



Ingeniería Electrónica

Informática

Apunte de Cátedra

Unidad 4: Modularización Funciones y Procedimientos

Francisco Murcia
Martin Morixe
Germán Tejero



Índice general

4	Modularización: Funciones y Procedimientos	3
4.1	Funciones	3
4.1.1	Conceptos	3
4.1.2	Valores de retorno	3
4.1.3	Parámetros	4
4.1.4	Uso de void (procedimientos)	5
4.1.5	Parámetros por referencia y por valor	6
4.1.6	Vectores, cadenas y matrices por parámetro	7
4.1.7	Pasaje de parámetros referencia / valor – síntesis	10
4.1.8	Ejemplos	11
4.2	Alcance de una variable	11
4.3	Recursión	12
4.4	Uso de parámetros en la función main	14
4.5	Guía práctica	14
4.6	Funciones útiles generales	14
4.7	Funciones útiles de vectores	15
4.8	Uso de funciones útiles de vectores	15
4.9	Funciones útiles de matrices	15
4.10	Uso de funciones útiles de matrices	16
4.11	Alcance de variables	16
4.12	Ejercicios extra	17
4.12.1	Funciones simples	17
4.12.2	Funciones simples	17
4.12.3	Funciones de vectores (y punteros)	17
4.12.4	Funciones de vectores (y punteros)	18
4.12.5	Funciones de matrices (y punteros)	18
4.12.6	Ejercicios de matrices	18



4. Modularización: Funciones y Procedimientos

4.1 Funciones

4.1.1 Conceptos

Hasta ahora vimos como hacer algoritmos (programas) utilizando únicamente la función main. Los problemas vistos se resolvían con una secuencia de pasos que comenzaba en la función main y finalizaba en el return del mismo.

Hasta ahora vimos como hacer algoritmos (programas) utilizando la función main (punto de entrada a todo programa en C). Hemos visto que nuestros programas pueden utilizar funciones de la biblioteca standard de C, como por ejemplo la función printf() o scanf() de la biblioteca stdio.h (recuerde otras bibliotecas: stdlib.h, string.h, etc).

El lenguaje C permite también que el programador implemente sus propias funciones. Al usar funciones definidas por el propio programador, los programas pueden estructurarse en partes más pequeñas, más sencillas. Cada una de estas partes debe responder a un único objetivo (por ejemplo, una función para sumar dos valores, otra para multiplicar dos valores). Estas funciones podrán ser luego reutilizadas en diferentes partes del programa.

Resolver un problema pensando en dividirlo en varias funciones, nos obliga a dividir al problema principal (complejo) en un conjunto de problemas más pequeños (y más sencillos). Esta división del problema nos dará varias ventajas:

1. Cada división independiente (función) será más fácil de programar y probar.
2. El problema principal es más sencillo de entender.
3. Podemos reutilizar cada una de las divisiones independientes (funciones).
4. Estas divisiones independientes (funciones) podrían ser programadas por diferentes personas.

4.1.2 Valores de retorno

Escriba funciones para leer un número entero, y un número real; utilícelas en un programa:

```
#include <stdio.h>

int leerNroEntero()
{
    int valor;

    printf("Ingrese un numero entero: ");
    scanf("%d", &valor);

    return valor;
}
```



```
}

float leerNroReal()
{
    float valor;

    printf("Ingrese un numero real: ");
    scanf("%f", &valor);

    return valor;
}

int main()
{
    int entero;
    float real;

    entero = leerNroEntero();
    real = leerNroReal();

    printf("Entero: %d\nReal: %f\n", entero, real);

    return 0;
}
```

Ejemplo de código 4.1: Vaor de retorno

Analicemos un poco el ejercicio anterior. Allí vemos dos funciones: main (como siempre) y leerNroEntero (una nueva función que creamos nosotros). Como vemos en su declaración:

```
int leerNroEntero()
{
    ...
}
```

Decimos que esta función retorna un entero (int) y que su nombre es leerNroEntero. La función espera que se ingrese un número por teclado, lo asigna a la variable valor, y retorna la variable valor como respuesta a quien la llame. Así entonces cuando en la función main se llama a esta función de la siguiente manera:

```
entero = leerNroEntero();
```

El valor retornado por leerNroEntero se asigna a la variable edad. Lo mismo sucede con la siguiente llamada.

