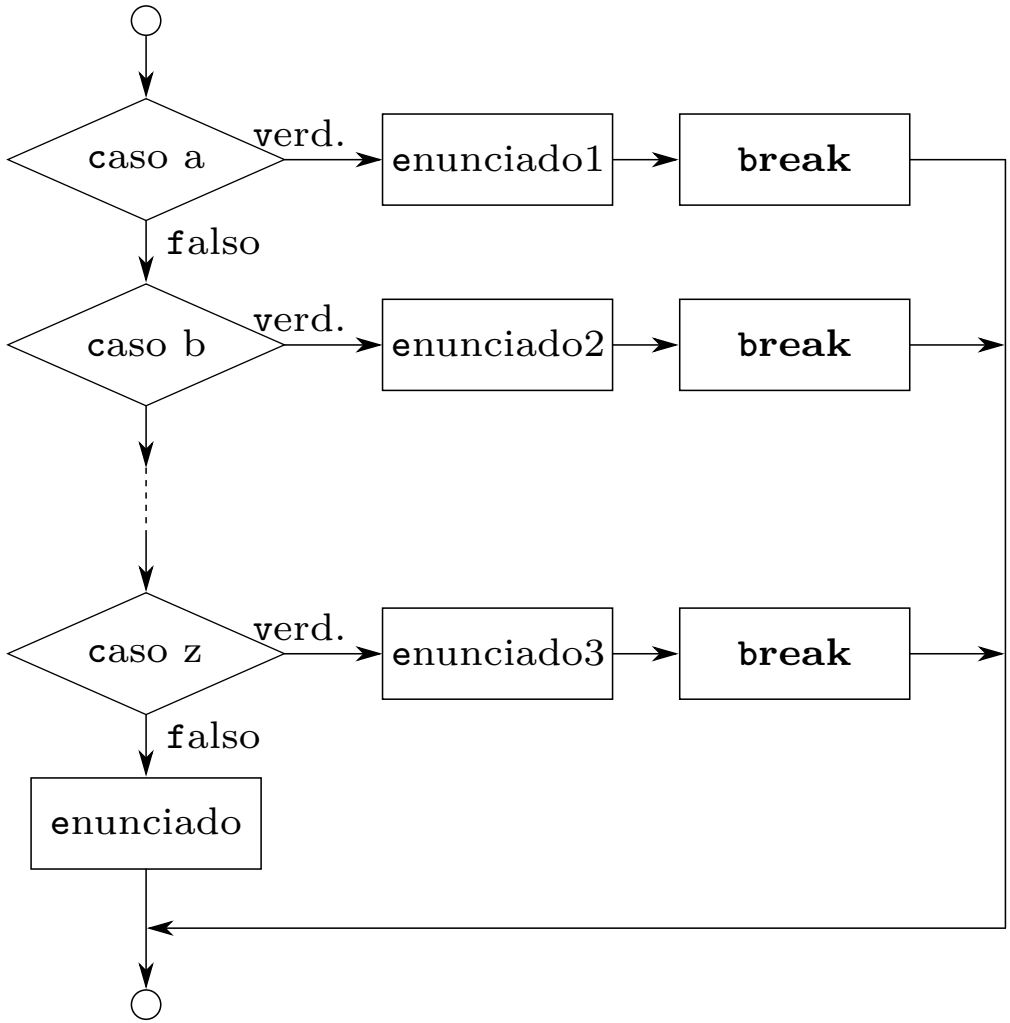
- ¿Qué sucede si no estuvieran las llaves en el **else**?
- ¿Qué sucede si se coloca un punto y coma luego de un **if**? . . . y el **if/else**?
- ¿Cómo sería el diagrama de flujo de estructuras **if/else** anidadas?

