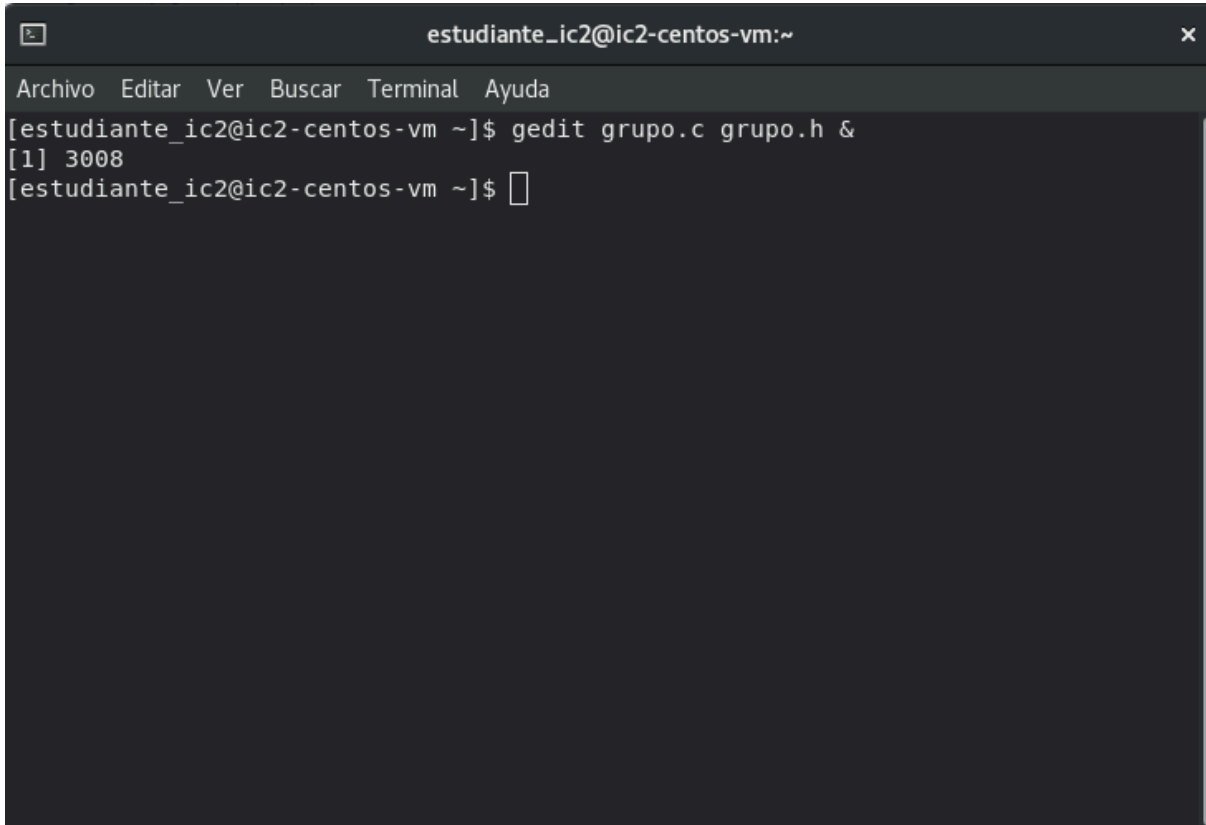


TRABAJO 1 IC2

Ayoze Ruano Alcántara, Enrique Reina Hernández, Fabio Nesta Arteaga Cabrera

En primer lugar, optamos por realizar la práctica siguiendo los requisitos para una calificación Apta. Empezamos por crear las librerías grupo.c y grupo.h.



```
estudiante_ic2@ic2-centos-vm:~  
Archivo Editar Ver Buscar Terminal Ayuda  
[estudiante_ic2@ic2-centos-vm ~]$ gedit grupo.c grupo.h &  
[1] 3008  
[estudiante_ic2@ic2-centos-vm ~]$
```

Vamos a distribuir el código de forma análoga al siguiente esquema:

Descomposición en módulos

opera.c

```
int suma(int a, int b){  
    int c;  
    c=a+b;  
    return c;  
}  
int resta(int a, int b){  
    int c;  
    c=a-b;  
    return c;  
}
```

opera.h

```
int suma(int a, int b);  
int resta(int a, int b);
```

test_opera.c

```
#include "opera.h"  
#include <stdlib.h>  
int main(){  
    int a=4, b=2, s, r;  
    s = suma(a,b);  
    printf("resultado suma=%d\n",s);  
    r = resta(a,b);  
    printf("resultado resta=%d\n",r);  
}
```

- Creación de grupo.c

```
#include <stdio.h>
#define MAX 50

int vector[];
int i;

// diferencia entre int n y la n que hay que inputear?
int inicializa_grupo(int n){
    for (i=0;i<n;i++){
        vector[i] = -1;
    }
}

// aun no ponemos void finaliza_grupo (porque no estamos con memdinam)

int matricula_alumno(int dni){
    for (i=0;i<n;i++){
        if (vector[i] == dni){
            return -2;
        }
        else if (vector[i] == -1){
            vector[i] = dni;
            return i;
        }
    }
    return -1; // problema: la n de antes no se guarda como variable global
}
}
```

En las instrucciones se pide que pasemos el tamaño del vector por teclado, pero eso lo haremos en el main. Pensamos que ese tamaño pasado era de alguna forma ese **int n**. En realidad son lo mismo, pero el valor del input se le pasa a ese **int n** más adelante.

También nos dimos cuenta que no podíamos utilizar esa n en siguientes funciones al no ser una variable global, así que creamos una variable extra, llamada a, para almacenar el contenido de n. Aparte tenemos el vector como tal, y una i (índice), para poder navegar el vector.

Nosotros obviamos el MAX, ya que el vector lo construimos con la propia n.

```
int vector[];
int i;
int a;

int inicializa_grupo(int n){
    a = n;
    for (i=0;i<a;i++){
        vector[i] = -1;
    }
}

// aun no ponemos void finaliza_grupo (porque no estamos con memdinam)
```

Función 1 terminada (en concepto) (Llena el vector de -1, según el tamaño del vector)

```

int matricula_alumno(int dni){
    for (i=0;i<a;i++){
        if (vector[i] == dni){
            return -2;
        }
        else if (vector[i] == -1){
            vector[i] = dni;
            return i;
        }
    }
    return -1;
}

```

En esta función, hacemos un for para recorrer el vector, **añadiéndonos al tamaño especificado en la función anterior**. Si encuentra el mismo DNI que se pasa, devuelve -2; y si encuentra un espacio libre (-1), asigna el DNI a esa posición y devuelve la posición. Si tras haber terminado de recorrer el vector no ha encontrado ninguna de esas cosas, devuelve -1.

Función 2 terminada (en concepto)

```

int desmatricula_alumno(int dni){
    for (i=0;i<a;i++){
        if (vector[i] == dni){
            vector[i] = -1;
            return i;
        }
    }
}

```

Esta función recorre el vector de forma que, si encuentra el dni pasado, desmatricula al alumno correspondiente, asignando un -1 en su posición y devolviendo tal posición.

Uno de nosotros sugirió que pusieramos un aviso en caso de que no encontrase el DNI en el vector.

```

int desmatricula_alumno(int dni){
    for (i=0;i<a;i++){
        if (vector[i] == dni){
            vector[i] = -1;
            return i;
        }
    }
    printf("No se ha encontrado el DNI solicitado en la lista")
}

```

La función entonces acabaría en el return en caso de encontrar el dni, o abajo en el printf, en caso contrario.

Función 3 terminada (en concepto)

```
int testea_asiento(int asiento){
    if (vector[asiento] == -1){
        return -1;
    }
    else if (asiento>a){
        return -2;
    }
    else{
        return vector[asiento];
    }
}
```

La función siguiente devuelve el DNI del alumno que esté en la posición pasada. Si el asiento está vacío, devuelve -1; y en caso de que el número de asiento sobrepase la posición máxima del vector (definida anteriormente) devuelve -2. Cabe destacar que no consideramos que el número del asiento pudiese ser negativo, ya que resulta ilógico.

Función 4 terminada (en concepto)

```
int plazas_libres(){
    counter = 0;
    for (i=0;i<a;i++){
        if (vector[i] == -1){
            counter += 1;
        }
    }
    return counter;
}
```

Para esta función, creamos una nueva variable global llamada counter, que sirve como contador. La función simplemente recorre el vector y añade 1 al counter cada vez que encuentre un espacio libre. Además, establecemos el counter a 0 para que tanto esta función como la siguiente puedan compartir esta variable.

Función 5 terminada (en concepto)

```
int plazas_ocupadas(){
    counter = 0;
    for (i=0;i<a;i++){
        if (vector[i] != -1){
            counter += 1;
        }
    }
    return counter;
}
```

La última función es una réplica de la anterior, sólo diferenciándose en que se cogen los valores distintos de -1 (espacios no vacíos). Como mencionamos antes, se asigna el counter a 0 por si, por ejemplo, se llamase a la función anterior y posteriormente a esta, o viceversa. Esto se hace para que el counter sea coherente, y se evita crear un nuevo counter también de esta manera.

