75.40 Algoritmos y Programación I Curso 4
Introducción a los Arreglos y Punteros

Dr. Mariano Méndez[1] and Correcciones: Ignacio Jordan, Martin Dardis[1]

[1]Facultad De Ingeniería. Universidad de Buenos Aires

24 de marzo de 2020

1. Punteros: Un Cuco... pero no tanto

Como se ha comentado anteriormente una variable es un espacio de memoria que se identifica mediante un nombre o identificador único por el cual se hace referencia a ella. Ampliando un poco más este concepto, podemos decir que en realidad la memoria está estructurada como un conjunto de celdas distribuidas linealmente, cada una de estas celdas posee una dirección única dentro de este conjunto, esta dirección se denomina **dirección de memoria** ver Figura 2.

El nombre de una variable es el equivalente humano al nombre de pila. Las computadoras no necesitan referirse a nombres pues con solo referenciar a un número que representa la dirección de memoria de una celda dentro de la estructura de la memoria pueden acceder a la información almacenada en la misma. Escribir programas que hagan directamente referencia a las direcciones de memoria de una computadora sería humanamente muy complejo, tedioso y acarrearía otros tipos de problemas que no vienen al caso. Aunque parezca mentira inicialmente con las primeras computadoras se escribían los programas referenciando a las direcciones de memoria directamente, esto resultaba en programas muy complejos de entender por ello en la actualidad, se utilizan nombres que posteriormente el compilador reemplaza por direcciones de memoria.

En muchos lenguajes de programación existe un tipo de dato que almacena exclusivamente direcciones de memoria, este tipo de dato se denomina **puntero**, ver Figura 1. Cabe destacar que la utilización de punteros agrega un nivel de complejidad al programa que se está desarrollando, ya que el programador, en el caso de C, debe controlar su uso. Como con todos los tipos de datos en C se puede declarar una variable tipo puntero.

En el caso específico de los punteros, la declaración se realiza de la siguiente forma:

```
tipo_de_dato_al_que_apuntar * nombre_del_puntero;
```

Concretamente :

```
int    *ptr_entero;      //un puntero a un entero
double *ptr_double;      //un puntero a un double
char   *ptr_char;        //un puntero a un caracter
float  *ptr_float;       //un puntero a un float
```

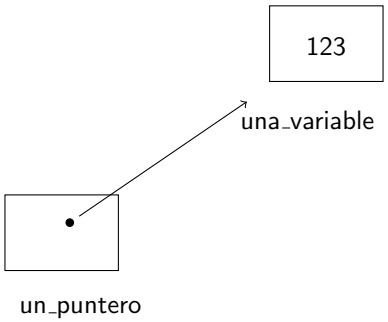


Figura 1: Esquema del concepto de Puntero

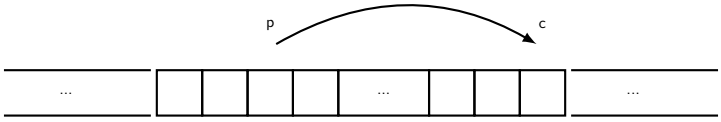


Figura 2: Esquema de distribución de la memoria

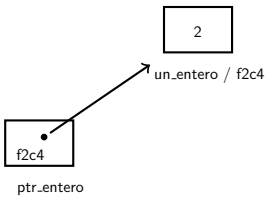


Figura 3: Diagrama del ejemplo

1.1. Operador de Dirección

El operador de dirección es un operador unario en C que devuelve la dirección del operando, este corresponde a "&".

```
int     un_entero=2;
int     *ptr_entero;              //un puntero a un entero

ptr_entero= &un_entero;
```

En el código escrito anteriormente se definen dos variables, una entera y otra que es un puntero a un entero. En la asignación, el operador de dirección le asigna la dirección de memoria en la cual esta definida la variable un_entero a ptr_entero.

1.2. Operador de Indirección

El operador de indirección devuelve el valor del objeto hacia el cual su operando apunta, dicho operando debe ser un puntero.

```
int     un_entero=2;
int     *ptr_entero;              //un puntero a un entero

ptr_entero= &un_entero;
printf("d",*ptr_entero);
```

¿Qué debería mostrar el código anterior? La respuesta correcta es 2. Veamos otro ejemplo

```
# include <stdio.h>


/*En el siguiente ejemplo se imprimira por pantalla el valor de la variable NUMERO_ENTERO y su
     poscion en la memoria*/
int main (){

    int numero_entero;
    int *puntero_a_numero_entero;

    numero_entero=7;
    puntero_a_numero_entero=&numero_entero;
    //puntero_a_numero_entero guarda la direccion de memoria de numero_entero.

```

```
    printf(" La direccion de memoria de NUMERO_ENTERO es %p \n ",&numero_entero);
    printf(" El valor de PUNTERO_A_NUMERO_ENTERO es %p \n ",puntero_a_numero_entero);

    /*Aqui se imprime en pantalla LA DIRECCION DE MEMORIA de la variable numero_entero, que
    esta almacenada en puntero_a_numero_entero.*/

    printf(" El valor de NUMERO_ENTERO  es %d \n " ,numero_entero);
    printf(" El contenido de la direccion de memoria apuntada por PUNTERO_A_NUMERO_ENTERO es %
    d ",*puntero_a_numero_entero);

    /*Aqui se imprime en pantalla el CONTENIDO de la variable a la que apunta
    puntero_a_numero_entero,es decir, el contenido de numero_entero.*/
return 0;
}
```

```
~/D/A/U/7/material ð 24_Introduccion_a_vectores ✎ ? apuntes_teoricos/4-Introducci
La direccion de memoria de NUMERO_ENTERO es 0x7ffe8b9b739c
 El valor de PUNTERO_A_NUMERO_ENTERO es 0x7ffe8b9b739c
 El valor de NUMERO_ENTERO  es 7
 El contenido de la direccion de memoria apuntada por PUNTERO_A_NUMERO_ENTERO es 7 ⏎
~/D/A/U/7/material ð 24_Introduccion_a_vectores ✎ ? apuntes_teoricos/4-Introducci
```

Figura 4: Salida por pantalla

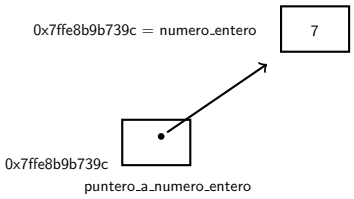


Figura 5: Diagrama del ejemplo

2. Pasaje de Parámetros por Referencia

Se ha explicado que en C sólo es posible pasar los parámetros por valor, y esta afirmación sigue siendo verdadera, entonces cómo se hace cuando se necesita pasar algún parámetro y que cualquier modificación hecha en la función que recibe el parámetro se vea plasmada o reflejada en la función llamadora, es decir pasar el parámetro por referencia. En C es posible utilizando el concepto de puntero. Si implementamos la siguiente función de intercambio de dos enteros:

```
#include <stdio.h>

void intercambio(int valor1, int valor2){
        int aux;

        aux = valor1;
        valor1=valor2;
        valor2=aux;
}


int main(){
        int un_numero,otro_numero;

        un_Numero=1;
        otro_Numero=2;

        printf("Antes de llamar a intercambio un_numero = %d , otro_numero=%d",un_numero,
    otro_numero);
        intercambio(un_numero,otro_numero);
        printf("Despues de llamar a intercambio un_numero = %d , otro_numero=%d",un_numero,
    otro_numero);
        return 0;
}
```

```
~/D/A/U/7/material ð 24_Introduccion_a_vectores ✎ ? apuntes_t
Antes de llamar a intercambio un_Numero = 1  otro_Numero=2
Despues de llamar a intercambio un_Numero = 1  otro_Numero=2
~/D/A/U/7/material ð 24_Introduccion_a_vectores ✎ ? apuntes_t
```

Figura 6: Salida por pantalla

tras compilar y ejecutar el programa propuesto, veremos sin asombro que el los valores impresos son :

```
1         un_numero=1
2         otro_numero=2
```

Ahora bien, ¿Cómo se plantea el programa si realmente se quiere intercambiar los valores de las variables? En realidad para que el pasaje de parámetros resulte por referencia, se debe implementar la función *intercambio()* de la siguiente forma:

Puntero a int

```
1
2 void intercambio( int *  valor1, int  * valor2){
3         int aux;
4
5         aux = * valor1;
6         *valor1=*valor2;
7         *valor2=aux;
8 }
```

El contenido de lo apuntado por

y la llamada a la función debe realizarse :

```
1 intercambio(&un_numero,&otro_numero);
```

Es decir, los parámetros formales de la función son punteros a dos enteros, y los parámetros actuales de la llamada son las direcciones de memoria de las variables que se desea pasar por referencia, ver Figura 7

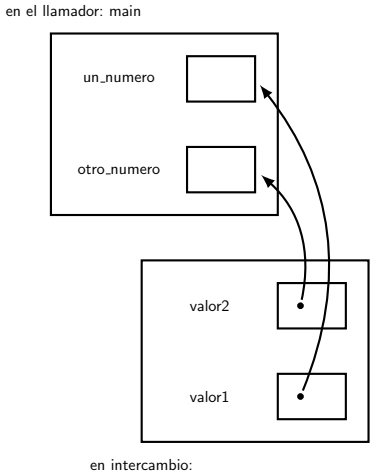


Figura 7: Diagrama ejemplo de pasaje de parámetros por referencia

Teniendo en cuenta ese cambio

```
~/D/A/U/7/material ð 24_Introduccion_a_vectores ✎ ? apuntes_teoric
Antes de llamar a intercambio un_Numero = 1  otro_Numero=2
Despues de llamar a intercambio un_Numero = 2  otro_Numero=1
~/D/A/U/7/material ð 24_Introduccion_a_vectores ✎ ? apuntes_teoric
```

captionsalida por pantalla

3. Constantes

Una constante no es nada más que una versión escrita o nombrada de un determinado valor que se utiliza en el programa. En ANSI C se pueden definir valores constantes mediante la utilización de la palabra reservada *const* de

la siguiente forma:

```
int const Pi = 3.1416;
const int Base =2;
```

¿Para qué sirve utilizar constantes si podemos escribir el valor por todo el programa? De hecho no es necesario usar constantes, pues los valores escritos directamente en el programa pueden ir tranquilamente. La utilización de constantes es una práctica de buena programación, por dos motivos:

1. Hacen más fácil la aplicación de cambios al programa.
2. Hacen más comprensible y auto-descriptivo el código fuente.
3. Ayudan al mantenimiento del código fuente.

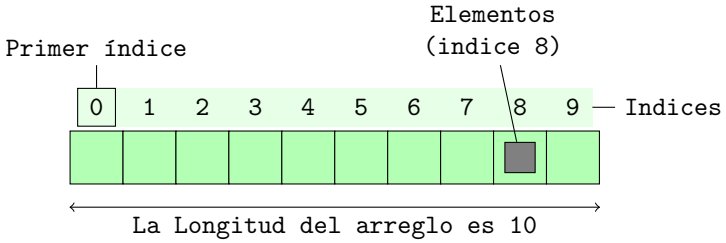
Otra forma de definir valores constantes, es mediante el uso del preprocesador de C. Es posible utilizar la directiva #define, a diferencia de utilizar la palabra reservada el preprocedador de C reemplazará el valor de la directiva por el nombre asignado.

```
#define TRUE 1
#define FALSE 0

....
....

if (es_primo==TRUE) ...
```

4. Vectores

ANSI C proporciona un tipo de dato estructurado capaz de almacenar una colección secuencial y finita de elementos del mismo tipo de dato. A este tipo de dato se lo denomina **arreglo** o **vector unidimensional**.

La idea detrás de los vectores consiste en que en vez de declarar n variables, por ejemplo, medicion1, medicion2, medicion3 , hasta medicion99 se declare una única variable llamada mediciones, que pueda almacenar todos los valores que sean necesarios (cuando se refiere a todos los valores necesarios no hay que olvidar que este valor DEBE ser finito).



Una vez declarada la variable de tipo vector, se podrán acceder a sus elementos mediante la utilización de un subíndice que corresponde al i-ésimo elemento del vector.

Para declarar un vector en C, el programador debe especificar el tipo de dato de los componente del vector (por definición, todos son del mismo tipo) y el número de elementos del vector o tamaño:

```
tipo_de_dato nombre_del_vector [ tamaño_del_vector ];
```

Algunos ejemplo de declaraciones de vectores :

```

int main(){

    int const CANTIDAD_MAXIMA_TEMP=50;
    int const DIAS_DEL_MES=31;
    int const MESES_DEL_ANIO=12;


    int mediciones[6];   //vector de 6 elementos enteros
    float temperaturas[CANTIDAD_MAXIMA_TEMP];
    float temperaturas_diarias[MESES_DEL_ANIO];
```

```
    char  palabra[12]; // vector de 12 caracteres

}
```

Nota:Las operaciones entre vectores **deben hacerse elemento por elemento**. No es posible en C poder asignar un vector a otro, ni sumar un escalar a un vector en forma automática.

Nota2: Los vectores se definen con una longitud fija y finita que *NO PUEDE SER CAMBIADA* a lo largo del programa.

Nota3: Los vectores o arreglos de caracteres se denominan Strings y en C tienen un tratamiento especial.

Nota4: Los vectores en C al igual que muchos tipos de variables pueden inicializarse en la misma línea en donde se declaran, para ello es necesario enumerar uno a uno los elementos del vector:

```
int mediciones[6]= {12, 23, 34, 45, 56, 67 };
```

Caso especial 1, inicialización de todos los elementos de un vector con el mismo valor:

```
int mediciones[6]= {0};
```

Dado que en C, si la lista de inicializadores es menor a la cantidad total de elementos del vector, aquellos elementos que no tengan inicializador serán automáticamente inicializados en 0. Ojo!!, este tipo de reglas no es igual en todos los lenguajes de programación, por lo tanto se desaconseja utilizarla.

Caso especial 2, inicialización de un arreglo al que no se le ha definido longitud:

```
int list_de_enteros[]= {5, 4, 3, 2, 1};
```

En este caso el arreglo tomará la longitud, fija, de la cantidad de elementos con inicializadores, en este caso la longitud será 5.

4.1. Operaciones Básicas con Vectores Unidimensionales

para esta sección se utilizará siempre el mismo tipo de datos, que será declarado a continuación:

```
int const MAXIMO = 10;
int vector[MAXIMO];
```

4.1.1. Inicializar

A continuación se realizará una de las operaciones clásicas con vectores que es inicializar todos los elementos del mismo. Para ello la forma de proceder correcta es la de pensar que para inicializar un vector debe hacerse lo **elemento por elemento**. Una posible forma sería creando una función:

```
void inicializar_vector( int vector[] ){
    int index;

    for(index=0; index<MAXIMO; index++) vector[index ]=0;
}
```

index=0,1,2,3...,MAXIMO-1

Cabe destacar que la realización de dicha operación está sujeta a la utilización de una estructura de control **iterativa**. Otra versión de la misma función podría llegar a ser :

```
void inicializar_vector( int vector[] ){
    int index;

    index=0;
    while(index<MAXIMO){
        vector[index]=0;
        index++;
    }
}
```

De la misma Forma podría realizarse una tercera función con la última estructura de control iterativa del lenguaje C:

```
void inicializar_vector( int vector[] ){
    int index;

    index=0;
    do{
        vector[index]=0;
        index++;
   }while(index<MAXIMO);
}
```

Nota: Cabe destacar que la inicialización puede realizarse de varias formas y con diversos valores.

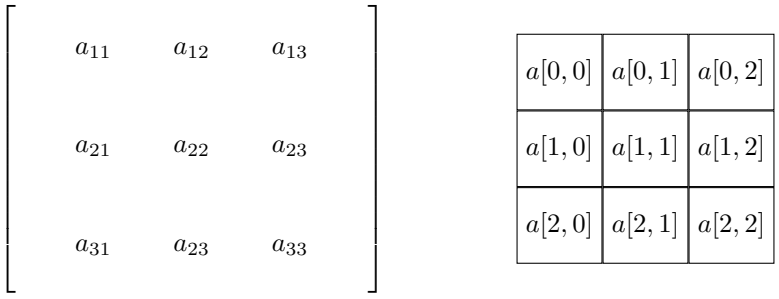


Figura 8: Una matriz de 3 filas y 3 columnas

4.1.2. Asignar

Esta es otra operación muy utilizada cuando se realizan algoritmos con vectores. Consiste en asignar valores a los elementos de un vector (podría verse común caso más general que el de la inicialización). A continuación se realizara una asignación entre los elementos de dos vectores:

```
void inicializar_vector( int un_vector[], int otro_vector[] ){
    int index;

   for(index=0; index<MAXIMO; index++)
       un_vector[index ] = un_vector[index ];
}
```

index=0,1,2,3,...,MAXIMO-1

4.2. Arreglos Multidimensionales

La primera aproximación a un arreglo multi-dimensional la tenemos en el concepto algebraico de matriz, en la Figura 8 puede observarse el concepto matemático dibujado a la izquierda y la representación computacional dibujada a la derecha.

