UACM

Universidad Autónoma
de la Ciudad de México

NADA HUMANO ME ES AJENO

Introducción a la Ingeniería de Software.

Profesor: Raúl Jara.

Práctica 4.

Velia Zavala Pérez.

Semestre 2025 – II.

Introducción

En esta práctica se desarrolla un programa en lenguaje C que utiliza arreglos dinámicos, memoria dinámica (malloc, calloc, realloc, free), así como structs, funciones, apuntadores, y manejo de buffers. El objetivo es gestionar información relacionada con:

- Alumnos
- Tareas
- Calificaciones

El proyecto debe:

- Permitir capturar, gestionar y mostrar todos los datos.
- Usar menús interactivos.
- Guardar datos mediante memoria dinámica.

El entregable incluye el análisis del problema, explicación del código, pruebas, defectos encontrados, tiempos de desarrollo y estructura del repositorio GitHub.

Desarrollo

Análisis del Problema

Se debe diseñar un sistema básico que permita:

- Registrar alumnos
- Registrar tareas
- Asignar calificaciones por alumno y tarea
- Mostrar toda la información captura.

Solución propuesta.

Código:

```
/* practica4.c

   Practica 4: Uso de Arreglos/Buffer, Funciones y apuntadores.

*/


#include <stdio.h>
```

```c
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define INIT_CAP 4

/* ---------- ESTRUCTURAS ---------- */
typedef struct {
    int id;
    char *title;   /* cadenas dinámicas */
    char *desc;
} Task;

typedef struct {
    int id;
    char *name;
} Student;

typedef struct {
    int student_id;
    int task_id;
    float grade; /* 0..100, -1 indica no asignada (no existe entrada) */
} Grade;

/* ---------- PROTOTIPOS ---------- */
/* util */
char *read_line_dynamic(void); /* usa malloc + realloc internamente */
void pause_anykey(void);
```

```
/* Tareas */
void fill_task(Task *t);
void add_task(Task **tasks, int *count, int *cap, int next_id);
void list_tasks(Task *tasks, int count);
int find_task(Task *tasks, int count, int id);

/* estudiante */
void fill_student(Student *s);
void add_student(Student **students, int *count, int *cap, int next_id);
void list_students(Student *students, int count);
int find_student(Student *students, int count, int id);

/* califi */
void add_or_update_grade(Grade **grades, int *gcount, int *gcap, int student_id, int task_id, float grade);
void assign_grade(Grade **grades, int *gcount, int *gcap, Student *students, int scount, Task *tasks, int tcount);
void list_grades(Grade *grades, int gcount, Student *students, int scount, Task *tasks, int tcount);
float get_grade_for(Grade *grades, int gcount, int student_id, int task_id);

/* limpia */
void free_task(Task *t);
void free_student(Student *s);
void free_all(Task *tasks, int tcount, Student *students, int scount, Grade *grades);

/* ---------- IMPLEMENTACION ---------- */
```

```
char *read_line_dynamic(void) {
    size_t cap = 64;
    size_t len = 0;
    char *buf = (char*) malloc(cap);
    if (!buf) return NULL;

    int c;
    while ((c = getchar()) != EOF && c != '\n') {
        buf[len++] = (char)c;
        if (len + 1 >= cap) {
            /* duplicar capacidad usando realloc */
            size_t newcap = cap * 2;
            char *tmp = (char*) realloc(buf, newcap);
            if (!tmp) { free(buf); return NULL; }
            buf = tmp;
            cap = newcap;
        }
    }
    buf[len] = '\0';
    /* Opcional: ajustar al tamaño exacto */
    char *trim = (char*) realloc(buf, len + 1);
    if (trim) buf = trim;
    return buf;
}

void pause_anykey(void) {
    printf("Presiona ENTER para continuar...");
    while (getchar() != '\n') ; /* consumir resto de la línea actual */
```

```
}

/* ---------- Tareas ---------- */
void fill_task(Task *t) {
    printf("Titulo: ");
    char *s = read_line_dynamic();
    if (!s) { puts("Error de memoria al leer titulo"); exit(1); }
    t->title = s;
    printf("Descripcion: ");
    s = read_line_dynamic();
    if (!s) { puts("Error de memoria al leer descripcion"); exit(1); }
    t->desc = s;
}

void add_task(Task **tasks, int *count, int *cap, int next_id) {
    if (*count >= *cap) {
        int newcap = (*cap) * 2;
        Task *tmp = (Task*) realloc(*tasks, sizeof(Task) * newcap);
        if (!tmp) {
            fprintf(stderr, "Error: memoria insuficiente al expandir tasks\n");
            return;
        }
        /* Inicializar la nueva porcion (evita punteros basura) */
        for (int i = *cap; i < newcap; ++i) {
            tmp[i].title = NULL;
            tmp[i].desc = NULL;
            tmp[i].id = 0;
        }
```

```
        *tasks = tmp;
        *cap = newcap;
    }
    Task *t = &((*tasks)[*count]);
    t->id = next_id;
    t->title = NULL;
    t->desc = NULL;
    fill_task(t);
    (*count)++;
    printf("Tarea agregada con id = %d\n", t->id);
}

void list_tasks(Task *tasks, int count) {
    if (count == 0) { printf("No hay tareas registradas.\n"); return; }
    printf("Tareas:\n");
    for (int i = 0; i < count; ++i) {
        printf("ID %d: %s\n  %s\n", tasks[i].id,
               tasks[i].title ? tasks[i].title : "(sin titulo)",
               tasks[i].desc ? tasks[i].desc : "");
    }
}

int find_task(Task *tasks, int count, int id) {
    for (int i = 0; i < count; ++i) if (tasks[i].id == id) return i;
    return -1;
}

/* ---------- estudiantes ---------- */
```

```
void fill_student(Student *s) {
    printf("Nombre del alumno: ");
    char *buf = read_line_dynamic();
    if (!buf) { puts("Error de memoria al leer nombre"); exit(1); }
    s->name = buf;
}

void add_student(Student **students, int *count, int *cap, int next_id) {
    if (*count >= *cap) {
        int newcap = (*cap) * 2;
        Student *tmp = (Student*) realloc(*students, sizeof(Student) * newcap);
        if (!tmp) {
            fprintf(stderr, "Error: memoria insuficiente al expandir students\n");
            return;
        }
        for (int i = *cap; i < newcap; ++i) {
            tmp[i].name = NULL;
            tmp[i].id = 0;
        }
        *students = tmp;
        *cap = newcap;
    }
    Student *st = &((*students)[*count]);
    st->id = next_id;
    st->name = NULL;
    fill_student(st);
    (*count)++;
    printf("Alumno agregado con id = %d\n", st->id);
```

