EJERCICIOS, PROBLEMAS, CASOS PRÁCTICOS

Universidad de Jaén

Escuela Politécnica Superior de Jaén

Go el lenguaje de la nube – Ejercicios básicos resueltos

Francisco Jesús Martínez Mimbrera

15/01/2026

CREA

GO, EL LENGUAJE DE LA NUBE. EJERCICIOS BÁSICOS RESUELTOS

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

FRANCISCO JESÚS MARTÍNEZ MIMBRERA

Título: Go, El Lenguaje de la Nube. Ejercicios básicos resueltos
Autores: Francisco J. Martínez Mimbrera
Versión: 1
Fecha: 15 de enero de 2026

Este manual ha sido desarrollado y maquetado utilizando LaTeX 2_ε.

Índice general

Capítulo 1

Introducción

El propósito general de este manual es proveer a los alumnos de las disciplinas de ingeniería informática, telecomunicaciones e industriales de un libro con problemas resueltos básicos de programación. La idea es que dicho material pueda servir como apoyo al estudio de las asignaturas a las que se enfrentarán durante su etapa en la Universidad. En este caso, se ha decidido apostar por el lenguaje de programación **Go** (a menudo referido como Golang). El lenguaje de programación Go es un proyecto de código abierto desarrollado inicialmente en Google en el año 2007 por Robert Griesemer, Rob Pike y Ken Thompson [7]. Fue anunciado públicamente en noviembre de 2009 y su versión 1.0 se lanzó en marzo de 2012 [19]. Go nació de la necesidad de mejorar la productividad de la programación en la era de los procesadores multinúcleo y las grandes redes de servidores, combinando la eficiencia de C++ con la legibilidad de lenguajes dinámicos como Python [3]. En la actualidad, Go se ha convertido en el lenguaje estándar para la infraestructura en la nube (*Cloud Computing*), microservicios y contenedores (Docker y Kubernetes están escritos en Go). Según el índice TIOBE y encuestas de Stack Overflow de 2025, Go se mantiene como uno de los lenguajes mejor pagados y con mayor crecimiento en el ámbito del desarrollo **backend** y de **sistemas**. Para la elaboración de este manual se han utilizado las versiones más recientes del lenguaje (Go 1.23+) y sus herramientas oficiales.

Estructura del manual

Para alcanzar el propósito planteado, este manual se va a estructurar en los siguientes capítulos para una mejor comprensión por parte del lector.

- **Capítulo 2. E/S por consola y operaciones aritméticas**. En este capítulo se pretende realizar ejercicios básicos de toma de contacto con el lenguaje, el uso del paquete `fmt` para la entrada y salida de datos, y la resolución de problemas que implican operaciones aritméticas básicas.

- **Capítulo 3. Flujos de control, funciones, cadenas, arrays y slices**. En este capítulo se van a presentar ejercicios relativos a las estructuras de control (bucles `for`, condicionales), el trabajo con funciones, cadenas de caracteres (`string`) y las estructuras de datos fundamentales en Go: los arrays y, especialmente, los *slices* y mapas.

- **Capítulo 4. Manejo de Ficheros, Punteros y Estructuras**. Este capítulo se dedicará a resolver problemas de persistencia de datos en ficheros, el uso de punteros y la definición de tipos de datos propios mediante `structs`.

- **Capítulo 5. Algoritmos de ordenación y búsqueda**. Como parte interesante de la resolución de problemas básicos, se abordarán algoritmos de ordenación y búsqueda, aprovechando tanto la implementación manual como las potentes herramientas de la biblioteca estándar de Go.

- **Capítulo 6. Ejercicios de criptografía básica**. Se incluye también un capítulo dedicado a problemas clásicos de criptografía, un área donde Go destaca por su robusta biblioteca estándar (`crypto`), para introducir al alumno en los fundamentos de la seguridad.

- **Capítulo 7. Concurrencia y Patrones Cloud Native**. En esta capítulo se abordará la característica estrella de Go. Se explorará el uso de goroutines, canales, el paquete `context` y patrones de diseño para la nube como *Worker Pools* y servidores HTTP.

- **Capítulo 8. Herramientas de Sistema y Automatización (CLI)**. En este capítulo nos centraremos en la ingeniería de sistemas. Aprenderemos a interactuar con el sistema operativo, procesar argumentos de línea de comandos, manipular ficheros de forma masiva y ejecutar procesos externos, creando herramientas de terminal robustas similares a las de UNIX.

- **Capítulo 9. Procesamiento de Imágenes y Esteganografía**. Por último, exploraremos las capacidades visuales de Go. Manipularemos imágenes a nivel de píxel (filtros, redimensionado, composición) y finalizaremos con un ejercicio avanzado de esteganografía para ocultar información secreta dentro de archivos gráficos.

Finalmente, el manual concluye con una **Recopilación Bibliográfica** de las fuentes más destacadas para repasar conceptos y sintaxis de programación en Go.

Herramientas de trabajo para programar en Go

Para la elaboración de los problemas de este manual, se ha utilizado el compilador oficial de Go (`gc`) [6] y el IDE Visual Studio Code de Microsoft con la extensión oficial de Go [13]. Ambos están disponibles para los tres sistemas operativos más importantes del mercado: Windows, macOS y Linux. Go favorece una gestión de proyectos simplificada. Para compilar y ejecutar los problemas, se debe crear el fichero con extensión `.go`. Go permite compilar y ejecutar en un solo paso durante el desarrollo o generar un binario nativo. Esta operación se puede hacer en la terminal utilizando los comandos "`go run fichero.go`" para ejecución directa, o "`go build fichero.go`" para generar el ejecutable. También se puede utilizar un IDE avanzado como GoLand [10] o VS Code.

Código fuente de los ejemplos

Para que el lector pueda utilizar más cómodamente los ejemplos de este libro, éstos han sido subidos a un repositorio público para que puedan ser descargados. Los ficheros `.go` están codificados con el juego de caracteres UTF-8 de forma nativa. En la siguiente dirección se podrán descargar los problemas resueltos.

`https://github.com/franmartinezmimbrera/problemasgo`

Capítulo 2

E/S por teclado y pantalla y operaciones aritméticas

En este capítulo se van a resolver una serie de problemas utilizando los distintos tipos de datos, operadores y expresiones en los lenguajes propuestos. Para estos problemas, las entradas de datos se realizarán mediante el teclado o con precarga de variables, mientras que las salidas se realizarán por pantalla. Los ejercicios resueltos que contiene servirán de base para familiarizarse con las operaciones básicas que se suelen realizar en un lenguaje de programación, así como para empezar a utilizar algunos paquetes externos de la biblioteca estándar, tales como el paquete para operaciones matemáticas. Como se ha resaltado en el capítulo de introducción, este manual no contiene teoría ni especificaciones profundas sobre el lenguaje de programación, mostrando únicamente soluciones a problemas propuestos. Para revisar la teoría y especificaciones propias del lenguaje Go para E/S por teclado y pantalla y operaciones aritméticas, se propone al lector revisar la documentación oficial en `golang.org` [1] o libros destacados como *The Go Programming Language* [22].

Nota: En estos ejercicios se utilizará el paquete `fmt` para la entrada y salida formateada. A diferencia de otros lenguajes, Go es muy estricto: declarar una variable o importar un paquete y no utilizarlo generará un error de compilación, lo que ayuda a mantener el código limpio. A continuación, se presentan los ejercicios resueltos para este capítulo.

2.1. Mi primer programa - Hola mundo

Este es el primer ejercicio/contacto de un programador con cualquier lenguaje de programación. Se pide realizar un programa que muestre *hola mundo* por pantalla.

```
// fichero holamundo.go
// Este programa muestra el saludo hola mundo por pantalla.
package main
import "fmt"

func main() {
    fmt.Println("¡Hola, Mundo!")
}
```

2.2. Saludo personalizado

Este ejercicio trata de capturar información desde el teclado (un nombre), para posteriormente almacenarla en una cadena de caracteres y mostrarla por pantalla en forma de saludo personalizado.

```
// fichero saludo.go
// Este programa hace un saludo personalizado
package main
import (
    "bufio"
    "fmt"
    "os"
)

func main() {

    fmt.Println("¡Hola! ¿Cómo te llamas?")
    scanner := bufio.NewScanner(os.Stdin)
    scanner.Scan()
    nombre := scanner.Text()
    fmt.Printf("¿Qué tal estás %s?\n", nombre)
}
```

2.3. Capturar distintos tipos de información desde teclado y mostrarla por pantalla

Este ejercicio trata de capturar distinto tipo de información desde el teclado, para posteriormente almacenarla en su variable correspondiente, y mostrar dicha información por pantalla.

```
// fichero capturar.go
// Captura información desde teclado y la muestra
package main
import (
    "bufio";"fmt";"os";"strconv";"strings"
)
func main() {
    var edad int ; var peso float64 ;var err error
    scanner := bufio.NewScanner(os.Stdin)
    fmt.Println("Dinos tu Edad, peso y color favorito:")
    fmt.Print("\n       Edad: ")
    if scanner.Scan() {
        textoEdad := scanner.Text()
        edad, err = strconv.Atoi(textoEdad)
        if err != nil {fmt.Println("Error: La edad debe ser un número entero.")
    ; os.Exit(1) }
    }
    fmt.Print("\n       Peso: ")
    if scanner.Scan() {
        textoPeso := scanner.Text()
        textoPeso = strings.Replace(textoPeso, ",", ".", 1)
        peso, err = strconv.ParseFloat(textoPeso, 64)
        if err != nil {
            fmt.Println("Error: El peso debe ser un número.")
            os.Exit(1)
        }
    }
    fmt.Print("\n Color favorito: ")
    scanner.Scan()
    color := scanner.Text()
    fmt.Printf("El %s!!! \n", color)
    fmt.Printf("¿Cómo puede gustarte el %s con %d años y pesando %.2fKg.?\n",
    color, edad, peso)
}
```

2.4. Calcular el área de un cuadrado a partir de un lado

Este ejercicio calcula el área de un cuadrado a partir de un lado dado por teclado y muestra el resultado por pantalla. La fórmula del área de un cuadrado [9] viene dada por $A = l^2$.

```
// fichero areacuadrado.go
// Este programa calcula el área de un cuadrado a partir de uno de sus lados
package main
import (
    "bufio";"fmt";"os";"strconv";"strings"
)
func main() {
    fmt.Print("Introduce valor lado de un cuadrado: ")
    scanner := bufio.NewScanner(os.Stdin)
    if scanner.Scan() {
        textoLado := scanner.Text()
        textoLado = strings.Replace(textoLado, ",", ".", 1)
        l1, err := strconv.ParseFloat(textoLado, 64)
        if err != nil {
            fmt.Println("Error: El lado debe de ser un número.")
            os.Exit(1)
        }
        if l1 <= 0 {
            fmt.Println("Error: El lado debe ser un número positivo >0")
            os.Exit(1)
        }
        area := l1 * l1
        fmt.Printf("El Área del cuadrado es: %v \n", area)
    }
}
```

2.5. Conversor de pesetas a euros

Este ejercicio propone crear un conversor de pesetas a euros. Partiendo de un número dado por teclado, se calculará su equivalencia a euros y se mostrará su cantidad expresada en euros por pantalla.

El cambio de euros a pesetas está fijado a razón de 1€ = 166,386 pesetas.

2.5. Conversor de pesetas a euros

```
// fichero pesetaseuros.go
// Este programa realiza la conversión de pesetas a euros
package main

import (
    "bufio"
    "fmt"
    "os"
    "strconv"
    "strings"
)

func main() {

    const tasaConversion = 166.386

    fmt.Print("Introduzca su valor en pesetas: ")
    scanner := bufio.NewScanner(os.Stdin)

    if scanner.Scan() {
        textoEntrada := scanner.Text()
        textoEntrada = strings.Replace(textoEntrada, ",", ".", 1)
        pesetas, err := strconv.ParseFloat(textoEntrada, 64)
        if err != nil {
            fmt.Println("ERROR: La entrada no es un número válido.")
            os.Exit(1)
        }
        if pesetas < 0 {
            fmt.Println("ERROR: El valor en pesetas no puede ser negativo.")
            os.Exit(1)
        }

        euros := pesetas / tasaConversion
        fmt.Printf("Su valor de %.2f pesetas equivale a: %.2f euros\n", pesetas
    , euros)
    }
}
```

2.6. Suma, resta, multiplicación y división de dos números

Este ejercicio propone, dados dos números por teclado, realizar todas las operaciones aritméticas básicas sobre ellos y mostrar los resultados por pantalla.

```
// fichero operacionesaritmeticas.go
// Este programa realiza operaciones aritméticas dados 2 números
package main
import ("bufio";"fmt";"os";"strconv";"strings")
func main() {
    var valor1, valor2 float64; var err error
    scanner := bufio.NewScanner(os.Stdin)
    fmt.Print("Introduce el valor primero: ")
    if scanner.Scan() {
        texto1 := strings.Replace(scanner.Text(), ",", ".", 1); valor1, err =
    strconv.ParseFloat(texto1, 64)
        if err != nil { fmt.Println("ERROR: El valor introducido no es un nú
    mero.");os.Exit(1)}
    }
    fmt.Print("Introduce el valor segundo: ")
    if scanner.Scan() {
        texto2 := strings.Replace(scanner.Text(), ",", ".", 1); valor2, err =
    strconv.ParseFloat(texto2, 64)
        if err != nil {fmt.Println("ERROR: El valor introducido no es un número
    .");os.Exit(1)}
    }
    fmt.Printf("El resultado de la suma (%v + %v) es: %v\n", valor1, valor2,
    valor1+valor2)
    fmt.Printf("El resultado de la resta (%v - %v) es: %v\n", valor1, valor2,
    valor1-valor2)
    fmt.Printf("El resultado de la multiplicación (%v * %v) es: %v\n", valor1,
    valor2, valor1*valor2)
    if valor2 == 0 {
        fmt.Printf("El resultado de la división (%v / %v) es: INDEFINIDO (
    División por Cero)\n", valor1, valor2)
    } else {
        fmt.Printf("El resultado de la división (%v / %v) es: %v\n", valor1,
    valor2, valor1/valor2)
    }
}
```

2.7. Precio medio de un producto en función de los precios de éste en otros establecimientos

Este ejercicio propone, dados por teclado los precios de un producto en 3 establecimientos, calcular el precio medio del producto y mostrarlo por pantalla.

```
// fichero preciomedio.go
// Este programa calcula la media de tres precios
package main
import ("bufio";"fmt";"os";"strconv";"strings")
func main() {
    var precio1, precio2, precio3 float64;var err error
    scanner := bufio.NewScanner(os.Stdin)
    fmt.Print("Introduzca el precio en establecimiento 1, en euros: ")
    if scanner.Scan() {
        texto := strings.Replace(scanner.Text(), ",", ".", 1); precio1, err =
    strconv.ParseFloat(texto, 64)
        if err != nil {fmt.Println("Error: El valor debe de ser un número.");os
    .Exit(1)}
        if precio1 < 0 {fmt.Println("Ningún precio puede ser negativo");os.Exit
    (1)}}
    fmt.Print("Introduzca el precio en establecimiento 2, en euros: ")
    if scanner.Scan() {
        texto := strings.Replace(scanner.Text(), ",", ".", 1);precio2, err =
    strconv.ParseFloat(texto, 64)
        if err != nil {fmt.Println("Error: El valor debe de ser un número.");os
    .Exit(1)}
        if precio2 < 0 {fmt.Println("Ningún precio puede ser negativo");os.Exit
    (1)}}
    fmt.Print("Introduzca el precio en establecimiento 3, en euros: ")
    if scanner.Scan() {
        texto := strings.Replace(scanner.Text(), ",", ".", 1);precio3, err =
    strconv.ParseFloat(texto, 64)
        if err != nil {fmt.Println("Error: El valor debe de ser un número.");os
    .Exit(1)}
        if precio3 < 0 {fmt.Println("Ningún precio puede ser negativo");os.Exit
    (1)}}
    media := (precio1 + precio2 + precio3) / 3.0
    fmt.Printf("El precio medio del producto es de %v euros\n", media)
}
```

2.8. Suma de los “n” primeros números naturales consecutivos

Este ejercicio propone, dado un número “n” por teclado, calcular los n primeros números naturales consecutivos y mostrar el resultado por pantalla. Para ello, se utilizará la fórmula de las series aritméticas notables de números naturales [9], conocida como la fórmula de Gauss:

$$\sum_{i=1}^{n} i = 1 + 2 + 3 + ... + n = \frac{n(n+1)}{2}$$

```
// fichero sumannumerosnaturales.go
// Este programa calcula la suma de los "n" primeros números naturales
package main
import (
    "bufio";"fmt";"os";"strconv"
)
func main() {
    var n int
    var suma int64
    fmt.Print("Introduzca número de números naturales a sumar: ")

    scanner := bufio.NewScanner(os.Stdin)
    if scanner.Scan() {
        texto := scanner.Text()
        val, err := strconv.Atoi(texto)
        if err != nil {
            fmt.Println("Error: El valor debe de ser un número.")
            os.Exit(1)
        }
        n = val
        if n <= 0 {
            fmt.Println("Error: El valor debe ser un número natural.")
            os.Exit(1)
        }
        suma = (int64(n) * (int64(n) + 1)) / 2
        fmt.Printf("La suma de los %d primeros números naturales es: %d\n", n,
    suma)
    }
}
```

2.9. Pasar de pulgadas a milímetros

Este ejercicio propone que, dado un valor numérico expresado en pulgadas por teclado, éste sea pasado a milímetros y el resultado mostrado por pantalla.

Para ello se tendrá en cuenta que 1 pulgada = 25.4 mm.

```
// fichero pulgadasmilimetros.go
// Este programa cambia pulgadas por milímetros
package main
import (
    "bufio"
    "fmt"
    "os"
    "strconv"
    "strings"
)
func main() {
    const factorConversion = 25.4

    fmt.Print("Introduzca valor en pulgadas: ")
    scanner := bufio.NewScanner(os.Stdin)
    if scanner.Scan() {
        texto := scanner.Text()
        texto = strings.Replace(texto, ",", ".", 1)
        pul, err := strconv.ParseFloat(texto, 64)
        if err != nil {
            fmt.Println("Error: El valor debe de ser un número.")
            os.Exit(1)
        }
        if pul < 0 {
            fmt.Println("Error: El valor debe ser un número positivo")
            os.Exit(1)
        }
        mil := factorConversion * pul
        fmt.Printf("El resultado en milímetros es: %v\n", mil)
    }
}
```

2.10. Pasar de grados Celsius a grados Fahrenheit

Este ejercicio propone que, dado por teclado un valor numérico expresado en grados Celsius (°C) (históricamente conocidos como centígrados), éste sea pasado a grados Fahrenheit (°F) y mostrado por pantalla.

Para ello se tendrá en cuenta que:

$$°F = \frac{9}{5}°C + 32$$

```
// fichero cambiagrados.go
// Este programa cambia grados centígrados por Fahrenheit
package main

import (
    "bufio"
    "fmt"
    "os"
    "strconv"
    "strings"
)

func main() {

  const factorCToF = 9.0 / 5.0
    fmt.Print("Introduzca valor en grados Centígrados : ")
    scanner := bufio.NewScanner(os.Stdin)

    if scanner.Scan() {
        texto := scanner.Text()
        texto = strings.Replace(texto, ",", ".", 1)
        c, err := strconv.ParseFloat(texto, 64)
        if err != nil {
            fmt.Println("Error: El valor debe de ser un número.")
            os.Exit(1)
        }
        f := (factorCToF * c) + 32
        fmt.Printf("El resultado en grados Fahrenheit es: %v\n", f)
    }
}
```

2.11. Calcular estadísticas de suspensos, aprobados, notables y sobresalientes de la clase

Este ejercicio propone, dados por teclado el número de suspensos, aprobados, notables y sobresalientes de una clase, calcular:

- El porcentaje de alumnos que superan la asignatura.
- El porcentaje de alumnos que suspenden la asignatura.
- El porcentaje de alumnos que sacan notable en la asignatura.
- El porcentaje de alumnos que sacan sobresaliente en la asignatura.
- El porcentaje de alumnos que han sacado un aprobado en la asignatura.

```
// fichero estadisticas.go
// Este programa calcula estadísticas sobre alumnos
package main
import ("bufio";"fmt";"os";"strconv";"strings")
func main() {
    var suspensos, aprobados, notables, sobresalientes float64; var err error
    scanner := bufio.NewScanner(os.Stdin)
    fmt.Print("Introduce el número total de suspensos: ")
    if scanner.Scan() {
        texto := strings.Replace(scanner.Text(), ",", ".", 1)
        suspensos, err = strconv.ParseFloat(texto, 64)
        if err != nil {fmt.Println("Error: El valor debe de ser un número.");os
    .Exit(1)}
        if suspensos < 0 {fmt.Println("Error: El valor debe ser un número
    positivo");os.Exit(1)}
    }
    fmt.Print("Introduce el número total de aprobados: ")
    if scanner.Scan() {
        texto := strings.Replace(scanner.Text(), ",", ".", 1)
        aprobados, err = strconv.ParseFloat(texto, 64)
        if err != nil {fmt.Println("Error: El valor debe de ser un número.");os
    .Exit(1)}
        if aprobados < 0 {fmt.Println("Error: El valor debe ser un número
    positivo");os.Exit(1)}
    }
    fmt.Print("Introduce el número total de notables: ")
    if scanner.Scan() {
```

```
        texto := strings.Replace(scanner.Text(), ",", ".", 1)
        notables, err = strconv.ParseFloat(texto, 64)
        if err != nil {fmt.Println("Error: El valor debe de ser un número.");os
    .Exit(1)}
        if notables < 0 {fmt.Println("Error: El valor debe ser un número
    positivo");os.Exit(1)}
    }
    fmt.Print("Introduce el número total de sobresalientes: ")
    if scanner.Scan() {
        texto := strings.Replace(scanner.Text(), ",", ".", 1)
        sobresalientes, err = strconv.ParseFloat(texto, 64)
        if err != nil {fmt.Println("Error: El valor debe de ser un número.");os
    .Exit(1)}
        if sobresalientes < 0 {fmt.Println("Error: El valor debe ser un número
    positivo");os.Exit(1)}
    }
    totalalumnos := suspensos + aprobados + notables + sobresalientes
    if totalalumnos == 0 {
      fmt.Println("No se han introducido alumnos (Total 0), no se pueden
    calcular estadísticas.")
        os.Exit(1)
    }
    poraprot := ((aprobados + notables + sobresalientes) / totalalumnos) * 100
    porsus := (suspensos / totalalumnos) * 100
    pornot := (notables / totalalumnos) * 100
    porsobre := (sobresalientes / totalalumnos) * 100
    porapro := (aprobados / totalalumnos) * 100
    fmt.Printf("El porcentaje de alumnos que superan la asignatura es: %v\n",
    poraprot)
    fmt.Printf("El porcentaje de alumnos que suspenden la asignatura es: %v\n",
     porsus)
    fmt.Printf("El porcentaje de alumnos que sacan notable en la asignatura es:
     %v\n", pornot)
    fmt.Printf("El porcentaje de alumnos que sobresalen en la asignatura es: %v
    \n", porsobre)
    fmt.Printf("El porcentaje de alumnos que han sacado un aprobado en la
    asignatura es: %v\n", porapro)
}
```

2.12. Cálculo del precio venta de un coche a partir del coste de fabricación, ganancia de la empresa e IVA

Este ejercicio propone, dado por teclado un valor numérico que expresa el coste de fabricar un coche, calcular el precio total de venta al que tendrían que venderlo, si se desea una ganancia fija de un 15 % y el valor del IVA del coche es de 21 %.

```
// fichero costecoche.cpp
// Este programa calcula el precio total de un coche

#include <iostream>
#include <cstdlib>
#include <iomanip>
using namespace std;
const double ganancia = 1.15;
const double impuesto = 1.21;

int main() {

   double costecoche,pvantesimpuestos,preciototal;

   cout<<"Introduce el coste del vehiculo: ";
   if (!(cin >> costecoche)) {
        cout << "Error: El valor debe de ser un número.\n";
        return EXIT_FAILURE;
   }
   if (costecoche < 0.0) {
        cout << "Error: El valor debe ser un número positivo\n";
        return EXIT_FAILURE;
   }
   pvantesimpuestos= costecoche*ganancia;
   preciototal= pvantesimpuestos*impuesto;
   cout << fixed << setprecision(2);
   cout<<"El precio total del automovil nuevo para el comprador es:"<<
    preciototal <<"\n";
   return EXIT_SUCCESS;
}
```