4.1.3 Parámetros

Pero además de retornar valores, una función puede recibir parámetros. Los parámetros son valores que toma la función en el momento en que es llamada. Los parámetros son variables que toman un valor específico en el momento en que se ejecuta el programa (igual que las otras variables que hemos estado viendo. Así, por ejemplo, podemos hacer un programa que calcule la suma de dos números:

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

    suma = a + b;

    return suma;
}
```

Ejemplo de código 4.2: Parámetros

Analicemos un poco la definición de esta función:

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

Esta función se llama suma, retorna un entero, y recibe dos parámetros enteros: el primer parámetro se llama a y el segundo b. Para utilizar la función suma debemos especificarle sus dos parámetros, que son los dos números que se deben sumar. Hay varias formas de especificar estos parámetros:

```
suma1 = suma(1, 2);  
suma2 = suma(n1, 2);  
suma3 = suma(n2, b);
```

Ejemplo de código 4.3: Ejemplo de invocación

El primer ejemplo asigna los valores 1 y 2 a las variables a y b de la función suma respectivamente. La función al finalizar retorna 3 y se lo asigna a la variable suma1.

El segundo ejemplo asigna los valores n1 y 2 a las variables a y b de la función suma respectivamente. Así por ejemplo, si n1 tuviera el valor 3, la variable a tomaría dicho valor. La función al finalizar retorna 5 y se lo asigna a la variable suma2.

El tercer ejemplo asigna los valores n1 y n2 a las variables a y b de la función suma respectivamente. Así por ejemplo, si n1 tuviera el valor 3, y n2 el valor 4, las variables a y b tomarían los valores 3 y 4 respectivamente. La función al finalizar retorna 7 y se lo asigna a la variable suma3.

El ejemplo completo quedaría así:

```
#include <stdio.h>  
  
int suma(int a, int b)  
{  
    int suma;  
  
    suma = a + b;  
  
    return suma;  
}  
  
int main()  
{  
    int n1 = 3;  
    int n2 = 4;  
    int suma1, suma2, suma3;  
  
    suma1 = suma(1, 2);  
    suma2 = suma(n1, 2);  
    suma3 = suma(n1, n2);  
  
    return 0;  
}
```

Ejemplo de código 4.4: Ejemplo completo

4.1.4 Uso de void (procedimientos)

Algunas veces necesitamos crear funciones que no retornen ningún valor. Los ejemplos más sencillos de éstos casos son referentes a impresión de datos en pantalla. Supongamos que queremos hacer una función que imprima un mensaje de bienvenida al programa:

```
void bienvenida()
{
    char aux[100];

    printf("Bienvenido al curso de UNRN\n");
    printf("Presione ENTER para comenzar\n");

    fgets(aux, 100, stdin);

    return;
}
```

Ejemplo de código 4.5: Procedimientos

El tipo de retorno void significa que la función no retorna ningún valor. En otros lenguajes de programación este tipo de funciones que no retornan ningún valor se llaman procedimientos. La forma de usar esa función en nuestro programa (función main) es la siguiente:

```
#include <stdio.h>

void bienvenida()
{
    char aux[100];

    printf("Bienvenido al curso de UNRN\n");
    printf("< Presione ENTER para comenzar >\n");

    fgets(aux, 100, stdin);
}

int main()
{
    bienvenida();

    return 0;
}
```

Ejemplo de código 4.6: Invocación de procedimiento

Ejercitar 4.1 — Hacer los ejercicios de la sección 4.6 Funciones útiles generales. ■

4.1.5 Parámetros por referencia y por valor

Algún alumno se preguntará en este momento, ¿que sucede si modifico el valor de la variable dentro de la función? ¿Se modifica también la variable asociada en la llamada a la función? Por ejemplo:

```
#include <stdio.h>

void incrementa(int a)
{
    a = a + 1;
}

int main()
{
    int x = 0;

    incrementa(x);
    incrementa(x);
    incrementa(x);
}
```

```
printf("El valor incrementado 3 veces es: %d\n", x);  
  
return 0;  
}
```

Ejemplo de código 4.7: Paso de parámetros

Si ejecutamos este ejemplo veremos que la variable `x` permanece con su valor asignado originalmente (valor 0) por lo que concluimos que al modificar el valor de un parámetro de una función, su variable equivalente en la llamada no se modifica, y permanece como estaba.

Esto es porque en C el pasaje de parámetros es por valor. El pasaje de parámetros por valor no cambia el valor de las variables originales al modificar el valor de los parámetros.

En otros lenguajes (por ejemplo en C++) existe el pasaje de parámetros por referencia. Cuando los parámetros se pasan por referencia, si cambiamos el valor dentro de la función, también se cambian en las variables de la llamada.

El lenguaje C tiene algo que a primera vista pareciera ser pasaje de parámetros por referencia, pero que en realidad es implementada de otra manera: a través de punteros. Si modificamos el ejercicio anterior para utilizar punteros, entonces si se logrará el resultado deseado.

- ❶ Para utilizar el puntero en la función utilizar el operador `*`.
- ❷ Para pasar por parámetro el puntero a una variable utilizar el operador `&`.