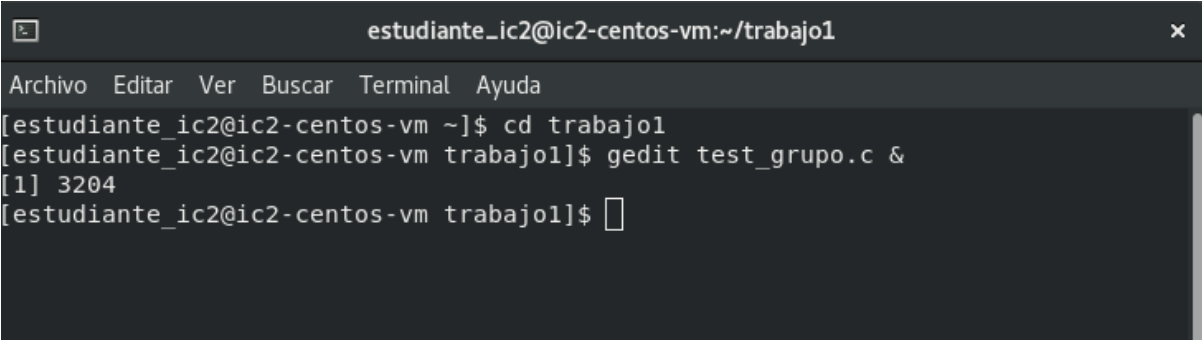
Función 6 terminada (en concepto)

- Creación de grupo.h

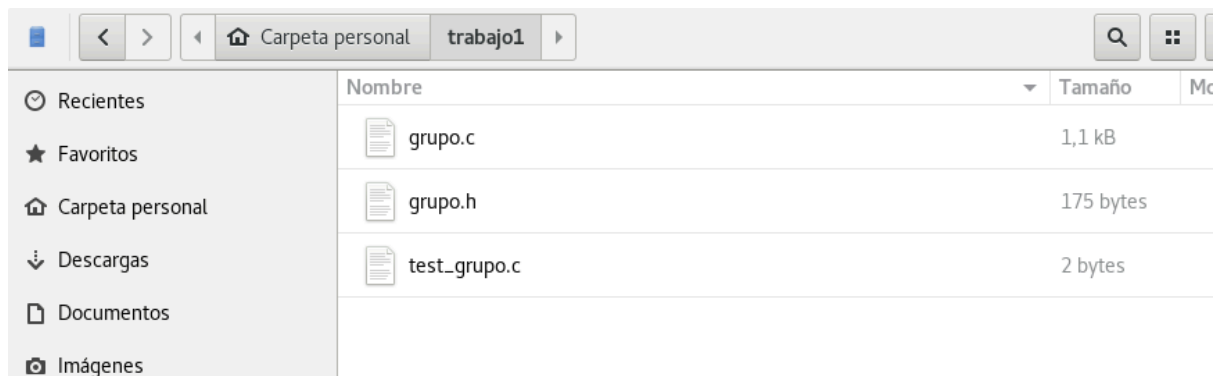
```
int inicializa_grupo(int n);
int matricula_alumno(int dni);
int desmatricula_alumno(int dni);
int testea_asiento(int asiento);
int plazas_libres();
int plazas_ocupadas();
```

Aquí simplemente listamos las funciones creadas en **grupo.c**

- Creación de test_grupo.c (Caso prueba)



```
estudiante_ic2@ic2-centos-vm:~/trabajo1
Archivo Editar Ver Buscar Terminal Ayuda
[estudiante_ic2@ic2-centos-vm ~]$ cd trabajo1
[estudiante_ic2@ic2-centos-vm trabajo1]$ gedit test_grupo.c &
[1] 3204
[estudiante_ic2@ic2-centos-vm trabajo1]$
```



```
#include "grupo.h" // esta va entre comillas porque la hemos creado nosotros
#include <stdio.h>

int aux;

int matricula(dni){
    if (plazas_libres() >= 1){
        if (matricula_alumno(dni) == -2){
            printf("El alumno con DNI %d ya está matriculado", dni);
        }
        else{
            aux = matricula_alumno(dni);
            printf("Se ha matriculado el alumno con DNI %d en el asiento/posición %d", dni, aux);
        }
    }
    else {
        printf("No hay plazas libres en el grupo");
    }
}
}
```

Antes de hacer el main, vamos a implementar las operaciones pedidas en este fichero. Esta sería la primera, donde utilizamos la función **plazas libres()** para ver si quedan plazas libres y si es posible matricular a la persona; y la función **matricula_alumno(dni)**, configurada anteriormente. También añadimos una variable auxiliar aux, para almacenar el contenido de la función matricula_alumno en caso de dar un índice (posición), pero tal vez no haga falta. Además, damos un mensaje de output personalizado para cada caso.

```
int desmatricula(dni){
    if (plazas_ocupadas() >= 1){
        aux = desmatricula_alumno(dni);
        printf("El alumno con DNI %d ha sido desmatriculado del grupo", dni);
    }
    else{
        printf("No hay alumnos matriculados en el grupo");
    }
}
}
```

En esta hacemos uso de las funciones **plazas_ocupadas()** y **desmatricula_alumno(dni)**. Primero se comprueba que haya algún integrante en el grupo, para más adelante chequear si algún DNI coincide con el solicitado. Este código estaría incompleto, ya que no hay un caso especificado por si no encuentra el DNI solicitado en la lista habiendo DNIs de otras personas en la misma.

Para arreglarlo, modificamos la función `desmatricula_alumno(dni)` en `grupo.c`.

```
// funcion 3
int desmatricula_alumno(int dni){
    for (i=0;i<a;i++){
        if (vector[i] == dni){
            vector[i] = -1;
            return i;
        }
    }
    return -3;
}
```

Modificamos la función para que devuelva un -3 en caso de no encontrar el DNI.

```
int desmatricula(dni){
    if (plazas_ocupadas() >= 1){
        if (desmatricula_alumno(dni) != -3){
            aux = desmatricula_alumno(dni);
            printf("El alumno con DNI %d y posición %d ha sido desmatriculado del grupo", dni, aux);
        }
        else{
            printf("No se ha encontrado el DNI solicitado en la lista");
        }
    }
    else{
        printf("No hay alumnos matriculados en el grupo");
    }
}
```

Ahora el caso es controlado por la propia función del main, de forma que avisa si no se encuentra el DNI pasado aunque haya DNIs de otras personas en la lista.

```
int matricula_multiple(int npersonas, int* lista_dni){
    if (plazas_libres() >= npersonas){
        for (i=0;i<npersonas;i++){
            matricula_alumno(*(lista_dni+i));
        }
    }
    else{
        printf("No hay plazas libres para todas las personas");
    }
}
```

Para esta última operación, preguntamos si hay plazas libres suficientes en la lista para meter al número de personas pedido, y si es así, hacemos un bucle de la función `matricula_alumno` por cada DNI del vector pasado.

```

int n;
int main(){

    printf("Introduzca el tamaño de la lista a crear: ");
    scanf("%d", n);
    inicializa_grupo(n);

    int npersonas = 5;
    int lista[ npersonas ] = {38808158, 17598594, 15002282, 97124473, 41678595};
    int* lista_dni = lista;
    matricula_multiple(npersonas, int* lista_dni);

    matricula(59317174);
    desmatricula(38808158);

    testea_asiento(5);
    testea_asiento(0);
    testea_asiento(1);

}

```

Este es nuestro primer main, donde probamos todas las funciones y algunos de los casos posibles. Al compilar:

```

[estudiante_ic2@ic2-centos-vm trabajo1]$ gcc -c test_grupo.c
test_grupo.c: En la función 'matricula':
test_grupo.c:7:5: aviso: el tipo de 'dni' es 'int' por defecto [-Wimplicit-int]
  int matricula(dni){
    ^~~~~~
test_grupo.c: En la función 'desmatricula':
test_grupo.c:22:5: aviso: el tipo de 'dni' es 'int' por defecto [-Wimplicit-int]
  int desmatricula(dni){
    ^~~~~~
test_grupo.c: En la función 'matricula_multiple':
test_grupo.c:39:8: error: 'i' no se declaró aquí (primer uso en esta función)
    for (i=0;i<npersonas;i++){
        ^
test_grupo.c:39:8: nota: cada identificador sin declarar se reporta sólo una vez para cada func
ce
test_grupo.c: En la función 'main':
test_grupo.c:57:2: error: un objeto de tamaño variable puede no ser inicializado
  int lista[ npersonas ] = {38808158, 17598594, 15002282, 97124473, 41678595};
  ^~~
test_grupo.c:57:26: aviso: exceso de elementos en el inicializador de matriz
  int lista[ npersonas ] = {38808158, 17598594, 15002282, 97124473, 41678595};
                        ^~~~~~
test_grupo.c:57:26: nota: (cerca de la inicialización de 'lista')
test_grupo.c:57:36: aviso: exceso de elementos en el inicializador de matriz
  int lista[ npersonas ] = {38808158, 17598594, 15002282, 97124473, 41678595};
                        ^~~~~~
test_grupo.c:57:36: nota: (cerca de la inicialización de 'lista')
test_grupo.c:57:46: aviso: exceso de elementos en el inicializador de matriz
  int lista[ npersonas ] = {38808158, 17598594, 15002282, 97124473, 41678595};
                        ^~~~~~
test_grupo.c:57:46: nota: (cerca de la inicialización de 'lista')
test_grupo.c:57:56: aviso: exceso de elementos en el inicializador de matriz
  int lista[ npersonas ] = {38808158, 17598594, 15002282, 97124473, 41678595};
                        ^~~~~~
test_grupo.c:57:56: nota: (cerca de la inicialización de 'lista')
test_grupo.c:57:66: aviso: exceso de elementos en el inicializador de matriz
  int lista[ npersonas ] = {38808158, 17598594, 15002282, 97124473, 41678595};
                        ^~~~~~
test_grupo.c:57:66: nota: (cerca de la inicialización de 'lista')
test_grupo.c:59:32: error: expected expression before 'int'
  matricula_multiple(npersonas, int* lista_dni);
                                ^

```



```
[estudiante_ic2@ic2-centos-vm trabajo1]$ gcc -c grupo.c
grupo.c:5:5: aviso: se asume que la matriz 'vector' tiene un elemento
int vector[];
    ^~~~~~
```

Corregimos los errores:

```
int n;
int i;
int main(){

    int npersonas = 5;
    int lista[5] = {38808158, 17598594, 15002282, 97124473, 41678595};
    int* lista_dni = lista;
    matricula_multiple(npersonas, lista_dni);
```

```
test_grupo.c: En la función 'matricula_multiple':
test_grupo.c:39:8: error: 'i' no se declaró aquí (primer uso en esta función)
    for (i=0;i<npersonas;i++){
        ^
```

La i tiene que estar declarada arriba de todo.

```
[estudiante_ic2@ic2-centos-vm trabajo1]$ gcc -c grupo.c
grupo.c:5:5: aviso: se asume que la matriz 'vector' tiene un elemento
int vector[];
    ^~~~~~

[estudiante_ic2@ic2-centos-vm trabajo1]$ gcc -c test_grupo.c
test_grupo.c: En la función 'matricula':
test_grupo.c:8:5: aviso: el tipo de 'dni' es 'int' por defecto [-Wimplicit-int]
int matricula(dni){
    ^~~~~~

test_grupo.c: En la función 'desmatricula':
test_grupo.c:23:5: aviso: el tipo de 'dni' es 'int' por defecto [-Wimplicit-int]
int desmatricula(dni){
    ^~~~~~

[estudiante_ic2@ic2-centos-vm trabajo1]$
```

Ahora daría esos warnings, pero vamos a ejecutar el código para ver si al menos corre.

```
[estudiante_ic2@ic2-centos-vm trabajo1]$ gcc -o test_grupo test_grupo.o grupo.o
[estudiante_ic2@ic2-centos-vm trabajo1]$ ./test_grupo
Introduzca el tamaño de la lista a crear: 10
Segmentation fault ('core' generado)
[estudiante_ic2@ic2-centos-vm trabajo1]$
```

```
printf("Introduzca el tamaño de la lista a crear: ");
scanf("%d", &n);
inicializa_grupo(n);
```

Pusimos el & en el scanf, se nos olvidó.