2.13. Calcular el perímetro de una circunferencia a partir del radio

Este ejercicio propone, dado por teclado el radio de una circunferencia, calcular el perímetro de la misma y mostrarlo por pantalla. Para ello se tendrá en cuenta que el perímetro de una circunferencia [9] viene dado por la siguiente fórmula $P = 2\pi r$.

```
// fichero perimetrocir.go
// Este programa calcula el perímetro de una circunferencia
package main

import ("bufio";"fmt";"math";"os";"strconv";"strings")
func main() {
    fmt.Print("Introduzca el radio: ")
    scanner := bufio.NewScanner(os.Stdin)
    if scanner.Scan() {
        texto := scanner.Text()
        texto = strings.Replace(texto, ",", ".", 1)
        radio, err := strconv.ParseFloat(texto, 64)
        if err != nil {
            fmt.Println("Error: El valor debe de ser un número.")
            os.Exit(1)
        }
        if radio <= 0 {
            fmt.Println("Error: El radio debe ser un número >=0")
            os.Exit(1)
        }
        perimetro := 2 * math.Pi * radio
        fmt.Printf("El perímetro es: %v\n", perimetro)
    }
}
```

2.14. Calcular la raíz cuadrada de un número dado

Este ejercicio propone, dado por teclado un número, calcular su raíz cuadrada y mostrarla por pantalla. Para ello se utilizará la función `math.Sqrt` que proporciona el paquete `math` de la biblioteca estándar de Go.

```
// fichero raizcuadrada.go
// Este programa calcula la raíz cuadrada de un número
package main

import (
    "bufio"
    "fmt"
    "math"
    "os"
    "strconv"
    "strings"
)

func main() {
    fmt.Print("Introduce el número a calcular la raíz cuadrada: ")
    scanner := bufio.NewScanner(os.Stdin)
    if scanner.Scan() {
        texto := scanner.Text()
        texto = strings.Replace(texto, ",", ".", 1)
        numero, err := strconv.ParseFloat(texto, 64)
        if err != nil {
            fmt.Println("Error: El valor debe de ser un número.")
            os.Exit(1)
        }
        if numero < 0 {
            fmt.Println("Error: El valor debe ser un número positivo")
            os.Exit(1)
        }
        resultado := math.Sqrt(numero)
        fmt.Printf("La raíz cuadrada de %v es: %v\n", numero, resultado)
    }
}
```

2.15. Calcular la raíz n-ésima de un número

Este ejercicio propone, dados por teclado un número entero y un exponente, calcular la raíz n-ésima de dicho número y mostrarla por pantalla. Para ello, debemos tener en cuenta que la raíz n-ésima de un número $\sqrt[n]{a}$ se puede reescribir como $a^{\frac{1}{n}}$ [9]. En Go se utilizará la función `math.Pow`.

```
// fichero raizenesima.go
// Este programa calcula la raíz n-ésima de un número
package main
import ("bufio";"fmt":"math";"os";"strconv";"strings")
func main() {
    var numero, resultado float64;var exponente int;var err error
    scanner := bufio.NewScanner(os.Stdin)
    fmt.Print("Introduce el número a calcular la raíz: ")
    if scanner.Scan() {
        texto := strings.Replace(scanner.Text(), ",", ".", 1)
        numero, err = strconv.ParseFloat(texto, 64)
        if err != nil {fmt.Println("Error: El valor debe de ser un número.");os
    .Exit(1)}
    }
    fmt.Print("Introduce exponente de raíz (un entero, ej: 2, 3..): ")
    if scanner.Scan() {
        texto := scanner.Text()
        exponente, err = strconv.Atoi(texto)
        if err != nil {fmt.Println("Error: El valor debe de ser un número
    entero.");os.Exit(1)}
    }
    if exponente == 0 {fmt.Println("Error: El exponente de la raíz no puede ser
     0");os.Exit(1)}
    if numero < 0 && exponente%2 == 0 {
        fmt.Println("Imposible cálculo de raíz par de un número < 0 en R")
        os.Exit(1)
    }
    basePositiva := math.Abs(numero)
    potencia := 1.0 / float64(exponente)
    resultado = math.Pow(basePositiva, potencia)
    if numero < 0 && exponente%2 != 0 { resultado = -resultado}
    fmt.Printf("La raíz %d ésima de %v es: %v\n", exponente, numero, resultado)
}
```

2.16. Calcular el área de un triángulo rectángulo

Este ejercicio propone, dados la base de un triángulo y la altura por teclado, calcular el área del mismo y mostrarla por pantalla.

Se utilizará la fórmula del área de un triángulo rectángulo [20]:

$$A = \frac{b \cdot a}{2}$$

```
// fichero areatriangulo1.go
// Este programa calcula el área de un triángulo rectángulo a partir de la base
     y la altura
package main
import ("bufio";"fmt";"os";"strconv";"strings")
func main() {
    var base, altura float64
    var err error
    scanner := bufio.NewScanner(os.Stdin)
    fmt.Print("Introduce la base del triángulo rectángulo: ")
    if scanner.Scan() {
        texto := strings.Replace(scanner.Text(), ",", ".", 1)
        base, err = strconv.ParseFloat(texto, 64)
        if err != nil {fmt.Println("Error: El valor debe de ser un número.");os
    .Exit(1)}
        if base <= 0 {fmt.Println("Error: El valor debe ser un número > que 0")
    ;os.Exit(1)}
    }
    fmt.Print("Introduce la altura del triángulo rectángulo: ")
    if scanner.Scan() {
        texto := strings.Replace(scanner.Text(), ",", ".", 1)
        altura, err = strconv.ParseFloat(texto, 64)
        if err != nil {fmt.Println("Error: El valor debe de ser un número.");os
    .Exit(1)}
        if altura <= 0 {fmt.Println("Error: El valor debe ser un número > que 0
    ");os.Exit(1)}
    }
    area := (base * altura) / 2.0
    fmt.Printf("El area del triángulo rectángulo es: %.2f\n", area)
}
```

2.17. Calcular el área de un triángulo equilátero

Este ejercicio propone, dado un lado de un triángulo equilátero por teclado, calcular el área del mismo y mostrarla por pantalla. Se utilizará la fórmula del área de un triángulo equilátero [20]:

$$A = \frac{\sqrt{3}}{4} \cdot a^2$$

```
// fichero areatriangulo2.go
// Este programa calcula el Área de un triángulo equilátero a partir de uno de
    sus lados
package main
import (
    "bufio"
    "fmt"
    "math"
    "os"
    "strconv"
    "strings"
)
func main() {
    var lado float64
    var err error
    fmt.Print("Introduce un lado del triángulo equilátero: ")
    scanner := bufio.NewScanner(os.Stdin)
    if scanner.Scan() {
        texto := scanner.Text()
        texto = strings.Replace(texto, ",", ".", 1)
        lado, err = strconv.ParseFloat(texto, 64)
        if err != nil {
            fmt.Println("Error: El valor debe ser un número.")
            os.Exit(1)
        }
        if lado <= 0 {
            fmt.Println("Error: El valor debe ser un número mayor que 0.")
            os.Exit(1)
        }
        area := (math.Sqrt(3.0) / 4.0) * lado * lado
        fmt.Printf("El Área del triángulo equilátero de lado %v es: %v\n", lado
    , area)
    }
}
```

2.18. Calcular el área de cualquier triángulo a partir de sus lados mediante la fórmula de Herón

Este ejercicio propone, dados por teclado los valores de los lados de un triángulo, calcular su área y mostrarla por pantalla utilizando la fórmula de Herón. El área de un triángulo según la fórmula de Herón [20] viene dada por:

$$A = \sqrt{s(s-l_1)(s-l_2)(s-l_3)}$$

donde

$$s = \frac{(l_1 + l_2 + l_3)}{2}$$

```
// fichero areatriangulo.go
// Calculamos el área de un triángulo mediante la fórmula de Herón
package main
import (
    "bufio"
    "fmt"
    "math"
    "os"
    "strconv"
    "strings"
)
func main() {
    var l1, l2, l3 float64
    var err error
    scanner := bufio.NewScanner(os.Stdin)
    fmt.Print("Introduce lo que mide el primer lado del triángulo: ")
    if scanner.Scan() {
        texto := strings.Replace(scanner.Text(), ",", ".", 1)
        l1, err = strconv.ParseFloat(texto, 64)
        if err != nil {
            fmt.Println("Error: El valor debe de ser un número.")
            os.Exit(1)
        }
        if l1 < 0 {
            fmt.Println("Error: El valor debe ser un número positivo")
            os.Exit(1)
        }
    }
    fmt.Print("Introduce lo que mide el segundo lado del triángulo: ")
```

```
    if scanner.Scan() {
        texto := strings.Replace(scanner.Text(), ",", ".", 1)
        l2, err = strconv.ParseFloat(texto, 64)
        if err != nil {
            fmt.Println("Error: El valor debe de ser un número.")
            os.Exit(1)
        }
        if l2 < 0 {
            fmt.Println("Error: El valor debe ser un número positivo")
            os.Exit(1)
        }
    }
    fmt.Print("Introduce lo que mide el tercer lado del triángulo: ")
    if scanner.Scan() {
        texto := strings.Replace(scanner.Text(), ",", ".", 1)
        l3, err = strconv.ParseFloat(texto, 64)
        if err != nil {
            fmt.Println("Error: El valor debe de ser un número.")
            os.Exit(1)
        }
        if l3 < 0 {
            fmt.Println("Error: El valor debe ser un número positivo")
            os.Exit(1)
        }
    }
    if l1+l2 <= l3 || l1+l3 <= l2 || l2+l3 <= l1 {
        fmt.Printf("\nError: Las longitudes (%v, %v, %v) no forman un triángulo
     válido (desigualdad triangular).\n", l1, l2, l3)
        os.Exit(1)
    }
    sp := (l1 + l2 + l3) / 2.0
    area := math.Sqrt(sp * (sp - l1) * (sp - l2) * (sp - l3))
    fmt.Printf("El area del triángulo es: %.4f\n", area)
}
```

2.19. Calcular el volumen de un cilindro a partir de su altura y diámetro

Este ejercicio propone, dados por teclado el diámetro y la altura de un cilindro, calcular el volumen del mismo y mostrarlo por pantalla. Para ello se tendrá en cuenta que $V = \pi \cdot r^2 \cdot H$ [9], siendo r el radio de la circunferencia y H la altura de la misma. El radio se obtiene a partir del diámetro tal que $r = d/2$.

```
// fichero volumencilindro.go
// Calculamos el volumen de un cilindro
package main
import ("bufio";"fmt";"math";"os";"strconv";"strings")
func main() {
    var D, H, R, V float64; var err error
    scanner := bufio.NewScanner(os.Stdin)
    fmt.Print("Introduzca el diámetro, en metros: ")
    if scanner.Scan() {
        texto := strings.Replace(scanner.Text(), ",", ".", 1)
        D, err = strconv.ParseFloat(texto, 64)
        if err != nil {
            fmt.Println("Error: El valor debe ser un número.")
            os.Exit(1)
        }
    }
    fmt.Print("Introduzca la altura, en metros: ")
    if scanner.Scan() {
        texto := strings.Replace(scanner.Text(), ",", ".", 1)
        H, err = strconv.ParseFloat(texto, 64)
        if err != nil {
            fmt.Println("Error: El valor debe ser un número.")
            os.Exit(1)
        }
    }
    if H <= 0 || D <= 0 {
        fmt.Println("Error: El diámetro y la altura debe ser un número > que 0"
    )
        os.Exit(1)
    }
    R = D / 2.0;V = math.Pi * R * R * H
    fmt.Printf("El volumen del cilindro es de %v m^3\n", V)
}
```

2.20. Calcular el valor del lado “a” de un triángulo rectángulo dados el valor de la hipotenusa “h” y del lado “b”

Este ejercicio propone, dados por teclado el valor del lado “b” de un triángulo rectángulo y el valor de su hipotenusa “h”, calcular el valor del lado “a” y mostrarlo por pantalla. Para ello se utilizará el Teorema de Pitágoras $h^2 = a^2 + b^2$ [20] que despejando nos queda que $a = \sqrt{h^2 - b^2}$.

```
// fichero ladotriangulo.go
// Este programa calcula el valor del lado a de un triángulo rectángulo
package main
import ("bufio";"fmt";"math";"os";"strconv";"strings")
func main() {
    var a, b, h float64;var err error
    scanner := bufio.NewScanner(os.Stdin)
    fmt.Print("Introduzca el valor del lado \"b\" del triángulo rectángulo: ")
    if scanner.Scan() {
        texto := strings.Replace(scanner.Text(), ",", ".", 1)
        b, err = strconv.ParseFloat(texto, 64)
        if err != nil {fmt.Println("Error: El valor debe ser un número.");os.
    Exit(1)}
        if b <= 0 {fmt.Println("Error: El lado 'b' debe ser un número mayor que
     0.");os.Exit(1)}
    }
    fmt.Print("Introduzca el valor de la hipotenusa del triángulo rectángulo: "
    )
    if scanner.Scan() {
        texto := strings.Replace(scanner.Text(), ",", ".", 1)
        h, err = strconv.ParseFloat(texto, 64)
        if err != nil {fmt.Println("Error: El valor debe ser un número.");os.
    Exit(1)}
        if h <= 0 {fmt.Println("Error: La hipotenusa debe ser un número mayor
    que 0.");os.Exit(1)}
    }
    if b >= h {
        fmt.Printf("Error: La hipotenusa (%v) debe ser mayor que el cateto 'b'
    (%v).\n", h, b)
        os.Exit(1)
    }
    a = math.Sqrt((h * h) - (b * b))
    fmt.Printf("El valor del lado a del triángulo rectángulo es: %v\n", a)
}
```

2.21. Calcular la altura de un triángulo equilátero

Este ejercicio propone, dado por teclado el lado de un triángulo equilátero, calcular la altura h del mismo y mostrar el resultado por pantalla. Para calcular la altura de un triángulo equilátero se aplicará el Teorema de Pitágoras obteniendo que $h = \frac{\sqrt{3}a}{2}$ [20].

```
// fichero alturatriangulo.go
// Este programa calcula la altura de un triángulo equilátero
package main
import ("bufio";"fmt";"math";"os";"strconv";"strings")

func main() {
    var l, h float64;var err error

    fmt.Print("Introduzca lado de un triángulo equilátero: ")
    scanner := bufio.NewScanner(os.Stdin)

    if scanner.Scan() {
        texto := scanner.Text()
        texto = strings.Replace(texto, ",", ".", 1)
        l, err = strconv.ParseFloat(texto, 64)
        if err != nil {
            fmt.Println("Error: El valor debe ser un número.")
            os.Exit(1)
        }
        if l <= 0 {
            fmt.Println("Error: El lado de un triángulo equilátero debe ser un
    número mayor que 0.")
            os.Exit(1)
        }
        h = (math.Sqrt(3) * l) / 2.0
        fmt.Printf("La altura de un triángulo equilátero de lado %v es: %v\n",
    l, h)
    }
}
```

2.22. Obtener el valor del binomio de suma al cuadrado $(a+b)^2$

Este ejercicio propone, dados por teclado dos números, calcular el binomio de suma al cuadrado $(a+b)^2$ [9] y mostrar el resultado por pantalla.

```
// fichero binomiosuma.go
// Este programa calcula el binomio de suma de a+b al cuadrado
package main
import (
    "bufio"
    "fmt"
    "os"
    "strconv"
    "strings"
)
func main() {
    var a, b, resultado float64
    var err error
    scanner := bufio.NewScanner(os.Stdin)
    fmt.Print("Introduce el valor de a: ")
    if scanner.Scan() {
        texto := strings.Replace(scanner.Text(), ",", ".", 1)
        a, err = strconv.ParseFloat(texto, 64)
        if err != nil {
            fmt.Println("Error: El valor debe ser un número.")
            os.Exit(1)
        }
    }
    fmt.Print("Introduce el valor de b: ")
    if scanner.Scan() {
        texto := strings.Replace(scanner.Text(), ",", ".", 1)
        b, err = strconv.ParseFloat(texto, 64)
        if err != nil {
            fmt.Println("Error: El valor debe ser un número.")
            os.Exit(1)
        }
    }
    resultado = (a * a) + (b * b) + (2.0 * a * b)
    fmt.Printf("El resultado del binomio de suma de (a+b)^2 es: %v\n",
    resultado)
}
```

Capítulo 3

Flujos de control, funciones, cadenas de caracteres, arrays y slices

Este capítulo presenta una serie de problemas resueltos que ilustran distintos tipos de flujos de control en Go. También se incluyen ejercicios sobre cómo resolver problemas utilizando funciones, así como su integración en el paquete principal (`main`). Por último, se abordarán problemas que implican el trabajo con cadenas de caracteres (*strings*), arrays y slices (rodajas).

A diferencia de C, C++, Java, etc., Go distingue entre **arrays** (tamaño fijo) y **slices** (tamaño dinámico). Para el trabajo con estos elementos, en estos ejercicios introductorios, se utilizarán mayoritariamente arrays de tamaño fijo para simular el comportamiento estático clásico, aunque se introducirá el uso de slices donde sea la práctica idiomática estándar del lenguaje. No se utilizarán paquetes externos complejos más allá de la biblioteca estándar.

Para revisar la teoría y especificaciones propias del lenguaje Go para abordar flujos de control, funciones y estructuras de datos, se propone al lector revisar textos fundamentales como *The Go Programming Language* [3], así como obras modernas enfocadas en la escritura de código idiomático y buenas prácticas, tales como *Learning Go* [2] o la guía de errores comunes *100 Go Mistakes* [8].

A continuación, se presentan los ejercicios resueltos para este capítulo.

3.1. ¿Número par o impar?

Este ejercicio propone, dado un número entero, calcular si éste es par o impar y mostrarlo por pantalla. Para ello se utilizará el flujo de control IF - ELSE.

```
// fichero parimpar.go
// Este programa dice si un número es par o impar
package main
import ("bufio";"fmt";"os";"strconv";"strings")
func main() {
    var numero int64;var err error
    fmt.Print("Introduzca un número entero: ")
    scanner := bufio.NewScanner(os.Stdin)
    if scanner.Scan() {
        texto := strings.TrimSpace(scanner.Text())
        numero, err = strconv.ParseInt(texto, 10, 64)
        if err != nil {
            fmt.Println("Entrada no válida (no se introdujo un número entero)")
            os.Exit(1)
        }
        if numero%2 == 0 {
            fmt.Println("ES PAR")
        } else {
            fmt.Println("ES IMPAR")
        }
    }
}
```

3.2. Calcular si un número es múltiplo de 3 y a la vez par

Este ejercicio propone, dado un número, calcular si es múltiplo de 3 y a la vez par y mostrarlo por pantalla. En Go, los operadores lógicos (&&, ||) funcionan de manera idéntica a C.

```
// fichero multiplo3.go
// Este programa dice si un número es múltiplo de 3 y a la vez par
package main
import ("bufio";"fmt";"os";"strconv";"strings")
func main() {
    var numero int64;var err error
    fmt.Print("Introduzca un número: ")
```

```
    scanner := bufio.NewScanner(os.Stdin)
    if scanner.Scan() {
        texto := strings.TrimSpace(scanner.Text())
        numero, err = strconv.ParseInt(texto, 10, 64)
        if err != nil {fmt.Println("ERROR: Entrada no válida (no se introdujo
    un número)");os.Exit(1)}
        if (numero%3 == 0) && (numero%2 == 0) {
            fmt.Printf("\n El número %d es MÚLTIPLO DE 3 Y TAMBIÉN ES PAR (Mú
    ltiplo de 6) \n", numero)
        } else if numero%3 == 0 {fmt.Printf("El número %d ES MÚLTIPLO DE 3\n",
    numero)
          } else {fmt.Printf("El número %d NO ES MÚLTIPLO DE 3\n", numero)}
    }
}
```

3.3. Calcular si un año es bisiesto

Este ejercicio propone, dado un año, calcular si es bisiesto o no y mostrarlo por pantalla. **Nota:** Un año es bisiesto en el calendario Gregoriano si es divisible entre 4, excepto aquellos divisibles entre 100 pero no entre 400.

```
// fichero bisiesto.go
// Este programa dice si un año es bisiesto o no
package main
import ("bufio";"fmt";"os";"strconv";"strings")
func main() {
    var anio int64;var err error
    fmt.Print("Introduzca un año: ")
    scanner := bufio.NewScanner(os.Stdin)
    if scanner.Scan() {
        texto := strings.TrimSpace(scanner.Text())
        anio, err = strconv.ParseInt(texto, 10, 64)
        if err != nil {fmt.Println("ERROR: La entrada no es un número entero");
    os.Exit(1)}
        if (anio%4 == 0 && anio%100 != 0) || (anio%400 == 0) {fmt.Println("ES
    BISIESTO")
        } else {fmt.Println("NO ES BISIESTO")}
    }
}
```

3.4. Programa que calcule el Máximo Común Divisor de dos números enteros no negativos mediante el algoritmo de Euclides

Este ejercicio propone, dados dos números enteros no negativos, calcular el máximo común divisor de dichos números utilizando el algoritmo de Euclides (más eficiente) y mostrarlo por pantalla. El **Algoritmo de Euclides** es un método eficiente para encontrar el Máximo Común Divisor (**MCD**) de dos números enteros no negativos a y b [9]. Se apoya en el principio de que el MCD de dos números también divide a su diferencia, lo que permite reducir el problema a números más pequeños.

El algoritmo se basa en la siguiente propiedad [9]: Si a y b son dos números enteros positivos con $a > b$, entonces MCD$(a, b) =$ MCD(b, r), donde r es el resto de la división euclídea de a entre b. Es decir: MCD$(a, b) =$ MCD$(b, a \bmod b)$

```
// fichero mcd.go
// Este programa calcula el MCD dados 2 números enteros usando el Algoritmo de
    Euclides
package main
import ("bufio";"fmt";"os";"strconv";"strings")
func main() {
    var a, b, a2, b2, temporal int64;var err error
    scanner := bufio.NewScanner(os.Stdin)
    fmt.Print("\n   Introduzca valor de a: ")
    if scanner.Scan() {
        texto := strings.TrimSpace(scanner.Text())
        a, err = strconv.ParseInt(texto, 10, 64)
        if err != nil {
            fmt.Println("ERROR: La entrada no es un número entero")
            os.Exit(1)
        }
    }
    fmt.Print("\n  Introduzca valor de b: ")
    if scanner.Scan() {
        texto := strings.TrimSpace(scanner.Text())
        b, err = strconv.ParseInt(texto, 10, 64)
        if err != nil {
            fmt.Println("ERROR: La entrada no es un número entero")
            os.Exit(1)
        }
    }
```

```
    a2 = a
    b2 = b
    if a < 0 {a = -a}
    if b < 0 {b = -b}
    if a == 0 || b == 0 {
        resultado := a
        if a == 0 {
            resultado = b
        }
        fmt.Printf("\n EL M.C.D de %d y %d es: %d\n", a2, b2, resultado)
        return
    }
    // ALGORITMO DE EUCLIDES
    for a != 0 {
        temporal = a
        a = b % a
        b = temporal
    }
    fmt.Printf(" EL M.C.D de %d y %d es: %d\n", a2, b2, b)
}
```

3.5. Función para calcular el Máximo Común Divisor de dos números enteros no negativos mediante el algoritmo de Euclides

Este ejercicio propone crear una función para, dados dos números enteros no negativos, calcular el máximo común divisor de dichos números utilizando el algoritmo de Euclides (visto en el ejercicio anterior) y que devuelva el resultado. En Go, las funciones se definen con la palabra clave `func`.

```
// fichero mcd1.go
// Este programa calcula el MCD dados 2 números mediante una función
package main
import ("bufio";"fmt";"os";"strconv";"strings")
// MCD calcula el Máximo Común Divisor usando el algoritmo de Euclides
func MCD(a, b int64) int64 {
    var temporal int64
    if a == 0 {
```

```
        return b
    }
    if b == 0 {
        return a
    }
    // Algoritmo de Euclides
    for a > 0 {
        temporal = a
        a = b % a
        b = temporal
    }
    return b
}
func main() {
    var a, b int64;var err error
    scanner := bufio.NewScanner(os.Stdin)
    fmt.Print("\n   Introduzca valor de a: ")
    if scanner.Scan() {
        texto := strings.TrimSpace(scanner.Text())
        a, err = strconv.ParseInt(texto, 10, 64)
        if err != nil {
            fmt.Println("ERROR: La entrada no es un número entero")
            os.Exit(1)
        }
    }
    fmt.Print("\n  Introduzca valor de b: ")
    if scanner.Scan() {
        texto := strings.TrimSpace(scanner.Text())
        b, err = strconv.ParseInt(texto, 10, 64)
        if err != nil {
            fmt.Println("ERROR: La entrada no es un número entero")
            os.Exit(1)
        }
    }
    if a < 0 {a = -a}
    if b < 0 {b = -b}
    fmt.Printf("EL M.C.D de %d y %d es: %d\n", a, b, MCD(a, b))
}
```