La declaración de un arreglo bidimensional en este caso se realiza como se describe a continuación:

```
int matriz[3][3];
```

Cabe destacar que para acceder al contenido de un elemento del arreglo bidimensional, necesitamos dos coordenadas i,j. Por ejemplo, si se desea imprimir el valor almacenado en la fila 2 columna 3 de la matriz, se deberá escribir:

```
   printf("%d", matriz[1,2]);
```

Asimismo también las matrices pueden ser inicializadas en su declaración :

```
int matriz[3][4] = {
{0, 1, 2, 3} ,   /*  initializers for row indexed by 0 */
{4, 5, 6, 7} ,   /*  initializers for row indexed by 1 */
{8, 9, 10, 11}   /*  initializers for row indexed by 2 */
};
```

En realidad no es necesario escribirlo de esta forma, también es posible inicializarla de la siguiente forma:

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

En C es posible declarar un arreglo de más de dos dimensiones, por ejemplo de tres dimensiones, ver Figura 9 esto se debe declarar de la siguiente forma:

4.3. Vectores, Punteros y Pasaje de Vectores como parámetros

En el lenguaje de Programación C existe una muy estrecha relación entre los vectores y los punteros [**?**, **?**]. A continuación se muestra la relación entre los vectores y los punteros en C:

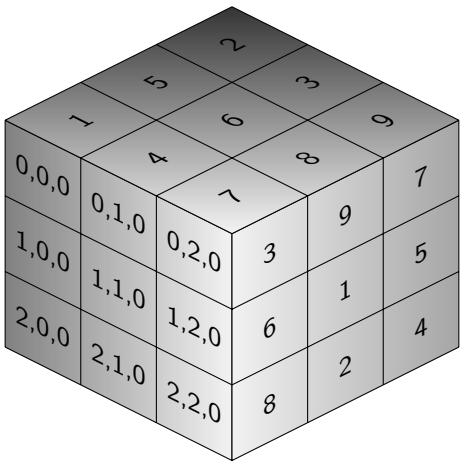


Figura 9: Una matriz de tres dimensiones

```
int     vector[10] = "12, 23, 34, 45, 56, 67, 78, 89, 90, 100";
int     *ptr_entero;
int     un_valor_entero;
```

Las siguientes expresiones son equivalentes:

ptr_entero= &vector[0]	ip=vector
un_valor_entero=*ptr_entero	*ptr_entero=un_valor_entero
*(ptr_entero+1)	v[1]
ptr_entero+i	&vector[i]

4.4. Pasaje de Vectores por Parámetros

En primer lugar, cuando se necesita definir un parámetro formal como un vector, se lo define de la siguiente forma:

```
int sumar_elementos_de_vector( int un_vector[] ){
        \*Operaciones*\
}
```

No es necesario poner la longitud del vector que se espera como parámetro. Como se ha visto en las igualdades anteriores la llamada a la función sumar puede ser escrita como :

```
 int suma=sumar_elementos_de_vector(&vector[0]);
 int suma=sumar_elementos_de_vector(vector);
```

esta última es la más utilizada. Como se puede apreciar los vectores o arreglos (cualquiera sea su dimensión) SIEMPRE se pasan por referencia.

5. Un Ejemplo Completo del Uso de Vectores

Dados dos arreglos cuyos elementos contienen valores enteros, realizar un programa en C que solucione los siguientes apartados:

1. Cargue ambos vectores con valores ingresados por el usuario
2. Calcule el vector suma
3. Calcule el vector resta
4. Calcule el producto escala entre ambos vectores
5. Calcule el módulo

```c
#include<stdio.h>
#include<math.h>

#define LONGITUD  10

void sumar(int vector_operando_uno[], int vector_operando_dos[], int vector_suma[]);
void restar(int vector_operando_uno[], int vector_operando_dos[], int vector_restar[] );
long producto_escalar (int vector_operando_uno[], int vector_operando_dos[]);
void leer_vector(int un_vector[]);
double modulo(int vector_operando_uno[]);
void mostrar_vector(int un_vector[]);

int main(){
    int un_vector[LONGITUD];
    int otro_vector[LONGITUD];
    int resultado[LONGITUD]={0};

    printf("Ingresar los datos del primer vector\n");
    leer_vector(un_vector);
    printf("Ingresar los datos del segundo vector\n");
    leer_vector(otro_vector);

    sumar(un_vector,otro_vector,resultado);
    mostrar_vector(resultado);
    restar(un_vector,otro_vector,resultado);
    mostrar_vector(resultado);
    printf("el producto escalar de los dos Vectores es : %ld\n", producto_escalar(un_vector,
    otro_vector) );
    printf("el modulo del primer vector es : %f\n", modulo(un_vector) );
    printf("el modulo del segundo vector es : %f\n",modulo(otro_vector) );
}

void sumar (int vector_operando_uno[], int vector_operando_dos[], int vector_suma[]){
    int i;
    for (i=0;i<LONGITUD;i++)
        vector_suma[i]=vector_operando_uno[i]+vector_operando_dos[i];
}

void restar (int vector_operando_uno[], int vector_operando_dos[], int vector_restar[]){
    int i;
    for (i=0;i<LONGITUD;i++)
        vector_restar[i]=vector_operando_no[i]-vector_perando_dos[i];
}

long producto_escalar (int vector_operando_uno[], int vector_operando_dos[]){
    int i;
    long sumatoria=0;
    for (i=0;i<LONGITUD;i++)
        sumatoria+=vector_operando_uno[i]*vector_operando_dos[i];
    return sumatoria;
}

void leer_vector(int un_vector[]){
    int i;
    for (i=0;i<LONGITUD;i++){
        printf("Vector[%d] = ",i);
        scanf("%d",&un_vector[i]);
    }
}

double modulo(int operando_uno[]){
    int i;
    double suma=0;
    for(i=0;i<LONGITUD;i++){
        suma+=operando_uno[i]*operando_uno[i];
    }
    return sqrt(suma);
}

void mostrar_vector(int un_vector[]){
    int i;
    for (i=0;i<LONGITUD;i++)
    printf("%d",un_vector[i] );
}
```

¿Cuál es la salida de este programa? ¿Que cosas mejoraría?

6. Simulación de Vectores de Longitud Variable

En C como en muchos otros lenguajes de programación no es posible declarar vectores que puedan variar la cantidad de elementos que almacenan. Por ello se dice que los vectores son tipos de datos estáticos, pues no se puede alterar su tamaño (en bytes) en tiempo de ejecución del programa.

De todas formas es muy común que sea necesario, dadas las características del problema que se necesite, variar la cantidad de elementos que contiene un vector en tiempos de ejecución ... Entonces, ¿Cómo lo hacemos si el lenguaje no lo permite? La respuesta es sencilla, se define un vector de una longitud conocida, a priori, que dentro del dominio del problema nunca podrá ser superada. En otras palabras, se define un vector muy grande. Y además, se utiliza una variable para almacenar la información del tamaño o tope utilizados del vector, ver Figura 10.