Además, vamos a hacer el Makefile, para facilitar las pruebas que hagamos con el programa.

```
test_grupo: test_grupo.o grupo.o
    gcc -o test_grupo test_grupo.o grupo.o
grupo.o: grupo.c
    gcc -c grupo.c
test_grupo.o: test_grupo.c
    gcc -c test_grupo.c
clean:
    rm *.o
```

```
[estudiante_ic2@ic2-centos-vm trabajo1]$ make
gcc -c test_grupo.c
test_grupo.c: En la función 'matricula':
test_grupo.c:8:5: aviso: el tipo de 'dni' es 'int' por defecto [-Wimplicit-int]
int matricula(dni){
    ^~~~~~
test_grupo.c: En la función 'desmatricula':
test_grupo.c:23:5: aviso: el tipo de 'dni' es 'int' por defecto [-Wimplicit-int]
int desmatricula(dni){
    ^~~~~~
gcc -o test_grupo test_grupo.o grupo.o
[estudiante_ic2@ic2-centos-vm trabajo1]$ ./test_grupo
Introduzca el tamaño de la lista a crear: 10
No hay plazas libres para todas las personasNo hay plazas libres en el grupoNo hay alumnos matriculados en el grupo
[estudiante_ic2@ic2-centos-vm trabajo1]$
```

Hasta ahora, han sido Ayoze y Fabio los que han ido haciendo el código, con varias puntualizaciones por parte de Enrique (él estaba de vacaciones). A partir de aquí, nos ponemos todos a intentar resolver los fallos en nuestro código.

Ayoze añadió una función llamada leer vector para saber cuál es el estado general del vector en cada momento, y también para hacer debug en cierto modo. Enrique sugirió printear la “a” para saber si el fallo se encontraba con esta variable.

```
// funcion 1
int inicializa_grupo(int n){
    a = n;
    printf("a = %d\n", a);
    for (i=0;i<a;i++){
        vector[i] = -1;
    }
}

//

int leer_vector(){
    printf("a = %d\n", a);
    for (i=0;i<a;i++){
        printf("%d,", vector[i]);
    }
}
```

```
[estudiante_ic2@ic2-centos-vm trabajo1]$ ./test_grupo
Introduzca el tamaño de la lista a crear: 10
a = 10
a = -1
No hay plazas libres para todas las personasNo hay plazas libres en el grupoNo h
ay alumnos matriculados en el grupo[estudiante_ic2@ic2-centos-vm trabajo1]$
```

Llegamos a la conclusión de que nuestro problema estaba en las variables. Enrique comentó acerca del extern (variables globales), así que vamos a intentarlo con eso.

```
extern int vector[];
int i; // indice
extern int n;
int counter; // contador

// funcion 1
int inicializa_grupo(int n){
    for (i=0;i<n;i++){
        vector[i] = -1;
    }
}
```

Tampoco nos funcionó así. Optamos por quitar los extern y definir el MAX 50, y meterlo como tamaño del vector inicial, pero tan solo cogeremos la n pasada para recorrer, de forma que maneje un vector de n elementos y no de 50.

```
#include <stdio.h>
#define MAX 50

int vector[MAX];
int i; // indice
int a; // almacena n
int counter;
```

Además hicimos algunas modificaciones en las funciones tanto del grupo.c como del test_grupo.c (el resto permanece igual, si no hay captura es que permanece igual que como estaba anteriormente):

```
int leer_vector(){
    for (i=0; i<a; i++){
        if (i == a-1){
            printf("%d\n", vector[i]);
        }
        else {
            printf("%d, ", vector[i]);
        }
    }
}
```

Modificamos esta función, (función extra implementada por Ayoze), para que muestre en pantalla como se encuentra el vector, para comprobar algún error en el vector.

```
int aux;
int i;
int matricula(int dni){
    if (plazas_libres() >= 1){
        aux = matricula_alumno(dni);
        if (aux == -2){
            printf("El alumno con DNI %d ya está matriculado\n", dni);
        }
        else{
            printf("Se ha matriculado el alumno con DNI %d en el asiento/posición %d\n", dni, aux);
        }
    }
    else {
        printf("No hay plazas libres en el grupo\n");
    }
}
```

En esta función del test_grupo.c nos dimos cuenta que el aux estaba mal, debido a que antes se estaba llamando una segunda vez al aux y daba un output incoherente, de esta forma solo se llama una vez a la función.

```

int desmatricula(int dni){
    if (plazas_ocupadas() >= 1){
        aux = desmatricula_alumno(dni);
        if (aux != -3){
            printf("El alumno con DNI %d y posición %d ha sido desmatriculado del grupo\n", dni, aux);
        }
        else{
            printf("No se ha encontrado el DNI solicitado en la lista\n");
        }
    }
    else{
        printf("No hay alumnos matriculados en el grupo\n");
    }
}

```

Aquí sucedía lo mismo que en la anterior función, por lo que solucionar el error fue lo mismo.

Además añadimos en prácticamente todas las frases de las funciones un “\n” en los printf para que sea más legible.

Vamos a proceder a probar el código, ahora creemos que no hay ningún error.

Aquí está el programa principal.

```

int main(){

    int n;
    printf("Introduzca el tamaño de la lista a crear: ");
    scanf("%d", &n);
    inicializa_grupo(n);
    leer_vector();

    int npersonas = 2;
    int lista[2] = {38808158, 17598594};
    int* lista_dni = lista;

    matricula_multiple(npersonas, lista_dni);
    matricula(59317174);
    desmatricula(38808158);

    testea_asiento(5);
    testea_asiento(0);
    testea_asiento(1);
}

```

```

[estudiante_ic2@ic2-centos-vm trabajo1]$ ./test_grupo
Introduzca el tamaño de la lista a crear: 10
-1,-1,-1,-1,-1,-1,-1,-1,-1,-1
Se ha matriculado el alumno con DNI 59317174 en el asiento/posición 2
El alumno con DNI 38808158 y posición 0 ha sido desmatriculado del grupo
[estudiante_ic2@ic2-centos-vm trabajo1]$ █

```

Podemos observar que da un resultado coherente, ahora realizaremos unas cuantas pruebas en el main para comprobar todos los casos posibles, por si hay que pulir el código en alguna situación. Aparte de modificaciones menores.

```
int leer_vector(){
    for (i=0;i<a;i++){
        if (i == a-1){
            printf("(%d)\n", vector[i]);
        }
        else {
            printf("(%d),", vector[i]);
        }
    }
}
```

Cambiamos el leer_vector (función extra) para que sea más legible.

```
int inicializa_grupo(int n){
    a = n;
    if(a<=MAX){
        for (i=0;i<a;i++){
            vector[i] = -1;
        }
    }
    else{
        printf("Se ha excedido el tamaño permitido por la clase.\n");
        exit(1);
    }
}
```

Le pusimos una condición que si el valor de n es mayor que 50 (lo permitido), te mande un mensaje explicando que te has pasado del máximo y a continuación se corta el código.

Nuevo main:

```

int main(){

    int n;
    printf("Introduzca el tamaño de la lista a crear: ");
    scanf("%d", &n);
    inicializa_grupo(n);
    leer_vector();

    printf("\n(Se intenta desmatricular cuando no hay nadie matriculado)\n");
    desmatricula(38808158);

    printf("\n(Se procede a hacer una matricula multiple)\n");
    int npersonas = 3;
    int lista[3] = {38808158, 17598594, 51489657};
    int* lista_dni = lista;
    matricula_multiple(npersonas, lista_dni);
    leer_vector();

    printf("\n(Se mete un matriculado nuevo(solo uno))\n");
    matricula(59317174);
    leer_vector();

    printf("\n(Se intenta matricular a alguien ya matriculado)\n");
    matricula(59317174);

    printf("\n(Se desmatricula el primero)\n");
    desmatricula(38808158);
    leer_vector();

    printf("\n(Se intenta desmatricular a alguien no matriculado)\n");
    desmatricula(38808158);

    printf("\n(Se procede a hacer una matricula multiple, que sobrepase el límite)\n");
    npersonas = 8;
    int lista2[8] = {38805658, 17863894, 54615168, 58394884, 65847219, 63514298, 98426571, 36951753};
    *lista_dni = lista2;
    matricula_multiple(npersonas, lista_dni);

    printf("\n(Se matricula a uno solo para luego comprobar que ocurre si esta todo ocupado)\n");
    matricula(38808158);
    leer_vector();

    printf("\n(Ahora se matricula a uno nuevo para ver si da ese error)\n");
    matricula(12356847);

    testea_asiento(5);
    testea_asiento(0);
    testea_asiento(1);
}

```

```
[estudiante_ic2@ic2-centos-vm trabajo1]$ ./test_grupo
Introduzca el tamaño de la lista a crear: 4
(-1),(-1),(-1),(-1)

(Se intenta desmatricular cuando no hay nadie matriculado)
No hay alumnos matriculados en el grupo

(Se procede a hacer una matricula multiple)
(38808158),(17598594),(51489657),(-1)

(Se mete un matriculado nuevo(solo uno))
Se ha matriculado el alumno con DNI 59317174 en el asiento/posición 3
(38808158),(17598594),(51489657),(59317174)

(Se intenta matricular a alguien ya matriculado)
No hay plazas libres en el grupo

(Se desmatricula el primero)
El alumno con DNI 38808158 y posición 0 ha sido desmatriculado del grupo
(-1),(17598594),(51489657),(59317174)

(Se intenta desmatricular a alguien no matriculado)
No se ha encontrado el DNI solicitado en la lista

(Se procede a hacer una matricula multiple, que sobrepase el límite)
No hay plazas libres para todas las personas

(Se matricula a uno solo para luego comprobar que ocurre si esta todo ocupado)
Se ha matriculado el alumno con DNI 38808158 en el asiento/posición 0
(38808158),(17598594),(51489657),(59317174)

(Ahora se matricula a uno nuevo para ver si da ese error)
No hay plazas libres en el grupo
```

Hemos implementado un caso de prueba para cada situación. De esta manera verificamos todos los casos posibles.