```
}

void list_students(Student *students, int count) {
    if (count == 0) { printf("No hay alumnos registrados.\n"); return; }
    printf("Alumnos:\n");
    for (int i = 0; i < count; ++i) {
        printf("ID %d: %s\n", students[i].id,
               students[i].name ? students[i].name : "(sin nombre)");
    }
}

int find_student(Student *students, int count, int id) {
    for (int i = 0; i < count; ++i) if (students[i].id == id) return i;
    return -1;
}

/* ---------- Calificacion ---------- */
void add_or_update_grade(Grade **grades, int *gcount, int *gcap, int student_id, int
task_id, float grade) {
    /* Si ya existe la combinacion, actualizar */
    for (int i = 0; i < *gcount; ++i) {
        if ((*grades)[i].student_id == student_id && (*grades)[i].task_id == task_id) {
            (*grades)[i].grade = grade;
            printf("Calificación actualizada.\n");
            return;
        }
    }
    /* Si no existe, agregar */
    if (*gcount >= *gcap) {
```

```
        int newcap = (*gcap) * 2;
        Grade *tmp = (Grade*) realloc(*grades, sizeof(Grade) * newcap);
        if (!tmp) {
            fprintf(stderr, "Error: memoria insuficiente al expandir grades\n");
            return;
        }
        *grades = tmp;
        *gcap = newcap;
    }
    (*grades)[*gcount].student_id = student_id;
    (*grades)[*gcount].task_id = task_id;
    (*grades)[*gcount].grade = grade;
    (*gcount)++;
    printf("Calificación agregada.\n");
}

void assign_grade(Grade **grades, int *gcount, int *gcap, Student *students, int scount, Task *tasks, int tcount) {
    if (scount == 0 || tcount == 0) {
        printf("Debe existir al menos un alumno y una tarea para asignar calificaciones.\n");
        return;
    }
    char *line;
    int sid, tid;
    printf("Ingresa ID del alumno: ");
    line = read_line_dynamic(); if (!line) return;
    sid = atoi(line); free(line);
    int si = find_student(students, scount, sid);
```

```
    if (si < 0) { printf("Alumno con ID %d no encontrado.\n", sid); return; }

    printf("Ingresa ID de la tarea: ");
    line = read_line_dynamic(); if (!line) return;
    tid = atoi(line); free(line);
    int ti = find_task(tasks, tcount, tid);
    if (ti < 0) { printf("Tarea con ID %d no encontrada.\n", tid); return; }

    printf("Ingresa la calificacion (0.0 - 100.0): ");
    line = read_line_dynamic(); if (!line) return;
    float g = (float) atof(line); free(line);
    if (g < 0.0f || g > 100.0f) { printf("Rango incorrecto. Debe ser 0-100.\n"); return; }

    add_or_update_grade(grades, gcount, gcap, sid, tid, g);
}

void list_grades(Grade *grades, int gcount, Student *students, int scount, Task *tasks, int tcount) {
    if (gcount == 0) { printf("No hay calificaciones registradas.\n"); return; }
    printf("Calificaciones (lista):\n");
    for (int i = 0; i < gcount; ++i) {
        int si = find_student(students, scount, grades[i].student_id);
        int ti = find_task(tasks, tcount, grades[i].task_id);
        const char *sname = (si >= 0 && students[si].name) ? students[si].name : "Alumno eliminado/desconocido";
        const char *ttitle = (ti >= 0 && tasks[ti].title) ? tasks[ti].title : "Tarea eliminada/desconocida";
        printf("Alumno: %s (id=%d) | Tarea: %s (id=%d) | Nota: %.2f\n",
               sname, grades[i].student_id, ttitle, grades[i].task_id, grades[i].grade);
```

```
    }
}

float get_grade_for(Grade *grades, int gcount, int student_id, int task_id) {
    for (int i = 0; i < gcount; ++i)
        if (grades[i].student_id == student_id && grades[i].task_id == task_id)
            return grades[i].grade;
    return -1.0f; /* no asignada */
}

/* ---------- LIMPIEZA ---------- */
void free_task(Task *t) {
    if (!t) return;
    free(t->title);
    free(t->desc);
    t->title = t->desc = NULL;
}

void free_student(Student *s) {
    if (!s) return;
    free(s->name);
    s->name = NULL;
}

void free_all(Task *tasks, int tcount, Student *students, int scount, Grade *grades) {
    /* liberar cada cadena dinámica dentro de tasks y students */
    if (tasks) {
        for (int i = 0; i < tcount; ++i) free_task(&tasks[i]);
```

```
    }
    if (students) {
        for (int i = 0; i < scount; ++i) free_student(&students[i]);
    }
    /* liberar arreglos principales */
    free(tasks);
    free(students);
    free(grades);
}

/* ---------- MENU ---------- */
int main(void) {
    /* Inicializar arreglos con calloc/malloc para demostrar uso */
    Task *tasks = (Task*) calloc(INIT_CAP, sizeof(Task));   /* calloc: inicializa en 0 */
    Student *students = (Student*) malloc(sizeof(Student) * INIT_CAP); /* malloc: sin inicializar */
    Grade *grades = (Grade*) calloc(INIT_CAP, sizeof(Grade)); /* calloc */

    if (!tasks || !students || !grades) {
        fprintf(stderr, "Error: memoria insuficiente al inicializar.\n");
        free(tasks); free(students); free(grades);
        return 1;
    }

    /* Asegurarse de inicializar campos de cadenas a NULL para seguridad */
    for (int i = 0; i < INIT_CAP; ++i) {
        tasks[i].title = tasks[i].desc = NULL;
        tasks[i].id = 0;
        students[i].name = NULL;
```

```
    students[i].id = 0;
}

int tcount = 0, scount = 0, gcount = 0;
int tcap = INIT_CAP, scap = INIT_CAP, gcap = INIT_CAP;
int next_task_id = 1, next_student_id = 1;

char *option = NULL;
while (1) {
    printf("\n--- MENU PRINCIPAL ---\n");
    printf("1. Gestionar Tareas (agregar, listar)\n");
    printf("2. Gestionar Alumnos (agregar, listar)\n");
    printf("3. Gestionar Calificaciones (asignar, listar)\n");
    printf("4. Mostrar todos los datos (matriz alumno x tarea)\n");
    printf("5. Salir\n");
    printf("Elige una opcion: ");
    option = read_line_dynamic();
    if (!option) break;

    if (strcmp(option, "1") == 0) {
        free(option); option = NULL;
        printf("\n-- Gestion Tareas --\n");
        printf("a) Agregar tarea\nb) Listar tareas\nc) Volver\nOpcion: ");
        option = read_line_dynamic();
        if (!option) break;
        if (strcmp(option, "a") == 0) {
            add_task(&tasks, &tcount, &tcap, next_task_id++);
        } else if (strcmp(option, "b") == 0) {
```