3.6. Función para calcular si un número entero es divisor de otro

Este ejercicio propone realizar una función que calcule si a es divisor de b. Un número entero d es divisor de otro número entero N si, al dividir N por d, el resto de la división es cero [9]. Para ello, se utilizará el operador módulo % de Go.

```
// fichero divisor.go
// Este programa calcula, dados 2 números, si a es divisor de b
package main
import ("bufio";"fmt";"os";"strconv";"strings")

// esDivisor comprueba si 'a' es divisor de 'b'
func esDivisor(a, b int64) bool {
    if a == 0 {return false}
    if a < 0 {a = -a}
    if b < 0 {b = -b}
    return b%a == 0
}
func main() {
    var a, b int64;var err error
    scanner := bufio.NewScanner(os.Stdin)
    fmt.Print("Introduzca valor de a (el divisor): ")
    if scanner.Scan() {
        texto := strings.TrimSpace(scanner.Text())
        a, err = strconv.ParseInt(texto, 10, 64)
        if err != nil {fmt.Println("ERROR: La entrada para 'a' no es un número
    entero válido.");os.Exit(1)}
    }
    fmt.Print("Introduzca valor de b (el dividendo): ")
    if scanner.Scan() {
        texto := strings.TrimSpace(scanner.Text())
        b, err = strconv.ParseInt(texto, 10, 64)
        if err != nil {fmt.Println("ERROR: La entrada para 'b' no es un número
    entero válido.");os.Exit(1)}
    }
    if esDivisor(a, b) {
        fmt.Printf("%d ES divisor de %d\n", a, b)
    } else {
        fmt.Printf("%d NO es divisor de %d\n", a, b)
    }
}
```

3.7. Pasar a función el ejercicio propuesto número 2.18

Este ejercicio toma como base el ejercicio 2.18 para crear una función que devuelva el valor del área de un triángulo usando la fórmula de Herón [20]. En Go, se utilizará el paquete `math` para la raíz cuadrada.

```
// fichero areatriangulo2.go
// Este programa calcula el área de un triángulo con la fórmula de Herón
package main
import ("bufio";"fmt";"os";"strconv";"strings";"math")
func AreaTrianguloHeron(l1, l2, l3 float64) float64 {
    if l1 <= 0 || l2 <= 0 || l3 <= 0 || l1+l2 <= l3 || l1+l3 <= l2 || l2+l3 <=
    l1 {
        return 0.0
    }
    sp := (l1 + l2 + l3) / 2.0
    // Fórmula de Herón: Raíz(s * (s-a) * (s-b) * (s-c))
    area := math.Sqrt(sp * (sp - l1) * (sp - l2) * (sp - l3))
    return area
}
func main() {
    var l1, l2, l3 float64;var err error
    scanner := bufio.NewScanner(os.Stdin)
    fmt.Print("Introduce lo que mide el primer lado del triángulo: ")
    if scanner.Scan() {
        l1, err = strconv.ParseFloat(strings.TrimSpace(scanner.Text()), 64)
        if err != nil {fmt.Println("ERROR: Entrada no válida (no es un número).
    ");os.Exit(1)}
    }
    fmt.Print("Introduce lo que mide el segundo lado del triángulo: ")
    if scanner.Scan() {
        l2, err = strconv.ParseFloat(strings.TrimSpace(scanner.Text()), 64)
        if err != nil {fmt.Println("ERROR: Entrada no válida (no es un número).
    ");os.Exit(1)}
    }
    fmt.Print("Introduce lo que mide el tercer lado del triángulo: ")
    if scanner.Scan() {
        l3, err = strconv.ParseFloat(strings.TrimSpace(scanner.Text()), 64)
        if err != nil {fmt.Println("ERROR: Entrada no válida (no es un número).
    ");os.Exit(1)}
    }
    areaFinal := AreaTrianguloHeron(l1, l2, l3)
```

```
    if areaFinal > 0.0 {
        fmt.Printf("El área del triángulo es: %.2f\n", areaFinal)
    } else {
        fmt.Println("Los lados introducidos NO forman un triángulo válido (la
    suma de 2 lados debe ser > que el tercero, y todos deben ser >=0)")
    }
}
```

3.8. Función para calcular el área de una circunferencia de radio válido

Este ejercicio propone realizar una función para calcular el área de una circunferencia [9]. Se hará uso de la constante `math.Pi` que proporciona mayor precisión que definirla manualmente.

```
// fichero areacircunferencia.go
// Este programa calcula el Área de una circunferencia
package main
import ("bufio";"fmt";"os";"strconv";"strings";"math")
func AreaCircunferencia(r float64) float64 {
    return math.Pi * math.Pow(r, 2)
}
func main() {
    var radio float64;var err error
    scanner := bufio.NewScanner(os.Stdin)
    for {
        fmt.Print("Introduce el radio de la circunferencia >0: ")
        if !scanner.Scan() {break }
        texto := strings.TrimSpace(scanner.Text())
        radio, err = strconv.ParseFloat(texto, 64)
        if err != nil {fmt.Println("ERROR: La entrada no es un número válido.")
    ;continue }
        if radio <= 0 {fmt.Println("ERROR: El radio debe ser mayor que cero");
    continue }
        break
    }
    area := AreaCircunferencia(radio)
    fmt.Printf("El área de la circunferencia es: %f\n", area)
}
```

3.9. Calcular la suma de los “n” primeros números naturales (Con bucle)

Este ejercicio propone realizar la suma de los “n” primeros números naturales [9] y mostrar el resultado de la suma por pantalla. Para ello, se ha utilizado un bucle **FOR**. En Go, `for` es la única estructura de bucle (funciona como `while` y `for` de C o C++ según la sintaxis). Como se puede observar, es mucho menos eficiente que el ejercicio planteado en 2.8

```
// fichero sumanaturales.go
// Este programa calcula la suma de los "n" primeros números naturales con for
package main
import ("bufio";"fmt";"os";"strconv";"strings")
func main() {
    var numero int64;var suma int64; var err error
    scanner := bufio.NewScanner(os.Stdin)
    for {
        fmt.Print("Introduzca un número n >0 de términos a sumar: ")
        if !scanner.Scan() {break}

        texto := strings.TrimSpace(scanner.Text())
        numero, err = strconv.ParseInt(texto, 10, 64)
        if err != nil {
            fmt.Println("ERROR: La entrada no es un número entero válido.")
            continue
        }
        if numero <= 0 {
            fmt.Println("ERROR: El número de términos debe ser >0.")
            continue
        }
        break
    }
    for i := int64(1); i <= numero; i++ {
        suma = suma + i
    }
    fmt.Printf("La suma de los %d primeros números naturales es: %d\n", numero,
     suma)
}
```

3.10. Realizar un programa que sume números introducidos por teclado hasta que se escriba un número concreto de parada

Este ejercicio propone realizar la suma de números introducidos por teclado, hasta que se introduzca un número de parada (en este caso el número -50) y muestre el resultado.

```
// fichero sumanumeros.go
// Este programa calcula la suma de los números introducidos hasta leer -50
package main
import ("bufio";"fmt";"os";"strconv";"strings")
func main() {
    var numero, suma float64;var err error
    scanner := bufio.NewScanner(os.Stdin)
    for {
        fmt.Print("Introduzca número a sumar (o -50 para terminar): ")
        if !scanner.Scan() {break }
        texto := strings.TrimSpace(scanner.Text())
        numero, err = strconv.ParseFloat(texto, 64)
        if err != nil {
            fmt.Println("ERROR: La entrada no es un número válido. Inténtelo de
     nuevo.")
            continue
        }
        if numero == -50.0 {
            break
        }
        suma += numero
    }
    fmt.Printf("La suma total de los números introducidos es: %f\n", suma)
}
```

3.11. Dados 10 números por teclado decir cuál es el mayor de todos utilizando una función

Este ejercicio propone, dados 10 números introducidos por teclado, calcular cuál es el mayor. Se resolverá esto guardando los datos en un **array** de enteros (`[10]int`) y aplicando una función. Se utilizará la cláusula `range` de Go, que simplifica la iteración sobre estructuras de datos.

```
// fichero mayor.go
// Este programa calcula el número mayor de 10 introducidos por teclado
package main
import ("bufio";"fmt";"os";"strconv";"strings")
const NumElementos = 10
func mayor(numeros []int64) int64 {
    if len(numeros) == 0 {return 0}
    max := numeros[0]
    for _, num := range numeros {
        if num > max {max = num}
    }
    return max
}
func main() {
    var numerosin []int64; var err error
    scanner := bufio.NewScanner(os.Stdin)
    for i := 0; i < NumElementos; i++ {
        for {
            fmt.Printf("Introduzca número %d de %d: ", i+1, NumElementos)
            if !scanner.Scan() {os.Exit(0)}
            texto := strings.TrimSpace(scanner.Text())
            numeroLeido, err := strconv.ParseInt(texto, 10, 64)
            if err != nil {fmt.Println("ERROR: La entrada no es un número
    entero válido. Inténtelo de nuevo");continue}
            numerosin = append(numerosin, numeroLeido)
            break
        }
    }
    fmt.Printf("\nEl número mayor de todos los introducidos es: %d\n", mayor(
    numerosin))
}
```

3.12. Calcular el tipo de triángulo en función de los lados

Este ejercicio propone, dados los lados de un triángulo, calcular el tipo de triángulo que es (equilátero, isósceles o escaleno) y mostrarlo por pantalla.

```
// fichero tipotriangulo.go
// Este programa calcula el tipo de triángulo en función de los lados
package main
import ("bufio";"fmt";"os";"strconv";"strings")
func main() {
    var l1, l2, l3 float64; var err error
    scanner := bufio.NewScanner(os.Stdin)
    fmt.Print("Introduce lo que mide el primer lado del triángulo: ")
    if scanner.Scan() {
        l1, err = strconv.ParseFloat(strings.TrimSpace(scanner.Text()), 64)
        if err != nil {fmt.Println("Error: la entrada no es un número válido");
    os.Exit(1)}
    }
    fmt.Print("Introduce lo que mide el segundo lado del triángulo: ")
    if scanner.Scan() {
        l2, err = strconv.ParseFloat(strings.TrimSpace(scanner.Text()), 64)
        if err != nil {fmt.Println("Error: la entrada no es un número válido");
    os.Exit(1)}
    }
    fmt.Print("Introduce lo que mide el tercer lado del triángulo: ")
    if scanner.Scan() {
        l3, err = strconv.ParseFloat(strings.TrimSpace(scanner.Text()), 64)
        if err != nil {fmt.Println("Error: la entrada no es un número válido");
    os.Exit(1)}
    }
    if l1+l2 <= l3 || l1+l3 <= l2 || l2+l3 <= l1 {
        fmt.Println("\nError Geométrico: Los lados NO forman un triángulo vá
    lido.");os.Exit(1)
    }
    if l1 == l2 && l2 == l3 {fmt.Println("El Triangulo es Equilatero")
    } else if l1 == l2 || l1 == l3 || l2 == l3 {
        fmt.Println("El Triangulo es Isoceles")
    } else {fmt.Println("El Triangulo es Escaleno")}
}
```

3.13. Resolver ecuaciones de segundo grado mediante la fórmula general (sólo soluciones reales)

Este ejercicio propone resolver ecuaciones de segundo grado, utilizando para ello la fórmula general $ax^2 + bx + c = 0$, donde $a \neq 0$. Se usará el paquete `math` y se controlará que el discriminante no sea negativo para evitar resultados `NaN` (Not a Number) en el dominio de los reales. Fórmula [9]:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

```
// fichero ecuaciones.go
// Resuelve ecuaciones de segundo grado por la fórmula general para números
    Reales
package main

import ("bufio";"fmt";"os";"strconv";"strings";"math")
func main() {
    var a, b, c, d, x1, x2 float64
    var err error
    scanner := bufio.NewScanner(os.Stdin)
    for {
        fmt.Print("Ingrese coeficiente a: ")
        if !scanner.Scan() {os.Exit(0)}
        texto := strings.TrimSpace(scanner.Text())
        a, err = strconv.ParseFloat(texto, 64)
        if err != nil {
            fmt.Fprintln(os.Stderr, "Error: la entrada no es un número válido")
            os.Exit(1)
        }
        if a == 0 {
            fmt.Println("El coeficiente 'a' no puede ser cero.")
            continue
        }
        break
    }
    fmt.Print("Ingrese coeficiente b: ")
    if scanner.Scan() {
        b, err = strconv.ParseFloat(strings.TrimSpace(scanner.Text()), 64)
        if err != nil {
            fmt.Fprintln(os.Stderr, "Error: la entrada no es un número válido")
            os.Exit(1)
```

```
        }
    }
    fmt.Print("Ingrese coeficiente c: ")
    if scanner.Scan() {
        c, err = strconv.ParseFloat(strings.TrimSpace(scanner.Text()), 64)
        if err != nil {
            fmt.Fprintln(os.Stderr, "Error: la entrada no es un número válido")
            os.Exit(1)
        }
    }
    d = math.Pow(b, 2) - 4*a*c
    if d > 0 {
        x1 = (-b + math.Sqrt(d)) / (2 * a)
        x2 = (-b - math.Sqrt(d)) / (2 * a)
        fmt.Printf("x1 = %f\n", x1)
        fmt.Printf("x2 = %f\n", x2)

    } else if d == 0 {
        x1 = -b / (2 * a)
        fmt.Printf("x1 = %f\n", x1)

    } else {
        fmt.Println("La ecuacion no tiene soluciones reales")
    }
}
```

3.14. Dados 10 números por teclado sumar los pares y los impares de manera separada y mostrar ambos resultados

Este ejercicio requiere diseñar un programa que lea una serie de números enteros introducidos por el usuario y los almacene en un slice o array. Posteriormente, el programa sumará por separado los números pares y los impares.

```
// fichero sumaparesimpares.go
// Calcula la suma de 10 números pasados por teclado distinguiendo entre los
    pares e impares
package main

import ("bufio";"fmt";"os";"strconv";"strings")
```

```
const TamanoArray = 10

func main() {
    numeros := make([]int64, TamanoArray)
    var sumaPares, sumaImpares int64;var err error
    scanner := bufio.NewScanner(os.Stdin)
    fmt.Printf("Por favor, introduce %d números enteros:\n", TamanoArray)
    for i := 0; i < TamanoArray; i++ {
        fmt.Printf("Número %d: ", i+1)
        if scanner.Scan() {
            texto := strings.TrimSpace(scanner.Text())
            numeros[i], err = strconv.ParseInt(texto, 10, 64)
            if err != nil {
                fmt.Fprintln(os.Stderr, "Error en la lectura o entrada no vá
    lida. Terminando...")
                os.Exit(1)
            }
        } else {
            os.Exit(1)
        }
    }
    for _, numeroActual := range numeros {
        if numeroActual%2 == 0 {
            sumaPares += numeroActual
            fmt.Printf("Añadiendo %d a la suma par.\n", numeroActual)
        } else {
            sumaImpares += numeroActual
            fmt.Printf("Añadiendo %d a la suma impar.\n", numeroActual)
        }
    }
    fmt.Print("Números ingresados: [")
    for i, num := range numeros {
        fmt.Print(num)
        if i < len(numeros)-1 {fmt.Print(", ")}
    }
    fmt.Println("]")
    fmt.Printf("Suma total de los números PARES: %d\n", sumaPares)
    fmt.Printf("Suma total de los números IMPARES: %d\n", sumaImpares)
}
```

3.15. Función para calcular el factorial de un número (iterativo)

Este ejercicio presenta el cálculo del factorial de manera iterativa.

Nota sobre Tipos de Datos: El factorial crece extremadamente rápido. En Go, el tipo `int` depende de la arquitectura (32 o 64 bits), por lo que se utilizarán explícitamente `int64` o `uint64` para maximizar la capacidad. Un `int64` (64 bits con signo) tiene un valor máximo de aprox. 9.2×10^{18}.

- 20! (aprox. 2.43×10^{18}) cabe en un `int64`.
- 21! (aprox. 5.1×10^{19}) provoca un desbordamiento (*overflow*).

```
// fichero factoriali.go
// Calcula el factorial de un número entero no negativo de forma iterativa
package main
import ("bufio";"fmt";"os";"strconv";"strings")
func factorialIterativo(n int64) int64 {
    if n == 0 || n == 1 {return 1}
    resultado := int64(1)
    for i := int64(2); i <= n; i++ {resultado *= i }
    return resultado
}
func main() {
    var numero int64;var err error
    scanner := bufio.NewScanner(os.Stdin)
    fmt.Print("Introduce un numero entero no negativo (max 20 para precision):
    ")
    if scanner.Scan() {
        texto := strings.TrimSpace(scanner.Text())
        numero, err = strconv.ParseInt(texto, 10, 64)
        if err != nil {
            fmt.Fprintln(os.Stderr, "Error: Entrada no valida.")
            os.Exit(1)
        }
    } else {
        return
    }
    if numero < 0 {
        fmt.Fprintln(os.Stderr, "Error: El factorial solo esta definido para
    numeros no negativos.")
        os.Exit(1)
    }
```

```
    if numero > 20 {
        fmt.Printf("Advertencia: El factorial de %d es muy grande y
    probablemente cause un resultado incorrecto por desbordamiento (overflow).\
    n", numero)
    }
    resultado := factorialIterativo(numero)
    fmt.Printf("El factorial de %d es: %d\n", numero, resultado)
}
```

Hasta aquí es la forma clásica de hacerlo con tipos de datos nativos int64, pero ahora es donde Go brilla, por ejemplo, frente a C++ y C. Go tiene un paquete nativo llamado math/big que permite manejar números de precisión arbitraria. A diferencia de lenguajes parecidos como C++, donde se necesitaría instalar bibliotecas externas (como GMP), en Go esto viene de serie. Se puede calcular el factorial de 1.000, 10.000 o 1.000.000 sin problemas, siempre que se tenga memoria RAM suficiente. A continuación, se muestra un ejemplo utilizando este paquete.

```
// fichero factoriali2.go
// Calcula el factorial de un número entero no negativo de forma iterativa
package main
import ("fmt";"math/big")

func main() {

    resultado := big.NewInt(1)
    limite := 100

    for i := 1; i <= limite; i++ {
        iBig := big.NewInt(int64(i))
        resultado.Mul(resultado, iBig)
    }

    fmt.Printf("El factorial de %d es:\n%s\n", limite, resultado.String())
}
```

3.16. Función para calcular el factorial de un número (recursivo)

Este ejercicio presenta la resolución del factorial de manera recursiva. Aunque Go soporta recursividad, no optimiza la recursión de cola (*tail call optimization*), por lo que una profundidad excesiva podría llenar la pila de llamadas (*stack overflow*), aunque para los límites del factorial (n=20) esto no es un problema.

```
// fichero factorialr.go
// Calcula el factorial de un número entero no negativo de forma recursiva.
package main
import ("bufio";"fmt";"os";"strconv";"strings")

func factorialRecursivo(n int64) int64 {
    if n <= 1 {return 1}
    return n * factorialRecursivo(n-1)
}
func main() {
    var numero int64;var err error
    scanner := bufio.NewScanner(os.Stdin)
    fmt.Print("Introduce un número entero no negativo (máx 20): ")
    if scanner.Scan() {
        texto := strings.TrimSpace(scanner.Text())
        numero, err = strconv.ParseInt(texto, 10, 64)
        if err != nil {fmt.Fprintln(os.Stderr, "Error: Entrada no válida.");os.
    Exit(1)}
    } else {
        return
    }
    if numero < 0 {
        fmt.Fprintln(os.Stderr, "Error: El factorial solo está definido para nú
    meros >0")
        os.Exit(1)
    }
    if numero > 20 {
        fmt.Printf("Advertencia: El factorial de %d es muy grande y
    probablemente cause un resultado incorrecto por desbordamiento (overflow).\
    n", numero)
    }
    resultado := factorialRecursivo(numero)
    fmt.Printf("El factorial de %d es: %d\n", numero, resultado)
}
```

3.17. Función para calcular el número combinatorio de N sobre M

Este ejercicio propone realizar el cálculo de un número combinatorio utilizando la fórmula del coeficiente binomial $\binom{N}{M} = \frac{N!}{M!(N-M)!}$ [9]. Se reutilizará la función factorial creada anteriormente.

```
// fichero combinatorio.go
// Calcula el número combinatorio C(n, r) o n sobre r, usando factoriales.
package main
import ("bufio";"fmt";"os";"strconv";"strings")
// Función factorial iterativa (igual que en el ejercicio anterior)
func factorial(n int64) int64 {
    if n == 0 || n == 1 {return 1}
    resultado := int64(1)
    for i := int64(2); i <= n; i++ {resultado *= i}
    return resultado
}
// Función para calcular un número combinatorio nCr
func nCr(n, r int64) int64 {
    // 1. Validaciones básicas de rango
    if r < 0 || r > n {return 0}
    // 2. Cálculo de los tres factoriales necesarios
    factN := factorial(n)
    factR := factorial(r)
    factNMinusR := factorial(n - r)
    // Si algún factorial dio negativo, es que "dio la vuelta" al int64 (superó
     el límite).
    if factN < 0 || factR < 0 || factNMinusR < 0 {
        fmt.Fprintln(os.Stderr, "Error: Un factorial intermedio es demasiado
    grande para 'int64'.")
        return -1 // Código de error
    }
    return factN / (factR * factNMinusR)
}
func main() {
    var n, m int64;var err error
    scanner := bufio.NewScanner(os.Stdin)
    fmt.Print("Introduce un numero entero no negativo para N (max 20 para
    precision): ")
    if scanner.Scan() {
        texto := strings.TrimSpace(scanner.Text())
```

```
        n, err = strconv.ParseInt(texto, 10, 64)
        if err != nil {
            fmt.Fprintln(os.Stderr, "Error: Entrada no válida para N.")
            os.Exit(1)
        }
    } else {
        return
    }
    if n < 0 {
        fmt.Fprintln(os.Stderr, "Error: N debe ser no negativo.");os.Exit(1)}
    if n > 20 {
        fmt.Printf("Advertencia: El factorial de N=%d causará desbordamiento.\n
    ", n)
    }
    fmt.Print("Introduce un numero entero no negativo para M (max 20 para
    precision): ")
    if scanner.Scan() {
        texto := strings.TrimSpace(scanner.Text())
        m, err = strconv.ParseInt(texto, 10, 64)
        if err != nil {
            fmt.Fprintln(os.Stderr, "Error: Entrada no válida para M.")
            os.Exit(1)
        }
    } else {return}
    if m < 0 {fmt.Fprintln(os.Stderr, "Error: M debe ser no negativo.");os.Exit
    (1)}
    if m > 20 {fmt.Printf("Advertencia: El factorial de M=%d causará
    desbordamiento.\n", m)}
    combinaciones := nCr(n, m)
    if combinaciones > 0 {
        fmt.Printf("El número combinatorio de %d sobre %d es: %d\n", n, m,
    combinaciones)
    } else if combinaciones == 0 {
        fmt.Printf("El número combinatorio de %d sobre %d es 0 (M fuera de
    rango 0 <= M <= N).\n", n, m)
    } else if combinaciones == -1 {
        fmt.Printf("No se pudo calcular el número combinatorio de %d sobre %d
    debido a desbordamiento.\n", n, m)
    }
}
```

3.18. Función para resolver $(a + b)^n$ utilizando el desarrollo del binomio de Newton

Este ejercicio propone resolver $(a + b)^n$ utilizando la fórmula del **binomio de Newton** [9]: $(a + b)^n = \sum_{k=0}^{n} \binom{n}{k} a^{n-k} b^k$. Se utilizarán funciones para el combinatorio y bucles para la sumatoria.

```
// fichero newton.go
// Calcula mediante el binomio de newton los datos dados
package main
import ("bufio";"fmt";"os";"strconv";"strings")

// Función factorial iterativa
func factorial(n int64) int64 {
    if n == 0 || n == 1 {return 1}
    resultado := int64(1)
    for i := int64(2); i <= n; i++ {resultado *= i}
    return resultado
}

// Función para calcular un número combinatorio (nCr)
func nCr(n, r int64) int64 {
    if r < 0 || r > n {return 0}
    factN := factorial(n)
    factR := factorial(r)
    factNMinusR := factorial(n - r)
    if factN < 0 || factR < 0 || factNMinusR < 0 {
        fmt.Fprintln(os.Stderr, "Error: Un factorial intermedio es demasiado
    grande para 'int64'.")
        return -1
    }
    return factN / (factR * factNMinusR)
}

// calcularPotencia implementa "Exponenciación Binaria"
// Es un algoritmo mucho más eficiente que multiplicar en bucle uno a uno.
func calcularPotencia(base, exponente int64) int64 {
    resultado := int64(1)
    for exponente > 0 {
        if exponente%2 == 1 {
            resultado *= base
```

```
        }
        base *= base
        exponente /= 2
    }
    return resultado
}

// binomioDeNewton calcula la expansión (a + b)^n
func binomioDeNewton(a, b, n int64) {
    if n < 0 || n > 20 {
        fmt.Fprintln(os.Stderr, "Error: El exponente N debe ser no negativo y
    <= 20.")
        return
    }
    fmt.Printf("\nExpansión de (%d + %d)^%d:\n", a, b, n)
    var suma int64 = 0
    for k := int64(0); k <= n; k++ {
        coeficiente := nCr(n, k)
        if coeficiente == -1 {
            fmt.Fprintln(os.Stderr, "Cálculo abortado debido a un error de
    factorial.")
            return
        }
        potenciaA := calcularPotencia(a, n-k)
        potenciaB := calcularPotencia(b, k)
        valorTermino := coeficiente * potenciaA * potenciaB
        suma += valorTermino
        fmt.Printf("  + (%d * %d^%d * %d^%d) = %d\n",
            coeficiente, a, n-k, b, k, valorTermino)
    }
    fmt.Printf("Valor Total del Binomio de Newton de (a+b)^n: %d\n", suma)
}

func main() {
    var a, b, n int64;var err error
    scanner := bufio.NewScanner(os.Stdin)
    fmt.Print("Introduce un numero entero no negativo para a: ")
    if scanner.Scan() {
        a, err = strconv.ParseInt(strings.TrimSpace(scanner.Text()), 10, 64)
        if err != nil || a < 0 {
            fmt.Fprintln(os.Stderr, "Error: Entrada no válida para a (debe ser
    no negativa).")
```

```
            os.Exit(1)
        }
    }
    fmt.Print("Introduce un numero entero no negativo para b: ")
    if scanner.Scan() {
        b, err = strconv.ParseInt(strings.TrimSpace(scanner.Text()), 10, 64)
        if err != nil || b < 0 {
            fmt.Fprintln(os.Stderr, "Error: Entrada no válida para b (debe ser
    no negativa).")
            os.Exit(1)
        }
    }
    fmt.Print("Introduce un numero entero no negativo para n: ")
    if scanner.Scan() {
        n, err = strconv.ParseInt(strings.TrimSpace(scanner.Text()), 10, 64)
        if err != nil || n < 0 {
            fmt.Fprintln(os.Stderr, "Error: Entrada no válida para n (debe ser
    no negativo).")
            os.Exit(1)
        }
    }
    // Nota: (a > 1 && n > 15) es una heurística simple para avisar que el nú
    mero será gigante
    if n > 20 || (a > 1 && n > 15) || (b > 1 && n > 15) {
        fmt.Println("\nAdvertencia: La expansión de (a+b)^n es muy grande. El
    resultado total puede desbordar 'int64'.")
    }
    binomioDeNewton(a, b, n)
}
```

3.19. Implementar la función de Fibonacci para valores enteros positivos

Este ejercicio propone calcular el valor de la sucesión de Fibonacci ($F(n) = F(n-1)+F(n-2)$) [9].