En este nuevo main vemos que cuando vamos a matricular un DNI ya matriculado si está lleno el grupo, nos dice que está lleno igualmente y no mira si está realmente el DNI. Se puede ver en las siguientes imágenes:

```
int matricula(int dni){
    if (plazas_libres() >= 1){
        aux = matricula_alumno(dni);
        if (aux == -2){
            printf("El alumno con DNI %d ya está matriculado\n", dni);
        }
        else{
            printf("Se ha matriculado el alumno con DNI %d en el asiento/posición %d\n", dni, aux);
        }
    }
    else {
        printf("No hay plazas libres en el grupo\n");
    }
}
```

```
(Se intenta matricular a alguien ya matriculado)
No hay plazas libres en el grupo
```


Para solucionarlo podemos implementar una condición en el else, la cual es muy parecida a la del if anterior.

```
int matricula(int dni){
    if (plazas_libres() >= 1){
        aux = matricula_alumno(dni);
        if (aux == -2){
            printf("El alumno con DNI %d ya está matriculado\n", dni);
        }
        else{
            printf("Se ha matriculado el alumno con DNI %d en el asiento/posición %d\n", dni, aux);
        }
    }
    else {
        aux = matricula_alumno(dni);
        if (aux == -2){
            printf("El alumno con DNI %d ya está matriculado\n", dni);
        }
        else{
            printf("No hay plazas libres en el grupo\n");
        }
    }
}
```

```
estudiante_ic2@ic2-centos-vm trabajo1]$ ./test_grupo
Introduzca el tamaño de la lista a crear: 4
(-1),(-1),(-1),(-1)

(Se intenta desmatricular cuando no hay nadie matriculado)
No hay alumnos matriculados en el grupo

(Se procede a hacer una matricula multiple)
(38808158),(17598594),(51489657),(-1)

(Se mete un matriculado nuevo(solo uno))
Se ha matriculado el alumno con DNI 59317174 en el asiento/posición 3
(38808158),(17598594),(51489657),(59317174)

(Se intenta matricular a alguien ya matriculado)
El alumno con DNI 59317174 ya está matriculado

(Se desmatricula el primero)
El alumno con DNI 38808158 y posición 0 ha sido desmatriculado del grupo
(-1),(17598594),(51489657),(59317174)

(Se intenta desmatricular a alguien no matriculado)
No se ha encontrado el DNI solicitado en la lista

(Se procede a hacer una matricula multiple, que sobrepase el límite)
No hay plazas libres para todas las personas

(Se matricula a uno solo para luego comprobar que ocurre si esta todo ocupado)
Se ha matriculado el alumno con DNI 38808158 en el asiento/posición 0
(38808158),(17598594),(51489657),(59317174)

(Ahora se matricula a uno nuevo para ver si da ese error)
No hay plazas libres en el grupo
```

Después de eso intentamos probar el código pero con más de 4 espacios para ver como funcionaba y añadimos un caso extra en el main para probar que pasa si introducimos otra lista de DNIs. Caso extra del main:

```

        printf("\n(Se procede a hacer otra matricula multiple)\n");
    npersonas = 4;
    int lista3[4] = {99808158, 99598594, 99489657, 99857465};
    *lista_dni = lista3;
    matricula_multiple(npersonas, lista_dni);
    leer_vector();

```

Lo cual devuelve cuando lo ejecutamos

```

(Se procede a hacer otra matricula multiple)
he mandado-149051840
(38808158),(17598594),(51489657),(-149051840),(-1),(-1),(-1),(-1),(-1),(-1),(-1),(-1),(-1),(-1),(-1)

```

Que en este caso en vez de poner los 4 Dnis devuelve un valor sin sentido.

Después de un tiempo nos dimos cuenta que se debía a que poníamos `*lista_dni = lista3` ya que de esta forma estamos guardando en `lista_dni` la posición en memoria de `lista3` en vez de el contenido de `lista 3`. Para solucionar esto quitamos algunos `*` que sobraban y modificamos cuando inicializamos `lista_dni` porque al hacerlo de esta forma el programa entiende que no quieres guardar la posición en memoria, lo que puede ser lioso, por lo que declaramos `lista_dni` de la siguiente forma:

```

printf("\n(Se procede a hacer una matricula multiple)\n");
int npersonas = 3;
int lista[3] = {38808158, 17598594, 51489657};
int* lista_dni;
lista_dni = lista;
matricula_multiple(npersonas, lista_dni);
leer_vector();

```

```

printf("\n(Se procede a hacer otra matricula multiple)\n");
npersonas = 4;
int lista3[4] = {99808158, 99598594, 99489657, 99857465};
lista_dni = lista3;
matricula_multiple(npersonas, lista_dni);
leer_vector();

```

```

printf("\n(Se procede a hacer una matricula multiple, que sobrepase el límite)\n");
npersonas = 8;
int lista2[8] = {38805658, 17863894, 54615168, 58394884, 65847219, 63514298, 98426571, 36951753};
lista_dni = lista2;
matricula_multiple(npersonas, lista_dni);
leer_vector();

```

```

(Se procede a hacer una matricula multiple)
he mandado38808158
he mandado17598594
he mandado51489657
(38808158),(17598594),(51489657),(-1),(-1),(-1),(-1),(-1),(-1),(-1),(-1),(-1),(-1),(-1)

(Se procede a hacer otra matricula multiple)
he mandado99808158
(38808158),(17598594),(51489657),(99808158),(-1),(-1),(-1),(-1),(-1),(-1),(-1),(-1),(-1),(-1),(-1)

```

Ahora nos dimos cuenta que solo nos enseñaba 1 dni cuando en la lista habíamos pasado 4.

No encontramos cuál podría ser el defecto del código, entonces pedimos una tutoría para ver si encontrábamos el error. Se llegó a la conclusión que el problema que había era que la `i` estaba implementada como variable global, lo que puede traer una serie de imperfecciones en el código, para solucionarlo quitamos la variable global y la pusimos en las funciones pertinentes.

```

int matricula_multiple(int npersonas, int* lista_dni){
    int i;
    if (plazas_libres() >= npersonas){
        for (i=0;i<npersonas;i++){
            printf("he mandado %d de %d\n", *(lista_dni+i), i);
            matricula_alumno(*(lista_dni+i));
        }
    }
    else{
        printf("No hay plazas libres para todas las personas\n");
    }
}

int inicializa_grupo(int n){
    int i;
    a = n;
    vector = (int*)malloc(n * sizeof(int));
    if(a<=MAX){
        for (i=0;i<a;i++){
            vector[i] = -1;
        }
    }
    else{
        printf("Se ha excedido el tamaño permitido por la clase.\n");
        exit(1);
    }
}

```

```

int leer_vector(){
    int i;
    for (i=0;i<a;i++){
        if (i == a-1){
            printf("(%d)\n", vector[i]);
        }
        else {
            printf("(%d),", vector[i]);
        }
    }
}

int matricula_alumno(int dni){
    int i;
    for (i=0;i<a;i++){
        if (vector[i] == dni){
            return -2;
        }
        else if (vector[i] == -1){
            vector[i] = dni;
            return i;
        }
    }
    return -1;
}

int desmatricula_alumno(int dni){
    int i;
    for (i=0;i<a;i++){
        if (vector[i] == dni){
            vector[i] = -1;
            return i;
        }
    }
    return -3;
}

```

```

int plazas_libres(){
    int i;
    int counter = 0;
    for (i=0;i<a;i++){
        if (vector[i] == -1){
            counter += 1;
        }
    }
    return counter;
}

int plazas_ocupadas(){
    int i;
    int counter = 0;
    for (i=0;i<a;i++){
        if (vector[i] != -1){
            counter += 1;
        }
    }
    return counter;
}

```

Cambie la función de testea asiento para que te imprima los casos que ocurren:

```

int testea_asiento(int asiento){
    if (vector[asiento] == -1){
        printf("El asiento %d está vacío\n", asiento);
        return -1;
    }
    else if (asiento>a){
        printf("El asiento %d no se encuentra en el rango\n", asiento);
        return -2;
    }
    else{
        printf("La persona con DNI: %d esta sentado en el asiento: %d\n", vector[asiento], asiento);
        return vector[asiento];
    }
}

```

main:

```

    testea_asiento(11);
    testea_asiento(28);
    testea_asiento(1);
}

```

```

El asiento 11 está vacío
El asiento 28 no se encuentra en el rango
La persona con DNI: 17598594 esta sentado en el asiento: 1

```

#####

Para implementar la memoria dinámica implementamos las siguientes funciones:

5. **Memoria dinámica** Una vez realizado el ejercicio, se puede modificar la biblioteca de forma que el número de alumnos máximo por grupo ((n_matriculados)) se pueda definir de forma dinámica, en tiempo de ejecución. Esto se consigue creando el vector mediante la función **malloc()**.

- **Malloc:** asigna memoria apuntada por el puntero:

```
char *str = (char *) malloc(15);
```

```
int *str = (int *) malloc(15);
```

- **void finaliza_grupo()**– En el caso de memoria dinámica, tiene que liberar dicha memoria.

- **Free:** libera la memoria
free(str)

```
int *vector;  
int i; // indice  
int a; // almacena n  
int counter;
```

|

```

int inicializa_grupo(int n){
    a = n;
    vector = (int *)malloc(n * sizeof(int));
    if(a<=MAX){
        for (i=0;i<a;i++){
            vector[i] = -1;
        }
    }
    else{
        printf("Se ha excedido el tamaño permitido por la clase.\n");
        exit(1);
    }
}

void finaliza_grupo(){
    free(vector);
}

```

Uno de los cambios que realizamos fue cambiar el vector a un puntero para ello quitamos los corchetes porque si no habría dado un error al realizar el malloc, luego en inicializar_grupo implementamos el malloc al vector y en él un sizeof para que nos haga que el tamaño del vector sea n.

La función finaliza_grupo lo que hace es liberar la memoria del vector, por tanto se podría decir que es la función que, como dice su nombre, acaba con toda la memoria que tenemos en el vector. Lo que luego nos permite poder crear un nuevo grupo.

Después modificamos el main para que finalizara el grupo y volviera a crear uno nuevo.

```

    testea_asiento(11);
    testea_asiento(28);
    testea_asiento(1);
    printf("\n");

finaliza_grupo();
//leer_vector();
printf("\n");

printf("Ahora creamos otro grupo: ");
scanf("%d", &n);
inicializa_grupo(n);
leer_vector();
}

```

[illegible]

Como se puede apreciar se crea un nuevo grupo cuando después de “Ahora creamos otro grupo: “ le pasamos un tamaño de 8 y se cambia el tamaño que iba antes (15) al nuevo que le pasamos.