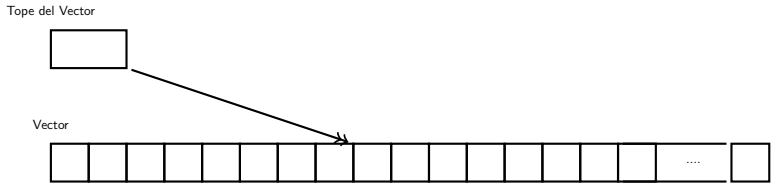


Figura 10: Uso acotado del tamaño de un vector

El par vector tope siempre van juntos durante la ejecución del programa. Teniendo en cuenta ahora esta forma de simular vectores de longitud variable o de tamaño variable, podemos modificar el ejemplo anterior para que el usuario seleccione la longitud de los dos vectores:

6.1. Ejemplo Completo de vectores de longitud variable

Dados dos arreglos cuyos elementos contienen valores enteros, y cuya dimensión se ingresa por teclado, realizar un programa en C que solucione los siguientes apartados:

1. Cargue ambos vectores con valores ingresados por el usuario
2. Calcule el vector suma
3. Calcule el vector resta
4. Calcule el producto escala entre ambos vectores
5. Calcule el modulo

```c
#include<stdio.h>
#include<math.h>

#define LONGITUD  1000

void sumar(int vector_operando_uno[], int vector_operando_dos[], int vector_suma[], int tope);
void restar(int vector_operando_uno[], int vector_operando_dos[], int vector_restar[], int tope );
long producto_escalar (int vector_operando_uno[], int vector_operando_dos[], int tope);
void leer_tope(int *tope);
void leer_vector(int un_vector[], int tope);
double modulo(int vector_operando_uno[], int tope);
void mostrar_vector(int un_vector[], int tope);

int main(){
    int un_vector[LONGITUD];
    int tope=0;
    int otro_vector[LONGITUD];
    int resultado[LONGITUD]={0};

    leer_tope(&tope);
    printf("Ingresar los datos del primer vector\n");
    leer_vector(un_vector,  tope);
    printf("Ingresar los datos del segundo vector\n");
    leer_vector(otro_vector, tope);

    sumar(un_vector,otro_vector,resultado, tope);
    mostrar_vector(resultado, tope);
    restar(un_vector,otro_vector,resultado, tope);
    mostrar_vector(resultado, tope);
    printf("el producto escalar de los dos Vectores es : %ld\n", producto_escalar(un_vector, otro_vector, tope) );
    printf("el modulo del primer vector es : %f\n", modulo(un_vector, tope) );
    printf("el modulo del segundo vector es : %f\n",modulo(otro_vector, tope) );
}
void sumar (int vector_operando_uno[], int vector_operando_dos[], int vector_suma[], int tope){
    for (int i=0;i<tope;i++)
        vector_suma[i]=vector_operando_uno[i]+vector_operando_dos[i];
}
void restar (int vector_operando_uno[], int vector_operando_dos[], int vector_suma[], int tope){
    for (int i=0;i<tope;i++)
        vector_suma[i]=vector_operando_uno[i]-vector_operando_dos[i];
}
long producto_escalar (int vector_operando_uno[], int vector_operando_dos[], int tope){
    long sumatoria=0;
    for (int i=0;i<tope;i++)
        sumatoria+=vector_operando_uno[i]*vector_operando_dos[i];
    return sumatoria;
}
void leer_tope(int *tope){
    printf("Ingrese la dimension de los vectores\n");
    scanf("%d",tope);
}
void leer_vector(int un_vector[], int tope){
    for (int i=0;i<tope;i++){
    printf("Vector[%d] = ",i);
    scanf("%d",&un_vector[i]);
    }
}
double modulo(int operando_uno[], int tope){
    double suma=0;
    for(int i=0;i<tope;i++){
        suma+=operando_uno[i]*operando_uno[i];
    }
    return sqrt(suma);
}
void mostrar_vector(int un_vector[], int tope){
    for (int i=0;i<tope;i++)
    printf("%d",un_vector[i] );
}
```

El mismo concepto es aplicable para las matrices y los arreglos multidimensionales, por cada dimensión es necesario almacenar el tope.

7. Registros

Un registro o struct en C es un tipo de dato estructurado, que define a una lista de variables agrupadas físicamente bajo un mismo nombre en un bloque de memoria, permitiendo de esta forma que estas sean accedidas mediante la utilización de un único nombre (que como ya sabemos en C equivale a la variable puntero que referencia a dicha estructura).

Un struct o registro puede contener ya sea a otros tipos de datos estructurados como a tipos de datos simples. Se debe tener en cuenta que los struct o registros permiten, al contrario de los vectores, agrupar variables de tipos de datos disímiles que guardan alguna relación entre sí.

```
struct un_nombre {
        tipo campo1;
        tipo campo2;
        tipo campo3;
        tipo campo4;
};
```

7.0.1. Typedef

También es posible utilizar y declarar estructuras utilizando la palabra reservada de C **typdef**. Su función es asignar un nombre alternativo a tipos existentes, a menudo cuando su declaración normal es aparatosa o potencialmente confusa, por ejemplo:

```

typedef struct fecha{
        unsigned short dia;
        unsigned short mes;
        unsigned int anio;
} fecha_t;

typedef char[9] documento_t;

typedef struct persona {
        char nombre[40];
        char apellido[40]
        documento_t documento
        fecha_t fecha_de_nacimiento;
}persona_t;

persona_t una_persona;
```

Cada una de las variables que componen a un struct o registro se denomina campo o miembro. Al igual que los arreglos estas variables pueden ser inicializadas:

```
/* Declaración anticipada del tipo de dato "punto". */
typedef struct punto punto_t;

/*Declaracion del struct o registro con los campos o miemobros x,y */
struct punto {
        int    x;
        int    y;
};
```

Inicialización al estilo C-89, solo puede ser hecha cuando los valores de los campos son ingresados en orden y en forma contigua:

```
/* define una variable de tipo Tpunto e inicializa sus miembros en el mismo lugar */
point p = {1,2};
```

Si la inicialización no es en orden, entonces:

```
/* Define una variable de tipo Tpunto e inicializa sus miembros*/
point p = {.y = 2, .x = 1};
```

7.1. Registros: Asignación

Como se puede suponer la asignación de una variable de tipo struct a otra puede hacerse en un único paso, con lo cual se asigna una variable a otra. O puede hacerse miembro a miembro:

```c
#include<stdio.h>

/*Declaracion del struct o registro con los campos o miemobros x,y */
typedef struct punto {
        int     x;
        int     y;
} punto_t;

int main(){

        punto_t un_punto={1,2};
        punto_t otro_punto;
        punto_t otro_punto_mas;

/* se asigna los valores de todos los miembros de una variable a otra */
        otro_punto = un_punto ;

/* se copia campo a campo los valores de una variable a otra */
        otro_punto_mas.x=un_punto.x;
        otro_punto_mas.y=un_punto.y;

        otro_punto.x+=1;

        if (otro_punto.x != otro_punto_mas.x)  printf("El campo x:%d  el Campo y :%d",
    un_punto.x,un_punto.y)

        return 0;
}
```

Nota : los registros en C no pueden ser comparados con ==, un struct es igual a otro si y solo si campo a campo se verifican la igualdad.

8. Abstracción

En las Ciencias de la Computación el término **abstracción** es de gran utilidad y es utilizado de varias formas en distintas ramas de esta ciencia (Programación, Ingeniería de Requerimientos, Sistemas Operativos, Sistema Distribuidos, entre otras). La abstracción según la RAE es *la acción de abstraer*:

> **1. tr.** Separar por medio de una operación intelectual un rasgo o una cualidad de algo para analizarlos aisladamente o considerarlos en su pura esencia o noción.
>
> **2. intr.** Hacer caso omiso de algo, o dejarlo a un lado. Centremos la atención en lo esencial abstrayendo DE consideraciones marginales.