Nota: Se implementará la versión iterativa en Go por ser mucho más eficiente en términos de memoria y CPU que la recursiva directa.

```
// fichero fibo.go
// Función iterativa para calcular el n-ésimo número de Fibonacci
package main
import ("fmt";"os")
// fibonacciIterativo calcula la serie usando un bucle.
func fibonacciIterativo(n int) int64 {
    var a int64 = 0
    var b int64 = 1
    var resultado int64 = 0
    if n <= 0 {return 0}
    if n == 1 {return 1}

    for i := 2; i <= n; i++ {
        resultado = a + b
        a = b
        b = resultado
    }
    return resultado
}
func main() {
    const N = 45
    resultado := fibonacciIterativo(N)
    fmt.Printf("El %d-ésimo número de Fibonacci (iterativo) es: %d\n", N,
    resultado)
    // El tipo int64 soporta hasta Fib(92).
    // Fib(93) ya desborda (Overflow).
    if N > 92 {
        fmt.Fprintf(os.Stderr, "Advertencia: N > 92. El resultado (%d)
    probablemente es incorrecto debido a desbordamiento de 'int64'.\n",
    resultado)
        os.Exit(1)
    }
}
```

3.20. Implementar un procedimiento para seleccionar y visualizar los K bits de orden inferior de un número entero proporcionado.

Este ejercicio tiene como objetivo extraer y mostrar por pantalla los K bits menos significativos. En Go, se utilizarán operadores a nivel de bit (*bitwise*) como `&` (AND) y desplazamientos ». Se probará por ejemplo con el número 45.

```go
// fichero nkbits.go
// Procedimiento que escribe los últimos k bits de un número entero.
package main
import ("fmt";"os";"strconv" )

func mostrarUltimosKBits(numero int, k int) {
    // strconv.IntSize nos da exactamente ese valor (32 o 64).
    tamanoInt := strconv.IntSize
    if k <= 0 {
        fmt.Fprintln(os.Stderr, "Error: k debe ser mayor que 0.")
        return
    }
    // Si k es mayor que los bits que tiene el número, lo limitamos
    if k > tamanoInt {
        k = tamanoInt
    }
    fmt.Printf("Los Últimos %d bits de %d son:", k, numero)

    for i := k - 1; i >= 0; i-- {
        bit := (numero >> i) & 1
        fmt.Print(bit)
    }
    fmt.Println()
}

func main() {
    num1 := 45
    mostrarUltimosKBits(num1, 8)
}
```

3.21. Implementar una función que invierta los últimos K bits de orden inferior de un número entero proporcionado

En este ejercicio se realizará una función que invierta los últimos K bits de un número. Se utilizarán máscaras y el operador XOR (^ en Go).

Creación de la Máscara

$1 \ll k$: Genera un número con el bit en la posición k encendido. Ejemplo con $k = 4$: 00010000 (valor decimal 16)

$(1 \ll k) - 1$: Al restar 1 de cualquier potencia de 2 (o valor con un solo bit encendido), se ponen a 1 todos los bits a la derecha de esa posición. Ejemplo con $k = 4$: $00010000 - 1 = 00001111$ (valor decimal 15)

Esta máscara, 00001111, tiene los 4 bits a la derecha listos para la inversión, y los 0's a la izquierda aseguran que el resto del número permanezca inalterado. **Aplicación del XOR**

La inversión se realiza mediante el operador ^ (XOR): resultado = numero ^ mascara. En Go ^ sirve tanto para XOR como para NOT dependiendo de qué lugar ocupe en la operación.

```
// fichero mascara.go
// Invierte (cambia 0->1 y 1->0) los últimos k bits de un número entero
package main
import ("fmt";"os")
func invertirUltimosKBits(numero int64, k int) int64 {
    const tamanoLL = 64
    if k <= 0 {return numero}
    if k >= tamanoLL {return ^numero}
    mascara := (uint64(1) << k) - 1
    return numero ^ int64(mascara)
}
func main() {
    var num int64 = 45
    k := 4
    resultado := invertirUltimosKBits(num, k)
    fmt.Printf("Número original:   %d\t(Binario: ...%08b)\n", num, num)
    fmt.Printf("K: %d\n", k)
    fmt.Printf("Número resultante: %d\t(Binario: ...%08b)\n", resultado,
    resultado)

}
```

3.22. Invertir una cadena de texto leída por teclado, sin utilizar funciones de inversión nativas

Este ejercicio propone invertir una cadena manualmente. **Importante en Go:** Las cadenas (`string`) en Go son inmutables y están codificadas en UTF-8. Para invertirla carácter a carácter de forma segura, primero se debe convertir a un slice de `rune` (caracteres Unicode), realizar el intercambio (swap) y volver a convertir a `string`.

```
// fichero invierte.go
// Este ejercicio implementa la inversión de una cadena de texto dada
package main
import ("bufio";"fmt";"os")
// En Go, no pasamos arrays por referencia para modificarlos in-situ (
    normalmente),
// sino que devolvemos el nuevo valor transformado.
func invertirCadena(s string) string {
    // Convertimos el string a un slice de 'runes'.Un 'rune' es un carácter
    Unicode completo.
    // Si usáramos bytes simples, romperíamos caracteres con tildes o emojis (
    ej: 'ñ').
    runes := []rune(s)
    longitud := len(runes)
    inicio := 0
    fin := longitud - 1
    for inicio < fin {runes[inicio], runes[fin] = runes[fin], runes[inicio];
    inicio++;fin--}
    // Convertimos los runes de vuelta a string
    return string(runes)
}
func main() {
    scanner := bufio.NewScanner(os.Stdin)
    fmt.Print("Introduce una cadena de texto: ")
    if scanner.Scan() {
        cadena := scanner.Text()
        if len(cadena) == 0 {fmt.Println("Cadena vacía.");os.Exit(0)}
        fmt.Printf("Cadena original:  \"%s\"\n", cadena)
        cadenaInvertida := invertirCadena(cadena)
        fmt.Printf("Cadena invertida: \"%s\"\n", cadenaInvertida)
    }
}
```

3.23. Leer una cadena de caracteres desde teclado, decir longitud y concatenar con otra utilizando funciones básicas

En Go, la longitud de un string se obtiene con la función nativa `len()` (que devuelve bytes) o `utf8.RuneCountInString()` (para contar caracteres reales). La concatenación se realiza con el operador +.

```
// fichero leelongi.go
// Este ejercicio lee una cadena desde teclado, dice la longitud y la concatena
     con otra.
package main
import ("bufio";"fmt";"os";"unicode/utf8")

func main() {

    scanner := bufio.NewScanner(os.Stdin)
    for {
        fmt.Println("Introduzca cadena (o solo Enter para terminar): ")
        if !scanner.Scan() {
            break
        }
        cadena := scanner.Text()
        if len(cadena) == 0 {
            break
        }
        lenBytes := len(cadena)
        lenCaracteres := utf8.RuneCountInString(cadena)
        fmt.Printf("La cadena contiene %d bytes y %d caracteres reales.\n",
    lenBytes, lenCaracteres)
        cadena += ".txt"
        fmt.Printf("Concatenación: %s \n", cadena)
    }
}
```

3.24. Convertir cadenas a letras mayúsculas y contar sus vocales

Este ejercicio consiste en leer cadenas, convertirlas a mayúsculas y contar vocales. En Go, se utilizará el paquete `strings` para la conversión (`ToUpper`) y un bucle sobre el slice de runas para el conteo.

```
// fichero conteo.go
// Este ejercicio convierte cadenas a mayúsculas y cuenta sus vocales.
package main
import ("bufio";"fmt";"os";"strings")
const numFrases = 4
func convertirAMayusculas(s string) string {
    return strings.ToUpper(s)
}
func contarVocales(s string) int {
    contador := 0
    // 'range' descompone el string en 'runes' (caracteres Unicode seguros).
    // Si usáramos un for clásico (i=0; i<len), estaríamos leyendo bytes, no
    letras.
    for _, letra := range s {
        // En Go, el switch no necesita 'break'.
        switch letra {
        case 'A', 'E', 'I', 'O', 'U':
            contador++
        }
    }
    return contador
}

func main() {
    listaFrases := make([]string, numFrases)
    scanner := bufio.NewScanner(os.Stdin)
    totalVocales := 0
    fmt.Printf("Introduce %d frases/líneas de texto:\n", numFrases)
    for i := 0; i < numFrases; i++ {
        fmt.Printf("Frase %d: ", i+1)
        if scanner.Scan() {
            listaFrases[i] = scanner.Text()
        } else {
            fmt.Fprintln(os.Stderr, "Error en la lectura o fin de archivo.")
            os.Exit(1)
```

```
        }
    }
    fmt.Println("\n Procesamiento y Conteo ")
    for i := 0; i < numFrases; i++ {
        listaFrases[i] = convertirAMayusculas(listaFrases[i])
        vocalesFrase := contarVocales(listaFrases[i])
        totalVocales += vocalesFrase
        fmt.Printf("Frase %d (MAYÚS): '%s' -> Vocales contadas: %d\n",
            i+1, listaFrases[i], vocalesFrase)
    }
    fmt.Println("\n Resumen")
    fmt.Printf("El número total de vocales en todas las frases es: %d\n",
    totalVocales)
}
```

3.25. Función para realizar la transposición de una matriz de números enteros dada

En este ejercicio se creará una función para trasponer una matriz. En Go, se representará la matriz como un array multidimensional (ej. `[N][M]int`). La **transposición de una matriz** A (denotada como A^T) [4] es una operación que **cambia las filas por las columnas** y viceversa. Si la matriz original A tiene dimensiones $m \times n$ (filas $\times$ columnas), su matriz transpuesta A^T tendrá dimensiones $n \times m$.

El elemento en la posición (i, j) de la matriz original A se convierte en el elemento en la posición (j, i) de la matriz transpuesta A^T [4].

$$A_{i,j} = (A^T)_{j,i}$$

```
// fichero trasposicion.go
// Función para trasponer una matriz MxN
package main
import ("fmt")

const MFilas = 3
const NColumnas = 4

type MatrizOriginal [MFilas][NColumnas]int
type MatrizTranspuesta [NColumnas][MFilas]int
```

```
func trasponerMatriz(original MatrizOriginal) MatrizTranspuesta {
    var transpuesta MatrizTranspuesta
    for i := 0; i < MFilas; i++ {
        for j := 0; j < NColumnas; j++ {
            transpuesta[j][i] = original[i][j]
        }
    }
    return transpuesta
}
func imprimirMatrizOriginal(matriz MatrizOriginal) {
    for i := 0; i < MFilas; i++ {
        for j := 0; j < NColumnas; j++ {
            fmt.Printf("%4d", matriz[i][j])
        }
        fmt.Println() // Salto de línea al terminar la fila
    }
}
func imprimirMatrizTranspuesta(matriz MatrizTranspuesta) {
    for i := 0; i < NColumnas; i++ {
        for j := 0; j < MFilas; j++ {
            fmt.Printf("%4d", matriz[i][j])
        }
        fmt.Println()
    }
}
func main() {
    A := MatrizOriginal{
        {1, 2, 3, 4},
        {5, 6, 7, 8},
        {9, 10, 11, 12},
    }
    fmt.Printf("Matriz Original A (%d x %d) \n", MFilas, NColumnas)
    imprimirMatrizOriginal(A)
    AT := trasponerMatriz(A)
    fmt.Printf("\n Matriz Transpuesta AT (%d x %d) \n", NColumnas, MFilas)
    imprimirMatrizTranspuesta(AT)
}
```

3.26. Función para multiplicar 2 matrices cuadradas de números enteros

En este ejercicio se multiplicarán dos matrices cuadradas. Para ello, seguiremos la explicación dada en [4], en la cual indica que la multiplicación de dos matrices cuadradas, A y B, de dimensión $N \times N$, da como resultado una matriz C, también de dimensión $N \times N$.

Para multiplicar $A \cdot B$, el número de columnas de A debe ser igual al número de filas de B. Si A y B son ambas $N \times N$, la condición siempre se cumple.

Cada elemento de la matriz resultante, C_{ij}, se calcula como el producto escalar (la suma de productos) de la fila i de A por la columna j de B [4].

$$C_{ij} = \sum_{k=1}^{N} A_{ik} B_{kj}$$

```
// fichero multiplicamatrices.go
// Multiplicación de dos matrices cuadradas de tamaño fijo.
package main
import ("fmt")

const N = 3
type Matriz [N][N]int

// Para matrices pequeñas (3x3) está bien, para grandes usaríamos punteros *
    Matriz.
func multiplicarMatrices(a, b Matriz) Matriz {
    var c Matriz
    for i := 0; i < N; i++ {
        for j := 0; j < N; j++ {
            for k := 0; k < N; k++ {
                c[i][j] += a[i][k] * b[k][j]
            }
        }
    }
    return c
}
func imprimirMatriz(matriz Matriz) {
    for i := 0; i < N; i++ {
        for j := 0; j < N; j++ {
            fmt.Printf("%4d ", matriz[i][j])
        }
```

```
        fmt.Println()
    }
}
func main() {

    a := Matriz{
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9},
    }
    b := Matriz{
        {9, 8, 7},
        {6, 5, 4},
        {3, 2, 1},
    }
    c := multiplicarMatrices(a, b)
    fmt.Printf("Matriz A (%dx%d):\n", N, N)
    imprimirMatriz(a)
    fmt.Printf("\nMatriz B (%dx%d):\n", N, N)
    imprimirMatriz(b)
    fmt.Println("\nMatriz Resultado C = A * B:")
    imprimirMatriz(c)
}
```

Capítulo 4

Manejo de ficheros y memoria dinámica

Aunque Go es un lenguaje moderno con recolección de basura (*Garbage Collection*), ofrece herramientas extremadamente potentes y sencillas para interactuar con los recursos del sistema operativo: la memoria (heap/stack) y el almacenamiento persistente (ficheros). Dominar estos conceptos es crucial para desarrollar aplicaciones que manejen grandes volúmenes de datos y mantengan información entre sesiones.

A diferencia de C y C++, donde la gestión de memoria es manual y propensa a errores, Go introduce el concepto de **Slices** (rebanadas) como abstracción dinámica sobre los arrays, y delega la liberación de memoria al *Garbage Collector*. Sin embargo, entender cuándo y cómo se asigna memoria (usando primitivas como `make` o `new`) sigue siendo vital para el rendimiento.

Con respecto a los ficheros, el paquete estándar `os` y `bufio` de Go simplifican enormemente la lectura y escritura, convirtiendo tareas que en otros lenguajes son verbosas en operaciones directas y seguras gracias al manejo de errores explícito y la sentencia `defer` para asegurar el cierre de recursos. Este capítulo explora cómo Go permite manejar la persistencia de datos y cómo utilizar estructuras dinámicas (Slices y Maps) para superar las limitaciones de los arrays estáticos. Para profundizar en las especificaciones del lenguaje, se recomienda revisar la documentación oficial de Go (`go.dev/doc`) o referencias clásicas como *The Go Programming Language* de Donovan y Kernighan [3].

4.1. Crear y escribir en un fichero

Este ejercicio propone crear un fichero de texto y escribir sobre él. Se aprenderá a utilizar el paquete `os` para crear archivos y la sentencia `defer` para asegurar que el fichero se cierre correctamente al finalizar la ejecución.

```go
// fichero creafiche.go
// Ejemplo para crear/escribir ficheros
package main
import (
    "fmt"
    "os"
)

func main() {

    fichero, err := os.Create("datos.txt")
    if err != nil {
        fmt.Fprintf(os.Stderr, "Error al abrir/crear el fichero datos.txt: %v\n
    ", err)
        os.Exit(1)
    }
    // 'defer' asegura que fichero.Close() se ejecute cuando la función main
    termine,
    // pase lo que pase (incluso si hay errores intermedios).
    defer fichero.Close()

    // Escritura de línea
    fichero.WriteString("Esta es la primera línea.\n")
    pi := 3.14159
    // Escritura con formato
    fmt.Fprintf(fichero, "El número PI es aproximadamente %.4f\n", pi)
    fichero.WriteString("Tercera línea de ejemplo.\n")
    fmt.Println("El archivo 'datos.txt' fue creado y escrito correctamente.")

}
```

4.2. Leer y mostrar contenido de un fichero de texto

Este ejercicio propone un procedimiento que lee un fichero de texto y muestra su contenido por pantalla. En Go, se utilizará el paquete `bufio` y su estructura `Scanner`, que es la forma idiomática y eficiente de leer texto línea a línea.

```go
// fichero leerfiche.go
// Ejemplo para leer ficheros de texto línea a línea
package main
import (
    "bufio"
    "fmt"
    "os"
)

func main() {

    // os.Open abre el archivo en modo "solo lectura" (Read Only).
    fichero, err := os.Open("datos.txt")
    if err != nil {
        fmt.Fprintf(os.Stderr, "Error al abrir el fichero datos.txt: %v\n", err
    )
        os.Exit(1)
    }
    defer fichero.Close()

    fmt.Println("\nContenido de 'datos.txt':")
    // El Scanner es un "iterador" que lee datos (por defecto, línea por línea)
    // Actúa como buffer intermedio para no hacer demasiadas llamadas al
    sistema.
    scanner := bufio.NewScanner(fichero)
    for scanner.Scan() {
        linea := scanner.Text()
        fmt.Println(linea)
    }
    // Siempre hay que comprobar scanner.Err() después del bucle.
    if err := scanner.Err(); err != nil {
        fmt.Fprintf(os.Stderr, "Error leyendo el fichero: %v\n", err)
    }
}
```

4.3. Contar caracteres y palabras de un fichero de texto

Este ejercicio propone leer el fichero `datos.txt` (previamente creado en el ejercicio 4.1) y contar el total de caracteres y el número de espacios en blanco. Se verá cómo `bufio.Scanner` puede configurarse para dividir la entrada por palabras o por bytes.

```
// fichero contarcara.go
// Ejemplo para contar caracteres y palabras leyendo una a una
package main
import ("bufio";"fmt";"io";"os";"unicode" )
func main() {
    fichero, err := os.Open("datos.txt")
    if err != nil {
        fmt.Fprintf(os.Stderr, "Error al abrir fichero datos.txt: %v\n", err)
        os.Exit(1)
    }
    defer fichero.Close()
    // Usamos un Reader en lugar de Scanner para leer carácter a carácter
    reader := bufio.NewReader(fichero)
    contadorCaracteres := 0;contadorPalabras := 0;dentroDePalabra := false
    for {
        // ReadRune lee el siguiente carácter UTF-8
        r, _, err := reader.ReadRune()
        if err == io.EOF {break}
        if err != nil {
            fmt.Fprintf(os.Stderr, "Error leyendo el archivo: %v\n", err)
            os.Exit(1)
        }
        contadorCaracteres++
        if unicode.IsSpace(r) {
            dentroDePalabra = false
        } else {
            if !dentroDePalabra {
                contadorPalabras++
                dentroDePalabra = true
            }
        }
    }
    fmt.Printf("Total caracteres leídos: %d\n", contadorCaracteres)
    fmt.Printf("Total palabras: %d\n", contadorPalabras)
}
```

4.4. Guardar una estructura en un fichero binario (Serialización)

Este ejercicio propone definir una estructura de datos (`struct`), inicializar una instancia y guardarla en un archivo binario. A diferencia de otros lenguajes de programación, en Go no se vuelca la memoria *cruda* de forma insegura; en su lugar, se utilizará el paquete `encoding/gob` (Go Binary) o `encoding/json` para serializar la estructura de forma segura y portable.

```
// fichero creabinario.go
// Ejemplo de serialización de una estructura en fichero binario
package main

import (
    "encoding/gob"
    "fmt"
    "os"
)
// Definición de la estructura.
// IMPORTANTE: Los campos deben empezar por Mayúscula (Exportados)
// para que el paquete 'gob' pueda leerlos y guardarlos.
type RegistroAlumno struct {
    ID     int
    Nombre string
    Nota   float32
}
const NombreFichero = "registro.gob"
// crearBinario crea el fichero y guarda la estructura serializada
func crearBinario() {
    alumnoOriginal := RegistroAlumno{
        ID:     101,
        Nombre: "Sofia Perez",
        Nota:   9.5,
    }
    fichero, err := os.Create(NombreFichero)
    if err != nil {
        fmt.Fprintf(os.Stderr, "Error al crear el fichero: %v\n", err)
        return
    }
    defer fichero.Close()

    // El encoder se encarga de traducir la estructura a bytes binarios
```

```
    encoder := gob.NewEncoder(fichero)

    // Go detecta automáticamente que hay un string, calcula su largo,
    // lo guarda y guarda los datos. No hace falta hacerlo manual.
    err = encoder.Encode(alumnoOriginal)
    if err != nil {
        fmt.Fprintf(os.Stderr, "Error al guardar los datos: %v\n", err)
        return
    }

    fmt.Println("Fichero binario creado correctamente.")
}
// leerBinario lee del fichero y reconstruye la estructura
func leerBinario() {

    var alumnoLeido RegistroAlumno
    fichero, err := os.Open(NombreFichero)
    if err != nil {
        fmt.Fprintf(os.Stderr, "Error al leer el fichero: %v\n", err)
        return
    }
    defer fichero.Close()

    decoder := gob.NewDecoder(fichero)
    // Pasamos un puntero (&) para que Go rellene la variable
    err = decoder.Decode(&alumnoLeido)
    if err != nil {
        fmt.Fprintf(os.Stderr, "Error al decodificar los datos: %v\n", err)
        return
    }

    fmt.Printf("Datos cargados del fichero: ID=%d, Nombre=%s, Nota=%.2f\n",
        alumnoLeido.ID, alumnoLeido.Nombre, alumnoLeido.Nota)
}
func main() {
    crearBinario()
    leerBinario()
}
```

4.5. Insertar al final de un fichero de texto (Append)

Este ejercicio propone, dado un fichero de texto `datos.txt`, escribir una nueva línea al final del mismo sin borrar el contenido anterior. Para ello se utilizarán las banderas (*flags*) de apertura `os.O_APPEND` y `os.O_WRONLY`.

```go
// fichero escribefinal.go
// Ejemplo para escribir al final de un fichero de texto (Append)
package main
import (
    "fmt"
    "os"
)
func main() {
    // Abrir fichero con banderas específicas (Flags)
    // os.O_APPEND: Escribir al final sin borrar lo anterior.
    // os.O_WRONLY: Solo escritura.
    // os.O_CREATE: Si no existe, créalo (comportamiento robusto).
    // 0644: Permisos del archivo (Lectura/Escritura para mí, Lectura para el
    resto).
    fichero, err := os.OpenFile("datos.txt", os.O_APPEND|os.O_WRONLY|os.
    O_CREATE, 0644)

    if err != nil {
        fmt.Fprintf(os.Stderr, "Error al abrir el fichero datos.txt en modo
    append: %v\n", err)
        os.Exit(1)
    }
    defer fichero.Close()
    // Escribir al final
    _, err = fichero.WriteString("Esta es la línea AÑADIDIDA al final (Append)\
    n")

    if err != nil {
        fmt.Fprintf(os.Stderr, "Error al escribir en el fichero: %v\n", err)
        os.Exit(1)
    }

    fmt.Println("\nSe ha añadido una línea al final de 'datos.txt'.")
}
```