Al final el código devuelve en este punto:

[illegible]

Esta sería la versión del código sin el struct ni manejo de errores: (Tienes su output justo arriba)

GRUPO.C

```
#include <stdio.h>
#define MAX 50

int *vector;
int a; // almacena n

int inicializa_grupo(int n){
    int i;
    a = n;
    vector = (int *)malloc(n * sizeof(int));
    if(a<=MAX){
        for (i=0;i<a;i++){
            vector[i] = -1;
        }
    }
    else{
        printf("Se ha excedido el tamaño permitido por la clase.\n");
        exit(1);
    }
}

void finaliza_grupo(){
    free(vector);
}

int leer_vector(){
    int i;
    for (i=0;i<a;i++){
        if (i == a-1){
            printf("(%d)\n", vector[i]);
        }
        else {
            printf("(%d),", vector[i]);
        }
    }
}
```

```

int matricula_alumno(int dni){
    int i;
    for (i=0;i<a;i++){
        if (vector[i] == dni){
            return -2;
        }
        else if (vector[i] == -1){

            vector[i] = dni;
            return i;
        }
    }
    return -1;
}

int desmatricula_alumno(int dni){
    int i;
    for (i=0;i<a;i++){
        if (vector[i] == dni){
            vector[i] = -1;
            return i;
        }
    }
    return -3;
}

int testea_asiento(int asiento){
    if (vector[asiento] == -1){
        printf("El asiento %d está vacio\n", asiento);
        return -1;
    }
    else if (asiento>a){
        printf("El asiento %d no se encuentra en el rango\n", asiento);
        return -2;
    }
    else{
        printf("La persona con DNI: %d esta sentado en el asiento:
%d\n", vector[asiento], asiento);
        return vector[asiento];
    }
}

```

```
int plazas_libres(){
    int i;
    int counter = 0;
    for (i=0;i<a;i++){
        if (vector[i] == -1){
            counter += 1;
        }
    }
    return counter;
}
```

```
int plazas_ocupadas(){
    int i;
    int counter = 0;
    for (i=0;i<a;i++){
        if (vector[i] != -1){
            counter += 1;
        }
    }
    return counter;
}
```

GRUPO.H

```
int inicializa_grupo(int n);
int matricula_alumno(int dni);
int desmatricula_alumno(int dni);
int testea_asiento(int asiento);
int plazas_libres();
int plazas_ocupadas();
```

TEST_GRUPO.C

```
#include <stdio.h>
#include "grupo.h"

int* n;
int aux;
int matricula(int dni){
    if (plazas_libres() >= 1){
        aux = matricula_alumno(dni);
        if (aux == -2){
            printf("El alumno con DNI %d ya está matriculado\n", dni);
        }
        else{
            printf("Se ha matriculado el alumno con DNI %d en el
asiento/posición %d\n", dni, aux);
        }
    }
    else {
        aux = matricula_alumno(dni);
        if (aux == -2){
            printf("El alumno con DNI %d ya está matriculado\n", dni);
        }
        else{
            printf("No hay plazas libres en el grupo\n");
        }
    }
}

int desmatricula(int dni){
    if (plazas_ocupadas() >= 1){
        aux = desmatricula_alumno(dni);
        if (aux != -3){
            printf("El alumno con DNI %d y posición %d ha sido
desmatriculado del grupo\n", dni, aux);
        }
        else{
            printf("No se ha encontrado el DNI solicitado en la
lista\n");
        }
    }
    else{
        printf("No hay alumnos matriculados en el grupo\n");
    }
}
```

```

}

int matricula_multiple(int npersonas, int* lista_dni){
    int i;
    if (plazas_libres() >= npersonas){
        for (i=0;i<npersonas;i++){
            //printf("he mandado %d de %d\n", *(lista_dni+i), i);
            matricula_alumno(*(lista_dni+i));
        }
    }
    else{
        printf("No hay plazas libres para todas las personas\n");
    }
}

int main(){

    printf("Introduzca el tamaño de la lista a crear: ");
    scanf("%d", &n);
    inicializa_grupo(n);
    leer_vector();

    printf("\n(Se intenta desmatricular cuando no hay nadie
matriculado)\n");
    desmatricula(38808158);

    printf("\n(Se procede a hacer una matricula multiple)\n");
    int npersonas = 3;
    int lista[3] = {38808158, 17598594, 51489657};
    int* lista_dni;
    lista_dni = lista;
    matricula_multiple(npersonas, lista_dni);
    leer_vector();

    printf("\n(Se procede a hacer otra matricula multiple)\n");
    npersonas = 5;
    int lista3[5] = {99808158, 99598594, 99489657, 99857465, 35248697};
    lista_dni = lista3;
    matricula_multiple(npersonas, lista_dni);
    leer_vector();

    printf("\n(Se mete un matriculado nuevo(solo uno))\n");
    matricula(59317174);

```

```

    leer_vector();

    printf("\n(Se intenta matricular a alguien ya matriculado)\n");
    matricula(59317174);
    leer_vector();

    printf("\n(Se desmatricula el primero)\n");
    desmatricula(38808158);
    leer_vector();

    printf("\n(Se intenta desmatricular a alguien no matriculado)\n");
    desmatricula(38808158);
    leer_vector();

    printf("\n(Se procede a hacer una matricula multiple, que sobrepase
el límite)\n");
    npersonas = 8;
    int lista2[8] = {38805658, 17863894, 54615168, 58394884, 65847219,
63514298, 98426571, 36951753};
    lista_dni = lista2;
    matricula_multiple(npersonas, lista_dni);
    leer_vector();

    printf("\n(Se matricula a uno solo para luego comprobar que ocurre
si esta todo ocupado)\n");
    matricula(38808158);
    leer_vector();
    printf("\n");
/*
    printf("\n(Ahora se matricula a uno nuevo para ver si da ese
error)\n");
    matricula(12356847);
    leer_vector();
*/
//Este caso se cumple en casos concretos

    testea_asiento(11);
    testea_asiento(28);
    testea_asiento(1);
    printf("\n");

```

```

finaliza_grupo();
/*
Mirar porque cuando liberamos el vector y luego lo leemos no nos sale
vacío
y nos devuelve algo raro.
*/
leer_vector();
printf("\n");

printf("Ahora creamos otro grupo: ");
scanf("%d", &n);
inicializa_grupo(n);
leer_vector();
}

```

#####

Para el **struct** hicimos lo siguiente:

```

#include <stdio.h>
#define MAX 50
#include <string.h>
#include

int *vector;
int a; // almacena n
int vecaux[MAX];

int inicializa_grupo(int n){
    int i;
    a = n;
    vector = (int *)malloc(n * sizeof(int));
    if(a<=MAX){
        for (i=0;i<a;i++){
            vector[i] = -1;
        }
    }
    else{
        printf("Se ha excedido el tamaño permitido por la clase.\n");
        exit(1);
    }
}

```

```

void finaliza_grupo(){
    free(vector);
}

int leer_vector(){
    int i;
    for (i=0;i<a;i++){
        if (i == a-1){
            printf("(%d)\n", vector[i]);
        }
        else {
            printf("(%d),", vector[i]);
        }
    }
}

```

// LAS DE ARRIBA SE QUEDAN IGUALES

```

int matricula_alumno(lista*){
    int i;
    for (i=0;i<a;i++){
        if (vector[i] == (lista*).dni){
            return -2;
        }
        else if (vector[i] == -1){
            vector[i] = {(lista*).dni, (lista*).nombre};
            vecaux[i] = (lista*);
            vecaux[i].numero_matricula = i;

            return i;
        }
    }
    return -1;
}

```

```

char* desmatricula_alumno(int dni){
    int i;
    char str[];
    for (i=0;i<a;i++){
        if (vecaux[i].dni == dni){
            str = vecaux[i].nombre;
            strcat(str, " ");
            strcat(str, vecaux[i].apellidos);
            vector[i] = -1;

            return str;
        }
    }
    return -3;
}

```


(El resto de grupo.c iguales)

```
#include <stdio.h>
#include "grupo.h"
#include <string.h>

int* n;
int aux;
struct new {
    int dni;
    char nombre[10];
    char apellidos[20];
    int numero_matricula;
};

int matricula(int dni, char* nom, char* apellido, int numero_matricula){
    if (plazas_libres() >= 1){

        struct new nuevo[1] = {dni, nom, apellido, numero_matricula};

        struct new* lista = nuevo;
        aux = matricula_alumno(lista*);
        if (aux == -2){
            printf("El alumno con DNI %d ya está matriculado\n", dni);
        }
        else{
            printf("Se ha matriculado el alumno con DNI %d en el asiento/posición %d\n", dni, aux);
        }
    }
    else {
        aux = matricula_alumno(dni);
        if (aux == -2){
            printf("El alumno con DNI %d ya está matriculado\n", dni);
        }
        else{
            printf("No hay plazas libres en el grupo\n");
        }
    }
}
```

```

int matricula_multiple(int npersonas, struct new* lista){
    int i;
    if (plazas_libres() >= npersonas){
        for (i=0;i<npersonas;i++){
            //printf("he mandado %d de %d\n", *(lista_dni+i), i);
            matricula_alumno(*(lista+i));
        }
    }
    else{
        printf("No hay plazas libres para todas las personas\n");
    }
}

```

```

int main(){

    printf("Introduzca el tamaño de la lista a crear: ");
    scanf("%d", &n);
    inicializa_grupo(n);
    leer_vector();

    printf("\n(Se intenta desmatricular cuando no hay nadie matriculado)\n");
    desmatricula(38808158);

    printf("\n(Se procede a hacer una matricula multiple)\n");
    int npersonas = 3;
    struct new lista3[3] = {{38808158, "Ayoze", "Ruano Alcántara"},{59874632, "Nesta", "Arteaga Cabrera"},{45179111, "Enrique", "Reina Hernández"},}
    struct new* lista_dni;
    lista_dni = lista;
    matricula_multiple(npersonas, lista_dni);
    leer_vector();

    printf("\n(Se procede a hacer otra matricula multiple)\n");
    npersonas = 2;
    struct new lista3[2] = {{66666666, "Cristina", "González López"},{55555555, "Manuel", "Delacruz Quintana"},}
    lista_dni = lista3;
    matricula_multiple(npersonas, lista_dni);
    leer_vector();

    printf("\n(Se mete un matriculado nuevo(solo uno))\n");
    matricula(59317174, "Isidro", "Velázquez Duarte", 0);
    leer_vector();

    printf("\n(Se intenta matricular a alguien ya matriculado)\n");
    matricula(59317174, "Isidro", "Velázquez Duarte", 0);
    leer_vector();

    printf("\n(Se desmatricula el primero)\n");
    desmatricula(38808158);
    leer_vector();

    printf("\n(Se intenta desmatricular a alguien no matriculado)\n");
    desmatricula(38808158);
    leer_vector();
}

