Laboratorio 2

Ana María Vega Angarita

Juan Diego Calderón Bermeo

Docente

Ronal Davilmar Montoya Montoya

Universidad de Antioquia

Facultad de ingeniería

Ingeniería de sistemas

Medellín, Colombia

2025

Introducción

El presente informe documenta el desarrollo de un programa implementado en los lenguajes Go y C++, cuyo propósito es manipular cadenas de texto utilizando punteros. Esta actividad busca reforzar la comprensión de la manipulación de memoria mediante referencias directas (punteros).

Ambos programas realizan operaciones similares sobre una cadena ingresada por el usuario, en primer lugar se debe invertir la cadena, luego el conteo de vocales y consonantes, y por último la sustitución de espacios por guiones bajos (_). Además, se incluye un filtro previo que elimina caracteres no alfabéticos.

Objetivos

- Implementar un programa en C++ y Go que manipule cadenas mediante el uso de punteros.
- Comparar el uso de punteros y manejo de memoria en C++ y Go.
- Aplicar validación previa para eliminar caracteres no alfabéticos.

Código Go

Elimina todos los caracteres no alfabéticos ni espacios de una cadena.

```go
package main

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

func limpiarCadena(s string) string {
    var limpia []rune
    for _, r := range s {
        if unicode.IsLetter(r) || r == ' ' {
            limpia = append(limpia, r)
        }
    }
    return string(limpia)
}
```

Invierte los caracteres de una cadena usando punteros a slices.

```go
func invertirCadena(s *[]rune) {
    i, j := 0, len(*s)-1
    for i < j {
        (*s)[i], (*s)[j] = (*s)[j], (*s)[i]
        i++
        j--
    }
}
```

Cuenta vocales, su frecuencia individual y consonantes en la cadena.

```
func contarVocalesYConsonantes(s []rune) (int, int, map[rune]int) {
    vocales := 0
    consonantes := 0
    vocalCount := map[rune]int{'a': 0, 'e': 0, 'i': 0, 'o': 0, 'u': 0}

    for _, r := range s {
        l := unicode.ToLower(r)
        if strings.ContainsRune("aeiou", l) {
            vocales++
            vocalCount[l]++
        } else if unicode.IsLetter(l) {
            consonantes++
        }
    }
    return vocales, consonantes, vocalCount
}
```

Sustituye los espacios por guiones bajos directamente en el slice usando punteros.

```
func reemplazarEspacios(s *[]rune) {
    for i, r := range *s {
        if r == ' ' {
            (*s)[i] = '_'
        }
    }
}
```

Se controla el flujo del programa: lee la entrada, limpia la cadena, invierte el texto, cuenta letras y modifica los espacios.

```
func main() {
    reader := bufio.NewReader(os.Stdin)
    fmt.Print("Ingrese una cadena: ")
    input, _ := reader.ReadString('\n')
    input = strings.TrimSpace(input)

    // Limpiar caracteres no alfabéticos ni espacios
    limpia := limpiarCadena(input)
```

```
    runes := []rune(limpia)

    // Invertir usando puntero
    invertirCadena(&runes)
    fmt.Println("Cadena invertida:", string(runes))

    // Contar vocales y consonantes
    vocales, consonantes, cuenta := contarVocalesYConsonantes(runes)
    fmt.Printf("Número de vocales: %d\n", vocales)
    fmt.Printf("Número de \"a\": %d, \"e\": %d, \"i\": %d, \"o\": %d, \"u\": %d\n",
        cuenta['a'], cuenta['e'], cuenta['i'], cuenta['o'], cuenta['u'])
    fmt.Printf("Número de consonantes: %d\n", consonantes)

    // Reemplazar espacios usando puntero
    reemplazarEspacios(&runes)
    fmt.Println("Cadena modificada:", string(runes))
}
```

Código C++

Elimina todos los caracteres no alfabéticos ni espacios de una cadena.

```
#include <iostream>
#include <cstring>
#include <cctype>
using namespace std;

void limpiarCadena(char* str) {
    char* dst = str;
    while (*str) {
        if (isalpha(*str) || *str == ' ') {
            *dst++ = *str;
        }
        str++;
    }
    *dst = '\0';
}
```

Invierte los caracteres de una cadena usando punteros a slices.

```
void invertirCadena(char* str) {
    char* inicio = str;
    char* fin = str + strlen(str) - 1;
    while (inicio < fin) {
        char temp = *inicio;
        *inicio = *fin;
        *fin = temp;
        inicio++;
        fin--;
    }
}
```

Cuenta vocales, su frecuencia individual y consonantes en la cadena.

```
void contarVocalesYConsonantes(char* str, int& vocales, int& consonantes, int
vocalCount[5]) {
    vocales = consonantes = 0;
    for (char* p = str; *p; p++) {
        char c = tolower(*p);
        if (c == 'a') { vocales++; vocalCount[0]++; }
        else if (c == 'e') { vocales++; vocalCount[1]++; }
        else if (c == 'i') { vocales++; vocalCount[2]++; }
        else if (c == 'o') { vocales++; vocalCount[3]++; }
        else if (c == 'u') { vocales++; vocalCount[4]++; }
        else if (isalpha(c)) consonantes++;
    }
}
```

Sustituye los espacios por guiones bajos directamente en el slice usando punteros.

```
void reemplazarEspacios(char* str) {
    for (char* p = str; *p; p++) {
        if (*p == ' ') *p = '_';
    }
}
```

Se ejecuta el programa.

```
int main() {
    char cadena[101];

    cout << "Ingrese una cadena (máx 100 caracteres): ";
    cin.getline(cadena, 101);

    limpiarCadena(cadena);
    invertirCadena(cadena);

    int vocales, consonantes;
    int vocalCount[5] = {0}; // a, e, i, o, u

    contarVocalesYConsonantes(cadena, vocales, consonantes, vocalCount);

    cout << "Cadena invertida: " << cadena << endl;
    cout << "Número de vocales: " << vocales << endl;
    cout << "Número de \"a\": " << vocalCount[0]
         << ", \"e\": " << vocalCount[1]
         << ", \"i\": " << vocalCount[2]
         << ", \"o\": " << vocalCount[3]
         << ", \"u\": " << vocalCount[4] << endl;
    cout << "Número de consonantes: " << consonantes << endl;

    reemplazarEspacios(cadena);
    cout << "Cadena modificada: " << cadena << endl;

    return 0;
}
```

Conclusiones

- El uso de punteros en C++ ofrece un control detallado de la memoria y permite manipular cadenas de forma directa, pero requiere mayor cuidado para evitar errores como accesos indebidos o corrupción de datos.

- En Go, el uso de slices y referencias simplifica el trabajo con colecciones, pero el uso explícito de punteros ayuda a comprender la semántica de paso por referencia y a modificar estructuras complejas en memoria.

- La práctica demostró que operaciones clásicas como invertir cadenas, contar vocales y reemplazar caracteres pueden realizarse de manera eficiente en ambos lenguajes.

- Comparar ambos enfoques permite comprender cómo se gestionan las cadenas y la memoria en lenguajes con distintos niveles de abstracción.