

Compilad ores

Introducción

Su escritura abarca...

- ✓ *Lenguajes de programación*
- ✓ *Arquitectura de computadoras*
- ✓ *Teoría de lenguajes*
- ✓ *Algoritmos*
- ✓ *Ingeniería de software*

Definición

- Un compilador es.

*Lengua
Programa*

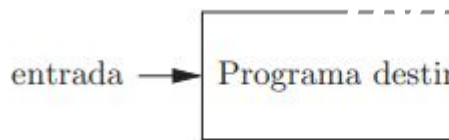


Figura 1.2: Ejecución del programa de destino

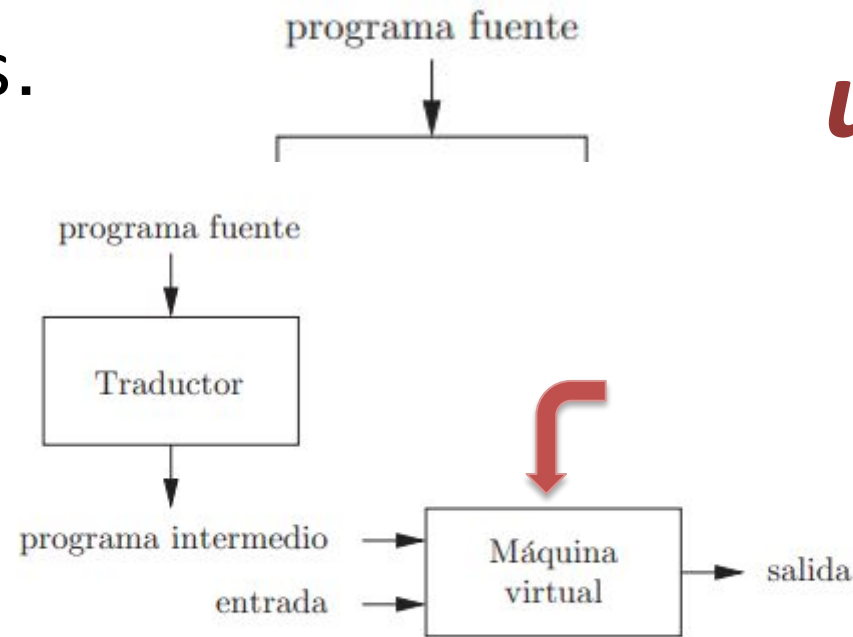


Figura 1.4: Un compilador híbrido

uaje de bajo nivel

ictor



Figura 1.3: Un intérprete

Fortran

- 1950
 - Compiladores son difíciles de implementar
 - 18 años, (Backus, 1975)
- Proceso
 - Análisis
 - Síntesis

Programas de apoyo

- Preprocesador, ensamblador y enlazador
- El preprocesador
 - Macros
 - `#define MAX 500`
 - `#define setBit(reg,bit) reg|=0x01<<((bit)-1)`
 - Expansión de código
- El ensamblador
- El enlazador