```

```

[estudiante_ic2@ic2-centos-vm ~]$ cd trabajo1
[estudiante_ic2@ic2-centos-vm trabajo1]$ make
gcc -c test_grupo.c
test_grupo.c: En la función 'matricula':
test_grupo.c:18:30: aviso: la inicialización de 'char' desde 'char *' crea un entero desde un puntero sin una conversión [-Wint-conversion]
    struct new nuevo[1] = {dni, nom, apellido, numero_matricula};
                                ^~~
test_grupo.c:18:30: nota: (cerca de la inicialización de 'nuevo[0].nombre[0]')
test_grupo.c:18:35: aviso: la inicialización de 'char' desde 'char *' crea un entero desde un puntero sin una conversión [-Wint-conversion]
    struct new nuevo[1] = {dni, nom, apellido, numero_matricula};
                                ^~~~~~
test_grupo.c:18:35: nota: (cerca de la inicialización de 'nuevo[0].nombre[1]')
test_grupo.c:21:38: error: expected expression before ')' token
    aux = matricula_alumno(lista*);
                                ^
test_grupo.c: En la función 'matricula_multiple':
test_grupo.c:59:21: error: tipo incompatible para el argumento 1 de 'matricula_alumno'
    matricula_alumno(*(lista+i));
                        ^~~~~~
In file included from test_grupo.c:3:
grupo.h:4:26: nota: se esperaba 'int' pero el argumento es de tipo 'struct new'
int matricula_alumno(int dni);
                        ~~~~~
test_grupo.c: En la función 'main':
test_grupo.c:71:22: aviso: el paso del argumento 1 de 'inicializa_grupo' crea un entero desde un puntero sin una conversión [-Wint-conversion]
    inicializa_grupo(n);
                        ^
In file included from test_grupo.c:3:
grupo.h:3:26: nota: se esperaba 'int' pero el argumento es de tipo 'int *'
int inicializa_grupo(int n);
                        ~~~~~
test_grupo.c:72:5: aviso: declaración implícita de la función 'leer_vector' [-Wimplicit-function-declaration]
    leer_vector();
    ^~~~~~
test_grupo.c:80:5: error: expected ',', or ';' before 'struct'
    struct new* lista_dni;
    ^~~~~~
test_grupo.c:81:2: error: 'lista_dni' no se declaró aquí (primer uso en esta función); ¿quiso decir 'lista'?
    lista_dni = lista;
    ^~~~~~
    lista
test_grupo.c:81:2: nota: cada identificador sin declarar se reporta sólo una vez para cada función en el que aparece
test_grupo.c:89:5: error: expected '}' before 'lista_dni'
    lista_dni = lista3;
    ^~~~~~
test_grupo.c:88:28: nota: to match this '{'
    struct new lista3[2] = {{66666666, "Cristina", "González López"},{55555555, "Manuel", "Delacruz Quintana"},
                        ^
test_grupo.c:148:1: error: expected ',', or ';' at end of input

```

Corregimos algunos errores, pero nos quedamos con este último, que no supimos arreglar.

```

int matricula_multiple(int npersonas, struct new* lista){
    int i;
    if (plazas_libres() >= npersonas){
        for (i=0;i<npersonas;i++){
            //printf("he mandado %d de %d\n", *(lista_dni+i), i);
            matricula_alumno(*(lista+i)); // pensamos que un error puede estar en que ahí se navegan structs SOLOS, y matricula alumno pide un puntero a struct
        }
    }
    else{
        printf("No hay plazas libres para todas las personas\n");
    }
}

```

Este es el error en concreto.

```

test_grupo.c: En la función 'matricula_multiple':
test_grupo.c:61:21: error: tipo incompatible para el argumento 1 de 'matricula_alumno'
    matricula_alumno(*(lista+i)); // pensamos que un error puede estar en que ahí se nav
                        ^~~~~~

```

Los errores que arreglamos fueron:

- Le quitamos el * a lista, que se mete en aux.

- Nos dimos cuenta que el puntero no estaba definido para el else, lo sacamos por fuera del if.

```
int matricula(int dni, char* nom, char* apellido, int numero_matricula){
    struct new nuevo[1] = {dni, nom, apellido, numero_matricula};

    struct new* lista;
    lista = nuevo;

    if (plazas_libres() >= 1){
        aux = matricula_alumno(lista);
        if (aux == -2){
            printf("El alumno con DNI %d ya está matriculado\n", dni);
        }
        else{
            printf("Se ha matriculado el alumno con DNI %d en el asiento/posición %d\n", dni, aux);
        }
    }
    else {
        aux = matricula_alumno(lista);
        if (aux == -2){
            printf("El alumno con DNI %d ya está matriculado\n", dni);
        }
        else{
            printf("No hay plazas libres en el grupo\n");
        }
    }
}
```

- Nos faltó una llave y dos puntos coma.

```
printf("\n(Se procede a hacer otra matricula multiple)\n");
npersonas = 2;
struct new lista3[2] = {{66666666, "Cristina", "González López"},{55555555, "Manuel", "Delacruz Quintana"},{}};
lista_dni = lista3;
matricula_multiple(npersonas, lista_dni);
leer_vector();
```

Con structs nos quedamos hasta este punto. Por último, vamos a implementar manejo de errores, pero en el código que sí nos ejecuta (versión sin struct).

En el grupo.h:

```
extern int errno;
```

```
int inicializa_grupo(int n);
int matricula_alumno(int dni);
int desmatricula_alumno(int dni);
int testea_asiento(int asiento);
int plazas_libres();
int plazas_ocupadas();
```

En el grupo.c

```
#include <stdio.h>
#include "grupo.h"
#include <errno.h>

int main(){
    printf("Introduzca el tamaño de la lista a crear: ");
    scanf("%d", &n);

    if (n<=0 || n>50){
        printf("Código de error: %d\n", errno);
        perror("Error en el input, el número debe estar comprendido entre 1 y 50\n");
    }

    else{
        inicializa_grupo(n);
        leer_vector();
    }
}
```

Antes de ejecutar el código normal, implementamos ese código.

```
gcc -o test_grupo test_grupo.o grupo.o
[estudiante_ic2@ic2-centos-vm humilde]$ ./test_grupo
Introduzca el tamaño de la lista a crear: -6
Código de error: 0
Error en el input, el número debe estar comprendido entre 1 y 50
: Success
[estudiante_ic2@ic2-centos-vm humilde]$
```

Este es su output.

#####

VERSIÓN FINAL DEL TRABAJO

Esta es la versión entera del trabajo SIN struct (con manejo de errores, el CodeBlock expresado anteriormente es sin manejo de errores):

GRUPO.C

```
#include <stdio.h>
#define MAX 50
```

```
int *vector;
int a; // almacena n

int inicializa_grupo(int n){

    int i;
    a = n;
    vector = (int *)malloc(n * sizeof(int));
    if(a<=MAX){
        for (i=0;i<a;i++){
            vector[i] = -1;
        }
    }
    else{
        printf("Se ha excedido el tamaño permitido por la clase.\n");
        exit(1);
    }
}

void finaliza_grupo(){
    free(vector);
}

int leer_vector(){
    int i;
    for (i=0;i<a;i++){
        if (i == a-1){
            printf("(%d)\n", vector[i]);
        }
        else {
            printf("(%d),", vector[i]);
        }
    }
}
```

```
int matricula_alumno(int dni){
    int i;
    for (i=0;i<a;i++){
        if (vector[i] == dni){
            return -2;
        }
        else if (vector[i] == -1){

            vector[i] = dni;
            return i;
        }
    }
    return -1;
}

int desmatricula_alumno(int dni){
    int i;
    for (i=0;i<a;i++){
        if (vector[i] == dni){
            vector[i] = -1;
            return i;
        }
    }
    return -3;
}

int testea_asiento(int asiento){
    if (vector[asiento] == -1){
        printf("El asiento %d está vacio\n", asiento);
        return -1;
    }
    else if (asiento>a){
        printf("El asiento %d no se encuentra en el rango\n", asiento);
        return -2;
    }
    else{
        printf("La persona con DNI: %d esta sentado en el asiento: %d\n", vector[asiento], asiento);
        return vector[asiento];
    }
}
```

```
int plazas_libres(){
    int i;
    int counter = 0;
    for (i=0;i<a;i++){
        if (vector[i] == -1){
            counter += 1;
        }
    }
    return counter;
}
```

```
int plazas_ocupadas(){
    int i;
    int counter = 0;
    for (i=0;i<a;i++){
        if (vector[i] != -1){
            counter += 1;
        }
    }
    return counter;
}
```

GRUPO.H

```
extern int errno;

int inicializa_grupo(int n);
int leer_vector();
int matricula_alumno(int dni);
int desmatricula_alumno(int dni);
int testea_asiento(int asiento);
int plazas_libres();
int plazas_ocupadas();
```


TEST_GRUPO.C

```
#include <stdio.h>
#include "grupo.h"
#include <errno.h>

int* n;
int aux;

int matricula(int dni){
    if (plazas_libres() >= 1){
        aux = matricula_alumno(dni);
        if (aux == -2){
            printf("El alumno con DNI %d ya está matriculado\n", dni);
        }
        else{
            printf("Se ha matriculado el alumno con DNI %d en el
asiento/posición %d\n", dni, aux);
        }
    }
    else {
        aux = matricula_alumno(dni);
        if (aux == -2){
            printf("El alumno con DNI %d ya está matriculado\n", dni);
        }
        else{
            printf("No hay plazas libres en el grupo\n");
        }
    }
}
```

```

int desmatricula(int dni){
    if (plazas_ocupadas() >= 1){
        aux = desmatricula_alumno(dni);
        if (aux != -3){
            printf("El alumno con DNI %d y posición %d ha sido
desmatriculado del grupo\n", dni, aux);
        }
        else{
            printf("No se ha encontrado el DNI solicitado en la
lista\n");
        }
    }
    else{
        printf("No hay alumnos matriculados en el grupo\n");
    }
}

int matricula_multiple(int npersonas, int* lista_dni){
    int i;
    if (plazas_libres() >= npersonas){
        for (i=0;i<npersonas;i++){
            //printf("he mandado %d de %d\n", *(lista_dni+i), i);
            matricula_alumno(*(lista_dni+i));
        }
    }
    else{
        printf("No hay plazas libres para todas las personas\n");
    }
}