4.6. Asignación dinámica con Slices

En Go, rara vez se gestionan punteros manualmente para crear arrays dinámicos como en otros lenguajes. En su lugar, se utilizan *Slices*. Este ejercicio muestra cómo crear un Slice de tamaño N usando la función incorporada `make`, cómo inicializarlo y cómo el **Garbage Collector** se encarga de la liberación de memoria automáticamente, eliminando la necesidad de un comando `delete` explícito.

```
// fichero asignalibera.go
// Ejemplo de asignación dinámica de memoria usando Slices
package main
import (
    "fmt"
)
func main() {
    N := 5

    // Asignación de Memoria
    // Usamos la función incorporada 'make'.
    // make([]int, N) crea un array subyacente de 5 enteros y nos devuelve un "
    Slice" que apunta a él.
    // El Slice es dinámico; el tamaño no necesita ser constante en compilación
    .
    vector := make([]int, N)
    fmt.Printf("Vector Dinámico (Slice) de %d Elementos\n", N)
    for i := 0; i < N; i++ {
        vector[i] = i * 10
        fmt.Printf("%d ", vector[i])
    }
    fmt.Println()
    // Liberación de Memoria
    // ¡NO HACEMOS NADA!
    // Go tiene Garbage Collector.
    fmt.Println("Memoria gestionada automáticamente por el Garbage Collector (
    No hace falta delete).")
}
```

4.7. Trabajo con matrices dinámicas 2D (Slices de Slices)

Este ejercicio nos propone crear una matriz $M \times N$ dinámica. En Go, esto se representa como un "Slice de Slices" (`[][]int`). Aprenderemos a inicializar esta estructura compuesta iterativamente.

```go
// fichero m2d.go
// Ejemplo de matrices dinámicas (Slice de Slices)
package main

import (
    "fmt"
)
func main() {
    M := 3 // Filas
    N := 4 // Columnas

    matriz := make([][]int, M)
    for i := 0; i < M; i++ {
        // Asignamos memoria para la fila 'i' con N columnas
        matriz[i] = make([]int, N)
        for j := 0; j < N; j++ {
            matriz[i][j] = i*N + j + 1
        }
    }
    fmt.Printf("\nMatriz Dinámica %dx%d\n", M, N)
    for i := 0; i < M; i++ {
        for j := 0; j < N; j++ {
            fmt.Printf("%3d ", matriz[i][j])
        }
        fmt.Println()
    }
    // Liberación
    // ¡NADA!
    // El Garbage Collector detecta cuando 'matriz' deja de usarse.
    fmt.Println("Memoria gestionada automáticamente por Go.")
}
```

4.8. Aumentar la capacidad de un Slice (Append)

Este ejercicio demuestra la potencia de la función append de Go. Se asignará un Slice inicial y luego se aumentará su tamaño dinámicamente agregando nuevos elementos. Go gestiona automáticamente la reasignación de memoria subyacente y la copia de datos (similar al `realloc` o `vector::push_back` de C o C++), abstrayendo la complejidad al programador.

```
// fichero aumentarvector.go
// Ejemplo de redimensionado dinámico usando append
package main
import (
    "fmt"
)
func main() {
    vector := make([]int, 3)
    for i := 0; i < len(vector); i++ {
        vector[i] = i + 10
    }
    fmt.Println("Vector Inicial:", vector)
    // len() es la longitud visible, cap() es la capacidad real del array
    subyacente
    fmt.Printf("Info Técnica -> Longitud: %d | Capacidad: %d\n", len(vector),
    cap(vector))
    // Redimensionado (La magia de Go)
    // Queremos llegar a 5 elementos.
    // En Go usamos 'append'.
    nInicial := 3
    nNuevo := 5
    fmt.Println("\n Ejecutando append ...")
    for i := nInicial; i < nNuevo; i++ {
        nuevoValor := i + 100
        vector = append(vector, nuevoValor)
    }
    fmt.Println("Vector Redimensionado:", vector)
    // Fíjate como la capacidad ha crecido (probablemente a 6 u 8, Go suele
    doblar la memoria)
    fmt.Printf("Info Técnica -> Longitud: %d | Capacidad: %d\n", len(vector),
    cap(vector))
    // Liberación. El Garbage Collector lo gestiona todo.
    fmt.Println("Memoria gestionada automáticamente.")
}
```

4.9. Manejo de memoria para almacenar estructuras de datos

Este ejercicio propone utilizar un Slice para almacenar estructuras de datos complejas en tiempo de ejecución. Se verá cómo los Slices pueden contener cualquier tipo de dato, incluyendo `structs` definidos por el usuario, simulando una base de datos en memoria que crece según la necesidad.

```go
// fichero memostruc.go
// Ejemplo de manejo de memoria para estructuras de datos
package main
import (
    "fmt"
)
// Definimos la estructura.
type Tarea struct {
    ID     int
    Nombre string
}
func main() {
    nTareas := 10

    listaTareas := make([]Tarea, nTareas)
    listaTareas[0].ID = 1
    listaTareas[0].Nombre = "Comprar"
    listaTareas[1] = Tarea{
        ID:     2,
        Nombre: "Estudiar",
    }
    listaTareas[2] = Tarea{ID: 3, Nombre: "Correr"}
    fmt.Println("Lista de Tareas:")
    // range devuelve índice (i) y una COPIA del valor (tarea).
    for i, tarea := range listaTareas {
        if tarea.ID != 0 {
            fmt.Printf(" - Tarea %d: %s\n", tarea.ID, tarea.Nombre)
        } else {
            fmt.Printf(" - Tarea %d: <No asignada>\n", i+1)
        }
    }
    // Liberación Automática por el Garbage Collector
    fmt.Println("Memoria de estructuras liberada correctamente (GC).")
}
```

4.10. Persistencia completa: memoria dinámica y ficheros

Este ejercicio combina todo lo aprendido: utilizar un Slice para almacenar estructuras en memoria durante la ejecución y, al finalizar, volcar esos datos a un fichero para recuperarlos posteriormente. Es el fundamento básico de cualquier aplicación que requiera guardar estado.

```
// fichero memostruc2.go
// Ejemplo completo: Estructuras en memoria dinámica y persistencia en fichero.
package main

import (
    "encoding/gob"
    "fmt"
    "io"
    "os"
)
type Tarea struct {
    ID     int
    Nombre string
}
const NombreFichero = "registroestructura.gob"

func main() {

    // Creamos un Slice con datos iniciales
    listaOriginal := []Tarea{
        {ID: 1, Nombre: "Comprar"},
        {ID: 2, Nombre: "Estudiar"},
        {ID: 3, Nombre: "Correr"},
    }
    // Creamos el fichero
    fichero, err := os.Create(NombreFichero)
    if err != nil {
        fmt.Fprintf(os.Stderr, "Error al crear fichero: %v\n", err)
        os.Exit(1)
    }
    // Creamos el codificador (Encoder)
    encoder := gob.NewEncoder(fichero)

    for _, tarea := range listaOriginal {
        err := encoder.Encode(tarea)
```

```
        if err != nil {
            fmt.Fprintf(os.Stderr, "Error guardando tarea: %v\n", err)
        }
    }

    fichero.Close()
    fmt.Println("Vector de escritura liberado automáticamente (GC).")
    fmt.Println("Datos guardados correctamente en disco.")

    // Abrimos el fichero para lectura
    ficheroLectura, err := os.Open(NombreFichero)
    if err != nil {
        fmt.Fprintf(os.Stderr, "Error al abrir fichero: %v\n", err)
        os.Exit(1)
    }
    defer ficheroLectura.Close()

    decoder := gob.NewDecoder(ficheroLectura)
    var listaLeida []Tarea
    var numRegistros int

    fmt.Println("\nLista de Tareas leídas del fichero:")
    for {
        var t Tarea
        err := decoder.Decode(&t)
        if err == io.EOF {
            break
        }
        if err != nil {
            fmt.Fprintf(os.Stderr, "Error de lectura: %v\n", err)
            break
        }
        listaLeida = append(listaLeida, t)
        fmt.Printf(" - Tarea %d: %s\n", t.ID, t.Nombre)
        numRegistros++
    }
    fmt.Printf("Se leyeron %d registros.\n", numRegistros)
    fmt.Println("Vector de lectura liberado automáticamente (GC).")
}
```

Capítulo 5

Algoritmos de ordenación y búsqueda

En este capítulo se realizará un repaso a los algoritmos más destacados de la literatura sobre ordenación y búsqueda implementados en Go [14] [11] [21]. Dichos algoritmos vienen a mejorar los métodos tradicionales para abordar este tipo de problemas y ayudan a optimizar los tiempos de ejecución, haciendo que los programas sean mucho más eficientes en coste de tiempo y recursos. Cuando se trabaja con datos, es muy importante que las estructuras utilizadas tengan unos tiempos de búsqueda eficientes; para ello, normalmente, se suelen utilizar métodos avanzados de inserción en la estructura de datos y de búsqueda. Muchas veces, el método de inserción hace uso de un método de ordenación para conseguir esa eficiencia en las búsquedas, de ahí la importancia de ver estos algoritmos utilizados en problemas.

Nota sobre Tipos en Go: En estos ejercicios se utilizará el tipo `int`. En Go, el tipo `int` depende de la plataforma (generalmente 32 o 64 bits), lo cual suele ser suficiente para la mayoría de algoritmos académicos. Para el uso de enteros más grandes o tipos decimales, simplemente habría que cambiar la firma de las funciones a `int64` o `float64`, ya que la lógica algorítmica permanece idéntica [3]. Aunque Go soporta *Generics* (a partir de la versión 1.18) para crear funciones que acepten cualquier tipo, en este capítulo el foco se centrará en tipos concretos para priorizar la claridad del algoritmo sobre la abstracción sintáctica.

5.1. Ordenación mediante el algoritmo de la Burbuja

Este ejercicio propone, dado un conjunto de números enteros desordenado (un Slice en Go), devolver el conjunto ordenado utilizando el **algoritmo de ordenación de la burbuja**, también conocido como **Bubble Sort** [14]. La simplicidad del algoritmo tiene un coste en eficiencia. La complejidad temporal del Método de la Burbuja es $O(n^2)$. Donde n es el número de elementos [14]. Esto lo hace poco práctico para conjuntos de datos grandes en comparación con otros algoritmos más eficientes como *Quick Sort* o *Merge Sort*.

```go
// fichero burbuja.go
// Implementación del algoritmo de ordenación de la Burbuja (Bubble Sort)
package main
import ("fmt")
// ordenarBurbuja ordena un slice de enteros de menor a mayor.
// En Go, los Slices son "referencias" a un array subyacente.
// Al modificar los elementos dentro de la función, se modifican en el original
    .
func ordenarBurbuja(vec []int) {
    n := len(vec)
    for i := 0; i < n-1; i++ {
        for j := 0; j < n-1-i; j++ {

            if vec[j] > vec[j+1] {
                // INTERCAMBIO (SWAP) IDIOMÁTICO EN GO
                vec[j], vec[j+1] = vec[j+1], vec[j]
            }
        }
    }
}
func main() {
    numeros := []int{10, 7, 8, 9, 1, 5}
    fmt.Print("\nConjunto original: ")
    for _, x := range numeros {fmt.Printf("%d ", x)}
    fmt.Println()
    ordenarBurbuja(numeros)
    fmt.Print("Conjunto ordenado (Bubble Sort): ")
    for _, x := range numeros {fmt.Printf("%d ", x)}
    fmt.Println()
}
```

5.2. Ordenación mediante el algoritmo QuickSort

Este ejercicio propone, dado un conjunto de números desordenado, devolver el conjunto ordenado utilizando el **algoritmo de ordenación Quicksort** [21] [14]. Quicksort es un algoritmo de ordenación extremadamente eficiente y ampliamente utilizado. Es conocido por ser uno de los más rápidos en la práctica para grandes conjuntos de datos. En el caso **Promedio y Mejor** su orden de complejidad es de $O(n \log n)$ y en el **peor de los casos** es de $O(n^2)$ [21]. Este último caso resulta en subproblemas muy desequilibrados ($n-1$ y 0), forzando al algoritmo a realizar un número de comparaciones similar al del *Bubble Sort*. Aunque Go proporciona funciones de ordenación optimizadas en sus paquetes estándar, en este ejercicio se implementará el algoritmo **manualmente** para comprender su mecanismo recursivo de partición.

```go
// fichero qsort.go
// Implementación manual del algoritmo QuickSort
package main

import (
    "fmt"
)

// particion toma el último elemento como pivote y coloca
// los menores a la izquierda y los mayores a la derecha.
func particion(arr []int, bajo, alto int) int {

    pivote := arr[alto] // Elegimos el último elemento como pivote
    i := bajo - 1       // Índice del elemento más pequeño

    for j := bajo; j < alto; j++ {
        // Si el elemento actual es menor o igual al pivote
        if arr[j] <= pivote {
            i++
            // Swap: arr[i] y arr[j]
            arr[i], arr[j] = arr[j], arr[i]
        }
    }
    // Colocamos el pivote en su posición correcta (i + 1)
    arr[i+1], arr[alto] = arr[alto], arr[i+1]
    return i + 1
}
```

```
// quicksort función principal recursiva
func quicksort(arr []int, bajo, alto int) {

    if bajo < alto {
        pi := particion(arr, bajo, alto)
        quicksort(arr, bajo, pi-1)
        quicksort(arr, pi+1, alto)
    }
}

func main() {

    datos := []int{10, 70, 8, 90, 1000, 5}
    n := len(datos)
    fmt.Print("Vector original: ")
    for _, x := range datos {
        fmt.Printf("%d ", x)
    }
    fmt.Println()
    quicksort(datos, 0, n-1)
    fmt.Print("Vector ordenado con Quicksort: ")
    for _, x := range datos {
        fmt.Printf("%d ", x)
    }
    fmt.Println()
}
```

5.3. Ordenación mediante el paquete estándar `slices` (PdQsort)

Este ejercicio propone, dado un conjunto de números desordenados, devolver el conjunto ordenado utilizando las herramientas optimizadas del lenguaje. En Go, esto se realiza habitualmente mediante el paquete `sort` (método clásico) o el paquete `slices` (método moderno introducido en Go 1.21). Estas implementaciones suelen utilizar una variante de Quicksort llamada **PdQsort** (Pattern-Defeating Quicksort) [15], que es híbrida y muy eficiente.

```
// fichero qsort2.go
// Ordenación utilizando Pattern-Defeating Quicksort (pdqsort)
package main

import (
    "fmt"
    "slices" // Paquete moderno para operaciones con slices (Go 1.21+)
)

func main() {

    numeros := []int{50, 10, 8, 20, 40}

    fmt.Print("Vector antes de ordenar: ")
    fmt.Println(numeros)

    // Go utiliza internamente 'pdqsort' (Pattern-Defeating Quicksort),
    // un algoritmo híbrido extremadamente rápido (O(n log n)).

    slices.Sort(numeros)

    fmt.Print("Vector después de ordenar: ")
    fmt.Println(numeros)
}
```

5.4. Ordenación mediante el algoritmo MergeSort

Este ejercicio propone, dado un conjunto de números desordenado, devolver el conjunto ordenado utilizando el **algoritmo de ordenación MergeSort** [21]. **MergeSort** es otro algoritmo de ordenación muy eficiente, basado en el principio de "Divide y Vencerás" [14]. A diferencia de Quicksort, su rendimiento es consistente: siempre $O(n \log n)$ en el mejor, promedio y peor caso [14]. En Go, el uso de *Slices* facilita enormemente la división de los arrays en subconjuntos sin necesidad de copiar memoria constantemente.

```
// fichero mergesort.go
// Implementación del algoritmo MergeSort (Divide y Vencerás)
package main

import ("fmt")
// funcion merge: Fusiona dos subconjuntos ordenados en un solo conjunto
    ordenado.
// Corresponde a la fase de "Vencer" o "Conquistar".
func merge(arr []int, inicio, medio, fin int) {

    tamano := fin - inicio + 1
    temp := make([]int, tamano)
    i := inicio    // Puntero para la mitad izquierda
    j := medio + 1 // Puntero para la mitad derecha
    k := 0         // Puntero para el array temporal
    for i <= medio && j <= fin {
        if arr[i] <= arr[j] {
            temp[k] = arr[i]
            i++
        } else {
            temp[k] = arr[j]
            j++
        }
        k++
    }
    // Si sobran elementos en la mitad izquierda, los copiamos
    for i <= medio {
        temp[k] = arr[i]
        k++
        i++
    }
    // Si sobran elementos en la mitad derecha, los copiamos
```

```
    for j <= fin {
        temp[k] = arr[j]
        k++
        j++
    }
    // Copiamos el array temporal ordenado de vuelta al array original
    // copy(destino, origen) es la forma eficiente en Go
    copy(arr[inicio:fin+1], temp)
}
// mergeSort: Función recursiva que divide el array
func mergeSort(arr []int, inicio, fin int) {
    if inicio < fin {
        // Encontramos el punto medio para dividir
        // Evitamos desbordamiento (overflow) usando esta fórmula en lugar de (
    inicio+fin)/2
        medio := inicio + (fin-inicio)/2
        // DIVIDIR
        mergeSort(arr, inicio, medio)
        mergeSort(arr, medio+1, fin)
        //VENCER (Mezclar las mitades ordenadas)
        merge(arr, inicio, medio, fin)
    }
}
func main() {

    datos := []int{38, 27, 43, 3, 9, 82, 10}
    n := len(datos)
    fmt.Println("Vector original desordenado:")
    for _, x := range datos {
        fmt.Printf("%d ", x)
    }
    fmt.Println()
    mergeSort(datos, 0, n-1)
    fmt.Println("\nVector ordenado con MergeSort:")
    for _, x := range datos {
        fmt.Printf("%d ", x)
    }
    fmt.Println()
}
```

5.5. Ordenación mediante el algoritmo ShellSort

Este ejercicio propone, dado un conjunto de números desordenado, devolver el conjunto ordenado utilizando el **algoritmo de ordenación ShellSort** [21]. El rendimiento del algoritmo **ShellSort** depende crucialmente de la elección de la secuencia de saltos o intervalos h. Una de las secuencias más eficientes y comunes, que optimiza significativamente el tiempo de ejecución en el peor caso, es la propuesta por D. Knuth [11]. La secuencia de intervalos de Knuth [11] se define recursivamente mediante la fórmula $h_{k+1} = 3h_k + 1$. El punto de partida de la secuencia es $h_1 = 1$. Los primeros términos de esta secuencia son: $1, 4, 13, 40, 121, 364, 1093, \ldots$

Utilizar la secuencia de Knuth [11] proporciona al ShellSort una complejidad temporal significativamente mejor que las secuencias triviales (como $h = N/2$): Se ha demostrado que con esta secuencia, el ShellSort alcanza un rendimiento cercano a $O(n^{3/2})$ en el peor caso, o incluso mejor, en torno a $O(n \log^2 n)$, superando con creces la complejidad $O(n^2)$ de los algoritmos de ordenación simples.

```go
// fichero shellsort.go
// Implementación de ShellSort usando la secuencia de Knuth (h = 3*h + 1)
package main

import (
    "fmt"
)

// shellsortKnuth ordena el slice in-place usando saltos variables.
func shellsortKnuth(arr []int) {

    n := len(arr)
    h := 1
    // Calcular el salto inicial (Secuencia de Knuth)
    // La secuencia es: 1, 4, 13, 40, 121...
    // Buscamos el h más grande posible que sea menor que n/3
    for h < n/3 {
        h = 3*h + 1
    }
    // vamos reduciendo el salto 'h'
    for h >= 1 {
        // Insertion Sort modificado con salto 'h'
        // En lugar de comparar con el vecino inmediato (i-1), comparamos con (
    i-h)
        for i := h; i < n; i++ {
```

```
            temp := arr[i]
            j := i
            // Desplazamos los elementos mayores hacia la derecha (saltando h
    posiciones)
            // j >= h verifica que no nos salgamos del array por la izquierda
            for j >= h && arr[j-h] > temp {
                arr[j] = arr[j-h]
                j -= h
            }
            // Colocamos el elemento temporal en su hueco
            arr[j] = temp
        }
        // Reducimos el salto según la inversa de Knuth
        h = h / 3
    }
}

func main() {

    datos := []int{90, 8, 70, 6, 50, 4, 30, 2, 10, 0, 85, 15, 65, 35}

    fmt.Println("Array original desordenado:")
    fmt.Println(datos)

    shellsortKnuth(datos)

    fmt.Println("\nArray ordenado con Shellsort (Knuth):")
    fmt.Println(datos)

}
```

5.6. Ordenación mediante el algoritmo RadixSort

Este ejercicio propone, dado un conjunto de números desordenado, devolver el conjunto ordenado utilizando el **algoritmo de ordenación RadixSort** [21] [11]. **Radix Sort** (ordenación por base o por dígitos), es un algoritmo de ordenación no comparativo que organiza los elementos procesando sus dígitos de forma secuencial [21]. En lugar de comparar directamente los valores entre sí (como hacen, por ejemplo, *QuickSort* o *MergeSort*), este algoritmo clasifica los números según cada posición decimal (unidades, decenas, centenas, etc.), utilizando un algoritmo de ordenación **estable** en cada paso, normalmente *Counting Sort*. Counting Sort (ordenación por conteo) [21] [11], es un algoritmo no comparativo que se utiliza para ordenar números enteros dentro de un rango limitado. En lugar de comparar elementos entre sí, cuenta cuántas veces aparece cada valor, y a partir de esas frecuencias reconstruye el slice ordenado. La complejidad temporal es $O(d \cdot (n + k))$ y la complejidad espacial es $O(n + k)$. El algoritmo *Radix Sort* es un método de ordenación eficiente para números enteros cuando los valores son de longitud limitada. Su rendimiento cercano a $O(n)$ lo convierte en una buena opción para grandes volúmenes de datos, siempre que la base y la longitud de los números no sean demasiado grandes [21] [11].

```go
// fichero radixsort.go
// Implementación de RadixSort (LSD - Least Significant Digit)
package main
import ("fmt")

func getMax(arr []int) int {
    max := arr[0]
    for _, v := range arr {
        if v > max {
            max = v
        }
    }
    return max
}
// countingSort realiza un ordenamiento estable basado en el dígito
    representado por 'exp'
// exp = 1 (unidades), 10 (decenas), 100 (centenas)...
func countingSort(arr []int, exp int) {
    n := len(arr)
    output := make([]int, n)
    count := make([]int, 10)

    // Contar frecuencias del dígito actual
```

```
    for i := 0; i < n; i++ {
        digit := (arr[i] / exp) % 10
        count[digit]++
    }
    //  Transformar conteo en posiciones (acumulado)
    for i := 1; i < 10; i++ {
        count[i] += count[i-1]
    }
    // IMPORTANTE: Recorremos el array original HACIA ATRÁS (n-1 hasta 0).
    // Esto garantiza que el algoritmo sea ESTABLE (Stable Sort).
    // Si dos números tienen el mismo dígito, el que aparecía después se
    mantiene después.
    for i := n - 1; i >= 0; i-- {
        digit := (arr[i] / exp) % 10
        output[count[digit]-1] = arr[i]
        count[digit]--
    }
    copy(arr, output)
}
// radixSort función principal
func radixSort(arr []int) {
    if len(arr) == 0 {
        return
    }
    m := getMax(arr)
    for exp := 1; m/exp > 0; exp *= 10 {
        countingSort(arr, exp)
    }
}
func main() {

    arr := []int{170, 45, 75, 90, 802, 24, 2, 66}
    fmt.Println("Array original:")
    fmt.Println(arr)
    radixSort(arr)
    fmt.Println("\nArray ordenado:")
    fmt.Println(arr)
}
```

5.7. Ordenación mediante el algoritmo HeapSort

Este ejercicio propone, dado un conjunto de números desordenado, devolver el conjunto ordenado utilizando el **algoritmo de ordenación HeapSort** [21]. **HeapSort** (Ordenación por Montículo) es un algoritmo de ordenación por comparación que utiliza la estructura de datos **montículo binario (heap)** para ordenar elementos. Es un algoritmo eficiente, con una complejidad temporal de $\mathcal{O}(n \log n)$ en el mejor, promedio y peor caso. HeapSort se basa en la propiedad de un **Max-Heap** (Montículo Máximo). Un Max-Heap [21] es un árbol binario completo donde:

- El **valor de cada nodo es mayor o igual** que los valores de sus hijos.
- El **elemento más grande** de todo el montículo siempre está en la **raíz** del árbol.