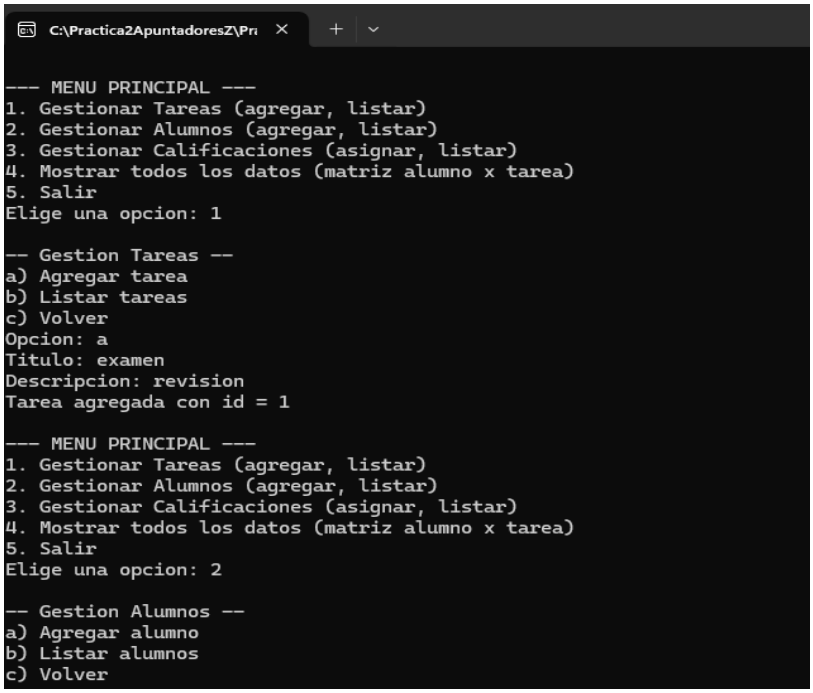
```
            list_tasks(tasks, tcount);
            pause_anykey();
        } else { /* volver */ }
    }
    else if (strcmp(option, "2") == 0) {
        free(option); option = NULL;
        printf("\n-- Gestion Alumnos --\n");
        printf("a) Agregar alumno\nb) Listar alumnos\nc) Volver\nOpcion: ");
        option = read_line_dynamic();
        if (!option) break;
        if (strcmp(option, "a") == 0) {
            add_student(&students, &scount, &scap, next_student_id++);
        } else if (strcmp(option, "b") == 0) {
            list_students(students, scount);
            pause_anykey();
        } else { /* volver */ }
    }
    else if (strcmp(option, "3") == 0) {
        free(option); option = NULL;
        printf("\n-- Gestion Calificaciones --\n");
        printf("a) Asignar calificacion\nb) Listar calificaciones\nc) Volver\nOpcion: ");
        option = read_line_dynamic();
        if (!option) break;
        if (strcmp(option, "a") == 0) {
            assign_grade(&grades, &gcount, &gcap, students, scount, tasks, tcount);
        } else if (strcmp(option, "b") == 0) {
            list_grades(grades, gcount, students, scount, tasks, tcount);
            pause_anykey();
```