Estructuras de selección

La estructura switch

Código C

```
switch(caracter)
{
   case 'a':
     enunciado1;
     break;
   case 'b':
     enunciado2;
     break;
   case 'z':
     enunciado3;
     break;
   default:
     enunciado;
}
```



Contenido

1. Introducción
2. Introducción al lenguaje C
3. Programación estructurada
4. Estructuras de selección
5. Estructura de repetición
   - La estructuras while
   - La estructuras do/while
   - La estructuras for
6. Funciones (para más adelante)

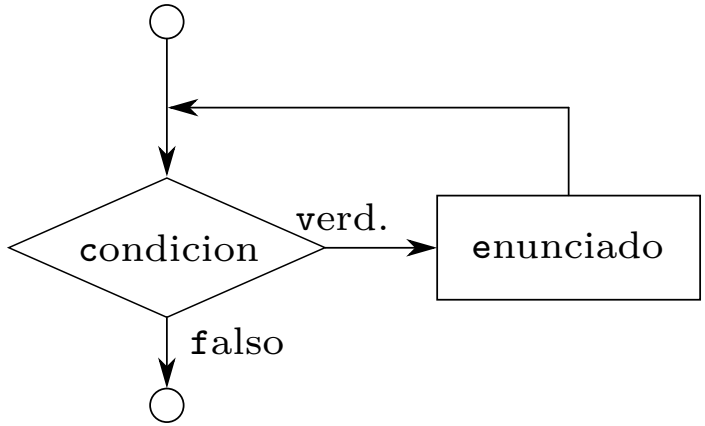
Estructura de repetición

La estructuras while

Código C

```
producto = 2;

while(producto <= 1000)
    producto = 2*producto;
```



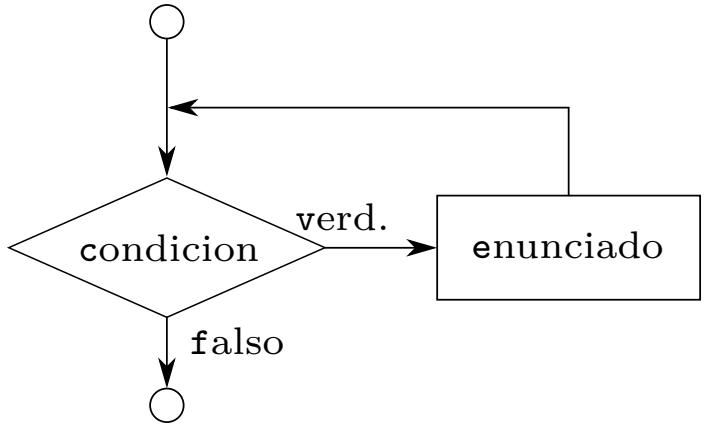
Estructura de repetición

La estructuras while

Código C

```
producto = 2;

while(producto <= 1000)
    producto = 2*producto;
```



- ¿Qué sucede si se coloca un punto y coma luego de un **while**?

Estructura de repetición

La estructuras do/while

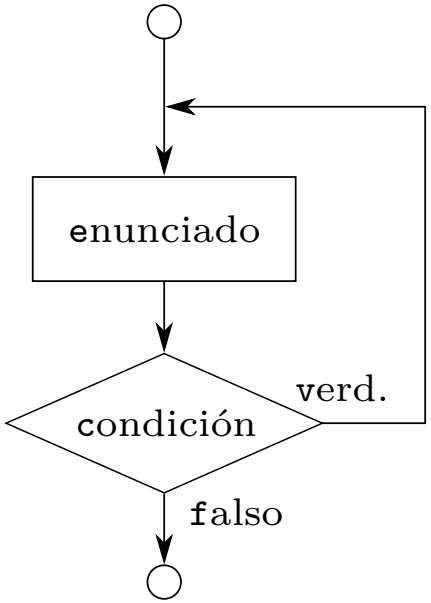
Código C

```c
/* Imprime del 1 al 10
   utilizando do/while */
#include <stdio.h>

int main(void)
{
  int num = 1;

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

  return 0;
}
```



Estructura de repetición

La estructuras do/while

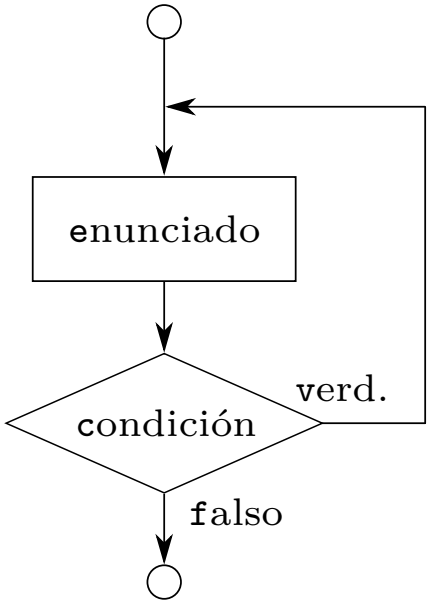
Código C

```c
/* Imprime del 1 al 10
   utilizando do/while */
#include <stdio.h>

int main(void)
{
  int num = 1;

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

  return 0;
}
```



Diferencia entre **while** y **do/while**

- En el **while** la condición de continuidad del ciclo se prueba al principio.
- En el **do/while** la condición de continuidad del ciclo se prueba luego de ejecutar el cuerpo.