```

```

int main(){

    printf("Introduzca el tamaño de la lista a crear: ");
    scanf("%d", &n);

    if (n<=0 || n>50){
        printf("Código de error: %d\n", errno);
        perror("Error en el input, el número debe estar comprendido entre 1
y 50\n");
    }

    else{

        inicializa_grupo(n);
        leer_vector();

        printf("\n(Se intenta desmatricular cuando no hay nadie
matriculado)\n");
        desmatricula(38808158);

        printf("\n(Se procede a hacer una matricula multiple)\n");
        int npersonas = 3;
        int lista[3] = {38808158, 17598594, 51489657};
        int* lista_dni;
        lista_dni = lista;
        matricula_multiple(npersonas, lista_dni);
        leer_vector();

        printf("\n(Se procede a hacer otra matricula multiple)\n");
        npersonas = 5;
        int lista3[5] = {99808158, 99598594, 99489657, 99857465, 35248697};
        lista_dni = lista3;
        matricula_multiple(npersonas, lista_dni);
        leer_vector();

        printf("\n(Se mete un matriculado nuevo(solo uno))\n");
    }
}

```

```

matricula(59317174);
leer_vector();

printf("\n(Se intenta matricular a alguien ya matriculado)\n");
matricula(59317174);
leer_vector();

printf("\n(Se desmatricula el primero)\n");
desmatricula(38808158);
leer_vector();

printf("\n(Se intenta desmatricular a alguien no matriculado)\n");
desmatricula(38808158);
leer_vector();

printf("\n(Se procede a hacer una matricula multiple, que sobrepase
el límite)\n");
npersonas = 8;
int lista2[8] = {38805658, 17863894, 54615168, 58394884, 65847219,
63514298, 98426571, 36951753};
lista_dni = lista2;
matricula_multiple(npersonas, lista_dni);
leer_vector();

/*
printf("\n(Se matricula a uno solo para luego comprobar que ocurre
si esta todo ocupado)\n");
matricula(38808158);
leer_vector();
*/

printf("\n");

/*
printf("\n(Ahora se matricula a uno nuevo para ver si da ese
error)\n");
matricula(12356847);
leer_vector();
*/

//Este caso se cumple en casos concretos

testea_asiento(11);
testea_asiento(28);
testea_asiento(1);

```

```

printf("\n");

finaliza_grupo();
/*
Mirar porque cuando liberamos el vector y luego lo leemos no nos sale
vacío
y nos devuelve algo raro.
*/
leer_vector();
printf("\n");

printf("Ahora creamos otro grupo: ");
scanf("%d", &n);

if (n<=0 || n>50){
    printf("Código de error: %d\n", errno);
    perror("Error en el input, el número debe estar comprendido entre 1
y 50\n");
}

else{
inicializa_grupo(n);
leer_vector();
}
}
}

```

En la entrega, tienes nuestros ficheros con la versión más avanzada con structs que pudimos realizar, pero tienes el código de la versión non struct más actualizado arriba porque no nos corre el código con structs.

Versión del trabajo con structs (lo más avanzada posible):

GRUPO.C

```
#include <stdio.h>
#define MAX 50
#include <string.h>

int *vector;
int a; // almacena n
int vecaux[MAX];

int inicializa_grupo(int n){
    int i;
    a = n;
    vector = (int *)malloc(n * sizeof(int));
    if(a<=MAX){
        for (i=0;i<a;i++){
            vector[i] = -1;
        }
    }
    else{
        printf("Se ha excedido el tamaño permitido por la clase.\n");
        exit(1);
    }
}

void finaliza_grupo(){
    free(vector);
}

int leer_vector(){
    int i;
    for (i=0;i<a;i++){
        if (i == a-1){
            printf("(%d)\n", vector[i]);
        }
        else {
            printf("(%d),", vector[i]);
        }
    }
}

// LAS DE ARRIBA SE QUEDAN IGUALES
```

```
int matricula_alumno(lista*){
    int i;
    for (i=0;i<a;i++){
        if (vector[i] == (lista*).dni){
            return -2;
        }
        else if (vector[i] == -1){
            vector[i] = {(lista*).dni, (lista*).nombre};
            vecaux[i] = (lista*);
            vecaux[i].numero_matricula = i;
            return i;
        }
    }
    return -1;
}

char* desmatricula_alumno(int dni){
    int i;
    char str[];
    for (i=0;i<a;i++){
        if (vecaux[i].dni == dni){
            str = vecaux[i].nombre;
            strcat(str, " ");
            strcat(str, vecaux[i].apellidos);
            vector[i] = -1;
            return str;
        }
    }
    return -3;
}
```

```
int testea_asiento(int asiento){
    if (vector[asiento] == -1){
        printf("El asiento %d está vacio\n", asiento);
        return -1;
    }
    else if (asiento>a){
        printf("El asiento %d no se encuentra en el rango\n", asiento);
        return -2;
    }
    else{
        printf("La persona con DNI: %d esta sentado en el asiento: %d\n", vector[asiento], asiento);
        return vector[asiento];
    }
}

int plazas_libres(){
    int i;
    int counter = 0;
    for (i=0;i<a;i++){
        if (vector[i] == -1){
            counter += 1;
        }
    }
    return counter;
}

int plazas_ocupadas(){
    int i;
    int counter = 0;
    for (i=0;i<a;i++){
        if (vector[i] != -1){
            counter += 1;
        }
    }
    return counter;
}
```


GRUPO.H

```
extern int errno;

int inicializa_grupo(int n);
int matricula_alumno(int dni);
int leer_vector();
int desmatricula_alumno(int dni);
int testea_asiento(int asiento);
int plazas_libres();
int plazas_ocupadas();
```

TEST_GRUPO.C

```
#include <stdio.h>
#include "grupo.h"
#include <string.h>

int* n;
int aux;
struct new {
    int dni;
    char nombre[10];
    char apellidos[20];
    int numero_matricula;
};

int matricula(int dni, char* nom, char* apellido, int
numero_matricula){
    struct new nuevo[1] = {dni, nom, apellido, numero_matricula};

    struct new* lista;
    lista = nuevo;

    if (plazas_libres() >= 1){

        aux = matricula_alumno(lista);
        if (aux == -2){
            printf("El alumno con DNI %d ya está matriculado\n", dni);
        }
        else{
```

```

        printf("Se ha matriculado el alumno con DNI %d en el
asiento/posición %d\n", dni, aux);
    }
}
else {
    aux = matricula_alumno(lista);
    if (aux == -2){
        printf("El alumno con DNI %d ya está matriculado\n", dni);
    }
    else{
        printf("No hay plazas libres en el grupo\n");
    }
}
}

int desmatricula(int dni){
    if (plazas_ocupadas() >= 1){
        aux = desmatricula_alumno(dni);
        if (aux != -3){
            printf("El alumno con DNI %d y posición %d ha sido
desmatriculado del grupo\n", dni, aux);
        }
        else{
            printf("No se ha encontrado el DNI solicitado en la
lista\n");
        }
    }
    else{
        printf("No hay alumnos matriculados en el grupo\n");
    }
}

int matricula_multiple(int npersonas, struct new* lista){
    int i;
    if (plazas_libres() >= npersonas){
        for (i=0;i<npersonas;i++){
            //printf("he mandado %d de %d\n", *(lista_dni+i), i);
            matricula_alumno(*(lista+i)); // pensamos que un error
puede estar en que ahí se navegan structs SOLOS, y matricula alumno
pide un puntero a struct
        }
    }
    else{
        printf("No hay plazas libres para todas las personas\n");
    }
}

```

```

    }
}

int main() {

    printf("Introduzca el tamaño de la lista a crear: ");
    scanf("%d", &n);

    if (n<0){
        printf("%d", errno);
        perror("Error en el input: El número debe ser positivo");
    }

    else if (n>50){
        printf("%d", errno);
        perror("Error en el input: El número debe ser menor de 50");
    }

    else{

        inicializa_grupo(n);
        leer_vector();

        printf("\n(Se intenta desmatricular cuando no hay nadie
matriculado)\n");
        desmatricula(38808158);

        printf("\n(Se procede a hacer una matricula multiple)\n");
        int npersonas = 3;
        struct new lista[3] = {{38808158, "Ayoze", "Ruano
Alcántara",},{59874632, "Nesta", "Arteaga Cabrera",},{45179111,
"Enrique", "Reina Hernández",}};
        struct new* lista_dni;
        lista_dni = lista;
        matricula_multiple(npersonas, lista_dni);
        leer_vector();
    }
}

```

```

printf("\n(Se procede a hacer otra matricula multiple)\n");
npersonas = 2;
struct new lista3[2] = {{66666666, "Cristina", "Gonzalez
López",},{55555555, "Manuel", "Delacruz Quintana",}};
lista_dni = lista3;
matricula_multiple(npersonas, lista_dni);
leer_vector();

printf("\n(Se mete un matriculado nuevo(solo uno))\n");
matricula(59317174, "Isidro", "Velázquez Duarte", 0);
leer_vector();

printf("\n(Se intenta matricular a alguien ya matriculado)\n");
matricula(59317174, "Isidro", "Velázquez Duarte", 0);
leer_vector();

printf("\n(Se desmatricula el primero)\n");
desmatricula(38808158);
leer_vector();

printf("\n(Se intenta desmatricular a alguien no matriculado)\n");
desmatricula(38808158);
leer_vector();

/*
printf("\n(Se procede a hacer una matricula multiple, que sobrepase
el límite)\n");
npersonas = 8;
int lista2[8] = {38805658, 17863894, 54615168, 58394884, 65847219,
63514298, 98426571, 36951753};
lista_dni = lista2;
matricula_multiple(npersonas, lista_dni);
leer_vector();
*/
/*
printf("\n(Se matricula a uno solo para luego comprobar que ocurre
si esta todo ocupado)\n");
matricula(38808158);
leer_vector();
printf("\n");
*/
/*