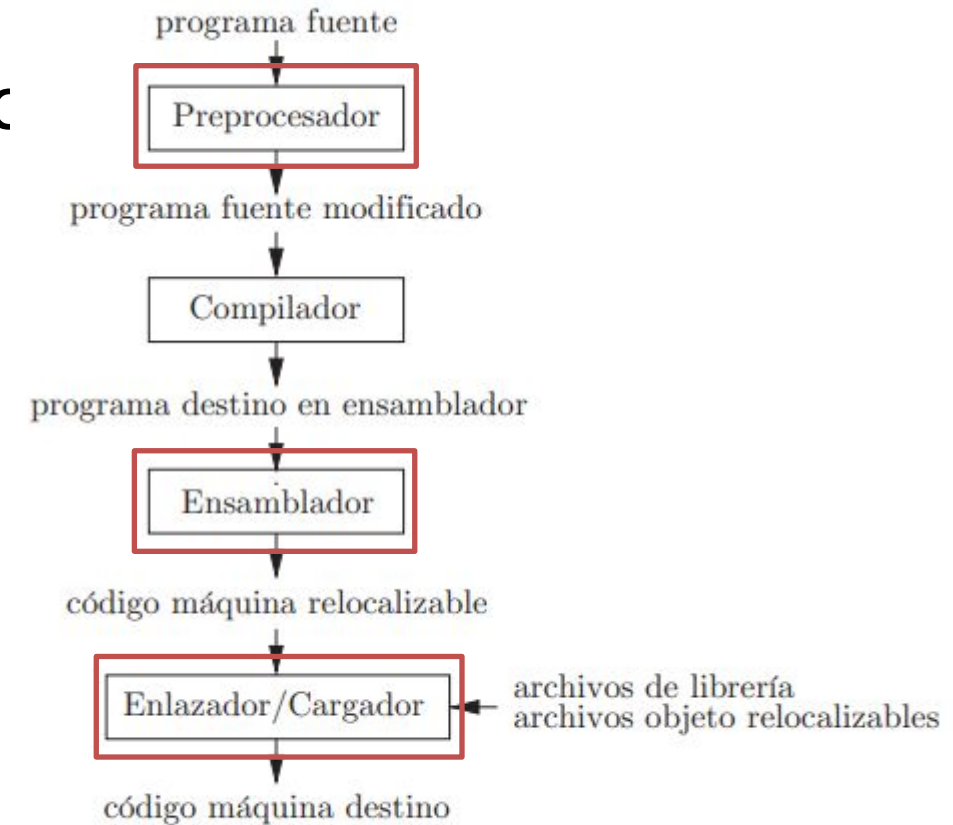


Figura 1.5: Un sistema de procesamiento de lenguaje

