# Informe de las prácticas de experimentación y aplicación de los aprendizajes

1. **Datos Informativos:**

|  |  |
| --- | --- |
| Facultad: | *CIENCIAS ADMINISTRATIVAS GESTIÓN EMPRESARIAL E INFORMÁTICA* |
| Carrera: | *SOFTWARE* |
| Asignatura: | *Algoritmos y lógica de programación* |
| Ciclo: | *Primero* |
| Docente: | *ING. MÓNICA BONILLA* |
| Título de la práctica: | *ARREGLOS UNIDIMENSIONALES Y BIDIMENSIONALES* |
| No. de práctica: | *No. 3* |
| Escenario o ambiente de aprendizaje de la practica | *DEV C++. - Entorno de desarrollo integrado para programar en lenguaje C/C++* |
| No. de horas: | ***10*** |
| Fecha: | ***28-06-2024*** |
| Estudiantes: | Ariel Alejandro Calderón  Jacson Antonio Narváez |
| Calificación |  |

1. **Introducción:**

En el ámbito de la programación y las ciencias de la computación, los arreglos son estructuras de datos fundamentales que permiten almacenar y manipular colecciones de elementos de un mismo tipo de manera eficiente. Los arreglos pueden ser de diversas dimensiones, siendo los más comunes los unidimensionales y bidimensionales. Comprender estos tipos de arreglos es crucial para cualquier programador, ya que son herramientas esenciales para la organización y procesamiento de datos.

1. **Objetivo de la práctica:**

Aplicar los arreglos unidimensionales y bidimensionales en lenguaje C++ en la solución de programas básicos.

1. **Descripción del desarrollo de la práctica:**

**4.1 Uso de Arreglos unidimensionales y bidimensionales en C**

**4.2 Arreglos Unidimensionales**

Un arreglo unidimensional, también conocido como vector, es una colección de elementos almacenados en una secuencia lineal. Cada elemento en el arreglo se puede acceder mediante un índice único, que generalmente comienza desde cero. Este tipo de arreglo es útil para almacenar listas de datos homogéneos, como números o cadenas de caracteres, y permite operaciones eficientes de acceso y modificación de elementos.

arreglo = [1, 2, 3, 4, 5]

**4.3 Arreglos Bidimensionales**

Un arreglo bidimensional, también conocido como matriz, es una colección de elementos organizados en filas y columnas. Este tipo de arreglo es esencial para representar tablas de datos y para trabajar con información que tiene una estructura más compleja, como imágenes o tablas de datos.

matriz = [[1, 2, 3],

[4, 5, 6],

[7, 8, 9]]

**4.4 Importancia y Aplicaciones**

La importancia de los arreglos unidimensionales y bidimensionales radica en su capacidad para almacenar grandes cantidades de datos de manera estructurada y permitir un acceso rápido y eficiente a estos datos. Se utilizan en una amplia variedad de aplicaciones, desde la resolución de problemas matemáticos y estadísticos hasta la manipulación de imágenes y la gestión de bases de datos.

**4.5.1 Problema**

Se desea desarrollar una estructura de datos que permita calcular el promedio de 4 calificaciones para un grupo de n estudiantes en una asignatura, utilizando matrices en la solución. La estructura debe ser capaz de visualizar los nombres de los estudiantes, sus calificaciones y los promedios obtenidos. Además, debe clasificar los promedios en dos categorías: mayor o igual a 7 (aprobado) y menor a 7 (reprobado o suspenso). Para los estudiantes reprobados, se debe imprimir un listado específico.

**4.5.2 Pseudocódigo**

Inicio

// Constantes y variables

const MAX\_ESTUDIANTES = 30

n = 0

// Controlador

Mientras n > MAX\_ESTUDIANTES O n < 1 Hacer

Escribir "Ingrese el número de estudiantes (máximo MAX\_ESTUDIANTES): "

Leer n

Fin Mientras

// Arreglos

Dimension nombres[MAX\_ESTUDIANTES][50]

Dimension calificaciones[MAX\_ESTUDIANTES][4]

Dimension promedios[MAX\_ESTUDIANTES]

// Ingresar datos

Para i = 0 Hasta n-1 Hacer

Escribir "Ingrese el NOMBRE del estudiante #" + (i+1) + ": "

Leer nombres[i]

Para j = 0 Hasta 3 Hacer

Repetir

Escribir "Ingrese la calificación #" + (j+1) + " de " + nombres[i] + " (entre 1 y 10): "

Leer calificaciones[i][j]

Si calificaciones[i][j] < 1 O calificaciones[i][j] > 10 Entonces

Escribir "Calificación inválida. Debe estar entre 1 y 10."