En otras palabras, para un programador **la abstracción es la acción de que despojar de los aspectos complejos, que no son importantes, para la resolución de un problema determinado**. Esta idea puede verse en la famosa obra de Pablo Picasso llamada El Toro:

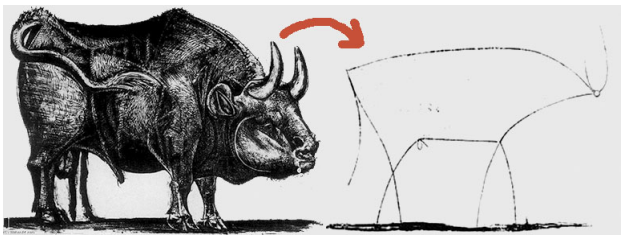


Figura 11: abstracción

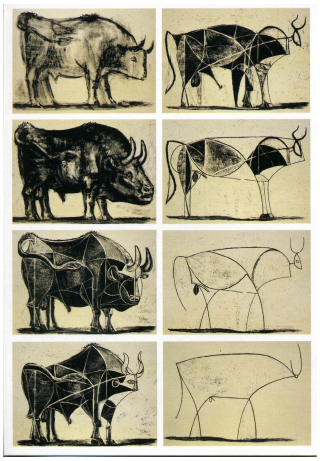


Figura 12: El Toro de Pablo Picasso

8.1. Un Ejemplo

Si se supone que se debe realizar una solución para un determinado problema en el que tiene como principal entidad a un equipo de fútbol, según lo visto en registros se puede crear un nuevo tipo de datos compuesto que represente al equipo. La pregunta del millón es como estaría conformado este registro, cuáles campos debe tener, cuales campos no debe tener. La respuesta a estas preguntas es **depende del problema**.

Se podría imaginar que un primer problema necesita que se almacenen los datos del campeonato actual,una posible definición del tipo de dato equipo seria:

```
// torneo actual
typedef struct equipo{
        char nombre_equipo[TAMAÑO] ;
        int goles_a_favor;
        int goles_en_contra;
        int diferencia;
        int puntaje;
}equipo_t;
```

Un segundo problema sería la necesidad de un club de almacenar los equipos segun sus divisiones,una posible definición del tipo de dato equipo seria:

```
// direccion tecnica
typedef struct equipo{
        char nombre_equipo[TAMAÑO];
        char division;
        int  cantidad_jugadores;
        int  sub_dad;
}equipo_t;
```

Un tercer problema podría ser la necesidad de los directivos del club de mantener el rendimiento histórico de sus equipos,una posible definición del tipo de dato equipo seria:

```
// historial
typedef struct equipo{
        char nombre_equipo[TAMAÑO];
        int campeonatos_jugados;
        int campeonatos_ganados;
        int mejor_puesto;
        float mejor_promedio_rendimiento;
}equipo_t;
```

Un cuarto problema podría ser que el plantel medico del club necesite saber si un equipo puede jugar o no dependiendo de los jugadores lesionados que tenga, una posible definición del tipo de dato equipo seria:

```
// ficha medica
typedef struct equipo {
        char nombre_equipo[TAMAÑO];
        int jugadores_totales;
        int jugadores_lesionados;
        bool puede_jugar;
}equipo_t;
```

Importante!: Si un programador se debe enfocar en uno solo de los problemas, dejando de lado los demás, cada definición anterior representaría un equipo de futbol **despojado de toda la complejidad innecesaria para ese problema**. Por otro lado si el mismo programado tuviera que resolver todos los problemas utilizaria otra definicion más compleja del **tipo de dato equipo**. Más aun podria ser una mezcla de todos los tipos anteriores:

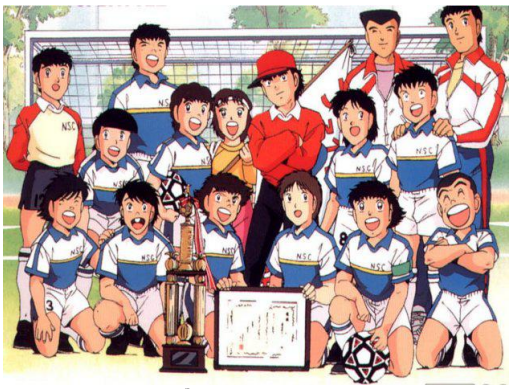
```
        // torneo actual
typedef struct equipo{

        char nombre_equipo[TAMAÑO] ;
        int goles_a_favor;
        int goles_en_contra;
        int diferencia;
        int puntaje;
        char division;
        int  cantidad_jugadores;
        int  sub_dad;
        int campeonatos_jugados;
        int campeonatos_ganados;
        int mejor_puesto;
        float mejor_promedio_rendimiento;
        int jugadores_totales;
        int jugadores_lesionados;
        bool puede_jugar;

}equipo_t;

```

Probablemente, este sea el registro que utilizaría el programador para resolver todos los problemas. Lo más importante en este ejemplo es ver como el registro o tipo de dato equipo es más complejo que en los casos anteriores. Y aquí es donde se ve lo potente del concepto de abstracción, a ningún programador en su sano juicio se le ocurriría utilizar esta definición de equipo para resolver uno solo de los problemas anteriores, pues estaría incurriendo en un error...habría hecho una mala abstracción, es decir no se habría despojado de toda la complejidad innecesaria para resolver el problema.

La **Abstracción** por ende es una de las herramientas más POTENTES que tiene un programador, y esta se mejora cuanto más se la utiliza como herramienta. Otro punto importante es que todas las definiciones de `equipo_t` corresponden a la definición de un equipo, solo que cada una utiliza el mecanismo de abstracción para alcanzar la solución que requerida.

```
// torneo actual
typedef struct equipo{

        char nombre_equipo[TAMAÑO] ;
        int goles_a_favor;
        int goles_en_contra;
        int diferencia;
        int puntaje;

}equipo_t;

```

```
// direccion tecnica
typedef struct equipo{

        char nombre_equipo[TAMAÑO];
        char division;
        int  cantidad_jugadores;
        int  sub_dad;


} equipo_t;

```



```
// historial
typedef struct equipo {

        char nombre_equipo[TAMAÑO];
        int campeonatos_jugados;
        int campeonatos_ganados;
        int mejor_puesto;
        float mejor_promedio_rendimiento;

}equipo_t;


```

```
// ficha medica
typedef struct equipo {

        char nombre_equipo[TAMAÑO];
        int jugadores_totales;
        int jugadores_lesionados;
        bool puede_jugar;

}equipos_t;


```

Se puede considerar que este tipo de dato más la cumplimentación de bibliotecas definidas por el programador, dan el puntapié inicial para una herramienta muy importante en programación que se denomina **TDA** Tipo de Datos Abstracto.

9. Registros: Punteros

Al igual que en cualquier otro tipo de dato, en C uno puede definir un puntero a un struct, se verá la operatoria en el ejemplo:

```
typedef struct punto {
        int x;
        int y;
}punto_t;//Declaro el tipo de dato punto_t

punto_t mi_punto = { 1, 3 }; // Declaro una variable punto_t con nombre mi_punto
punto_t* ptr_punto = &mi_punto; //Declaro un puntero del tipo de dato punto_t y le asigno la
    direccion de memoria de mi_punto

(*ptr_punto).x = 8;   /* acceder al primer miembro del registro */
ptr_punto->x = 8;    /* otra forma de hacer lo mismo (Notese que esta forma es mas intuitiva
    que la anterior)*/
```

10. Ejemplos Resueltos Vectores

10.1. Hora trooper

En su camino al centro de operaciones para recibir sus órdenes del día, los Stormtroopers deben usar el servicio de trenes levogiratorio que los llevará en forma gratuita, pero es de vital importancia para optimizar la velocidad del mismo, que distribuyan el peso total de la manera más homogénea posible. Dado que es el mismo tren el que los lleva y el que los distribuye una vez asignados a una tarea, los vagones podrían quedar completamente vacíos debido al descenso en grupo de unidades trooper. Para lograr equiparar estas diferencias, se tiene un sistema de tickets que le indica al trooper en qué vagón se tiene que subir, en función de la carga de cada uno.

Se le ha encargado programar una función que sepa determinar qué número de vagón deberá usar el trooper cuando vaya a obtener su ticket, a partir de la información de la carga del tren, recibida en un vector, organizada de manera que cada posición del vector representa a cada vagón, con la salvedad de que los troopers saben contar desde 1. Además contará con la cantidad de vagones del tren.

Ej:
cantidad de vagones: 15
vagones: [35, 12, 42, 6, 15, 22, 66, 12, 34, 16, 37, 47, 24, 13, 20]
resultado: 4 (el vagón con 6 pasajeros).