Estructura de un compilador

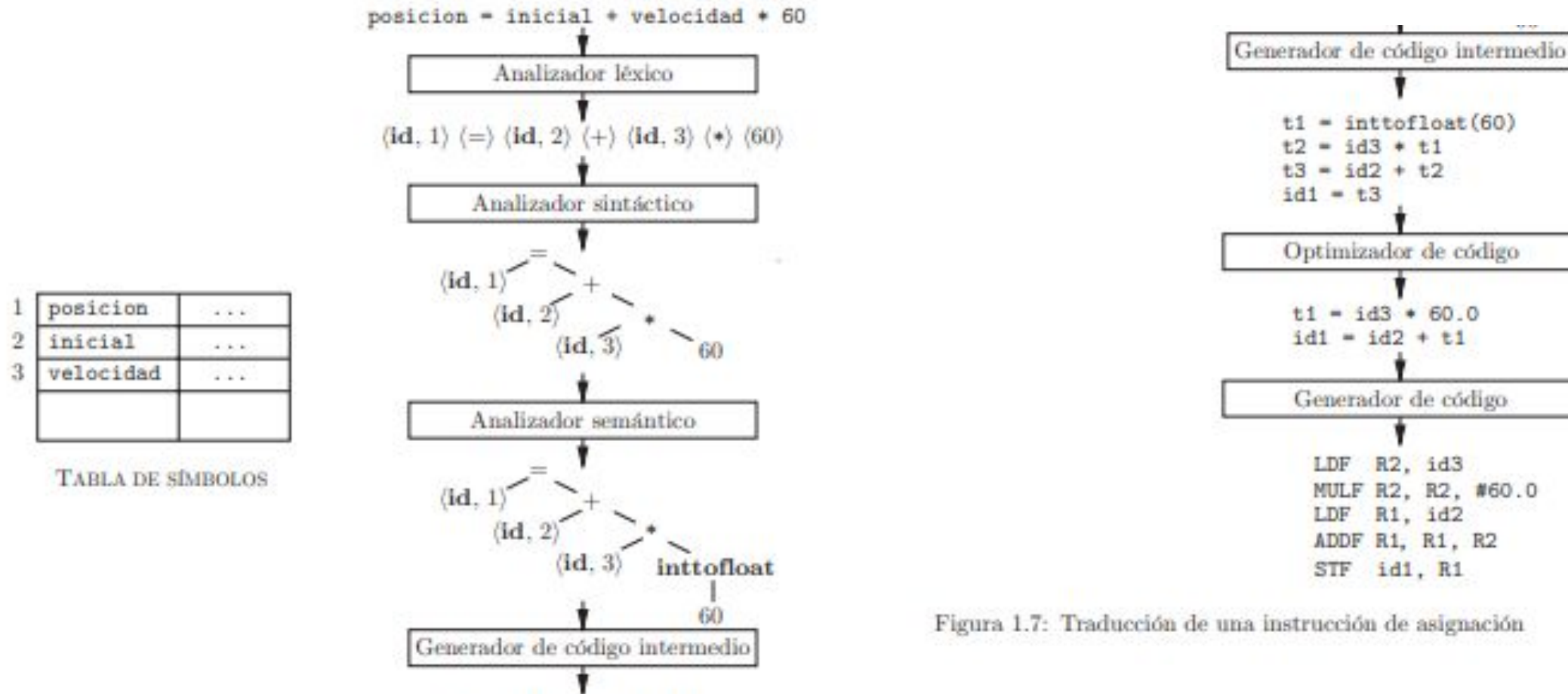
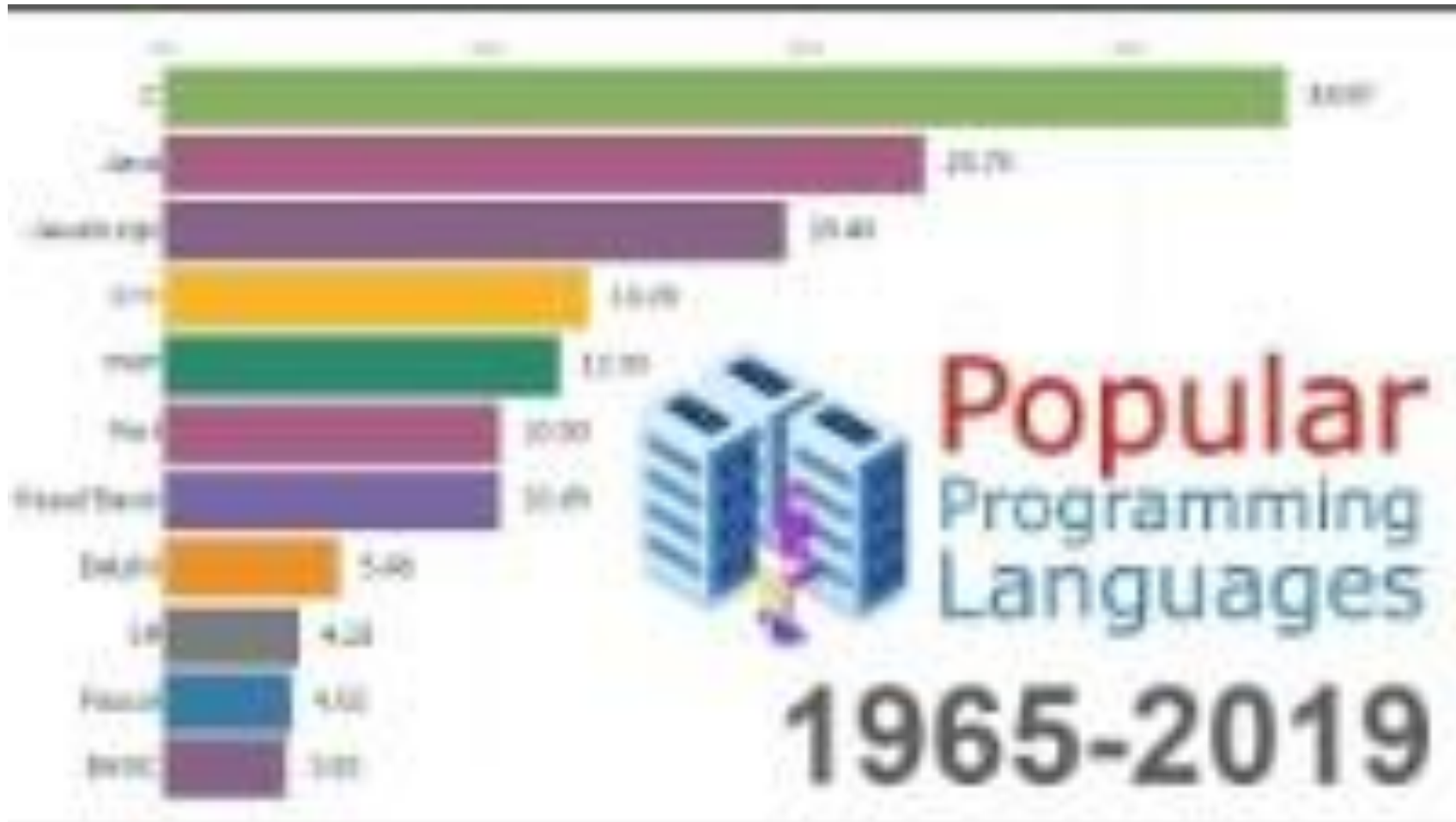