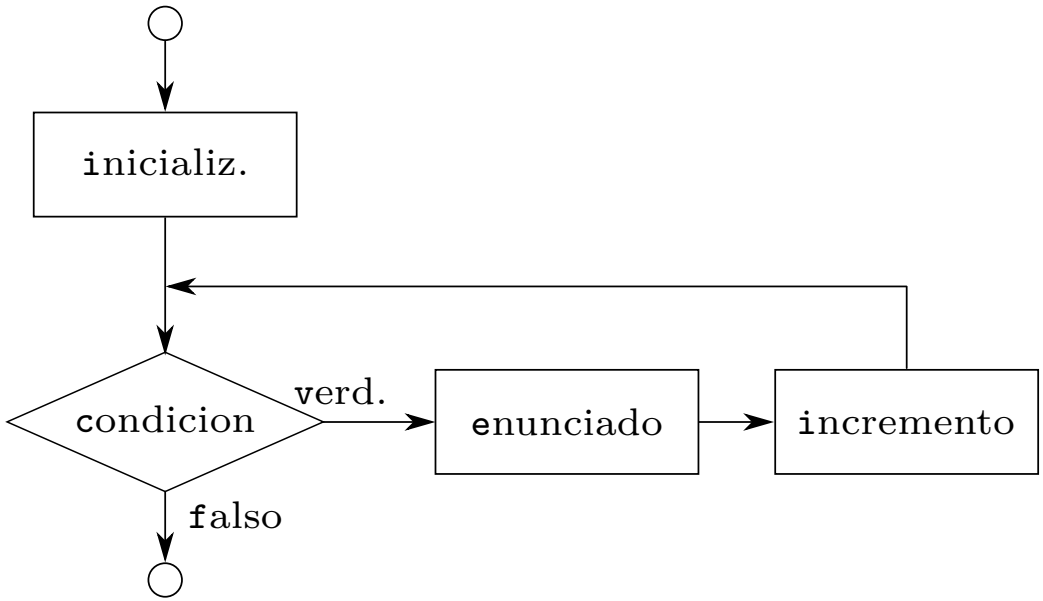
Código C

```c
/* Imprime del 1 al 10 utilizando for */
#include <stdio.h>

int main(void) {
  int num;

  for(num = 1; num <= 10; num++)
    printf("%d ", num);

  return 0;
}
```



Maneja todas los detalles de la repetición controlada por contador (repetición definida).

Formato general de la estructura for

```
for(expresion1; expresion2; expresion3)
   enunciado;
```

Maneja todas los detalles de la repetición controlada por contador (repetición definida).

Formato general de la estructura for

```
for(expresion1; expresion2; expresion3)
    enunciado;
```

Equivalente con estructura while

```
expresion1;
while(expresion2) {
    enunciado;
    expresion;
}
```

Estructura de repetición

La estructuras for

```c
/* Sumatoria con estructura for */
#include <stdio.h>

int main(void)
{
  int sum = 0, num;

  for(num = 2; num <= 100; num += 2)
    sum += num;

  printf("La suma es igual a: %d\n", sum);

  return 0;
}
```

Estructura de repetición

La estructuras for

```c
/* Sumatoria con estructura for */
#include <stdio.h>

int main(void)
{
  int sum = 0, num;

  for(num = 2; num <= 100; num += 2)
    sum += num;

  printf("La suma es igual a: %d\n", sum);

  return 0;
}
```

```
La suma es igual a: 2550
```

Estructura de repetición

La estructuras for

```c
/* Sumatoria con estructura for */
#include <stdio.h>

int main(void)
{
  int sum = 0, num;

  for(num = 2; num <= 100; sum += num, num += 2)
    ;

  printf("La suma es igual a: %d\n", sum);

  return 0;
}
```

```
La suma es igual a: 2550
```

Contenido

1. Introducción
2. Introducción al lenguaje C
3. Programación estructurada
4. Estructuras de selección
5. Estructura de repetición
6. Funciones (para más adelante)