10.1.1. Posible Solución

```
// 9 - Hora Trooper - Realizado por Alejandro Nicolás Smith
int acomodar_trooper(int cantidad_de_vagones, int vagones[]) {
        int vagon_menos_cargado = 0;
        for (int i=0; i<cantidad_de_vagones;i++){
                if (vagones[i]<vagones[vagon_menos_cargado]){
                        vagon_menos_cargado=i;
                }
        }

        return (vagon_menos_cargado+1);
}
```

10.2. La muralla de Maz Kanata

Durante la batalla de Takodana, Kylo Ren tiene sitiado el castillo de Maz Kanata, y tiene posicionadas y listas para el ataque sus armas de asedio. Las defensas del castillo también están listas para recibir el ataque y resistir.

Las murallas del castillo tienen una resistencia total, y las armas de asedio tienen un poder de ataque. En cada ataque, las armas de asedio gastan su munición y reducen la resistencia total de las defensas.

Kylo Ren está preocupado por saber si la munición que tiene será suficiente para derribar las defensas del castillo, y quiere que se simule la batalla antes de comenzar a gastar municiones para poder decidir si cambiar su estrategia de antemano. Por esto te ha delegado la responsabilidad de ayudarlo a calcular la efectividad de sus ataques.

Hacer una función que reciba un vector con la intensidad de los ataques de sus 10 armas de asedio (cada una es distinta), y reste de la resistencia total de la muralla, sus ataques, indicando como resultado si se logró destruir la muralla. De más está decir que la resistencia de la muralla no puede ser menor a 0.

Nota: La función se podrá usar repetidas veces según la cantidad de municiones.

10.2.1. Posible Solución

```
// 10 - La muralla de Maz Kanata - Realizado por Alejandro Nicolás Smith
#define CANTIDAD_ARMAS 10

```

```c
int atacar_muralla(unsigned int ataques[CANTIDAD_ARMAS], unsigned int *
    resistencia_de_la_muralla) {
        // restar todos los ataques modificando la resistencia de la muralla
        int destruida=1;
        if(*resistencia_de_la_muralla>0){
                for (int i=0; i<CANTIDAD_ARMAS;i++){
                        (*resistencia_de_la_muralla)-=ataques[i];
                        if(*resistencia_de_la_muralla<=0){
                                destruida=1;
                        }
                        if(*resistencia_de_la_muralla>0){
                                destruida=0;
                        }
                }
        }return destruida;
}
```

10.3. Nace un nuevo líder

Un rebelde logró infiltrarse entre las tropas de Darth Vader y está instaurando un mensaje entre los enemigos, y es, que él es el verdadero Líder a seguir. Para que todo el ejército le crea su mensaje, debe ganarle al Piedra, Papel o Tijeras a más del 50 % de los Stormtroopers que se acercan a desafiarlo.

El sigue una táctica que no ha sido descubierta, que es jugar siempre tijera (La buena tijera... Nada le gana).

Se busca crear una función que reciba la cantidad de Stormtroopers que juegan contra el rebelde (a una partida cada uno) y determine si es el nuevo Líder o no. La jugadas jugadas de los Stormtroopers vienen en un vector.

Las jugadas posibles son:

- 'R': Piedra.
- 'P': Papel
- 'T': Tijera

Nota : Devuelva 0 (cero) si se determina que no es el nuevo líder o un valor distinto de cero si lo es.

10.3.1. Posible Solución

```c
// 11 - Nace un nuevo lider - Realizado por Bruno Lucena

int calcular_liderazgo(char jugadas[],unsigned int cant_soldados){
        int partidas_ganadas = 0;
        int resultado;

        for (int a = 0; a < cant_soldados; a++) {
                if (jugadas[a] == 'P') {
                        partidas_ganadas++;
                }
        }

        if (partidas_ganadas > (cant_soldados / 2)) {
                resultado = 1;
        } else {
                resultado = 0;
        }

        return resultado;
}
```

10.4. Orientando cañones

Han Solo está planeando su intervención en la batalla de Miratrella, para lo que está tratando de maximizar la eficiencia de ataque del Halcón Milenario, mediante un mecanismo de predicción del flanco en que se acercarán las naves enemigas, para optimizar la orientación de la nave y sus cañones más poderosos.

El software del Halcón Milenario es capaz de reconocer y almacenar las direcciones de los últimos 10 ataques, pero Han Solo necesita que le resuelvas el problema de la póxima dirección de ataque en una función que tome los datos disponibles.

A partir de un vector de 10 posiciones de los ataques recientes, calcule el valor menos frecuente, dentro de un conjunto de posibilidades y prediga de donde vendrá el próximo ataque, entendiendo que será el lado opuesto al más frecuente.

Las direcciones de ataque posibles son, en pares opuestos:

- 'E' $->$ 'H' (arriba y abajo)
- 'D' $->$ 'S' (derecha e izquierda)
- 'V' $->$ 'R' (frente y cola)
- 'I' (imposible predecir)

En caso de haber empate, la dirección más probable es la menos frecuente. En caso de seguir habiendo empate, se deberá emitir el resultado 'I' (indeterminado).

10.4.1. Posible Solución

```c
// 12 - Orientando cañones - Realizado por Gisela Ramos
#define CANTIDAD_ATAQUES 10
#define CANT_DIRECCIONES 6
char predecir_flanco(char ataques[CANTIDAD_ATAQUES]) {

        int acumulador_ataques[CANT_DIRECCIONES];
        int i, j, pos_maximo, pos_minimo, empate_max = 0 , empate_min = 0;
        char direccion_contrario[CANT_DIRECCIONES]= {'H' , 'S' , 'R' , 'E','D','V'};
        char direccion_contrario_2[CANT_DIRECCIONES]= {'E' , 'D' , 'V' , 'H','S','R'};
        char direccion ;

        for(i = 0 ; i < CANT_DIRECCIONES ; i++){
                acumulador_ataques[i] = 0;
        }

        for(i = 0 ; i < CANTIDAD_ATAQUES ; i++){
                direccion = ataques[i];
                switch(direccion){
                        case 'E':
                        acumulador_ataques[0]++;
                        //direccion_contrario[0] = 'H';
                        break;
                        case 'D':
                        acumulador_ataques[1]++;
                        //direccion_contrario[1] = 'S';
                        break;
                        case 'V':
                        acumulador_ataques[2]++;
                        //direccion_contrario[2] = 'R';
                        break;
                        case 'H':
                        acumulador_ataques[3]++;
                        //direccion_contrario[3] = 'E';
                        break;
                        case 'S':
                        acumulador_ataques[4]++;
                        //direccion_contrario[4] = 'D';
                        break;
                        case 'R':
                        acumulador_ataques[5]++;
                        //direccion_contrario[5] = 'V';
                        break;
                }
        }

```

```
46          pos_maximo = 0;
47          i = 1;
48          while( (i < CANT_DIRECCIONES) && (empate_max == 0) ){
49                  if(acumulador_ataques[i] == acumulador_ataques[pos_maximo])
50                  empate_max = 1;
51                  if(acumulador_ataques[i] > acumulador_ataques[pos_maximo])
52                  pos_maximo = i;
53                  i++;
54          }
55
56          pos_minimo = 0;
57          j = 1;
58
59          while( (j < CANT_DIRECCIONES) && (empate_min == 0) ){
60                  if(acumulador_ataques[j] == acumulador_ataques[pos_minimo])
61                  empate_min = 1;
62                  if(acumulador_ataques[j] < acumulador_ataques[pos_minimo])
63                  pos_minimo = j;
64                  j++;
65          }
66
67
68          if(empate_max == 0)
69          return direccion_contrario[pos_maximo];
70          else if (empate_min == 0)
71          return direccion_contrario_2[pos_minimo];
72          else
73          return 'I';
74  }
```

11. Ejemplo Resueltos Matrices

11.1. Apuntando los cañones

(esto es un producto entre matrices, tranquilamente se puede borrar gran parte del enunciado)

En el campo de entrenamiento se están probando unos cañones experimentales de tiro oblicuo que usan trayectorias de atracción a concentraciones de midiclorias para maximizar su efectividad sobre seres biológicos. Esta característica hace que el objetivo de los disparos sea uno dentro de un rango de posibilidades.

Los técnicos que diseñaron la maquinaria trabajan junto con el equipo de matemática aplicada para conseguir un algoritmo que les permita ajustar los resultados y poder predecir el punto de impacto de cada disparo. Actualmente necesitan ensayar una teoría que involucra tomar en cuenta las midiclorias dispersas en pequeñas cantidades en el ambiente, tomando en cuenta tanto las dimensiones verticales como las horizontales sobre el área de efecto del disparo.