Figura 1.7: Traducción de una instrucción de asignación

Evolución de los lenguajes de programación



Consideraciones

- Las generaciones de lenguajes de programación se dividen en cinco niveles principales:
 1. **Primera generación:** los lenguajes de máquina, que consisten en códigos binarios entendidos directamente por el hardware.
 2. **Segunda generación:** los lenguajes ensambladores, que utilizan abreviaturas mnemónicas para representar instrucciones de máquina.
 3. **Tercera generación:** los lenguajes de alto nivel, como COBOL, Fortran, C y Pascal, que permiten una programación más cercana al lenguaje humano.
 4. **Cuarta generación:** lenguajes de programación de propósito específico, como bases de datos o sistemas expertos.
 5. **Quinta generación:** lenguajes de programación orientados a la inteligencia artificial y la programación basada en la lógica.

Consideraciones

El término *lenguaje von Neumann* se aplica a los lenguajes de programación cuyo modelo se basa en la arquitectura de computadoras descrita por von Neumann. Muchos de los lenguajes de la actualidad, como Fortran y C, son lenguajes von Neumann.

Un *lenguaje orientado a objetos* es uno que soporta la programación orientada a objetos, un estilo de programación en el que un programa consiste en una colección de objetos que interactúan entre sí. Simula 67 y Smalltalk son de los primeros lenguajes orientados a objetos importantes. Los lenguajes como C++, C#, Java y Ruby son los lenguajes orientados a objetos más recientes.

Los *lenguajes de secuencias de comandos (scripting)* son lenguajes interpretados con operadores de alto nivel diseñados para “unir” cálculos. Estos cálculos se conocían en un principio como “*secuencias de comandos (scripts)*”. Awk, JavaScript, Perl, PHP, Python, Ruby y Tcl son ejemplos populares de lenguajes de secuencias de comandos. Los programas escritos en

Consideraciones

Otra de las clasificaciones de los lenguajes utiliza el término *imperativo* para los lenguajes en los que un programa especifica *cómo* se va a realizar un cálculo, y *declarativo* para los lenguajes en los que un programa especifica *qué* cálculo se va a realizar. Los lenguajes como C, C++, C# y Java son lenguajes imperativos. En los lenguajes imperativos hay una noción de estado del programa, junto con instrucciones que modifican ese estado. Los lenguajes funcionales como ML y Haskell, y los lenguajes de lógica de restricción como Prolog, se consideran a menudo como lenguajes declarativos.