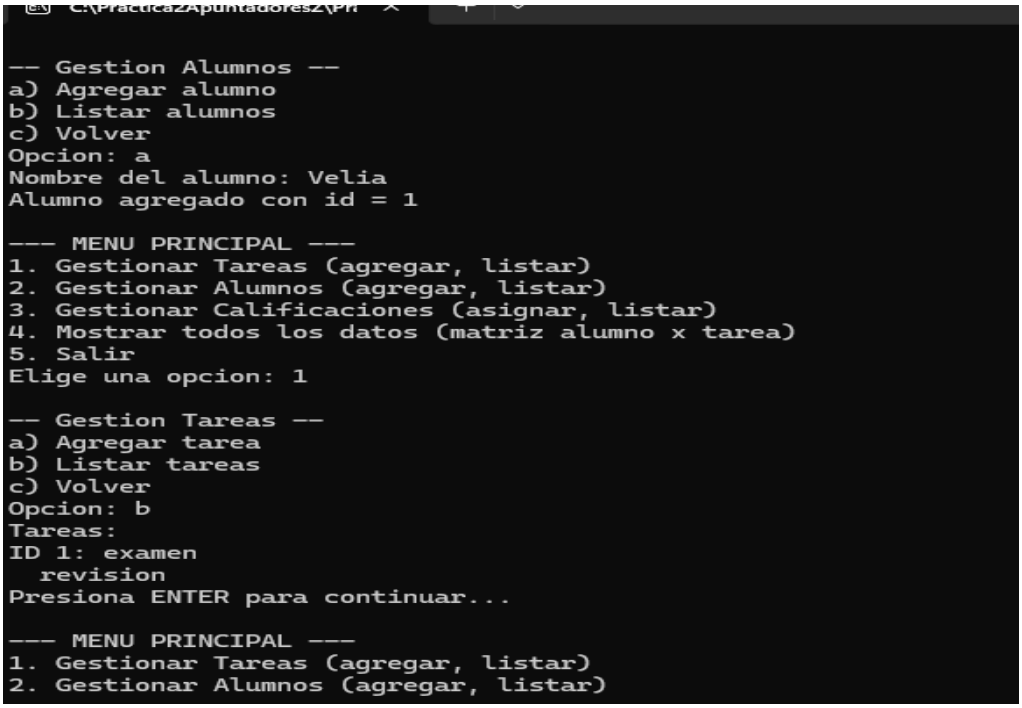
```
        } else { /* volver */ }
    }
    else if (strcmp(option, "4") == 0) {
        free(option); option = NULL;
        printf("\n==== MATRIZ DE RESULTADOS (Alumnos x Tareas) ====\n");
        list_students(students, scount);
        list_tasks(tasks, tcount);
        if (scount == 0 || tcount == 0) {
            printf("Se requieren al menos 1 alumno y 1 tarea para mostrar la matriz completa.\n");
        } else {
            /* Mostrar encabezado de tareas */
            printf("\nAlumno\\Tarea");
            for (int j = 0; j < tcount; ++j) printf(" | %d", tasks[j].id);
            printf("\n");
            for (int i = 0; i < scount; ++i) {
                printf("%s (id=%d)", students[i].name ? students[i].name : "(sin nombre)", students[i].id);
                for (int j = 0; j < tcount; ++j) {
                    float g = get_grade_for(grades, gcount, students[i].id, tasks[j].id);
                    if (g < 0.0f) printf(" |  -- ");
                    else printf(" | %6.2f", g);
                }
                printf("\n");
            }
        }
        pause_anykey();
    }
    else if (strcmp(option, "5") == 0) {
```