```
#include <stdio.h>  
  
void incrementa(int *a)  
{  
    *a = *a + 1;  
}  
  
int main()  
{  
    int x = 0;  
  
    incrementa(&x);  
    incrementa(&x);  
    incrementa(&x);  
  
    printf("El valor incrementado 3 veces es: %d\n", x);  
  
    return 0;  
}
```

Ejemplo de código 4.8: Paso por referencia o dirección

4.1.6 Vectores, cadenas y matrices por parámetro

Además de tipos de datos primitivos (`char`, `int`, `float` `double`, etc) las funciones pueden recibir por parámetros vectores, cadenas y matrices. Para esto lo que se debe hacer indicar que se trata de un vector, cadena, o matriz incluyendo los corchetes como se muestra en los siguientes ejemplos.

Problema 4.1 haga una función que sume los elementos del vector:

```
int sumaVector(int vec[], int cantElementos)  
{  
    int i, suma;  
  
    suma = 0;  
    for(i = 0; i < cantElementos; i++)  
    {
```

```
        suma += vec[i];
    }

    return suma;
}
```

Ejemplo de código 4.9: Paso de vectores

Como vemos en la declaración de la función `sumaVector`, el primer parámetro es un vector de enteros. Esto queda determinado por los corchetes `[]` (se trata de un vector) y por su tipo `int` (se trata de enteros). Nótese que además se incluye otro parámetro que es la cantidad de elementos que tiene el vector. SIEMPRE que pasamos un vector por parámetro debemos incluir su cantidad de elementos.

Nota: en el caso de los vectores es posible utilizar un `define` en el programa que especifique esta cantidad, y luego especificar esto en la declaración del vector. Entonces en la función utilizaríamos esta constante `N`. El siguiente ejemplo ilustra esto:

```
int sumaVector(int vec[N])
{
    int i, suma;

    suma = 0;
    for(i = 0; i < N; i++)
    {
        suma += vec[i];
    }

    return suma;
}
```

Ejemplo de código 4.10: Uso de constantes

Las diferencias de estos dos ejemplos anteriores son: • en el primero se indica la cantidad de elementos del vector a través de un parámetro, en el segundo se hace con una constante de programa • en el primero se utiliza la variable que indica la cantidad de elementos para recorrer el vector, en el segundo se utiliza la constante de programa

Vemos que la primera tiene ventajas con respecto a la segunda: podemos utilizar la función para vectores de cualquier tamaño. Mientras que en el segundo caso sólo podremos usar esa función para vectores del mismo tamaño `N` dentro del mismo programa. Es por esto que preferimos siempre la primer opción.

El ejemplo completo, que muestra también como utilizar la función es el siguiente:

```
#include <stdio.h>

#define N 5

int sumaVector(int vec[], int cantElementos)
{
    int i, suma;

    suma = 0;
    for(i = 0; i < cantElementos; i++)
    {
        suma += vec[i];
    }

    return suma;
}

int main()
```



```
{
    int vector[N] = { 1, 2, 3, 4, 5 };
    int suma;

    suma = sumaVector(vector, N);
    printf("El vector suma: %d\n", suma);

    return 0;
}
```

Ejemplo de código 4.11: Ejemplo de paso de vectores

Cadena de caracteres: como las cadenas de caracteres son vectores, aplica exactamente lo mismo que lo visto para vectores.

Ejercitar 4.2 — Hacer los ejercicios de las secciones 4.7 Funciones útiles de vectores y 4.8 Uso de funciones útiles de vectores.

Problema 4.2 Matrices: haga una función que sume un valor escalar a una matriz, retornando el resultado en una segunda matriz:

```
#define N 3
#define M 3

void sumaEscalar(int mat[N][M], int escalar, int resultado[N][M])
{
    int fila, col;

    for(fila = 0; fila < N; fila++)
    {
        for(col = 0; col < M; col++)
        {
            resultado[fila][col] = mat[fila][col] + escalar;
        }
    }
}
```

Ejemplo de código 4.12: Ejemplo sobre matrices

En la declaración de la función observamos varias cosas:

```
void sumaEscalar(int mat[N][M], int escalar, int resultado[N][M])
```

La función es de tipo procedimiento ya que no retorna ningún valor (es de tipo void). Esta función se llama sumaEscalar, y recibe tres parámetros. El primero y el tercero son matrices enteras de N filas por M columnas. El segundo parámetros es un número entero.

Observe la declaración del parámetro mat: si un parámetro es una matriz SIEMPRE se debe indicar el tamaño de filas y columnas (entre corchetes). Nota: en realidad el tamaño de filas puede o no indicarse (quedando esta declaración optativa así: int mat[][M]), pero a los efectos de este curso y para simplificar al alumno, siempre declararemos ambos atributos simultáneamente.

El ejemplo completo, que muestra también como utilizar la función es el siguiente:

```
#include <stdio.h>

#define N 3
#define M 3

void sumaEscalar(int mat[N][M], int escalar, int resultado[N][M])
{
    int fila, col;
```

```
for(fila = 0; fila < N; fila++)
{
    for(col = 0; col < M; col++)
    {
        resultado[fila][col] = mat[fila][col] + escalar;
    }
}

void mostrarMatriz(int mat[N][M])
{
    int fila, col;

    for(fila = 0; fila < N; fila++)
    {
        for(col = 0; col < M; col++)
        {
            printf(" %d ", mat[fila][col]);
        }
        printf("\n");
    }
    printf("\n");
}

int main()
{
    int a = 5;
    int m1[N][M] = {
        { 1, 2, 3 },
        { 4, 5, 6 },
        { 7, 8, 9 },
    };
    int m2[N][M];

    sumaEscalar(m1, a, m2);
    mostrarMatriz(m2);

    return 0;
}
```

Ejemplo de código 4.13: Ejemplo completo

En el ejemplo se ve también como implementar una función para imprimir una matriz.

En este ejemplo se ve también que, en el caso de matrices, al modificar el valor del parámetro dentro de la función, SI se modifica la matriz original (es el efecto explicado en pasaje de parámetros por referencia). Pero recuerde que C no tiene pasaje de parámetros por referencia, sino que este efecto se debe a que las matrices se pasan por parámetro como punteros (como ya se mencionó, esto no se verá en este capítulo).

Esta última observación (al modificar el valor del parámetro también se modifica la variable original) aplica tanto para matrices como para vectores (y por lo tanto cadena de caracteres).

4.1.7 Pasaje de parámetros referencia / valor – síntesis

El pasaje de parámetros de datos nativos (char, int, float, double, etc) es siempre por valor: la variable original mantiene su valor aunque el parámetro se modifique en la función.

El pasaje de parámetros de vectores, cadenas y matrices es siempre por puntero: al modificar el valor de una posición en el vector, cadena o matriz, se modifica la original.

4.1.8 Ejemplos

Queda para el alumno hacer algunos ejemplos de funciones indicados a continuación y que se encuentran en el práctico:

1. hallar el valor máximo de un vector
2. hallar el valor máximo de una matriz
3. hallar el valor mínimo de un vector
4. hallar el valor mínimo de una matriz
5. contar la cantidad de ocurrencias de un número dado en un vector
6. contar la cantidad de ocurrencias de un número dado en una matriz
7. sumar dos vectores en un tercer vector resultado
8. sumar dos matrices en una tercer matriz resultado
9. multiplicar dos matrices en una matriz resultado

Ejercitar 4.3 — Hacer los ejercicios de las secciones 4.9 Funciones útiles de matrices y 4.10 Uso de funciones útiles de matrices. ■

4.2 Alcance de una variable

El alcance de una variable determina el ámbito en el que está puede ser llamada. Así tenemos variables locales y globales.

Hasta ahora todas las variables que vimos son locales: todas están declaradas dentro de una función (por ejemplo, dentro de la función main, o de la función sumarVector, etc). Decimos que el ámbito de estas funciones es el ámbito de la función: estas variables pueden ser utilizadas únicamente dentro de la función en donde se declaran. Diferentes funciones que declaren variables del mismo nombre hacen referencia a variables diferentes. Estas funciones viven únicamente dentro de la función en donde se declaran.

Pero también podemos tener variables que puedan ser usadas dentro de todo un programa, o dentro de una serie de funciones (a partir de que estas variables son declaradas). Estas variables se llaman variables globales. La forma de usar variables globales se ilustra en el siguiente ejemplo:

```
#include <stdio.h>

int var;

void incrementar()
{
    var = var + 1;
}

int main()
{
    var = 3;
    printf("var = %d\n", var);

    incrementar();
    printf("var = %d\n", var);

    incrementar();
    printf("var = %d\n", var);

    return 0;
}
```

Ejemplo de código 4.14: Variables globales

En el ejemplo se observa que la variable var se declara fuera de todas las funciones, no pertenece a ninguna, sino que es global. Esta variable se usa entonces en la función incrementar y en la función

main.

Preferimos las variables locales frente a las variables globales. Las variables globales se prestan a confusión, y hace que el código de algunas funciones dependan del código de otras funciones. Esto suele conducir a defectos en los programas. En el curso vamos a evitar el uso de variables globales.

Ejercitar 4.4 — Hacer los ejercicios de la sección 4.11 Alcance de variables. ■

4.3 Recursión

¿Pero qué sucedería si una función se llamara a si misma? La respuesta es recursión. Esta es una técnica de programación utilizada para resolver problemas del tipo factorial, series, etc.

Pero para que una función se pueda llamar así misma es necesario que cumpla dos características: 1. debe existir un caso, llamado caso base, por el que el procedimiento no se llama así mismo y finaliza (evitando llamarse así mismo infinitas veces); 2. cada vez que el procedimiento se llame a si mismo (directa o indirectamente), debe estar más cerca del criterio base.

Veamos en un único programa varios ejemplos de funciones recursivas. En el siguiente ejemplo se muestra como implementar una función que muestre una cuenta regresiva, tanto en forma recursiva como en forma iterativa. También se ilustra como hallar un número factorial de forma recursiva y una sumatoria de manera recursiva. Y por último se ejemplifica el uso de la recursión para hallar series, mostrando la serie de Fibonacci:

```
#include <stdio.h>

/* Funcion que calcula e imprime una cuenta regresiva de forma recursiva */
void cuentaRegresivaRecursiva(int count)
{
    printf("%d\t", count);

    /* Siempre tienen que tener una condicion de fin,
    que tiene relacion a la variable recibida. */
    if (count != 0)
    {
        /* Vuelve a llamarse a si misma, por eso se dice que es recursiva */
        cuentaRegresivaRecursiva(count - 1);
    }
}

/* Funcion que calcula e imprime una cuenta regresiva de forma iterativa */
void cuentaRegresivaIterativa(int count)
{
    int i;

    /* Iterador for, por eso se dice que es iterativa. */
    for(i = count; i > 0; i--)
    {
        printf("%d\t", i);
    }
}

int factorial(int n)
{
    if (n == 1) /* caso base */
    {
        return(1);
    }
}
```

```
    return(n * factorial(n - 1)); /* recursion */
}

int sumatoria(int n)
{
    if( n == 1 )
    {
        return 1;
    }

    return n + sumatoria(n - 1);
}

int fibonacci(int n)
{
    if (n == 0) /* primer caso base */
    {
        return(0);
    }

    if ( n==1 ) /* segundo caso base */
    {
        return(1);
    }

    return(fibonacci(n-1) + fibonacci(n-2));
}

int main()
{
    int sum, fac, fib, nro;

    printf("Ingrese Numero: \n");
    scanf("%d", &nro);

    cuentaRegresivaRecursiva(nro);

    printf("\n\n");

    sum = sumatoria(nro);
    fac = factorial(nro);
    fib = fibonacci(nro);

    printf("Sumatoria(%d): %d\n", nro, sum);
    printf("Factorial(%d): %d\n", nro, fac);
    printf("Fibonacci(%d): %d \n", nro, fib);

    return(0);
}
```

Ejemplo de código 4.15: Ejemplo de funciones recursivas

Cambie el ejercicio de manera que se obtenga un único resultado dada una operación elegida por el operador. De esta manera, el operador debe ingresar dos números: primero la operación, y luego el número base para la operación: • Si el operador ingresa 0 – salir del programa • Si el operador ingresa 1 – Sumatoria • Si el operador ingresa 2 – Factorial • Si el operador ingresa 3 – Fibonacci

4.4 Uso de parámetros en la función main

La función main recibe dos parámetros del sistema operativo al ser ejecutado. El primero es un entero que indica la cantidad de parámetros recibidos. El segundo es un vector de punteros a cadenas de caracteres con cada uno de los parámetros recibidos. El siguiente ejemplo ilustra esto:

```
#include <stdio.h>

int main(int n, char *argv[]) {
    printf("Cantidad de parametros: %d\n", n);

    for(int i = 0; i < n; i++) {
        printf("argv[%d] = %s\n", i, argv[i]);
    }
    return 0;
}
```

Ejemplo de código 4.16: Impresión de parámetros

Al ejecutar este programa con distinta combinación de parámetros se obtendrán distintos resultados. Ejemplos:

```
prompt> main.exe -x -param2
```

Cantidad de parametros: 3

```
argv[0] == "main.exe"
argv[1] == "-x"
argv[2] == "-param2"
```

```
main.exe -param1 -param2 param3 param4 a b c
```

Cantidad de parametros: 8

```
argv[0] == "main.exe"
argv[1] == "-param1"
argv[2] == "-param2"
argv[3] == "param3"
argv[4] == "param4"
argv[5] == "a"
argv[6] == "b"
argv[7] == "c"
```

4.5 Guía práctica

4.6 Funciones útiles generales

1. Escriba la función `int leerEntero(char mensaje[])` que imprima el mensaje en la pantalla, lea un entero por teclado y lo retorne. Haga un programa main que utilice dicha función para inicializar un vector de 5 edades de árboles y luego los muestre.
2. Escriba la función `float leerFloat(char mensaje[])` que imprima el mensaje en la pantalla, lea un float por teclado y lo retorne. Haga un programa main que utilice dicha función para inicializar un vector de 5 diámetros de árboles y luego los muestre.
3. Escriba la función `char leerChar(char mensaje[])` que imprima el mensaje en la pantalla, lea un char por teclado y lo retorne. Haga un programa main que utilice dicha función para inicializar un vector de 5 caracteres y luego los muestre.
4. Escriba la función `void leerCadena(char mensaje[], char cadena[], int largoMax)` que imprima el mensaje en la pantalla, lea una cadena por teclado y la retorne en el parámetro `cadena`. Haga un programa main que utilice dicha función para inicializar un vector (¿matriz?) de 5 nombres y apellidos de personas y luego los muestre.

5. Escriba la función `int azar(int min, int max)` que retorne número al azar entre `min` y `max`. Haga un programa `main` que inicialice un vector de 5 elementos con valores al azar entre 2 y 6, y luego los muestre.
6. Escriba una función que retorne la suma de 3 enteros pasados por parámetro. Utilícela para sumar los valores (a) 2, 3 y 4; (b) 3, 4 y 5; (c) -2, -1 y -3; (d) el resultado de las 3 sumas anteriores.

4.7 Funciones útiles de vectores

7. Escriba la función (procedimiento) `void mostrarVectorEnteros(int vector[], int largo)` que muestre el vector en la pantalla. Haga un programa `main` que utilice dicha función para imprimir un vector inicializado estáticamente con 5 números.
8. Escriba la función (procedimiento) `void mostrarVectorFloats(float vector[], int largo)` que muestre el vector en la pantalla. Haga un programa `main` que utilice dicha función para imprimir un vector inicializado estáticamente con 5 números.
9. Escriba la función (procedimiento) `void mostrarVectorChars(char vector[], int largo)` que muestre el vector en la pantalla. Haga un programa `main` que utilice dicha función para imprimir un vector inicializado estáticamente con las 5 letras vocales.
10. Escriba la función `int buscarMayorEntero(int vector[], int n)` que retorne el número mayor del vector. Haga un programa `main` que inicialice un vector de 5 elementos al azar y utilice dicha función para mostrar el mayor.
11. Escriba la función `float buscarMayorFloat(float vector[], int n)` que retorne el número mayor del vector. Haga un programa `main` que inicialice un vector de 5 elementos al azar y utilice dicha función para mostrar el mayor.
12. Escriba la función `int buscarMenorEntero(int vector[], int n)` que retorne el número menor del vector. Haga un programa `main` que inicialice un vector de 5 elementos al azar y utilice dicha función para mostrar el menor.
13. Escriba la función `float buscarMenorFloat(float vector[], int n)` que retorne el número menor del vector. Haga un programa `main` que inicialice un vector de 5 elementos al azar y utilice dicha función para mostrar el menor.
14. Escriba la función `int sumarVectorEnteros(int vector[], int n)` que retorne la suma de todos los elementos del vector. Haga un programa `main` que inicialice un vector de 5 elementos al azar y utilice dicha función para mostrar la suma.
15. Escriba la función `float sumarVectorFloats(float vector[], int n)` que retorne la suma de todos los elementos del vector. Haga un programa `main` que inicialice un vector de 5 elementos al azar y utilice dicha función para mostrar la suma.