Consideraciones

- Lenguaje de alto nivel

Alta abstracción
Fácil programación
Lentos de ejecutar
Portables

- Lenguajes de bajo nivel

Más eficiente
Difíciles de escribir
Menos portables

Consideraciones adicionales

- Arquitectura de computadoras
 - CISC,RISC
 - Reduced/complex instructions set computer
 - Paralelismo
 - A nivel de instrucción
 - A nivel de procesador

Consideraciones adicionales

- Jerarquías de memoria

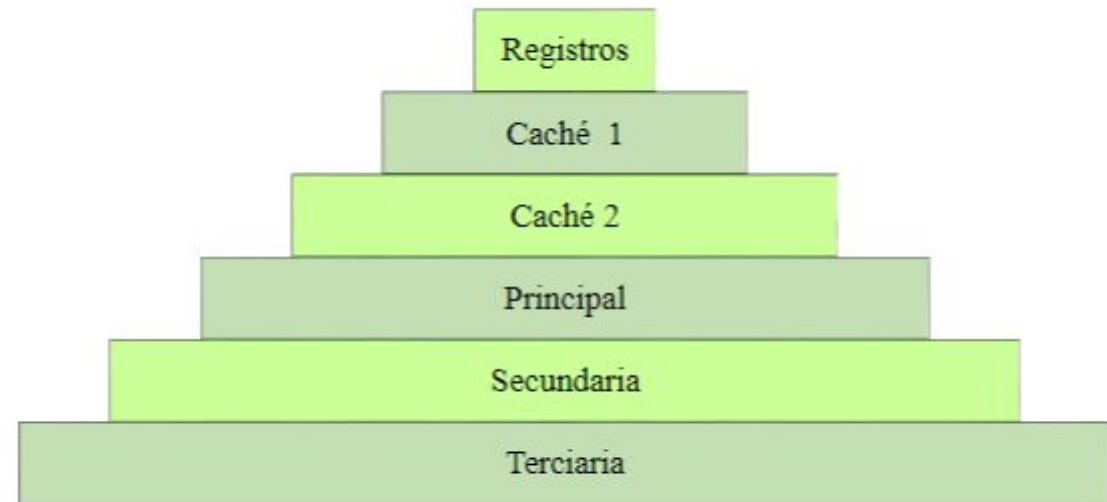


Figura 3.2: Niveles de jerarquía.

Fundamentos de lenguajes

- Dinámico y estático
- Alcance de variables (bloques)
- Nombres, Variables, identificadores
- Procedimientos funciones y método
- Control de acceso:
 - Public, private, protected
- Declaraciones y definiciones