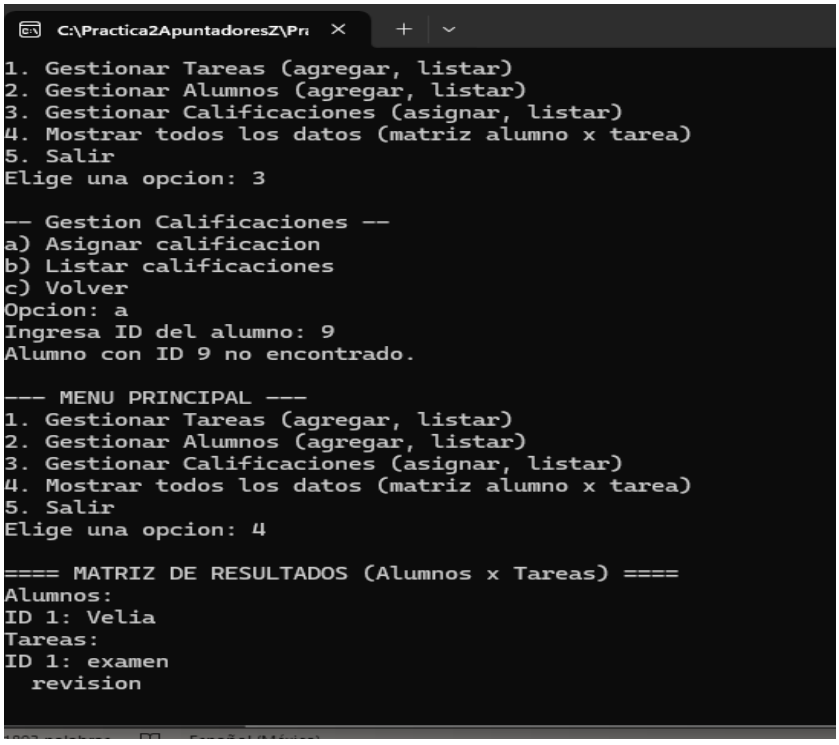
```
            free(option); option = NULL;
            printf("Saliendo...\n");
            break;
        }
        else {
            printf("Opción no valida.\n");
        }
        free(option); option = NULL;
    }

    /* Free de todo */
    free_all(tasks, tcount, students, scount, grades);
    return 0;
}
```