16. Escriba la función `int contarEnVectorEnteros(int vector[], int n, int buscar)` que retorne la cantidad de veces que encontró el número `buscar` en el vector. Haga un programa `main` que inicialice un vector de 5 elementos con los números 1, 1, 2, 2, 2 y utilice dicha función para mostrar la cantidad de veces que aparece el número 1, el 2 y el 3.

4.8 Uso de funciones útiles de vectores

17. Escriba un programa que pida nombre científico, edad y circunferencia de 5 árboles. Almacene los datos en vectores paralelos. Indique el nombre y edad del árbol más ancho, y el nombre y circunferencia del árbol más viejo.

4.9 Funciones útiles de matrices

18. Escriba la función (procedimiento) `void mostrarMatrizEnteros(int matr[N][M])` que muestre la matriz en la pantalla. Haga un programa `main` que utilice dicha función para imprimir una

matriz N=3 y M=4 inicializada estáticamente con valores elegidos por el alumnos.

19. Escriba la función (procedimiento) void mostrarMatrizFloats(matriz[N][M]) que muestre la matriz en la pantalla. Haga un programa main que utilice dicha función para imprimir una matriz N=3 y M=4 inicializada estáticamente con valores elegidos por el alumno.
20. Escriba la función (procedimiento) void mostrarMatrizChars(char matr[N][M]) que muestre la matriz en la pantalla. Haga un programa main que utilice dicha función para imprimir una matriz N=3 y M=4 inicializada estáticamente con valores elegidos por el alumno.
21. Escriba la función int buscarMayorMatriz(int matr[N][M]) que retorne el número mayor de la matriz. Haga un programa main que inicialice una matriz N=3 y M=4 con elementos al azar y utilice dicha función para mostrar el mayor.
22. Escriba la función int buscarMenorMatriz(int matriz[N][M]) que retorne el número menor de la matriz. Haga un programa main que inicialice una matriz N=3 y M=4 con elementos al azar y utilice dicha función para mostrar el menor.
23. Escriba la función (procedimiento) void sumarMatrizEscalar(int matriz[N][M], int escalar, int result[N][M]) que sume un escalar en una matriz, guardando los resultados en la matriz result. Escriba un programa main que inicialice dos matrices N=3 y M=4 con números al azar entre 1 y 9), y que utilice la función para
 - a. sumar un valor escalar ingresado por teclado a la primer matriz, guardando el resultado en una tercer matriz,
 - b. sumar el valor ingresado por teclado a la segunda matriz, guardando el resultado en la misma matriz,
 - c. mostrar las 3 matrices.
24. Escriba la función (procedimiento) void sumarMatriz (int m1[N][M], int m2[N][M], int result[N][M]) que sume dos matrices, guardando los resultados en la matriz result. Escriba un programa main que inicialice dos matrices N=3 y M=4 con números al azar entre 1 y 9), y que luego las sume. Mostrar las 3 matrices.
25. Escriba la función int obtenerSumaMatriz (int m1[N][M]) que sume todos los elementos de la matriz y la retorne. Escriba un programa main que inicialice una matriz N=3 y M=4 con números al azar entre 1 y 9, y que luego obtenga la suma. Mostrar también la matriz.
26. Escriba la función int contarEnMatriz(int mat[N][M], int buscar) que retorne la cantidad de veces que encontró el número buscar en la matriz. Haga un programa main que inicialice una matriz N=3 y M=4 con números al azar entre 1 y 9. Utilice dicha función para mostrar la cantidad de veces que aparece un número ingresado por teclado. Mostrar la matriz.
27. Escriba la función int buscaMayorDiagonalPrincipal(int matr[N][N]) que retorne el número mayor de la diagonal principal de la matriz. Haga un programa main que inicialice una matriz N=3 con elementos al azar y utilice dicha función para mostrar el mayor de la diagonal.
28. Escriba la función int buscaMenorDiagonalSecundaria(int matr[N][N]) que retorne el número mayor de la diagonal secundaria de la matriz. Haga un programa main que inicialice una matriz N=3 con elementos al azar y utilice dicha función para mostrar el mayor de la diagonal.

4.10 Uso de funciones útiles de matrices

29. Llenar 3 matrices de 4x4 con números al azar (del 0 al 9), mostrarlas y hallar la sumatoria de cada una. Mostrar cada resultado.

4.11 Alcance de variables

30. Escriba un programa que, utilizando funciones y vectores paralelos como variables globales, pida <número de dni y salario>de 3 pesonas y luego las imprima por pantalla.

4.12 Ejercicios extra

4.12.1 Funciones simples

Dada la siguiente función:

```
int calcular (float x, int y) {  
    int a, b, c;  
    a = (int) (x * 0.5);  
    b = (int) (x + 0.4);  
    c = (a + b) / y;  
    return c;  
}
```

Indicar:

1. Tipo de Retorno.
2. Es una función o procedimiento.
3. Parámetros o argumentos.
4. Tipo de pasaje de cada parámetro (por valor, por referencia, puntero).
5. Variables locales.
6. Tipos de las variables locales.
7. Nombre de la función.
8. Cuerpo de la función.
9. Cuerpo de la función.

4.12.2 Funciones simples

1. Escriba una función que retorne un número al azar entre 2 y 5.
2. Escriba una función que retorne un número al azar entre un mínimo y un máximo dados por parámetro.
3. ¿Cuál de las dos funciones anteriores le parece más útil? ¿Por qué?
4. Escriba una función que pida un número entero entre 2 y 5. Si el valor ingresado es incorrecto vuelva a pedirlo.
5. Escriba una función que pida un número entero entre un mínimo y un máximo dados por parámetro. Si el valor ingresado es incorrecto vuelva a pedirlo.
6. ¿Cuál de las dos funciones anteriores le parece más útil? ¿Por qué?
7. Use las funciones 2 y 5 en un programa.

Nota: resuelva el ejercicio utilizando variables locales y luego repita la solución utilizando variables globales.

4.12.3 Funciones de vectores (y punteros)

Dada la siguiente función:

```
void duplicarVec (float x[], int n) {  
    int i;  
    for (i = 0; i < n; i++) {  
        x[i] *= 2;  
    }  
}
```

Indicar:

1. Firma o Prototipo de la función
2. Tipo de Retorno.
3. Es una función o procedimiento.
4. Parámetros o argumentos.
5. Tipo de pasaje de cada parámetro (por valor, por referencia, puntero).

6. Variables locales.
7. Tipos de las variables locales.
8. Nombre de la función.
9. Cuerpo de la función.

Sólo IE, IT Es lo mismo declarar la función así:

```
void duplicarVec (float *x, int n) {...}
```

4.12.4 Funciones de vectores (y punteros)

1. Escriba un procedimiento que incremente un número pasado por parámetro.
2. Escriba una función que inicialice un vector con números al azar entre un mínimo y un máximo. Utilice las funciones de la sección anterior.
3. Escriba una función que imprima un vector.
4. Escriba una función que retorne la suma de los valores de un vector.
5. Escriba una función que, dado un vector, obtenga un 2do vector que duplique el valor de cada elemento del 1er vector.
6. Escriba un programa que utilice las funciones anteriores.

Nota: resuelva el ejercicio utilizando variables locales y luego repita la solución utilizando variables globales.

4.12.5 Funciones de matrices (y punteros)

Dada la siguiente función:

```
void duplicarMat (float m[FILAS][COL]) {  
    int f, c;  
    for (f = 0; f < FILAS; f++) {  
        for (c = 0; c < COL; c++) {  
            m[f][c] *= 2;  
        }  
    }  
    return;  
}
```

Indicar:

1. Firma o Prototipo de la función.
2. Tipo de Retorno.
3. Es una función o procedimiento.
4. Parámetros o argumentos.
5. Tipo de pasaje de cada parámetro (por valor, por referencia, puntero).
6. Variables locales.
7. Tipos de las variables locales.
8. Nombre de la función.
9. Cuerpo de la función.

Sólo IE, IT Es lo mismo declarar la función así:

```
void duplicarMat (float *x) {...}
```

4.12.6 Ejercicios de matrices

1. Escriba una función que inicialice una matriz con números al azar entre un mínimo y un máximo. Utilice las funciones de secciones anteriores.
2. Escriba una función que imprima una matriz.
3. Escriba una función que retorne la suma de los valores de una matriz.

4. Escriba una función que, dada una matriz, obtenga una 2da matriz que triplique el valor de cada elemento de la 1er matriz.
5. Escriba un programa que utilice las funciones anteriores.

Nota: resuelva el ejercicio utilizando variables locales y luego repita la solución utilizando variables globales.