Go posee un paquete estándar `container/heap`, pero en primer lugar se implementará la lógica manualmente para fines educativos y seguidamente se verá cómo usar el paquete.

```
// fichero heapsort.go
// Implementación manual de HeapSort (Ordenación por Montículo)
package main

import ("fmt")

// heapify asegura que el subárbol con raíz en el índice 'i' cumpla la
    propiedad de Max-Heap.
// n es el tamaño del heap "activo" (va disminuyendo conforme ordenamos).
func heapify(arr []int, n int, i int) {
    largest := i       // Inicializamos el más grande como la raíz
    left := 2*i + 1    // Hijo izquierdo
    right := 2*i + 2   // Hijo derecho

    // Si el hijo izquierdo es más grande que la raíz
    if left < n && arr[left] > arr[largest] {
        largest = left
    }
    // Si el hijo derecho es más grande que el más grande hasta ahora
    if right < n && arr[right] > arr[largest] {
        largest = right
    }
    // Si el más grande no es la raíz
    if largest != i {
        // Hacemos SWAP: la raíz baja y el hijo grande sube
```

```
        arr[i], arr[largest] = arr[largest], arr[i]
        // Recursivamente aplicamos heapify al subárbol afectado
        heapify(arr, n, largest)
    }
}

// heapSort es la función principal
func heapSort(arr []int) {
    n := len(arr)

    // Construir el Heap (Build Max-Heap)
    // Empezamos desde el último nodo que NO es hoja (n/2 - 1) hacia arriba.
    for i := n/2 - 1; i >= 0; i-- {
        heapify(arr, n, i)
    }
    // Extraer elementos del heap uno a uno
    for i := n - 1; i > 0; i-- {
        // Movemos la raíz actual (el máximo) al final del array
        arr[0], arr[i] = arr[i], arr[0]
        // Llamamos a heapify en el heap reducido (tamaño i)
        // para restaurar la propiedad de Max-Heap en la raíz.
        heapify(arr, i, 0)
    }
}

func main() {

    arr := []int{12, 11, 13, 5, 6, 7}
    fmt.Println("Array original:")
    fmt.Println(arr)
    heapSort(arr)
    fmt.Println("Array ordenado con Heapsort:")
    fmt.Println(arr)
}
```

HeapSort usando el paquete container/heap (más simple)

```
// fichero heapsort2.go
// Ejemplo de uso de container/heap para ordenar
package main
import ("container/heap";"fmt")
// Definimos un tipo personalizado (un slice de enteros)
type IntHeap []int
// Implementamos la interfaz sort.Interface
func (h IntHeap) Len() int { return len(h) }
// Less define el orden.
// Si usamos < (menor que), construimos un Min-Heap (el menor arriba).
// Al extraer elementos de un Min-Heap uno a uno, obtendremos una lista
    ordenada ascendente.
func (h IntHeap) Less(i, j int) bool { return h[i] < h[j] }
func (h IntHeap) Swap(i, j int) { h[i], h[j] = h[j], h[i] }
// Implementamos Push y Pop (requisitos de heap.Interface)
// Usa un puntero (*IntHeap) porque Push y Pop modifican el tamaño del slice.
func (h *IntHeap) Push(x any) {*h = append(*h, x.(int))}
func (h *IntHeap) Pop() any {
    old := *h
    n := len(old)
    x := old[n-1]     // Tomamos el último elemento
    *h = old[0 : n-1] // Reducimos el slice
    return x
}
func main() {
    // Creamos el slice y lo convertimos en un puntero a nuestra estructura
    nums := &IntHeap{12, 11, 13, 5, 6, 7}
    fmt.Println("Datos originales:", *nums)
    // heap.Init reordena el slice para cumplir la propiedad de montículo. O(n)
    heap.Init(nums)
    fmt.Println("Tras heap.Init (Min-Heap):", *nums)
    fmt.Print("Extrayendo ordenadamente: ")
    // Extracción ordenada (HeapSort)
    // Mientras queden elementos, sacamos el mínimo (Pop)
    for nums.Len() > 0 {
        val := heap.Pop(nums)
        fmt.Printf("%d ", val)
    }
    fmt.Println()
}
```

5.8. Búsqueda Lineal: Encontrar un número en un conjunto de datos

Este ejercicio propone, dado un conjunto de números, buscar un número perteneciente al conjunto utilizando el **algoritmo de búsqueda lineal**. La Búsqueda Lineal (o secuencial) [14] es el algoritmo de búsqueda más simple. Su complejidad temporal es de orden **lineal** ($O(n)$) porque el tiempo de ejecución crece directamente con el número de elementos (n) del array. La complejidad es $O(1)$ si el elemento se encuentra en la **primera posición**. En promedio, se espera buscar $\frac{n}{2}$ elementos, lo que sigue siendo proporcional a n, que es el peor de los casos. Para conjuntos de datos grandes y **ordenados**, la **Búsqueda Binaria** es mucho más eficiente, con una complejidad temporal de $O(\log n)$. En este ejercicio se verán 2 métodos: la implementación mediante función y la propia de Go para los slices.

```
// fichero blineal.go
// Implementación del algoritmo de Búsqueda Lineal (Secuencial)
package main
import ("fmt";"slices")

// busquedaLineal recorre el slice elemento a elemento.
// Retorna el índice si lo encuentra, o -1 si no existe.
func busquedaLineal(arr []int, objetivo int) int {
    // range devuelve (índice, valor)
    for i, v := range arr {
        if v == objetivo {
            return i // Encontrado
        }
    }
    return -1 // No encontrado tras recorrer todo
}
func main() {
    datos := []int{10, 5, 20, 15, 8, 30}
    objetivo1 := 15
    fmt.Printf("Conjunto de datos: %v\n", datos)
    //Método manual
    indice := busquedaLineal(datos, objetivo1)
    if indice != -1 {
        fmt.Printf("Resultado para %d: Encontrado en índice %d.\n", objetivo1,
    indice)
    } else {
```

```
        fmt.Printf("Resultado para %d: No encontrado\n", objetivo1)
    }
    //Usando la fórmula de slices.
    indice = slices.Index(datos, objetivo1)
    if indice != -1 {
        fmt.Printf("Resultado para %d: Encontrado en índice %d.\n", objetivo1,
    indice)
    } else {
        fmt.Printf("Resultado para %d: No encontrado\n", objetivo1)
    }
}
```

5.9. Búsqueda Binaria: Encontrar un número en un conjunto ordenado

Este ejercicio propone, dado un conjunto de números ordenados, buscar un número perteneciente al conjunto utilizando el **algoritmo de búsqueda binaria** [14]. La Búsqueda Binaria utiliza la estrategia de "Divide y Vencerás" [14]. La principal ventaja de la Búsqueda Binaria sobre la Búsqueda Lineal es su velocidad. La complejidad temporal es logarítmica, lo que significa que el tiempo de ejecución crece muy lentamente a medida que aumenta el tamaño de los datos, siendo esta $O(\log n)$. En este ejercicio se verán 2 métodos: la implementación mediante función y la propia de Go para los slices.

```
// fichero bbinaria.go
// Implementación del algoritmo de Búsqueda Binaria (Iterativa)
package main

import ("fmt";"slices" )

// busquedaBinaria asume que el slice 'arr' está ORDENADO de menor a mayor.
// Devuelve el índice del elemento o -1 si no existe.
func busquedaBinaria(arr []int, objetivo int) int {
    bajo := 0
    alto := len(arr) - 1
    for bajo <= alto {
        // Calculamos el punto medio.
        // Usamos "bajo + (alto-bajo)/2" en lugar de "(bajo + alto)/2"
        medio := bajo + (alto-bajo)/2
```

```
        if arr[medio] == objetivo {
            return medio // ¡Encontrado!
        }
        if arr[medio] < objetivo {
            // El objetivo está en la mitad superior
            bajo = medio + 1
        } else {
            // El objetivo está en la mitad inferior
            alto = medio - 1
        }
    }
    return -1 // No encontrado
}

func main() {

    datosOrdenados := []int{5, 8, 12, 15, 20, 30, 40}
    objetivo1 := 20
    fmt.Printf("Conjunto de datos (Ordenado): %v\n", datosOrdenados)
    indice := busquedaBinaria(datosOrdenados, objetivo1)
    if indice != -1 {
        fmt.Printf("Resultado para %d: Encontrado (función manual) en el índice
     %d.\n", objetivo1, indice)
    } else {
        fmt.Printf("Resultado para %d: No encontrado.\n", objetivo1)
    }
    // slices.BinarySearch devuelve el índice y un booleano 'found'
    idx, found := slices.BinarySearch(datosOrdenados, objetivo1)
    if found { indice = idx } else { indice = -1 }
    if indice != -1 {
        fmt.Printf("Resultado para %d: Encontrado (función de slices) en el í
    ndice %d.\n", objetivo1, indice)
    } else {
        fmt.Printf("Resultado para %d: No encontrado.\n", objetivo1)
    }
}
```

5.10. Búsqueda por Interpolación: Encontrar un número en un conjunto ordenado

Este ejercicio propone, dado un conjunto de números enteros no negativos ordenados, buscar un número perteneciente al conjunto utilizando el **algoritmo de búsqueda por interpolación** [21] [11]. Éste es un algoritmo de búsqueda diseñado para conjuntos de datos ordenados que asume una distribución uniforme de los elementos. Es una mejora de la Búsqueda Binaria, ya que no solo divide el espacio de búsqueda a la mitad, sino que estima la posición del elemento objetivo basándose en su valor. En el caso promedio, y cuando los datos están uniformemente distribuidos, la Búsqueda por Interpolación es más rápida que la Búsqueda Binaria ($O(\log \log n)$ vs $O(\log n)$). En el peor de los casos son idénticos $O(n)$.

```
// fichero binterpola.go
// Implementación del algoritmo de Búsqueda por Interpolación
package main
import ("fmt")
// busquedaInterpolacion busca un elemento estimando su posición basándose en
    su valor.
// Requiere que el array esté ORDENADO y uniformemente distribuido.
func busquedaInterpolacion(arr []int, objetivo int) int {
    bajo := 0
    alto := len(arr) - 1
    // El bucle continúa mientras el objetivo esté dentro del rango posible de
    valores
    // arr[bajo] <= objetivo <= arr[alto]
    for bajo <= alto && objetivo >= arr[bajo] && objetivo <= arr[alto] {
        // Caso base: si el rango se ha reducido a un solo punto
        if bajo == alto {
            if arr[bajo] == objetivo {
                return bajo
            }
            return -1
        }
        // Si los valores extremos son iguales, evitamos división por cero
        if arr[alto] == arr[bajo] {
            if arr[bajo] == objetivo { return bajo }
            return -1
        }
        // FÓRMULA DE INTERPOLACIÓN:
        // pos = bajo + ((alto - bajo) / (arr[alto] - arr[bajo])) * (objetivo -
```

```
    arr[bajo])
        // En Go, la división de enteros devuelve un entero (trunca).
        // Para obtener la precisión decimal necesaria para la interpolación,
        // DEBEMOS convertir explícitamente a float64.
        numerador := float64(alto - bajo)
        denominador := float64(arr[alto] - arr[bajo])
        distanciaValor := float64(objetivo - arr[bajo])
        fraction := numerador / denominador
        pos := bajo + int(fraction * distanciaValor)
        if pos < bajo { pos = bajo }
        if pos > alto { pos = alto }
        if arr[pos] == objetivo {
            return pos
        }
        if arr[pos] < objetivo {
            bajo = pos + 1
        } else {
            alto = pos - 1
        }
    }
    return -1
}
func main() {
    datosOrdenados := []int{10, 20, 30, 40, 50, 60, 70, 80, 90, 100}
    objetivo1 := 70
    indice := busquedaInterpolacion(datosOrdenados, objetivo1)
    fmt.Printf("\nResultado para %d: ", objetivo1)
    if indice != -1 {
        fmt.Printf("Encontrado en el índice %d.\n", indice)
    } else {
        fmt.Println("No encontrado.")
    }
}
```

Además de todos los algoritmos revisados anteriormente, existen otros algoritmos de búsqueda sobre estructuras de datos más complejas como árboles o grafos, los cuales quedan fuera de este libro de problemas básicos. Para más información sobre algoritmos avanzados véase [21] [14] [11].

Capítulo 6

Ejercicios de Criptografía Básica

La criptografía es la ciencia que se encarga de proteger la información mediante el uso de algoritmos matemáticos, asegurando que el mensaje sólo pueda ser leído por el destinatario previsto. En el ámbito de la programación en Go, la implementación de algoritmos criptográficos básicos es un excelente ejercicio para dominar el manejo de **strings**, **slices de bytes**, la aritmética modular y las operaciones a nivel de bits (bitwise).

Históricamente, la criptografía se divide en clásica y moderna. En este capítulo el foco se centrará principalmente en la **criptografía clásica** y **simétrica básica**, que, aunque no es segura para estándares modernos, sienta las bases lógicas de los sistemas actuales. Go posee una biblioteca estándar extremadamente potente para criptografía moderna (paquetes `crypto/sha256`, `crypto/aes`, etc.), pero aquí se implementará la lógica manualmente con fines educativos. Para profundizar en la teoría matemática y estándares modernos, se recomienda consultar obras de referencia como las de Bruce Schneier [16], Douglas Stinson [17] o Manuel Lucena [12].

A continuación, se presenta una serie de ejercicios que van desde el cifrado por desplazamiento hasta la manipulación de bits mediante operaciones XOR.

6.1. Implementar el cifrado César (Desplazamiento)

Este ejercicio propone implementar el famoso **Cifrado César** [12], uno de los métodos de codificación más antiguos y simples. Es un tipo de cifrado por sustitución en el que una letra en el texto original es reemplazada por otra letra que se encuentra un número fijo de posiciones más adelante en el alfabeto. Para cifrar una letra x con un desplazamiento n:

$$E_n(x) = (x + n) \mod 26$$

El programa debe leer una cadena y un entero n (la clave), y mostrar la cadena cifrada. En Go, se utilizará el tipo `rune` para manejar los caracteres Unicode y el paquete `strings.Builder`.

```
// fichero cesar_cifrar.go
package main
import ("bufio";"fmt";"os";"strings";"unicode")
// cifrarCesar desplaza cada letra 'n' posiciones.
func cifrarCesar(texto string, n int) string {
    var sb strings.Builder
    shift := n % 26
    for _, r := range texto {
        if unicode.IsLetter(r) {
            base := 'A'
            if unicode.IsLower(r) {base = 'a'}
            nuevo := base + (r-base+rune(shift))%26
            sb.WriteRune(nuevo)
        } else {
            sb.WriteRune(r)
        }
    }
    return sb.String()
}
func main() {
    reader := bufio.NewReader(os.Stdin)
    fmt.Print("Introduce el mensaje a cifrar: ")
    mensaje, _ := reader.ReadString('\n')
    mensaje = strings.TrimSpace(mensaje)
    fmt.Print("Introduce la clave (número de desplazamiento): ")
    var n int
    fmt.Scanf("%d", &n)
    cifrado := cifrarCesar(mensaje, n)
    fmt.Printf("\nMensaje Cifrado: %s\n", cifrado)
}
```

6.2. Descifrar el código César con clave conocida

Este ejercicio es la operación inversa al anterior 6.1. Dado un texto cifrado y la clave n utilizada, se debe recuperar el mensaje original. La fórmula para el descifrado es [12]:

$$D_n(x) = (x - n) \mod 26$$

Es importante manejar correctamente los valores negativos en Go al realizar la operación módulo, ya que el operador `%` puede devolver negativos. Se debe normalizar el resultado para asegurar que apunte a un índice válido del alfabeto.

```
// fichero cesar_descifrar.go
package main
import ("bufio";"fmt";"os";"strings";"unicode")
func descifrarCesar(texto string, n int) string {
    var sb strings.Builder
    shift := n % 26
    for _, r := range texto {
        if unicode.IsLetter(r) {
            base := 'A'
            if unicode.IsLower(r) {base = 'a'}
            tmp := (r - base - rune(shift)) % 26
            if tmp < 0 {tmp += 26}
            nuevo := base + tmp
            sb.WriteRune(nuevo)
        } else {
            sb.WriteRune(r)
        }
    }
    return sb.String()
}
func main() {
    reader := bufio.NewReader(os.Stdin)
    fmt.Print("Introduce el mensaje cifrado: ")
    mensaje, _ := reader.ReadString('\n')
    mensaje = strings.TrimSpace(mensaje)
    fmt.Print("Introduce la clave original: ")
    var n int
    fmt.Scanf("%d", &n)
    claro := descifrarCesar(mensaje, n)
    fmt.Printf("\nMensaje Descifrado: %s\n", claro)
}
```

6.3. Ataque de fuerza bruta al cifrado César

Este ejercicio propone, dado un texto cifrado mediante el método César del cual se desconoce la clave, mostrar por pantalla **todas las posibles combinaciones** de descifrado (las 25 rotaciones posibles). Este es un ejemplo básico de criptoanálisis mediante fuerza bruta, demostrando la debilidad de los cifrados con un espacio de claves pequeño. El usuario deberá leer la salida y determinar visualmente cuál es el mensaje con sentido.

```go
// fichero cesar_bruta.go
package main
import ("bufio";"fmt";"os";"strings";"unicode")
func descifrar(texto string, n int) string {
    var sb strings.Builder
    shift := n % 26
    for _, r := range texto {
        if unicode.IsLetter(r) {
            base := 'A'
            if unicode.IsLower(r) {base = 'a'}
            tmp := (r - base - rune(shift)) % 26
            if tmp < 0 { tmp += 26 }
            sb.WriteRune(base + tmp)
        } else {
            sb.WriteRune(r)
        }
    }
    return sb.String()
}
func main() {
    reader := bufio.NewReader(os.Stdin)
    fmt.Print("Introduce el mensaje cifrado para atacar: ")
    mensaje, _ := reader.ReadString('\n')
    mensaje = strings.TrimSpace(mensaje)
    fmt.Println("\n Iniciando Ataque de Fuerza Bruta ")
    for k := 1; k < 26; k++ {
        candidato := descifrar(mensaje, k)
        fmt.Printf("Clave %2d: %s\n", k, candidato)
    }
    fmt.Println("Fin del Ataque")
    fmt.Println("Busca visualmente la frase que tenga sentido.")
}
```

6.4. Cifrado y descifrado mediante operación XOR

Este ejercicio introduce el concepto de cifrado de flujo simétrico utilizando la operación **XOR** ($\oplus$) [16]. La operación XOR tiene una propiedad fundamental para la informática: $(A \oplus K) \oplus K = A$ es su propia inversa. El programa debe solicitar una cadena de texto y una clave. En Go, se convertirá el string a un slice de bytes (`[]byte`) para aplicar el operador ^ (XOR) byte a byte. Al aplicar la misma operación al resultado, se obtendrá el mensaje original.

```
// fichero xor_cifra_descifra.go
package main
import ("bufio";"fmt";"os";"strings")
// cifrarDescifrarXOR aplica la operación XOR byte a byte.
// Al ser XOR una operación involutiva ((A ^ B) ^ B = A), la misma función
    sirve para cifrar y descifrar.
func cifrarDescifrarXOR(input, clave string) string {
    inputBytes := []byte(input)
    claveBytes := []byte(clave)
    output := make([]byte, len(inputBytes))
    for i := 0; i < len(inputBytes); i++ {
        output[i] = inputBytes[i] ^ claveBytes[i%len(claveBytes)]
    }
    return string(output)
}
func main() {
    reader := bufio.NewReader(os.Stdin)
    fmt.Print("Introduce el mensaje: ")
    mensaje, _ := reader.ReadString('\n')
    mensaje = strings.TrimSpace(mensaje)
    fmt.Print("Introduce la clave: ")
    clave, _ := reader.ReadString('\n')
    clave = strings.TrimSpace(clave)
    resultado := cifrarDescifrarXOR(mensaje, clave)
    fmt.Println("\nResultado")
    fmt.Printf("Hex:   %x\n", resultado)
    recuperado := cifrarDescifrarXOR(resultado, clave)
    fmt.Printf("\nVerificación (Descifrando de nuevo): %s\n", recuperado)
}
```

6.5. Implementar el cifrado de Vigenère

Este ejercicio propone implementar el **Cifrado de Vigenère** [12], un método de cifrado polialfabético que mejora la seguridad del César utilizando una palabra clave en lugar de un sólo número. La clave se repite tantas veces como sea necesario para igualar la longitud del mensaje. Si el mensaje es M y la clave es K, el carácter cifrado C_i en la posición i es: $C_i = (M_i + K_{i \pmod L}) \bmod 26$ donde L es la longitud de la clave.

```
//vigenere.go
package main
import ("bufio";"fmt";"os";"strings";"unicode")
func vigenere(texto, clave string, cifrar bool) string {
    var sb strings.Builder
    claveRunes := []rune(strings.ToUpper(clave))
    kLen := len(claveRunes)
    kIndex := 0 // Índice para recorrer la clave
    for _, r := range texto {
        if unicode.IsLetter(r) {
            base := 'A'
            if unicode.IsLower(r) {base = 'a'}
            valTexto := int(r - base)
            valClave := int(claveRunes[kIndex%kLen] - 'A')
            var nuevoVal int
            if cifrar {
                nuevoVal = (valTexto + valClave) % 26
            } else {
                nuevoVal = (valTexto - valClave + 26) % 26
            }
            sb.WriteRune(base + rune(nuevoVal))
            kIndex++
        } else {
            sb.WriteRune(r)
        }
    }
    return sb.String()
}
func main() {
    reader := bufio.NewReader(os.Stdin)
    fmt.Print("Mensaje: ")
    msg, _ := reader.ReadString('\n')
    msg = strings.TrimSpace(msg)
```

```
    fmt.Print("Clave: ")
    key, _ := reader.ReadString('\n')
    key = strings.TrimSpace(key)
    cifrado := vigenere(msg, key, true)
    fmt.Printf("Cifrado: %s\n", cifrado)
    descifrado := vigenere(cifrado, key, false)
    fmt.Printf("Descifrado: %s\n", descifrado)
}
```

6.6. Cifrado por sustitución monoalfabética con clave aleatoria

A diferencia del César, donde el alfabeto se desplaza, este ejercicio propone utilizar un alfabeto completamente desordenado como clave. Se debe generar (o introducir) una cadena de 26 caracteres únicos que represente la permutación del alfabeto normal. Ejemplo de correspondencia: A $\rightarrow$ Q, B $\rightarrow$ W, C $\rightarrow$ E ...