Fin Si

Hasta que calificaciones[i][j] >= 1 Y calificaciones[i][j] <= 10

Fin Para

Fin Para

// Promedios

Para i = 0 Hasta n-1 Hacer

suma = 0

Para j = 0 Hasta 3 Hacer

suma = suma + calificaciones[i][j]

Fin Para

promedios[i] = suma / 4

Fin Para

// Resultados

Escribir "\nResultados:"

Escribir "Nombre\t\tCalificaciones\t\t\tPromedio\tEstado"

Para i = 0 Hasta n-1 Hacer

Escribir nombres[i] + "\t\t"

Para j = 0 Hasta 3 Hacer

Escribir calificaciones[i][j] + " "

Fin Para

Escribir "\t" + promedios[i] + "\t\t"

Si promedios[i] >= 7 Entonces

Escribir "Aprobado"

Sino

Escribir "Reprobado"

Fin Si

Fin Para

// Lista de reprobados

Escribir "\nListado de estudiantes reprobados:"

Para i = 0 Hasta n-1 Hacer

Si promedios[i] < 7 Entonces

Escribir nombres[i]

Fin Si

Fin Para

Fin

**4.5.3 Diagrama de flujo**

**4.5.4 Codificación**

#include <stdio.h>

int main()

{

    const int MAX\_ESTUDIANTES = 30;

    int n = 0;

    while (n > MAX\_ESTUDIANTES || n < 1)

    {

        printf("Ingrese el número de estudiantes (máximo %d): ", MAX\_ESTUDIANTES);

        scanf("%d", &n);

    }

    char nombres[MAX\_ESTUDIANTES][50];

    float calificaciones[MAX\_ESTUDIANTES][4];

    float promedios[MAX\_ESTUDIANTES];

    // Ingresar datos

    for (int i = 0; i < n; i++)

    {

        printf("Ingrese el NOMBRE del estudiante #%d: ", i + 1);

        scanf("%s", nombres[i]);

        for (int j = 0; j < 4; j++)

        {

            do

            {

                printf("Ingrese la calificación #%d de %s (entre 1 y 10): ", j + 1, nombres[i]);

                scanf("%f", &calificaciones[i][j]);

                if (calificaciones[i][j] < 1 || calificaciones[i][j] > 10)

                {

                    printf("Calificación inválida. Debe estar entre 1 y 10.\n");

                }

            } while (calificaciones[i][j] < 1 || calificaciones[i][j] > 10);

        }

    }

    // Calcular promedios

    for (int i = 0; i < n; i++)

    {

        float suma = 0;

        for (int j = 0; j < 4; j++)

        {

            suma += calificaciones[i][j];

        }

        promedios[i] = suma / 4;

    }

    // Imprimir resultados

    printf("\nResultados:\n");

    printf("Nombre\t\tCalificaciones\t\t\tPromedio\tEstado\n");

    for (int i = 0; i < n; i++)

    {

        printf("%s\t\t", nombres[i]);

        for (int j = 0; j < 4; j++)

        {

            printf("%.2f ", calificaciones[i][j]);

        }

        printf("\t%.2f\t\t", promedios[i]);

        if (promedios[i] >= 7)

        {

            printf("Aprobado\n");

        }

        else

        {

            printf("Reprobado\n");

        }

    }

    // Imprimir lista de reprobados

    printf("\nListado de estudiantes reprobados:\n");

    for (int i = 0; i < n; i++)

    {

        if (promedios[i] < 7)

        {

            printf("%s\n", nombres[i]);

        }

    }

    return 0;