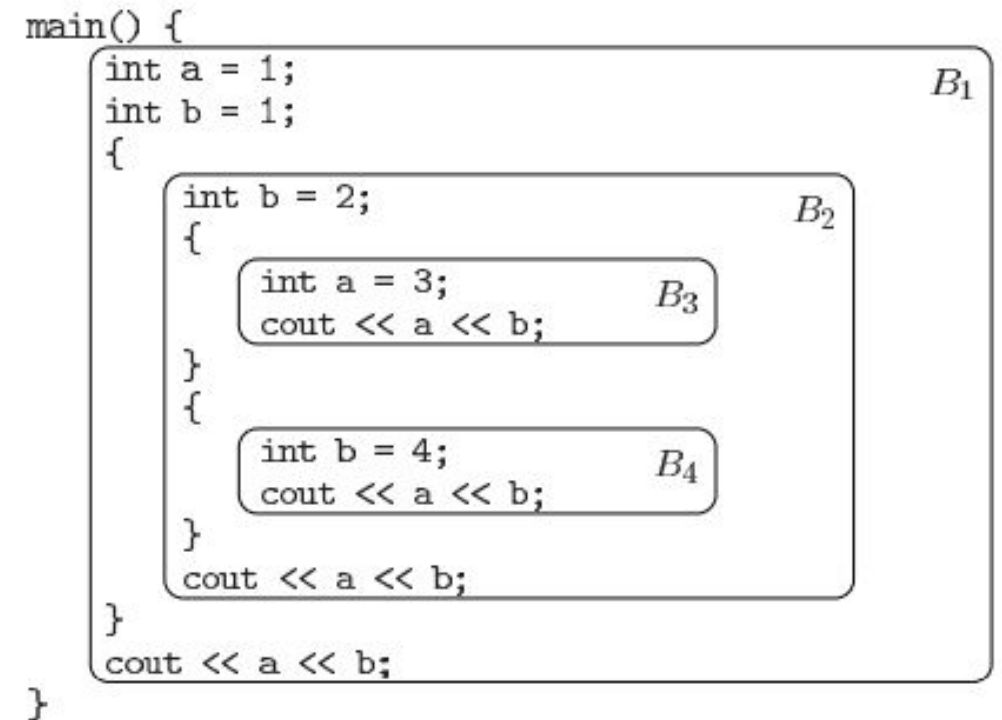


Figura 1.10: Bloques en un programa en C++

Bloques

```
main() {  
  int a = 1;  
  int b = 1;  
  {  
    int b = 2;  
    {  
      int a = 3;  
      cout << a << b;  
    }  
    {  
      int b = 4;  
      cout << a << b;  
    }  
    cout << a << b;  
  }  
}
```

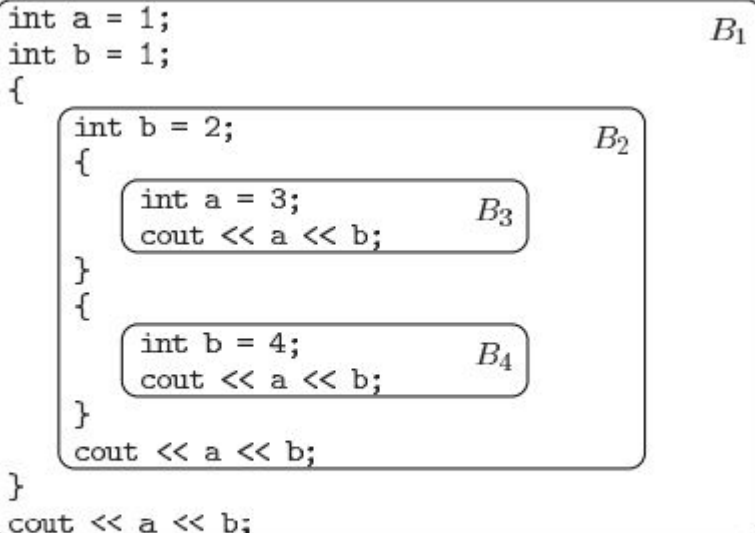


Figura 1.10: Bloques en un programa en C++

```
int w, x, y, z;  
int i = 4; int j = 5;  
{  
  int j = 7;  
  i = 6;  
  w = i + j;  
}  
x = i + j;  
{  
  int i = 8;  
  y = i + j;  
}  
z = i + j;
```

(a) Código para el ejercicio 1.6.1

```
int w, x, y, z;  
int i = 3; int j = 4;  
{  
  int i = 5;  
  w = i + j;  
}  
x = i + j;  
{  
  int j = 6;  
  i = 7;  
  y = i + j;  
}  
z = i + j;
```

(b) Código para el ejercicio 1.6.2

Figura 1.13: Código estructurado por bloques

valores asignados a w , x , y y z .

Repaso

compilador

Optimización de código

Partes de un compilador

Procesador

Alcance de una variable

Fases del Análisis

Referencias

- Aho,
- Imágenes
 - Sanchez,J.J.(2018). Diferencias entre private, protected y public.
<https://janpierrsanchez.medium.com/diferencias-entre-private-protected-y-public-fb3ddb72c5e0>
 - Aho/Setti Text book