El programa sustituirá cada letra del mensaje original por su correspondiente en el alfabeto de sustitución, utilizando un `map` o arrays de correspondencia.

```
//Fichero sustitucion.go
package main
import (
    "bufio"
    "fmt"
    "os"
    "strings"
    "unicode"
)

// Alfabeto desordenado de ejemplo (Clave)
const alfabetoNormal = "abcdefghijklmnopqrstuvwxyz"
const alfabetoClave  = "qwertyuiopasdfghjklzxcvbnm"

func sustitucion(texto string, mapeo map[rune]rune) string {

    var sb strings.Builder
    for _, r := range texto {
        // Convertimos a minúscula para buscar en el mapa
        lower := unicode.ToLower(r)
        if val, ok := mapeo[lower]; ok {
            // Si es mayúscula, convertimos el resultado a mayúscula
```

```
            if unicode.IsUpper(r) {
                sb.WriteRune(unicode.ToUpper(val))
            } else {
                sb.WriteRune(val)
            }
        } else {
            sb.WriteRune(r)
        }
    }
    return sb.String()
}

func main() {

    mapero := make(map[rune]rune)
    runesNormal := []rune(alfabetoNormal)
    runesClave := []rune(alfabetoClave)

    for i := 0; i < 26; i++ {
        mapero[runesNormal[i]] = runesClave[i]
    }

    reader := bufio.NewReader(os.Stdin)
    fmt.Println("Clave de sustitución: ", alfabetoClave)
    fmt.Print("Mensaje a cifrar: ")
    msg, _ := reader.ReadString('\n')
    resultado := sustitucion(strings.TrimSpace(msg), mapero)
    fmt.Printf("Resultado: %s\n", resultado)
}
```

6.7. Cifrado por transposición columnar

Este ejercicio se enfoca en la **transposición** [12], donde no se cambian las letras, sino su posición. Se inspira en la antigua escítala espartana. El algoritmo consiste en escribir el mensaje en una matriz de ancho fijo (determinado por la clave) y leer el mensaje columna por columna. En Go, se puede simular esta matriz utilizando lógica de índices sobre el slice de runas.

```
// transposicion.cpp
package main
import ("fmt";"math";"strings")

func cifrarTransposicion(mensaje string, ancho int) string {
    // Limpiamos espacios para el ejemplo clásico de Escítala
    mensaje = strings.ReplaceAll(mensaje, " ", "")
    runes := []rune(mensaje)
    n := len(runes)
    // Calculamos la altura necesaria (redondeo hacia arriba)
    filas := int(math.Ceil(float64(n) / float64(ancho)))
    var sb strings.Builder
    for col := 0; col < ancho; col++ {
        for fila := 0; fila < filas; fila++ {
            index := fila*ancho + col
            if index < n {
                sb.WriteRune(runes[index])
            }
        }
    }
    return sb.String()
}
func main() {
    msg := "ESTO ES UN EJEMPLO DE TRANSPOSICION"
    ancho := 4 // Clave
    fmt.Printf("Original: %s\n", msg)
    fmt.Printf("Ancho (Clave): %d\n", ancho)
    cifrado := cifrarTransposicion(msg, ancho)
    fmt.Printf("Cifrado:  %s\n", cifrado)
}
```

6.8. Análisis de frecuencias básico

Este ejercicio es una herramienta de criptoanálisis. Dado un texto cifrado (presumiblemente por sustitución), el programa debe contar la aparición de cada letra y mostrar el porcentaje de frecuencia de cada una. En el idioma español, las letras 'E' y 'A' son las más frecuentes. Este programa ayuda a identificar qué símbolo del texto cifrado corresponde probablemente a estas letras. Se utilizará un `map[rune]int` para el conteo eficiente.

```
//fichero frecuencia.go
package main
import ("fmt";"unicode")

func analizarFrecuencia(texto string) {
    conteo := make(map[rune]int)
    totalLetras := 0
    for _, r := range texto {
        if unicode.IsLetter(r) {
            r = unicode.ToUpper(r)
            conteo[r]++
            totalLetras++
        }
    }
    fmt.Printf("Análisis de '%s'...\n", texto)
    fmt.Printf("Total letras: %d\n\n", totalLetras)
    for letra := 'A'; letra <= 'Z'; letra++ {
        val := conteo[letra]
        porcentaje := (float64(val) / float64(totalLetras)) * 100
        if val > 0 {
            fmt.Printf("%c: %d (%.2f%%)\n", letra, val, porcentaje)
        }
    }
}
func main() {
    texto := "HOLA ESTO ES UNA PRUEBA DE FRECUENCIA EN GO LENGUAJE INTERESANTE"
    analizarFrecuencia(texto)
}
```

6.9. Generación de un Hash simple (Checksum)

La criptografía no sólo protege la confidencialidad, sino también la **integridad**. Este ejercicio propone implementar una función de hash sencilla. El programa debe leer una cadena y devolver un valor numérico único (hash) [12]. Si se cambia un solo carácter de la cadena, el valor del hash debe cambiar, demostrando cómo se detectan modificaciones en los datos.

```
// hash_simple.go
package main
import ("bufio";"fmt";"os";"strings")
// hashSimple implementa un checksum básico.
// Suma los valores de los bytes multiplicados por su posición.
func hashSimple(s string) uint64 {
    var hash uint64 = 0
    for i, r := range s {
        // Multiplicamos por (i+1) para que "AB" sea diferente de "BA"
        // Usamos bitwise XOR y desplazamiento para mezclar bits
        hash += uint64(r) * uint64(i+1)
        hash = (hash << 5) ^ hash
    }
    return hash
}
func main() {
    reader := bufio.NewReader(os.Stdin)
    fmt.Print("Introduce texto para hashear: ")
    texto1, _ := reader.ReadString('\n')
    texto1 = strings.TrimSpace(texto1)
    h1 := hashSimple(texto1)
    fmt.Printf("Hash 1: %d\n", h1)
    fmt.Print("Introduce texto ligeramente diferente: ")
    texto2, _ := reader.ReadString('\n')
    texto2 = strings.TrimSpace(texto2)
    h2 := hashSimple(texto2)
    fmt.Printf("Hash 2: %d\n", h2)
    if h1 != h2 {
        fmt.Println("¡Integridad verificada! Los hashes son distintos.")
    } else {
        fmt.Println("Colisión o textos idénticos.")
    }
}
```

6.10. Generador de contraseñas aleatorias seguras

Este ejercicio propone crear un programa que genere contraseñas seguras de una longitud dada por el usuario. La contraseña se generará seleccionando aleatoriamente elementos de un conjunto de caracteres (*charset*). Para asegurar la calidad criptográfica, se utilizará el paquete `crypto/rand` de Go, que accede al generador de números aleatorios seguro del sistema operativo (CSPRNG), en lugar del paquete `math/rand` que es determinista y predecible. Esto garantiza que las contraseñas generadas sean resistentes a la predicción.

```
package main
import ("crypto/rand";"fmt";"math/big")
const (
    letrasMay = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
    letrasMin = "abcdefghijklmnopqrstuvwxyz"
    numeros   = "0123456789"
    simbolos  = "!@#$%^&*()-_=+"
    todo      = letrasMay + letrasMin + numeros + simbolos
)
// generarPassword usa crypto/rand para seguridad criptográfica
func generarPassword(longitud int) (string, error) {
    bytes := make([]byte, longitud)
    max := big.NewInt(int64(len(todo)))
    for i := 0; i < longitud; i++ {
        num, err := rand.Int(rand.Reader, max)
        if err != nil {return "", err}
        bytes[i] = todo[num.Int64()]
    }
    return string(bytes), nil
}
func main() {
    longitud := 16
    password, err := generarPassword(longitud)
    if err != nil {
        fmt.Println("Error generando password:", err)
        return
    }
    fmt.Printf("Generador de Contraseñas Seguras (crypto/rand)\n")
    fmt.Printf("Longitud: %d\n", longitud)
    fmt.Printf("Password: %s\n", password)
}
```

Capítulo 7

Concurrencia y Patrones Cloud Native

Go se ha convertido en el lenguaje "de facto" de la computación en la nube. Herramientas como **Docker, Kubernetes, Terraform o Prometheus** están escritas en Go. ¿Por qué? La respuesta no es solo su velocidad, sino su capacidad para gestionar la red y la concurrencia de forma nativa. En el desarrollo Cloud Native moderno (microservicios, serverless, contenedores), las aplicaciones pasan la mayor parte del tiempo **"esperando"**: esperando una respuesta de una base de datos, esperando a una API de terceros o esperando una petición HTTP de un usuario. En lenguajes tradicionales, cada "espera" consume un hilo del sistema operativo (caro y pesado) [23] [18]. En Go, el runtime gestiona esto con **Goroutines**, permitiendo que un servidor modesto maneje decenas de miles de conexiones simultáneas sin despeinarse [3].

En este capítulo dejaremos atrás los scripts secuenciales para construir software que se comporta como un servicio real en la nube:

- **Escalabilidad (Goroutines):** Cómo atender a múltiples usuarios a la vez con hilos ligeros (apenas 2KB de RAM).
- **Orquestación (WaitGroups):** El patrón *Scatter-Gather*, vital para llamar a múltiples microservicios en paralelo y reunir sus respuestas.
- **Resiliencia (Context):** Cómo evitar que un servicio lento colapse toda tu plataforma mediante *Timeouts* y cancelación de peticiones.
- **Comunicación (Canales):** Cómo pasar datos entre procesos de forma segura sin usar bloqueos de memoria complejos.
- **Infraestructura (Graceful Shutdown):** Cómo apagar un servicio correctamente sin cortar conexiones activas, un requisito obligatorio para aplicaciones que viven en orquestadores como Kubernetes.

7.1. Hola Mundo Concurrente (Goroutines)

En este ejercicio se pretende mostrar cómo trabajar con las goroutines para lanzar 2 trabajos a la vez. Es el ejercicio conocido como hola mundo concurrente. La unidad básica de ejecución en Go es la Goroutine. A diferencia de un hilo de sistema operativo, una goroutine es gestionada por el propio Go. Cuando lanzamos una función con la palabra clave `go`, el programa no espera a que termine; sigue ejecutando la siguiente línea inmediatamente [3]. Esto es la base de los servidores web: el `main` escucha en un puerto, y cada petición entrante se lanza en su propia goroutine.

```
// hola_goroutine.go
package main
import (
    "fmt"
    "time"
)
func imprimir(mensaje string) {
    for i := 1; i <= 5; i++ {
        time.Sleep(100 * time.Millisecond) // Simula trabajo
        fmt.Printf("%s: %d\n", mensaje, i)
    }
}
func main() {
    // "go" lanza la función en un hilo aparte
    go imprimir("Goroutine")
    // Ejecutamos otra cosa en el hilo principal
    imprimir("Main")
    time.Sleep(1 * time.Second)
    fmt.Println("Fin del programa")
}
```

7.2. Orquestación de tareas con WaitGroup

En este ejercicio se pretende asegurar la sincronización de trabajo entre 3 gourutines distintas haciendo que esperen los workers para sincronizarse. En la nube, a menudo necesitamos realizar varias tareas en paralelo (ej. pedir datos al servicio de Usuarios, al de Pagos y al de Inventario simultáneamente) y esperar a que todas terminen antes de enviar una respuesta al cliente. Dormir el programa (`Sleep`) es inaceptable. La herramienta correcta es `sync.WaitGroup`, que actúa como un contador de tareas pendientes [**?**].

```
//fichero waitgroup.go
package main

import (
    "fmt"
    "sync"
    "time"
)
// worker simula una tarea que tarda un tiempo
func worker(id int, wg *sync.WaitGroup) {
    // defer asegura que Done() se ejecute al salir de la función
    defer wg.Done()

    fmt.Printf("Worker %d iniciando...\n", id)
    time.Sleep(time.Second) // Simular trabajo pesado
    fmt.Printf("Worker %d completado.\n", id)
}

func main() {
    var wg sync.WaitGroup

    // Vamos a lanzar 3 goroutines
    for i := 1; i <= 3; i++ {
        wg.Add(1) // Incrementamos el contador
        go worker(i, &wg)
    }
    fmt.Println("Esperando a que terminen los workers...")
    // Wait() bloquea hasta que el contador del WaitGroup llegue a 0
    wg.Wait()
    fmt.Println("Todos los procesos han terminado.")
}
```

7.3. Comunicación mediante Canales (Channels)

En este ejercicio, presentamos el trabajo con canales en go. Si las Goroutines son los trabajadores, los **Canales** son las tuberías por las que se pasan el trabajo. La filosofía de Go es: *"No comuniques compartiendo memoria; comparte memoria comunicando"*. Esto evita los peligrosos bloqueos (locks) [23] de los lenguajes clásicos y permite crear arquitecturas de flujo de datos (Stream Processing) muy robustas.

```
//fichero canales_basico.go
package main

import "fmt"

func main() {
    // Creamos un canal de strings
    mensajes := make(chan string)

    // Goroutine anónima que envía datos
    go func() {
        mensajes <- "Ping" // Enviar al canal (bloqueante si no hay quien
    reciba)
        mensajes <- "Pong"
        close(mensajes)    // Cerramos el canal para indicar que no hay más
    datos
    }()

    // Recibir datos del canal
    // El bucle 'range' itera hasta que el canal se cierra
    for msg := range mensajes {
        fmt.Println("Recibido:", msg)
    }
}
```

7.4. Control de Timeouts con Select

En este ejercicio vamos a revisar cómo es el control de Timouts en Go. En sistemas distribuidos, **la red no es fiable**. Un servicio externo puede quedarse colgado y nunca responder. Si tu programa espera indefinidamente, se quedará bloqueado [18]. La sentencia `select` es fundamental en la nube: nos permite esperar por una respuesta **O** por un tiempo límite (Timeout), lo que ocurra primero.

```go
//fichero select_timeout.go
package main

import (
    "fmt"
    "math/rand"
    "time"
)

// simularServidor responde después de un tiempo aleatorio
func simularServidor(nombre string, ch chan string) {
    delay := time.Duration(rand.Intn(1000)) * time.Millisecond
    time.Sleep(delay)
    ch <- fmt.Sprintf("%s respondió en %v", nombre, delay)
}
func main() {
    c1 := make(chan string)
    c2 := make(chan string)

    go simularServidor("Servidor A", c1)
    go simularServidor("Servidor B", c2)
    // select espera al PRIMERO que esté listo
    select {
    case res := <-c1:
        fmt.Println("Ganó:", res)
    case res := <-c2:
        fmt.Println("Ganó:", res)
    case <-time.After(500 * time.Millisecond):
        // Si nadie responde en 500ms, se ejecuta este caso
        fmt.Println("Error: Timeout. Los servidores son demasiado lentos.")
    }
}
```

7.5. Seguridad en datos compartidos (Mutex)

En este ejercicio vamos a presentar cómo manejar la exclusión mutua [23] en el lenguaje go. A veces, necesitamos mantener un estado global en memoria, como un contador de visitas o una caché de datos en un microservicio. Si múltiples goroutines intentan escribir en la misma variable a la vez, ocurre una **Race Condition** (condición de carrera) [23] [18] y los datos se corrompen. `sync.Mutex` nos permite "echar el cerrojo" a una variable para garantizar la integridad de los datos en entornos concurrentes.

```go
//fichero mutex_contador.go
package main
import ("fmt";"sync")
// SafeCounter es seguro para usar concurrentemente
type SafeCounter struct {
    mu sync.Mutex ;v  map[string]int
}
// Inc incrementa el contador para una clave dada
func (c *SafeCounter) Inc(key string) {
    // Bloqueamos el acceso para que solo UNA goroutine escriba a la vez
    c.mu.Lock()
    defer c.mu.Unlock()
    c.v[key]++
}
// Value devuelve el valor actual del contador
func (c *SafeCounter) Value(key string) int {
    c.mu.Lock()
    defer c.mu.Unlock()
    return c.v[key]
}
func main() {
    c := SafeCounter{v: make(map[string]int)}
    var wg sync.WaitGroup
    // 1000 goroutines intentando incrementar el mismo contador a la vez
    for i := 0; i < 1000; i++ {
        wg.Add(1)
        go func() {c.Inc("visitas");wg.Done()
        }()
    }
    wg.Wait()
    fmt.Println("Valor final (debería ser 1000):", c.Value("visitas"))
}
```

7.6. Patrón Worker Pool (Colas de trabajo)

Este ejercicio nos propone implementar una cola de trabajo utilizando el patrón productor-consumidor. En go esto se implementa mediante **Worker Pool**, el cual es una generalización del patrón de arquitectura productor-consumidor clásico [23]. Si recibes 10,000 peticiones por segundo, no puedes lanzar 10,000 procesos simultáneos a tu base de datos o la tumbarás. El patrón **Worker Pool** crea un número fijo de "trabajadores" (ej. 5 goroutines) que van consumiendo tareas de una cola (un canal). Esto permite procesar grandes volúmenes de datos controlando el uso de recursos (CPU/Memoria).

```
//fichero worker_pool.go
package main
import (
    "fmt"
    "time"
)
// worker es el "empleado" que procesa tareas
func worker(id int, jobs <-chan int, results chan<- int) {
    for j := range jobs {
        fmt.Printf("Worker %d comenzó tarea %d\n", id, j)
        time.Sleep(500 * time.Millisecond) // Simula trabajo costoso
        fmt.Printf("Worker %d terminó tarea %d\n", id, j)
        results <- j * 2
    }
}
func main() {
    const numJobs = 5
    jobs := make(chan int, numJobs)
    results := make(chan int, numJobs)
    // Lanzamos 3 workers (goroutines fijas)
    for w := 1; w <= 3; w++ {
        go worker(w, jobs, results)
    }
    // Enviamos 5 tareas
    for j := 1; j <= numJobs; j++ {jobs <- j}
    close(jobs) // Cerramos canal para que los workers sepan que no hay más
    trabajo
    // Recogemos resultados
    for a := 1; a <= numJobs; a++ {<-results}
    fmt.Println("Todas las tareas procesadas.")
}
```

7.7. El paquete Context: El estándar de la Nube

En este ejercicio se enseña un ejemplo de como trabajar con el paquete context en Go. El paquete `context` es quizás la librería más importante para el desarrollo backend en Go. Permite transportar señales de cancelación, plazos (deadlines) y valores de autenticación a través de las llamadas de API y goroutines. Si un usuario cierra su navegador, el servidor debe enterarse y cancelar inmediatamente cualquier consulta a base de datos que estuviera haciendo para ese usuario. Eso es lo que hace `context`.

```
//fichero contexto_cancel.go
package main

import (
    "context"
    "fmt"
    "time"
)
func operacionLenta(ctx context.Context) {
    // select permite escuchar al canal Done() del contexto
    select {
    case <-time.After(2 * time.Second):
        fmt.Println("Operación completada con éxito")
    case <-ctx.Done():
        // Si el contexto se cancela o expira, entramos aquí inmediatamente
        fmt.Println("Operación cancelada:", ctx.Err())
    }
}
func main() {
    // Creamos un contexto que expira automáticamente en 1 segundo
    // (Timeout)
    ctx, cancel := context.WithTimeout(context.Background(), 1*time.Second)

    // Es buena práctica llamar a cancel al salir, aunque haya expirado
    defer cancel()

    fmt.Println("Iniciando operación que tarda 2 segundos...")
    // Pasamos el contexto a la función
    operacionLenta(ctx)

    fmt.Println("Programa terminado")
}
```

7.8. Patrón Pipeline (Procesamiento de Streams)

Este ejercicio nos propone encadenar salidas-entradas de gourutines de manera secuencial. Inspirado en las tuberías de Unix (|), este patrón conecta goroutines de forma que la salida de una es la entrada de la siguiente. Es ideal para arquitecturas **ETL** (Extract, Transform, Load) y procesamiento de datos en tiempo real, permitiendo procesar archivos gigantescos sin tener que cargarlos enteros en la memoria RAM.

```
//fichero pipeline.go
package main

import "fmt"

// Genera números
func generar(nums ...int) <-chan int {
    out := make(chan int)
    go func() {
        for _, n := range nums {out <- n}
        close(out)
    }()
    return out
}
// Calcula el cuadrado
func cuadrado(in <-chan int) <-chan int {
    out := make(chan int)
    go func() {
        for n := range in {out <- n * n}
        close(out)
    }()
    return out
}
func main() {

    // entrada -> generar -> cuadrado -> salida
    c1 := generar(2, 3, 4, 5)
    c2 := cuadrado(c1)
    // Consumimos el resultado final
    for result := range c2 {
        fmt.Println(result)
    }
}
```

7.9. API REST con JSON (Microservicio Básico)

En este ejercicio se propone crear un microservicio como API que reciba y devuelva en formato JSON, la base de la comunicación moderna entre servicios. Go tiene un servidor web de producción integrado en su librería estándar (`net/http`). No hace falta Apache, Nginx ni frameworks pesados para servir tráfico.

```
//fichero servidor_json.go
package main
import (
    "encoding/json"
    "fmt"
    "net/http"
    "time"
)
// Estructura para la respuesta JSON
type Respuesta struct {
    Mensaje string    `json:"mensaje"`
    Hora    time.Time `json:"hora"`
    Status  int       `json:"status"`
}
func handler(w http.ResponseWriter, r *http.Request) {
    fmt.Println("Petición recibida")
    // Preparamos datos
    data := Respuesta{
        Mensaje: "Hola desde Go!",
        Hora:    time.Now(),
        Status:  200,
    }
    // Configuramos cabeceras
    w.Header().Set("Content-Type", "application/json")
    w.WriteHeader(http.StatusOK)
    // Codificamos struct a JSON y enviamos
    json.NewEncoder(w).Encode(data)
}
func main() {
    http.HandleFunc("/api/hola", handler)
    fmt.Println("Servidor escuchando en http://localhost:8880/api/hola")
    // Bloquea indefinidamente
    http.ListenAndServe(":8880", nil)
}
```

7.10. Graceful Shutdown (Apagado controlado en Kubernetes)

En este ejercicio se propone realizar un apagado controlado en Kubernetes. En la nube, los servidores son efímeros. Kubernetes puede decidir matar tu pod en cualquier momento para moverlo a otro nodo. Una aplicación "Cloud Native" debe escuchar las señales del sistema operativo (SIGTERM), dejar de aceptar nuevas peticiones y terminar las que están en curso antes de morir. Si cortas una conexión a mitad de una transacción de pago, pierdes datos (y dinero). Esto se llama **Graceful Shutdown**.

```
//fichero graceful_shutdown.go
package main
import ("context";"fmt";"net/http";"os";"os/signal";"time")
func main() {
    srv := &http.Server{Addr: ":8880"}
    http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
        time.Sleep(2 * time.Second) // Simula una petición lenta
        fmt.Fprintln(w, "Hola Mundo")
    })
    // Lanzamos el servidor en una goroutine para que no bloquee
    go func() {
        fmt.Println("Servidor iniciado en puerto 8880 (Ctrl+C para parar)")
        if err := srv.ListenAndServe(); err != nil && err != http.
    ErrServerClosed {
            fmt.Printf("Error: %v\n", err)
        }
    }()
    // Canal para escuchar señales del sistema operativo
    quit := make(chan os.Signal, 1)
    // Notificarnos si llega SIGINT (Ctrl+C) o SIGTERM (Kubernetes stop)
    signal.Notify(quit, os.Interrupt)
    // Bloqueamos aquí hasta recibir la señal
    <-quit
    fmt.Println("\nApagando servidor...")
    // Damos 5 segundos de margen para terminar peticiones vivas
    ctx, cancel := context.WithTimeout(context.Background(), 5*time.Second)
    defer cancel()
    if err := srv.Shutdown(ctx); err != nil {
        fmt.Println("Forzando cierre:", err)
    }
    fmt.Println("Servidor apagado correctamente.")
}
```

Capítulo 8

Herramientas de Sistema y Automatización (CLI)

Una de las principales razones por las que Go ha desplazado a Python y Bash en el mundo de la infraestructura (DevOps) es su capacidad para crear herramientas de línea de comandos (CLI) robustas, rápidas y que se compilan en un solo binario estático sin dependencias.

Como ingenieros, a menudo necesitamos automatizar tareas repetitivas: procesar logs, realizar copias de seguridad, limpiar directorios temporales o interactuar con el sistema operativo. En este capítulo aprenderemos a interactuar con el entorno mediante los paquetes `os`, `flag` y `path/filepath`.

Aprenderemos a:

- Leer argumentos y "banderas"desde la terminal (ej: `-verbose`, `-output`).
- Manipular el sistema de ficheros (crear, borrar, renombrar masivamente).
- Ejecutar comandos externos del sistema operativo.
- Procesar ficheros de texto estructurados (CSV) y comprimir datos.