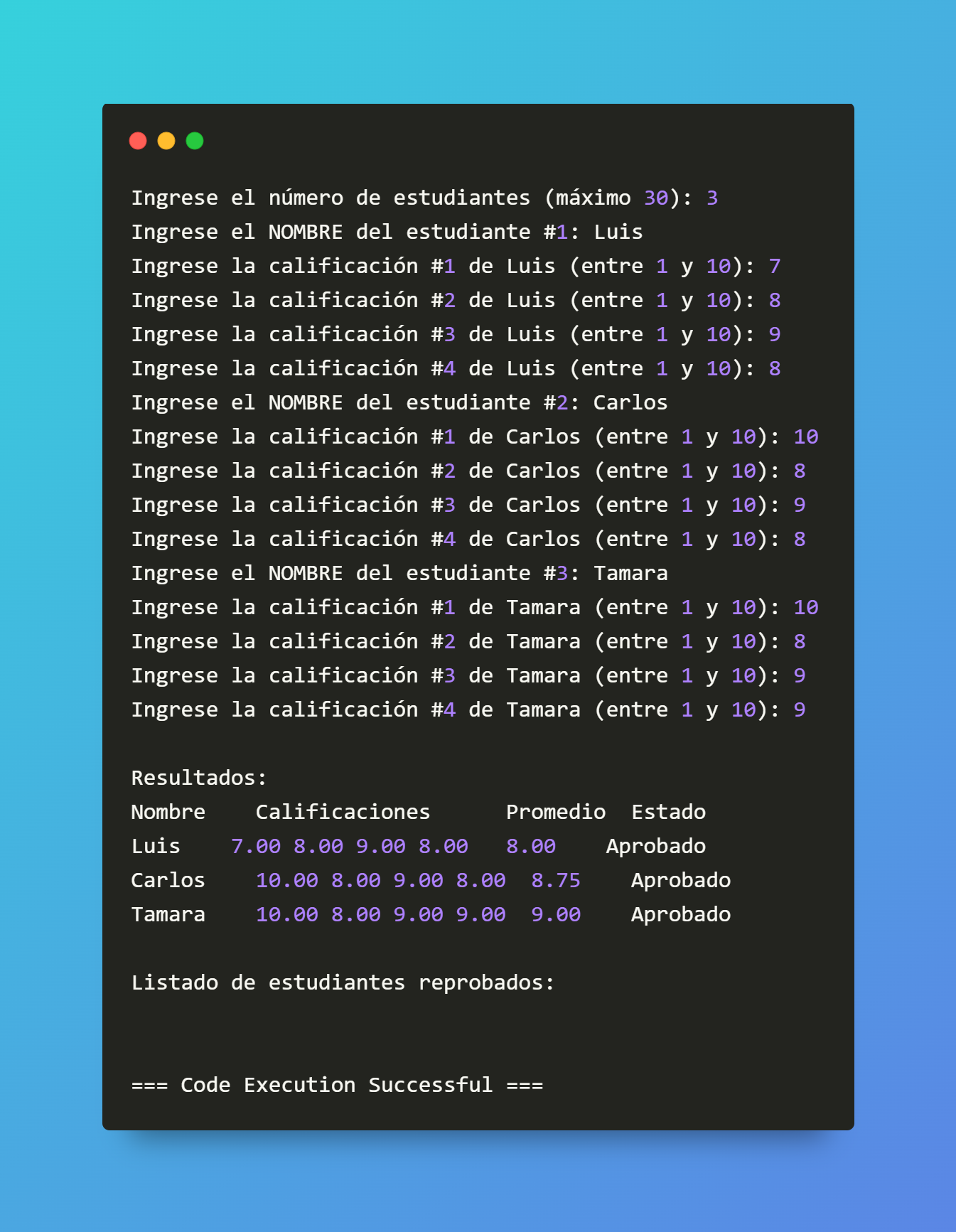
}

1. **Metodología:**

Experimentar con proyectos minimalistas donde se pueda poner en práctica el uso de arreglos unidimensionales y bidimensionales para optimizar el flujo de un programa.

1. **Resultados obtenidos:**

**Ejecución del programa en C**

****

1. **Conclusiones:**

Los arreglos unidimensionales y bidimensionales son componentes esenciales en la programación, proporcionando una base sólida para el manejo de datos y la implementación de algoritmos eficientes. Su comprensión y uso adecuado son habilidades indispensables para cualquier programador que busque desarrollar aplicaciones complejas y eficientes.

1. **Recomendaciones:**

* **Declaración y Inicialización**: Asegurarse de declarar el tamaño del arreglo correctamente. El tamaño debe ser conocido en tiempo de compilación para arreglos estáticos.
* **Índices válidos**: Los índices de los arreglos comienzan en 0 y van hasta n-1 donde n es el tamaño del arreglo. Asegurarse de no acceder a índices fuera de este rango, ya que esto puede causar comportamiento indefinido.
* **Manejo de Memoria**: Para arreglos cuyo tamaño no se conoce en tiempo de compilación, usar memoria dinámica con malloc y free.
* **Evitar Desbordamiento:** Siempre verificar que los índices estén dentro del rango permitido para evitar desbordamiento de búfer, lo que puede causar fallos en el programa o vulnerabilidades de seguridad.
* **Uso de Constantes y Macros**: Esto hace que el código sea más fácil de mantener y modificar.

1. **Bibliografía:**



1. **Anexos:**