Pruebas – evidencia.




-- Gestion Alumnos --
a) Agregar alumno
b) Listar alumnos
c) Volver
Opcion: a
Nombre del alumno: Velia
Alumno agregado con id = 1
--- MENU PRINCIPAL ---
1. Gestionar Tareas (agregar, listar)
2. Gestionar Alumnos (agregar, listar)
3. Gestionar Calificaciones (asignar, listar)
4. Mostrar todos los datos (matriz alumno x tarea)
5. Salir
Elige una opcion: 1
-- Gestion Tareas --
a) Agregar tarea
b) Listar tareas
c) Volver
Opcion: b
Tareas:
ID 1: examen
revision
Presiona ENTER para continuar...
--- MENU PRINCIPAL ---
1. Gestionar Tareas (agregar, listar)
2. Gestionar Alumnos (agregar, listar)


C:\Practica2ApuntadoresZ\Pr
1. Gestionar Tareas (agregar, listar)
2. Gestionar Alumnos (agregar, listar)
3. Gestionar Calificaciones (asignar, listar)
4. Mostrar todos los datos (matriz alumno x tarea)
5. Salir
Elige una opcion: 3
-- Gestion Calificaciones --
a) Asignar calificacion
b) Listar calificaciones
c) Volver
Opcion: a
Ingresa ID del alumno: 9
Alumno con ID 9 no encontrado.
--- MENU PRINCIPAL ---
1. Gestionar Tareas (agregar, listar)
2. Gestionar Alumnos (agregar, listar)
3. Gestionar Calificaciones (asignar, listar)
4. Mostrar todos los datos (matriz alumno x tarea)
5. Salir
Elige una opcion: 4
==== MATRIZ DE RESULTADOS (Alumnos x Tareas) ====
Alumnos:
ID 1: Velia
Tareas:
ID 1: examen
revision

Tiempos estimados vs tiempos reales.

Actividad.	**Tiempo estimado.**	**Tiempo real.**
Análisis del problema.	40 min	30 min
Diseño de estructuras.	1 hora	40 minutos
Programación.	4 horas	5 horas
Pruebas.	1 hora	40 minutos
Documentación.	2 horas	1 hora
Total	8 horas 40 minutos	7 horas 50 minutos