8.1. Argumentos de línea de comandos (Flags)

En este ejercicio se pide implementar, mediante dicho paquete, entradas de parámetros que pueden cambiar en cada ejecución del mismo. Asimismo, se definirá unos valores por defecto para el caso que no se decida meter ningun parámetro de entrada. Para que un programa sea útil, debe ser configurable sin necesidad de recompilarlo. El paquete `flag` nos permite definir parámetros de entrada estándar UNIX.

```
//fichero hola_flags.go
package main

import (
    "flag"
    "fmt"
)

func main() {
    // Definimos las banderas (nombre, valor_defecto, descripcion)
    nombre := flag.String("nombre", "Mundo", "Nombre a saludar")
    edad := flag.Int("edad", 0, "Edad del usuario")
    esHacker := flag.Bool("hacker", false, "Modo hacker activado")

    // Importante: Parse procesa los argumentos de la terminal
    flag.Parse()

    fmt.Printf("Hola, %s.\n", *nombre) // nombre es un puntero, usamos *

    if *edad > 0 {
        fmt.Printf("Tienes %d años.\n", *edad)
    }

    if *esHacker {
        fmt.Println("ACCESO AL SISTEMA CONCEDIDO... 1010101")
    }
}
// Ejecutar: go run hola_flags.go -nombre="Fran" -edad=30 -hacker
```

8.2. Variables de Entorno

En este ejercicio se va a implementar peticiones para permitir acceder a las variables de entorno del sistema operativo (en este ejemplo Linux) y poder establecer nuevos valores. En entornos de nube y servidores, las configuraciones sensibles (contraseñas, puertos) no se pasan por argumentos, sino mediante variables de entorno. Go accede a ellas nativamente.

```
//fichero variables_entorno.go
package main

import (
    "fmt"
    "os"
)

func main() {
    // Obtener una variable de entorno específica
    usuario := os.Getenv("USER") // En Windows suele ser USERNAME
    if usuario == "" {
        usuario = os.Getenv("USERNAME")
    }

    path := os.Getenv("PATH")

    fmt.Printf("Usuario del sistema: %s\n", usuario)
    fmt.Printf("Primeras rutas del PATH: %.50s...\n", path)

    // Establecer una variable temporalmente para este proceso
    os.Setenv("MI_APP_PORT", "8080")
    fmt.Println("Puerto configurado:", os.Getenv("MI_APP_PORT"))
}
```

8.3. Lectura de Ficheros (Clon de 'cat')

Este ejercicio nos propone recrear la herramienta clásica de Unix `cat`. Leeremos un fichero completo y lo volcaremos en la salida estándar, gestionando los errores de ficheros inexistentes.

```
//fichero mi_cat.go
package main

import (
    "fmt"
    "io"
    "os"
)

func main() {
    // os.Args[0] es el nombre del programa, Args[1] el primer argumento
    if len(os.Args) < 2 {
        fmt.Println("Uso: go run mi_cat.go <fichero>")
        os.Exit(1)
    }

    filename := os.Args[1]

    // Abrir el fichero
    file, err := os.Open(filename)
    if err != nil {
        fmt.Printf("Error al abrir el fichero: %v\n", err)
        os.Exit(1)
    }
    defer file.Close()

    // Copiar el contenido del fichero a la salida estándar (Terminal)
    // io.Copy es muy eficiente, no carga todo en memoria RAM a la vez
    _, err = io.Copy(os.Stdout, file)
    if err != nil {
        fmt.Println("Error leyendo fichero:", err)
    }
}
```

8.4. Contador de Palabras (Clon de 'wc')

En este ejercicio proponemos realizar un contador de palabras al estilo de la herramienta wc de Linux. Utilizando `bufio.Scanner`, podemos procesar ficheros línea a línea o palabra a palabra de forma muy eficiente, ideal para ficheros de logs gigantescos.

```
//fichero mi_wc.go
package main

import (
    "bufio"
    "fmt"
    "os"
    "strings"
)
func main() {
    if len(os.Args) < 2 {
        fmt.Println("Por favor, indica un fichero.")
        return
    }
    file, err := os.Open(os.Args[1])
    if err != nil {
        panic(err)
    }
    defer file.Close()

    scanner := bufio.NewScanner(file)
    lineas := 0
    palabras := 0
    caracteres := 0
    for scanner.Scan() {
        linea := scanner.Text()
        lineas++
        caracteres += len(linea)
        // Fields divide por espacios en blanco
        palabras += len(strings.Fields(linea))
    }

    fmt.Printf("Líneas: %d\nPalabras: %d\nCaracteres (aprox): %d\n", lineas,
    palabras, caracteres)
}
```

8.5. Recorrer directorios recursivamente

En este ejercicio mostramos cómo realizar una búsqueda de ficheros específicos en el árbol de directores. Para ello, la función `filepath.WalkDir` hará todo el trabajo sucio de la recursividad por nosotros.

```
//fichero buscar_archivos.go
package main

import (
    "fmt"
    "io/fs"
    "os"
    "path/filepath"
)

func main() {
    root := "." // Directorio actual
    if len(os.Args) > 1 {
        root = os.Args[1]
    }

    fmt.Printf("Buscando ficheros .go en '%s'...\n", root)
    // WalkDir recorre recursivamente el árbol
    err := filepath.WalkDir(root, func(path string, d fs.DirEntry, err error)
    error {
        if err != nil {
            return err
        }
        // Si no es directorio y acaba en .go
        if !d.IsDir() && filepath.Ext(path) == ".go" {
            info, _ := d.Info()
            fmt.Printf("[GO] %s (%d bytes)\n", path, info.Size())
        }
        return nil
    })
    if err != nil {
        fmt.Println("Error:", err)
    }
}
```

8.6. Ejecutar comandos del sistema

En el siguiente ejercicio vamos a realizar el lanzamiento de un comando del sistema para realizar ping sobre google.com y ver qué nos devuelve. A veces necesitamos invocar otros programas (como `ping`, `git` o `ffmpeg`) desde nuestro código. El paquete `os/exec` permite lanzar subprocesos y capturar su salida.

```go
//fichero ejecutar_cmd.go
package main

import (
    "fmt"
    "os/exec"
)

func main() {

    // Vamos a ejecutar un comando 'ping' (o 'ls' / 'dir')
    // Nota: En Windows usar "ping", "-n", "3", "google.com"

    cmd := exec.Command("ping", "-c", "3", "google.com")

    fmt.Println("Ejecutando comando externo...")

    // Output ejecuta el comando y devuelve la salida estándar capturada
    salida, err := cmd.Output()

    if err != nil {
        fmt.Println("El comando falló:", err)
        return
    }

    fmt.Println("Resultado:")
    fmt.Println(string(salida))
}
```

8.7. Procesamiento de datos CSV

El siguiente ejercicio que nos presenta cómo crear un fichero csv y posteriormente leer los datos en él almacenados. El formato CSV es el estándar de facto para intercambio de datos en ingeniería. Go tiene un parser robusto que maneja comillas y casos especiales automáticamente.

```
//fichero procesar_csv.go
package main
import ("encoding/csv";"fmt";"os";"strconv")

func main() {
    // Creamos un CSV de ejemplo en memoria
    archivoNombre := "datos.csv"
    f, _ := os.Create(archivoNombre)
    f.WriteString("Nombre,Edad,Nota\nAna,22,8.5\nLuis,24,9.0\nPepe,21,4.5")
    f.Close()
    // Abrimos para leer
    file, _ := os.Open(archivoNombre)
    defer file.Close()
    lector := csv.NewReader(file)
    registros, _ := lector.ReadAll() // Lee todo a memoria (slice de slices)

    var sumaNotas float64;var count int

    fmt.Println("Alumnos aprobados:")
    // Saltamos la cabecera (i=1)
    for i := 1; i < len(registros); i++ {
        fila := registros[i] // [Nombre, Edad, Nota]
        nota, _ := strconv.ParseFloat(fila[2], 64)

        if nota >= 5.0 {
            fmt.Printf("- %s (%.1f)\n", fila[0], nota)
        }
        sumaNotas += nota
        count++
    }
    fmt.Printf("\nPromedio de la clase: %.2f\n", sumaNotas/float64(count))
    // Limpieza
    os.Remove(archivoNombre)
}
```

8.8. Renombrado masivo de ficheros

Un ejercicio práctico de mantenimiento: crear una herramienta que añada un prefijo o cambie la extensión a cientos de ficheros de una sola vez.

```
//fichero renombrador.go
package main

import (
    "fmt"
    "os"
    "path/filepath"
    "strings"
)

func main() {
    // Simulación: Creamos ficheros de prueba
    os.Create("imagen1.txt")
    os.Create("imagen2.txt")

    patron := "*.txt"
    archivos, _ := filepath.Glob(patron)

    fmt.Println("Renombrando ficheros .txt a .bak...")

    for _, viejo := range archivos {
        // Cambiamos extensión
        nuevo := strings.TrimSuffix(viejo, ".txt") + ".bak"

        err := os.Rename(viejo, nuevo)
        if err == nil {
            fmt.Printf("%s -> %s\n", viejo, nuevo)
        }
    }

    // Limpieza para el ejercicio
    files, _ := filepath.Glob("*.bak")
    for _, f := range files { os.Remove(f) }
}
```

8.9. Backup con compresión de ficheros (ZIP)

En este ejercicio se propone crear una herramienta para realizar copias de seguridad comprimidas. Go incluye soporte para `archive/zip` en su librería estándar.

```go
//fichero compresor.go
package main
import (
    "archive/zip"
    "fmt"
    "io"
    "os"
)
func main() {
    archivos := []string{"hola_flags.go", "mi_cat.go"}
    zipName := "codigos.zip"
    nuevoZip, err := os.Create(zipName)
    if err != nil { panic(err) }
    defer nuevoZip.Close()
    writer := zip.NewWriter(nuevoZip)
    defer writer.Close()

    for _, archivo := range archivos {
        fmt.Printf("Comprimiendo %s...\n", archivo)

        // Abrir fichero original
        f, err := os.Open(archivo)
        if err != nil {
            fmt.Println("Saltando (no existe):", archivo)
            continue
        }
        defer f.Close()

        // Crear entrada dentro del zip
        w, _ := writer.Create(archivo)

        // Copiar contenido
        io.Copy(w, f)
    }
    fmt.Println("Backup creado:", zipName)
}
```

8.10. Monitor de espacio en disco

En este ejercicio pretendemos crear un pequeño monitor para visualizar el espacio que nos queda en el disco duro utilizando la herramienta comentada. Para ello, utilizaremos las funciones de información de ficheros (`FileInfo`), las cuales nos permiten analizar qué ficheros están consumiendo más espacio en una carpeta dada.

```
//fichero analizar_disco.go
package main

import (
    "fmt"
    "os"
)

func main() {
    dir := "."
    entries, _ := os.ReadDir(dir)

    var total int64
    fmt.Printf("Analizando '%s'...\n", dir)

    for _, e := range entries {
        info, _ := e.Info()
        size := info.Size()
        total += size

        if size > 1000 { // Destacar ficheros > 1KB
            fmt.Printf("! GRANDE: %-20s %d bytes\n", e.Name(), size)
        } else {
            fmt.Printf("  Pequeño: %-20s %d bytes\n", e.Name(), size)
        }
    }
    fmt.Printf("\nTotal ocupado: %d bytes\n", total)
}
```

Capítulo 9

Procesamiento de Imágenes y Esteganografía

Go dispone de una excelente biblioteca estándar para el manejo de imágenes (`image`), que permite decodificar, manipular y codificar formatos como JPEG, PNG y GIF de forma nativa. Esto es especialmente útil en ingeniería para tareas de visión artificial básica, generación de gráficos dinámicos o preprocesamiento de datos visuales.

En este capítulo exploraremos cómo los ordenadores entienden las imágenes: como matrices de píxeles. Manipularemos estos píxeles matemáticamente para aplicar filtros, redimensionar fotos y finalmente realizaremos un ejercicio avanzado de **Esteganografía**: la técnica de ocultar información secreta dentro de una imagen aparentemente normal, modificando los bits menos significativos de sus píxeles.

Para profundizar más sobre teoría de procesamiento de información visual véase [5]

9.1. Generación de una imagen sintética

Antes de procesar fotos, vamos a crear una desde cero. En este ejercicio, se pretende generar un patrón visual (un degradado o tablero de ajedrez) definiendo el color de cada píxel individualmente y guardándolo como PNG.

```go
//fichero generar_patron.go
package main

import (
    "image"
    "image/color"
    "image/png"
    "os"
)

func main() {
    width, height := 256, 256
    // Creamos un lienzo RGBA en blanco
    img := image.NewRGBA(image.Rect(0, 0, width, height))

    for y := 0; y < height; y++ {
        for x := 0; x < width; x++ {
            // Generamos un degradado:
            // Rojo aumenta con X, Verde aumenta con Y, Azul fijo
            c := color.RGBA{
                R: uint8(x),
                G: uint8(y),
                B: 100,
                A: 255, // Alfa opaco
            }
            img.Set(x, y, c)
        }
    }

    f, _ := os.Create("patron.png")
    defer f.Close()
    png.Encode(f, img)
}
```

9.2. Lectura de metadatos y configuración

En este ejercicio vamos a analizar los datos de la imagen creada anteriormente. A menudo necesitamos saber las dimensiones o el formato de una imagen sin cargar todos sus megabytes en memoria. La función `image.DecodeConfig` es vital para esto.

```
//fichero leer_info.go
package main

import (
    "fmt"
    "image"
    _ "image/jpeg" // Importar codecs para que Go sepa leer JPG
    _ "image/png"  // Importar codecs para que Go sepa leer PNG
    "os"
)

func main() {
    // Asegúrate de tener una imagen 'input.jpg' o usa 'patron.png' del ej
    anterior
    f, err := os.Open("patron.png")
    if err != nil {
        fmt.Println("No encuentro la imagen:", err)
        return
    }
    defer f.Close()

    // DecodeConfig lee solo la cabecera (muy rápido)
    config, format, err := image.DecodeConfig(f)
    if err != nil {
        fmt.Println("Error decodificando:", err)
        return
    }

    fmt.Printf("Formato detectado: %s\n", format)
    fmt.Printf("Dimensiones: %d x %d píxeles\n", config.Width, config.Height)
}
```

9.3. Conversión a escala de grises

En este ejercicio se pretende convertir una imagen a color en blanco y negro promediando los canales RGB o utilizando la fórmula de luminancia para la percepción humana. Esto es la base de la visión artificial, ya que para trabajar se suele utilizar imágenes en blanco y negro para casi todos los aspectos.

```
// fichero ejercicioscapitulo9/escala_grises.go
package main

import ("image";"image/jpeg";_ "image/png";"os")

func main() {
    // Abrimos el fichero original (que es un PNG)
    f, err := os.Open("patron.png")
    if err != nil {
        panic(err) // Si falla (ej. no existe el archivo), paramos
    }
    defer f.Close()
    // Al tener el import _ "image/png", el Decode sabe leer PNGs automá
    ticamente
    src, _, err := image.Decode(f)
    if err != nil {
        panic(err)
    }
    bounds := src.Bounds()
    // Creamos nueva imagen en escala de grises
    // NewGray crea un lienzo donde cada píxel es solo luz (0-255)
    grayImg := image.NewGray(bounds)
    for y := bounds.Min.Y; y < bounds.Max.Y; y++ {
        for x := bounds.Min.X; x < bounds.Max.X; x++ {
            // Leemos el color original
            originalColor := src.At(x, y)
            grayImg.Set(x, y, originalColor)
        }
    }
    out, _ := os.Create("salida_gris.jpg")
    defer out.Close()
    // Guardamos con calidad por defecto (nil)
    jpeg.Encode(out, grayImg, nil)
}
```

9.4. Inversión de colores (Negativo)

En este ejercicio vamos a realizar una manipulación básica de píxeles de inversión de colores de la imagen patrón que venimos utilizando. Para ello restaremos el valor de cada canal al máximo posible (255), obtenemos el negativo fotográfico de la imagen.

```
//fichero filtro_negativo.go
package main

import (
    "image"
    "image/color"
    "image/png"
    "os"
)
func main() {
    f, _ := os.Open("patron.png")
    src, _, _ := image.Decode(f)
    defer f.Close()

    bounds := src.Bounds()
    dst := image.NewRGBA(bounds)
    for y := bounds.Min.Y; y < bounds.Max.Y; y++ {
        for x := bounds.Min.X; x < bounds.Max.X; x++ {
            r, g, b, a := src.At(x, y).RGBA()
            // Nota: .RGBA() devuelve valores de 16-bits (0-65535)
            // Convertimos a 8-bits
            r8, g8, b8 := uint8(r>>8), uint8(g>>8), uint8(b>>8)
            // Invertimos el color (255 - valor)
            negColor := color.RGBA{
                R: 255 - r8,
                G: 255 - g8,
                B: 255 - b8,
                A: uint8(a >> 8),
            }
            dst.Set(x, y, negColor)
        }
    }
    out, _ := os.Create("salida_negativo.png")
    png.Encode(out, dst)
}
```

9.5. Recorte de imágenes (Cropping)

En este ejercicio, vamos a realizar un recorte sobre la imagen patrón de (50,50,150,150). Go utiliza una interfaz `SubImage` que permite definir una región rectangular de interés (ROI) sin necesidad de copiar memoria, simplemente redefiniendo los límites lógicos.

```
//fichero recortar.go
package main

import (
    "image"
    "image/png"
    "os"
)

func main() {
    f, _ := os.Open("patron.png")
    src, _, _ := image.Decode(f)
    defer f.Close()

    // Interfaz SubImage (si la imagen la soporta)
    type SubImager interface {
        SubImage(r image.Rectangle) image.Image
    }

    if img, ok := src.(SubImager); ok {
        // Definimos el rectángulo de recorte (x0, y0, x1, y1)
        rect := image.Rect(50, 50, 150, 150)
        crop := img.SubImage(rect)

        out, _ := os.Create("salida_crop.png")
        png.Encode(out, crop)
    }
}
```

9.6. Redimensionado simple (Downsampling)

En este ejercicio implementaremos un algoritmo básico de "vecino más cercano" o submuestreo para reducir el tamaño de una imagen [5], reduciendo así su peso en disco.

```
//fichero redimensionar.go
package main

import (
    "image"
    "image/png"
    "os"
)

// redimensionar reduce a la mitad (algoritmo simple: saltar píxeles)
func redimensionar(src image.Image) image.Image {
    bounds := src.Bounds()
    w, h := bounds.Dx(), bounds.Dy()

    newW, newH := w/2, h/2
    dst := image.NewRGBA(image.Rect(0, 0, newW, newH))

    for y := 0; y < newH; y++ {
        for x := 0; x < newW; x++ {
            // Tomamos el píxel de la coordenada doble (2x, 2y)
            dst.Set(x, y, src.At(x*2, y*2))
        }
    }
    return dst
}

func main() {
    f, _ := os.Open("patron.png")
    src, _, _ := image.Decode(f)
    f.Close()

    peque := redimensionar(src)

    out, _ := os.Create("salida_peque.png")
    png.Encode(out, peque)
}
```

9.7. Composición y Marca de Agua

En este ejercicio vamos a poner una marca de agua a nuestra imagen generada patron. Para ello, utilizaremos el paquete `image/draw`, el cual nos permite superponer una imagen sobre otra (compositing) con transparencia, ideal para añadir logos o marcas de agua automáticamente.

```
//fichero marga_agua.go
package main

import (
    "image"
    "image/color"
    "image/draw"
    "image/png"
    "os"
)
func main() {
    // Cargar imagen base
    f, _ := os.Open("patron.png")
    base, _, _ := image.Decode(f)
    f.Close()

    b := base.Bounds()
    // Necesitamos una imagen editable (DrawImage)
    lienzo := image.NewRGBA(b)
    // Copiamos la base al lienzo
    draw.Draw(lienzo, b, base, image.Point{}, draw.Src)
    // Crear la marca de agua (un cuadrado rojo semitransparente)
    marca := image.NewRGBA(image.Rect(0, 0, 50, 50))
    rojoTransparente := color.RGBA{255, 0, 0, 128} // Alpha 128 (50%)
    draw.Draw(marca, marca.Bounds(), &image.Uniform{rojoTransparente}, image.
    Point{}, draw.Src)

    // Dibujar marca sobre el lienzo (offset 20,20)
    offset := image.Pt(20, 20)
    destRect := image.Rect(20, 20, 70, 70) // Dónde va a ir
    // draw.Over superpone mezclando el canal Alpha
    draw.Draw(lienzo, destRect, marca, image.Point{}, draw.Over)
    out, _ := os.Create("salida_watermark.png")
    png.Encode(out, lienzo)
}
```

9.8. Análisis de Histograma

En este ejercicio presentamos una análisis de datos del histográma. Para ello, recorreremos la imagen para contar la frecuencia de cada nivel de intensidad. Esto nos dice si una foto está "quemada"(muy clara) o subexpuesta (muy oscura).

```go
//fichero histograma.go
package main

import ("fmt";"image";_ "image/png";"os")

func main() {
    f, _ := os.Open("patron.png")
    img, _, _ := image.Decode(f)
    f.Close()

    // Histograma de Luminosidad (0-255)
    var histograma [256]int
    bounds := img.Bounds()

    for y := bounds.Min.Y; y < bounds.Max.Y; y++ {
        for x := bounds.Min.X; x < bounds.Max.X; x++ {
            r, g, b, _ := img.At(x, y).RGBA()
            // Convertir a 8 bits y calcular luminosidad simple
            val := (uint8(r>>8) + uint8(g>>8) + uint8(b>>8)) / 3
            histograma[val]++
        }
    }

    fmt.Println("Distribución de luminosidad (resumen):")
    fmt.Printf("Oscuros (0-50): %d px\n", sum(histograma[0:50]))
    fmt.Printf("Medios (51-200): %d px\n", sum(histograma[51:200]))
    fmt.Printf("Claros (201-255): %d px\n", sum(histograma[201:255]))
}

func sum(arr []int) int {
    s := 0
    for _, v := range arr { s += v }
    return s
}
```

9.9. Esteganografía: Ocultar texto

En este ejercicio, vamos a esconder un mensaje de texto dentro de una imagen. Usaremos la técnica LSB (Least Significant Bit), modificando el último bit del canal azul de cada píxel, algo imperceptible al ojo humano.

```
//fichero esteganografia_ocultar.go
package main
import ("fmt";"image";"image/color";"image/png";"os")
func main() {
    mensaje := "Este es un mensaje secreto top secret!!"
    // Añadimos un caracter nulo al final para saber dónde parar de leer
    mensaje += string(rune(0))
    f, _ := os.Open("patron.png")
    img, _, _ := image.Decode(f)
    f.Close()
    bounds := img.Bounds();nuevo := image.NewRGBA(bounds)
    // Convertimos mensaje a bits
    bits := make([]int, 0)
    for _, char := range mensaje {
        for i := 0; i < 8; i++ {bit := (int(char) >> i) & 1;bits = append(bits,
     bit)}
    }
    bitIndex := 0
    for y := bounds.Min.Y; y < bounds.Max.Y; y++ {
        for x := bounds.Min.X; x < bounds.Max.X; x++ {
            r, g, b, a := img.At(x, y).RGBA()
            c := color.RGBA{uint8(r>>8), uint8(g>>8), uint8(b>>8), uint8(a>>8)}
            if bitIndex < len(bits) {
            // Ponemos a 0 el último bit de Azul y sumamos el bit del mensaje
            // (c.B & 0xFE) borra el último bit
                c.B = (c.B & 0xFE) | uint8(bits[bitIndex])
                bitIndex++
            }
            nuevo.Set(x, y, c)
        }
    }
    out, _ := os.Create("secreto.png")
    png.Encode(out, nuevo)
    fmt.Println("Mensaje oculto en secreto.png")
}
```

9.10. Esteganografía: Recuperar texto

En este ejercicio, se propone el proceso inverso al anterior. Leeremos la imagen "hackeada", extraeremos los bits menos significativos y reconstruiremos el mensaje secreto original.

```
//fichero esteganografia_leer.go
package main
import ("fmt";"image";_ "image/png";"os")
func main() {
    f, _ := os.Open("secreto.png")
    if f == nil {
        fmt.Println("Ejecuta primero el 9.9 para generar 'secreto.png'")
        return
    }
    img, _, _ := image.Decode(f)
    f.Close()
    bounds := img.Bounds();var bits []int
    for y := bounds.Min.Y; y < bounds.Max.Y; y++ {
        for x := bounds.Min.X; x < bounds.Max.X; x++ {
            _, _, b, _ := img.At(x, y).RGBA()
            b8 := uint8(b >> 8)
            // Extraemos último bit
            bits = append(bits, int(b8 & 1))
        }
    }
    var mensaje string
    for i := 0; i < len(bits)-8; i += 8 {
        var charVal int
        for bit := 0; bit < 8; bit++ {
            if bits[i+bit] == 1 {
                charVal |= (1 << bit)
            }
        }
        if charVal == 0 {
            break
        }
        mensaje += string(rune(charVal))
    }
    fmt.Println("Mensaje recuperado:")
    fmt.Println(mensaje)
}
```

Bibliografía

[1] The Go Authors. *The Go Programming Language Specification*.

[2] Jon Bodner. *Learning Go: An Idiomatic Approach to Real-World Go Programming*. O'Reilly Media, 2024.

[3] Alan A. A. Donovan and Brian W. Kernighan. *The Go Programming Language*. Addison-Wesley Professional, 2015.

[4] Merino L. Santos E. *Álgebra Lineal con métodos elementales*. Paraninfo, 1997.

[5] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Pearson, fourth edition, 2018.

[6] Google. Command go. `https://pkg.go.dev/cmd/go`, 2025. Herramienta oficial de compilación y gestión de paquetes.

[7] Robert Griesemer, Rob Pike, and Ken Thompson. The go programming language: History. `https://go.dev/doc/history`, 2009. Accessed: 2026-01-12.

[8] Teiva Harsanyi. *100 Go Mistakes and How to Avoid Them*. Manning Publications, 2022.

[9] Stewart James. *Cálculo - Trascendentes Tempranas*. CENGAGE, eight edition, 2018.

[10] JetBrains. Goland: Capable and ergonomic go ide. `https://www.jetbrains.com/go/`, 2025.

[11] Donald E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley Professional, second edition, 1998.

[12] Lucena López Manuel. *Criptografía y Seguridad en computadores*. Universidad de Jaén, fifth edition, 2023.

[13] Microsoft. *Visual Studio Code https://code.visualstudio.com/*. 2025.

[14] G. Brassard P. Bratley. *Fundamentos de algoritmia*. Prentice Hall., 1998.

[15] Orson Peters. Pattern-defeating quicksort. `https://github.com/orlp/pdqsort`, 2015.

[16] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., 2nd edition, 1996.

[17] Douglas R. Stinson and Maura B. Paterson. *Cryptography: Theory and Practice*. CRC Press, 4th edition, 2018.

[18] Andrew S. Tanenbaum and Herbert Bos. *Sistemas operativos modernos*. Pearson, 4 edition, 2016.

[19] The Go Team. Go 1 release notes. `https://go.dev/doc/go1`, 2012. Accessed: 2026-01-12.

[20] Kahan W. *Miscalculating Area and Angles of Needle-like Triangle*. Berkeley, https://people.eecs.berkeley.edu/ wkahan/triangle.pdf edition, 2014.

[21] Mark Allen Weiss. *Data Structures and Algorithm Analysis in C*. Person, second edition, 2003.

[22] Alan A. A. Donovan y Brian W. Kernighan. *The Go Programming Language*. Addison-Wesley Professional, 2015.

[23] Palma Méndez José Tomás y otros. *Programación Concurrente*. Thomson, first edition, 2006.