```

```

        printf("\n(Ahora se matricula a uno nuevo para ver si da ese
error)\n");
        matricula(12356847);
        leer_vector();
    */
    //Este caso se cumple en casos concretos

    testea_asiento(11);
    testea_asiento(28);
    testea_asiento(1);
    printf("\n");

finaliza_grupo();
/*
Mirar porque cuando liberamos el vector y luego lo leemos no nos sale
vacío
y nos devuelve algo raro.
*/
leer_vector();
printf("\n");

printf("Ahora creamos otro grupo: ");
scanf("%d", &n);
inicializa_grupo(n);
leer_vector();

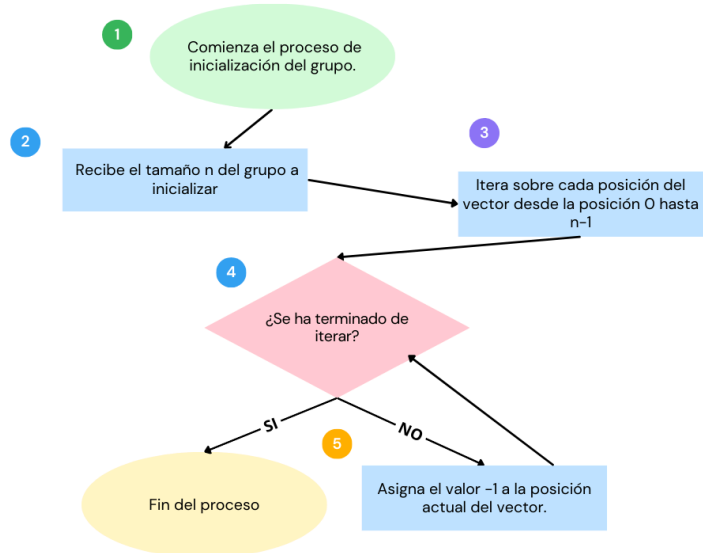
}
}

```

DIAGRAMAS DE FLUJO

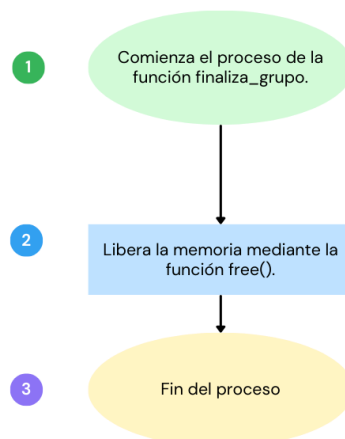
int inicializa grupo(int n)

Pone todas las posiciones del vector a -1



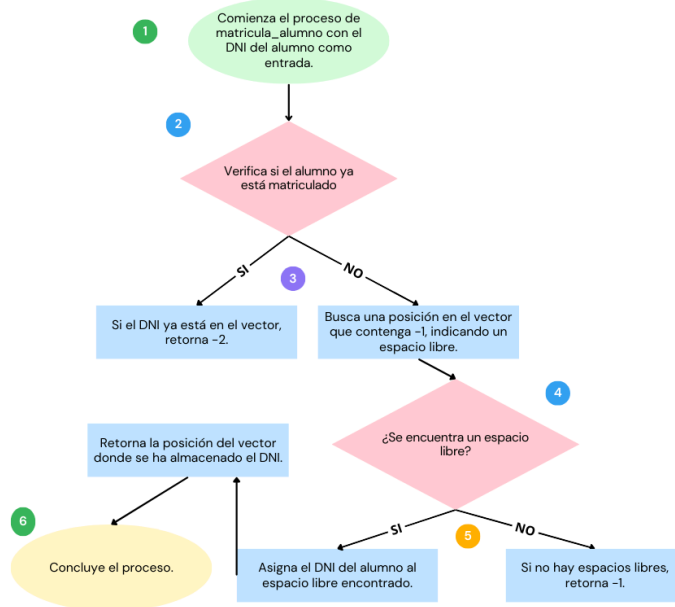
void finaliza_grupo()

En el caso de memoria dinámica, tiene que liberar dicha memoria.



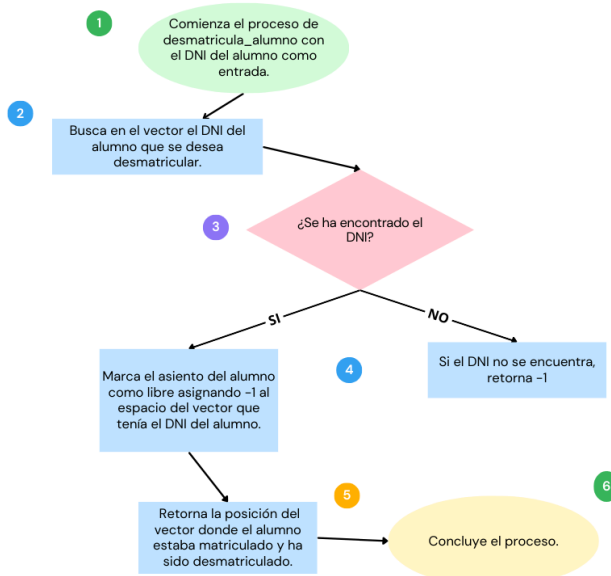
int matricula_alumno(int dni)

Busca asignarle una plaza al alumno



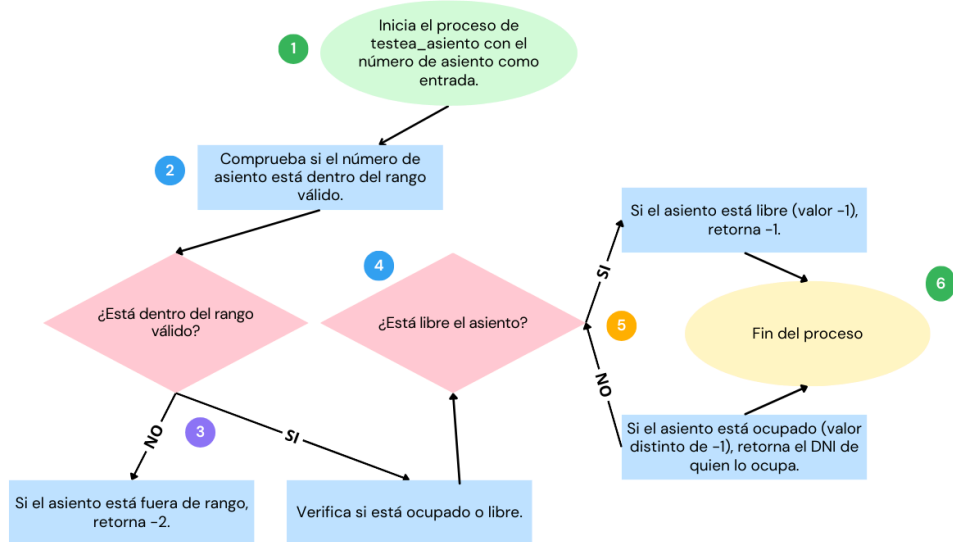
int desmatricula_alumno(int dni)

Liberar un asiento en un grupo usando el DNI de la persona, devolviendo la posición del asiento.



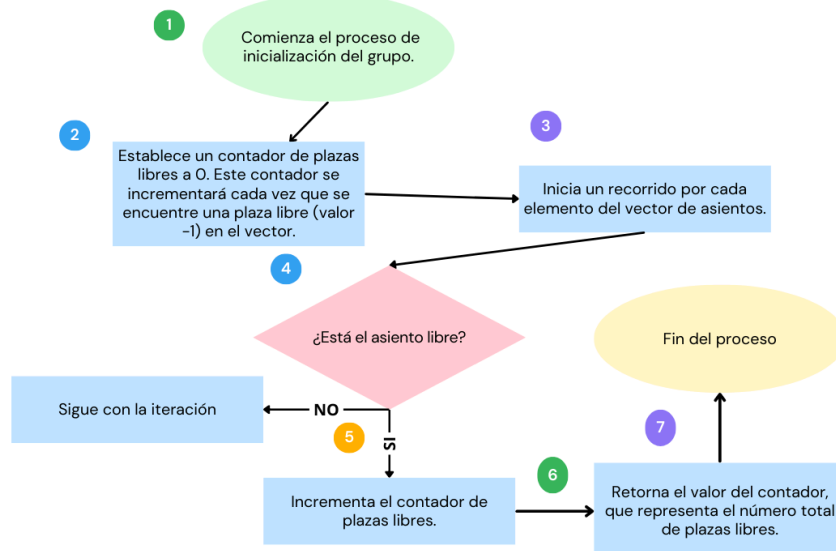
int testea_asiento(int asiento)

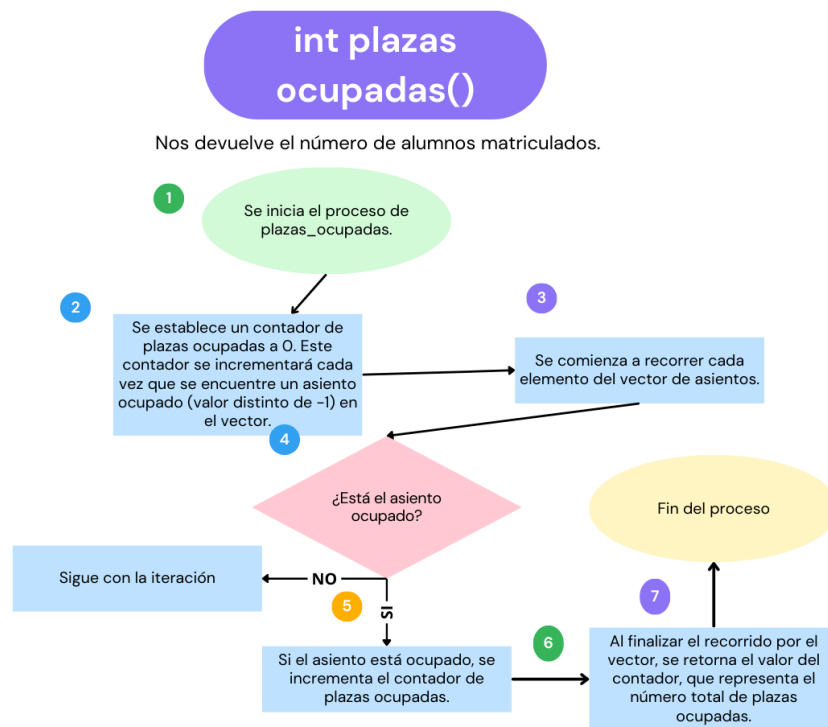
Consultar el estado de un asiento por su número, devolviendo el DNI del ocupante, -1 si está libre, o -2 si está fuera de rango.



int plazas_libres()

Devuelve el número de plazas libres.





REPARTO DEL TRABAJO:

Ayoze: Diseñó la interfaz entera (funciones y operaciones), básicamente el grupo.c, grupo.h y Makefile enteros; y test_grupo.c HASTA justo antes del main(). *Todo esto tanto para la versión con struct como para la versión non struct.* Con interfaz entera, se refiere al algoritmo y diseño de cada función.

Enrique: Ayudó en la gran mayoría de errores que se iban encontrando, tanto en la versión con structs como sin structs. Comentó una gran parte del código en la memoria (documento del trabajo). Hizo el main() (programa de prueba / casos de prueba) en ambos casos (struct vs non struct).

Nesta: Memoria dinámica, tratamiento de errores, gráficos de diagramas de flujo, aportación de ideas durante todo el proceso, detección de algunos errores, etc.