La teoría a ensayar consiste, matemáticamente, en el producto matricial de las medidas tomadas por múltiples sensores incorporados en los cañones. Por la cantidad y sensibilidad de sensores incorporados, las matrices obtenidas son de 3 x 3 (2 dimensiones).

Codifique una función que efectúe este producto matricial (ver referencia para obtener información sobre la operación matemática), recibiendo las matrices de mediciones verticales, mediciones horizontales, y devuelva una tercera matriz con las probabilidades de recibir el tiro de cada área del rango de efecto.

Referencias: `http://www.vitutor.com/algebra/matrices/producto.html`

11.1.1. Posible Solución

```
1
2  #define TAMANIO_MATRIZ 3
3
4  void producto_matricial(double densidades_verticales[TAMANIO_MATRIZ][TAMANIO_MATRIZ], double
       densidades_horizontales[TAMANIO_MATRIZ][TAMANIO_MATRIZ],
5  double producto[TAMANIO_MATRIZ][TAMANIO_MATRIZ]) {
6          int i, j, k;
7          for(i=0; i<TAMANIO_MATRIZ; i++){
8                  for(j=0; j<TAMANIO_MATRIZ; j++){
9                          producto[i][j]=0;
10                         for(k=0; k<TAMANIO_MATRIZ; k++){
11                                 producto[i][j]+=(densidades_verticales[i][k]*
       densidades_horizontales[k][j]);
12                         }
13                 }
14         }
15 }
```

11.2. Traidor entre líneas

Darth Vader recibió información de que en uno de los batallones de Stormtroopers hay un rebelde infiltrado, y también el número de ID que está usando. Como el desfile de inauguración de la Estrella de la muerte se aproxima y se le asignó un orden determinado a cada batallón, decidió que se encargaría de eliminar en público al infiltrado, durante la ceremonia. Para lograrlo, necesita saber la posición en la que el rebelde se va a encontrar durante el desfile.

Se le encargó realizar una función que reciba un batallón de stormtroopers, el ID que usa el infiltrado y las posiciones del traidor(una para las filas y otra para las columnas). En caso de hallar el infiltrado en el batallón, la función tiene que actualizar los parámetros de posición recibidos con la posición del infiltrado, si no se encuentra en ese batallón, se tiene que asignar -1 en dichos parámetros.

11.2.1. Posible Solución

```
1
2  #define FILAS 5
3  #define COLUMNAS 5
4
5  void buscar_el_traidor(int batallon[FILAS][COLUMNAS], int id_rebelde, int *fila_del_rebelde,
       int *columna_del_rebelde){
6          *fila_del_rebelde=-1;
```

```
        *columna_del_rebelde=-1;

        for(int fila=0; fila<FILAS; fila++){
                for(int columna=0; columna<COLUMNAS; columna++){
                        if (batallon[fila][columna]==idRebelde){
                                *fila_del_rebelde=fila;
                                *columna_del_rebelde=columna;
                        }
                }
        }
}
```

11.3. Organizando batallones

El imperio se percató de que sus minas eran fácilmente removibles por nuestro equipo de soldados. Por lo tanto crearon un nuevo tipo de minas imposible de extraer por nuestro equipo. Como no nos encontramos en condiciones de extraerlas, tenemos que colocar banderas alrededor de estas para no pisarlas. Además, sabemos que las minas están alejadas al menos 2 casillas de distancias.
Por ejemplo, Si recibimos:

```
0 0 X 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 X 0 0 0 0
0 0 0 0 0 0 0 0 X 0
0 0 0 0 X 0 0 0 0 0
0 0 X 0 0 0 0 0 0 0
0 0 0 0 0 0 0 X 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

Debería quedar:

```
0 B X B 0 0 0 0 0 0
0 0 B 0 0 B 0 0 0 0
0 0 0 0 B X B 0 B 0
0 0 0 0 B B 0 B X B
0 0 B B X B 0 0 B 0
0 B X B B 0 0 B 0 0
0 0 B 0 0 0 B X B 0
0 0 0 0 0 0 0 B 0 0
0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
```

11.3.1. Posible Solución

```

#define TAM 10
#define BANDERA ?B?

void poner_banderas (char campo_minado[][TAM], int fila, int columna){
        if (fila>0) campo_minado[fila-1][columna]=BANDERA;
        if (fila<TAM-1) campo_minado[fila+1][columna]=BANDERA;
        if (columna>0) campo_minado[fila][columna-1]=BANDERA;
        if (columna<TAM-1) campo_minado[fila][columna+1]=BANDERA;
}

void buscar_minas (char campo_minado[][TAM]){
        int fila, columna;
        for (fila=0; fila<TAM; fila++){
                for (columna=0; columna<TAM; columna++){
                        if (campo_minado[fila][columna]=='X'){
                                poner_banderas(campo_minado, fila, columna);
                        }
```

```
                }
        }
}
```

11.4. Equipando batallones

Un cargamento con nuevos rifles láser llegó a la estrella de la muerte! El emperador ordenó a todos los batallones reportarse para revisar el equipamiento de cada uno de sus integrantes.

Se decidió que un soldado solamente va a recibir uno de los nuevos rifles si la cantidad de armas que posee es menor a 3.

Se le encargó realizar una función que reciba como parámetros un batallón (con sus respectivos topes) y la cantidad de nuevos rifles disponibles. Esta función debe revisar el equipamiento de cada soldado y, en caso de tener menos de 3 armas, debe equiparlo con uno de los nuevos rifles de lujo, y restarlo del total de nuevos rifles disponibles.

Aclaraciones: cada elemento de la matriz corresponde a un arma: el 2 significa que tiene un arma y el -1 significa que no tiene un arma.

Si un soldado tiene alguna arma, esta va a estar ubicada en la primera columna, no puede pasar por ejemplo que una fila sea así: (-1 2 -1). Si el soldado tiene un arma siempre se está antes de los espacios disponibles.

ejemplo de batallón:

$$\begin{array}{rrrl} 2 & 2 & -1 & (soldado1) \\ 2 & 2 & 2 & (soldado2) \\ 2 & -1 & -1 & (soldado3) \\ -1 & -1 & -1 & (soldado4) \end{array}$$

Al terminar nuestra función, la matriz debería quedar así:

$$\begin{array}{rrr} 2 & 2 & 2 \\ 2 & 2 & -1 \\ 2 & -1 & -1 \end{array}$$

11.4.1. Solución Posibles

```

#define MAX_SOLDADOS 1000
#define MAX_ARMAS 300

void equipar_batallon(int batallon[MAX_SOLDADOS][MAX_ARMAS], int cantidad_soldados, int
    cantidad_armas, int* nuevos_rifles_disponibles){
        int armas_brindadas_a_soldado_actual;
        for (int fila=0; fila<cantidad_soldados; fila++){
                armas_brindadas_a_soldado_actual=0;
                if (*nuevos_rifles_disponibles>0){
                        for (int columna=0; columna<cantidad_armas; columna++){
                                if (batallon[fila][columna]==-1 &&
    armas_brindadas_a_soldado_actual==0){
                                        batallon[fila][columna]=2;
                                        (*nuevos_rifles_disponibles)--;
                                        armas_brindadas_a_soldado_actual=1;
                                }
                        }
                }
        }
}
```

11.4.2. Posible Solución

la mejora posiuble es sacar números mágicos del código:

- -1 significa **sin arma**. SIN_ARMA
- 2 significa **con arma** . CON_ARMA

```c

#define MAX_SOLDADOS 1000
#define MAX_ARMAS 300
#define CON_ARMA 2
#define SIN_ARMA -1

void equipar_batallon(int batallon[MAX_SOLDADOS][MAX_ARMAS], int cantidad_soldados, int cantidad_armas, int* nuevos_rifles_disponibles){
    int armas_brindadas_a_soldado_actual;
    for (int fila=0; fila<cantidad_soldados; fila++){
        armas_brindadas_a_soldado_actual=0;
        if (*nuevos_rifles_disponibles>0){
            for (int columna=0; columna<cantidad_armas; columna++){
                if ( batallon[fila][columna]==SIN_ARMA &&  armas_brindadas_a_soldado_actual==0 ){
                    batallon[fila][columna]=CON_ARMA;
                    (*nuevos_rifles_disponibles)--;
                    armas_brindadas_a_soldado_actual=1;
                }
            }
        }
    }
}
```

12. Ejemplo Resueltos Matrices

12.1. Clasificación de estrellas

El Líder Supremo Snoke le ha encargado al General Hux un relevamiento de las estrellas cercanas para poder determinar cuáles son combustible viable para la nueva super arma: Starkiller Base.

Para esto es necesario saber de qué clase es, el radio en kilómetros y la edad. La clase, consiste en dos datos, una familia, identificada con letras 'A', 'B', 'C', y un número de subtipo (un entero sin signo); la edad es el entero positivo más grande que conozca; y del radio nos interesan hasta 6 cifras significativas.

Defina el tipo de registro que necesitará el General Hux para cumplir con su tarea.

12.1.1. Solución Posibles

```
1
2  //17 - Clasificación de estrellas - Realizado por Lorenzo Gimenez
3  typedef struct estrella {
4          // campos
5          char familia;
6          unsigned int subtipo;
7          unsigned long int edad;
8          double radio;
9
10 } t_estrella;
```

12.2. Equipo trooper

La formación de un Trooper no es cosa simple. En la Academia, los soldados recién llegados deben aprender lo básico, antes de comenzar a manejar armas y practicar maniobras.

En esta ocasión, deberán aprender a reconocer las partes de su equipamento para no olvidar nada en el momento de partir para una misión.

El equipamiento elemental de todo Trooper consta de 5 elementos, y cada soldado debe conocer exactamente sus características.

Botas: Número de las botas (entero) Armadura: Los talles son S, M y L Casco: La circunferencia interna del casco (entero en centímetros) Arma de alcance: Modelo del arma (una cadena de caracteres de hasta 6 caracteres, ej: DC-15A) Municiones: Cantidad de municiones para el arma de alcance (entero) Arma de melé: Modelo del arma (una cadena de caracteres de hasta 4 caracteres, ej: Z6-2)

Declare un nuevo tipo de dato (struct) para guardar la información del equipo de cada soldado usando la convención de nomenclatura usada en la materia (snake_case)

Extras: `http://starwars.wikia.com/wiki/E-11_blaster_rifle` `https://scifi.stackexchange.com/questions/115022/what-was-the-weapon-that-this-tormtrooper-used-when-he-fought-finn`

12.2.1. Solución Posibles

```
1  // 18 - Equipo trooper - Realizado por Lorenzo Gimenez
2  typedef struct equipo_trooper {
3          // campos
4
5          int botas;
6          char armadura;
7          int casco;
8          char arma_de_alcance[7];
9          int municiones;
10         char arma_de_mele[5];
11
12 } t_equipo_trooper;
```

12.3. Kylo, el perverso

El despiadado Kylo Ren, guiado por el líder supremo Snoke ha desarrollado un sistema de extracción de información mediante interrogatorios basados en el uso de tortura mediante la fuerza. Diversos métodos de tortura han sido desarrollados y la veracidad mínima de la información obtenida mediante cada uno ha sido cuantizada. No pudiendo contenerse en su oscuro afán, han lanzado múltiples operaciones para capturar prisioneros de la Resistencia en todos los planetas conocidos por ser bastiones de la misma, y planea aplicarles sus nuevos métodos a todos ellos para ensayarlos antes de un ataque teledirigido a la cúpula de mando de la misma Resistencia.

Los prisioneros ingresados, serán catalogados con la siguiente información prisionero:

- género: un string indicando su género
- nombre
- alias: sus pseudónimos conocidos
- operación de captura: la operación en la que fue capturado que tendrá los datos:
  - código: alfanumérico de hasta 12 caracteres
  - planeta: el planeta donde se efectuó la operación
  - id: un número natural capaz de contar todos los planetas del universo conocido
  - nombre
  - sistema: el nombre del sistema planetario al que pertenece
  - cuadrante: el identificador del cuadrante donde se ubica (alfanumérico de hasta 24 caracteres)

Estos, serán apareados con los métodos de disuasión utilizados durante el interrogatorio: tortura:

- nombre
- nivel: entero que sirve para catalogarla
- veracidad: entero no signado de grado de veracidad obtenido en la información extraída con este método; la escala no es muy alta.

Finalmente, la Primera Orden necesita catalogar la información obtenida para su posterior análisis por los analistas de inteligencia para determinar futuras acciones militares. información:

- prioridad, en números enteros
- prisionero, con toda su información
- tortura, aplicada para obtener esta información
- detalle, explicando la información obtenida, en prosa.

Declare todos los tipos necesarios para el análisis de los recursos de información de la Primera Orden.

12.3.1. Solución Posibles

```
1
2  // 19 - Kylo, el perverso - Realizado por Gisela Ramos
3  typedef struct planeta{
4          unsigned int id;
5          char nombre[20];
6          char sistema[20];
7          char cuadrante[25];
8  }t_planeta;
9
10 typedef struct operacion{
11         char codigo[13];
12         t_planeta planeta;
13 }t_operacion;
14
15 typedef struct prisionero{
16         char genero[10];
17         char nombre[30];
18         char alias[10];
19         t_operacion operacion_de_captura;
```

```
}t_prisionero;

typedef struct tortura{
        char nombre[20];
        unsigned int nivel;
        unsigned int veracidad;
}t_tortura;

typedef struct informacion{
        int prioridad;
        t_prisionero prisionero;
        t_tortura tortura;
        char detalle[257];
}t_informacion;
```

12.4. La mejor defensa es un buen ataque

Un asedio no es tarea sencilla, como determinar la mejor estrategia de defensa para el mismo, tampoco lo es, y Maz Kanata lo sabe muy bien. Para organizar la defensa de su castillo, está confiada en la resistencia de sus murallas, tanto como lo está en los cañones defensivos apostados en los puestos de artillería. Cada cañón tiene sus características particulares: propósito (artillería, antiaéreo, etc), tipo de munición y cantidad de cartuchos. Además, para operarlo, hace falta un equipo de operadores especializado, que cuenta con un oficial al mando, y entre 2 y 5 soldados, según la complejidad del equipo.

Para llevar cuenta del estado de cada uno, Maz kanata necesita tener sus datos reunidos en el centro de comando, y nos ha encargado definir la estructura que usará su computadora para poder informarla sobre la situación.

Puesto de artillería:

- propósito
- cantidad de caniones
- ubicación
- flanco: 'norte', 'sur', 'este', 'oeste'
- posicion: numero de 1 a 25
- munición
- clase: alfanumérico
- subclase: 'A', 'B', 'C'
- tipo de danio: alfabético
- poder: numérico
- cantidad de cartuchos
- equipo de operación
- oficial
- rango: alfanumérico
- número: alfanumérico
- nombre: alfabético
- experiencia: cantidad de misiones desde su última promoción de rango
- equipo: array
- función: 'artillero', 'asistente', etc
- rango
- número

Defina la estructura en C siguiendo los lineamientos indicados en el enunciado.

12.4.1. Solución Posibles

```
1
2  // 20 - La mejor defensa es un buen ataque - Realizado por Gisela Ramos
3  typedef struct ubicacion{
4          char flanco[6];
5          unsigned int posicion;
6  }t_ubicacion;
7
8  typedef struct municion{
9          char clase[30];
10         char subclase;
11         char tipo_de_danio[20];
12         int poder;
13 }t_municion;
14
15 typedef struct oficial{
16         char rango[20];
17         char numero[20];
18         char nombre[30];
19         unsigned int experiencia;
20 }t_oficial;
21
22 typedef struct equipo_de_operacion{
23         char funcion[20];
24         char rango[20];
25         char numero[20];
26 }t_equipo_de_operacion;
27
28 typedef struct equipos{
29         t_oficial oficial;
30         t_equipo_de_operacion equipo[4];
31 }t_equipo;
32
33
34 typedef struct artilleria{
35         char proposito[20];
36         unsigned int cantidad_de_caniones;
37         t_ubicacion ubicacion;
38         t_municion municion;
39         unsigned int cantidad_de_cartuchos;
40         t_equipo equipo_de_operacion;
41 }t_puesto_artilleria;
```

Referencias