Licenciatura en Sistemas
Programación de computadoras

Equipo docente: Jorge Golfieri, Natalia Romero, Romina Masilla y Nicolás Perez
Mails: jgolfieri@hotmail.com , nataliab_romero@yahoo.com.ar , romina.e.mansilla@gmail.com, nperez_dcao_smn@outlook.com
Facebook: https://www.facebook.com/groups/171510736842353
Git: ***http://***github.com/UNLASistemasProgramacion/Programacion-de-Computadoras

---

Unidad 4:

Tipos de datos compuestos. Arreglos unidimensionales y bidimensionales. Cadenas: Concepto de cadena, Inicialización, lectura, funciones de cadena, conversión de cadenas a números y viceversa. Algoritmos de ordenación: burbuja, selección e inserción. Problemas: implementación y testeo.

Bibliografía citada:

Fundamentos de programación: libro de problemas. Algoritmos, estructuras de datos y objetos (2a. ed.) Luis Joyanes Aguilar, Luis Rodríguez Baena y Matilde Fernández Azuela. McGraw-Hill España. Capítulo 6-7-10-11.

C algoritmos, programación y estructuras de datos. Luis Joyanes Aguilar, Andrés Castillo Sanz, y Lucas Sánchez García. Capítulo 9-10.

Unidad 4: Arreglos y ordenamiento

Variables de tipo cadena, String en C

Hasta ahora veníamos trabajando con ejercicios donde manipulábamos datos del tipo numérico, ya sean int, float o algún doublé, pero recordemos que la primera clase les nombramos que había otro tipo de datos, los denominados char. Estos nos ayudan a trabajar e incorporar a nuestros códigos palabras, letras o símbolos.

Para **declarar una cadena**, por ejemplo, si queremos declarar una cadena de longitud 20 caracteres se haría:

```
char nombre[20];
```

No podemos entonces introducir más de 20 elementos en la cadena. Vamos a ver un ejemplo para mostrar el nombre del usuario en pantalla:

Nuestro código quedaría así:

```
char nombre[20];

printf( "Introduzca su nombre (20 letras maximo): " );

scanf( "%s", nombre );

printf( "\n El nombre que ha escrito es: %s \n", nombre );
```

Obsérvese que en la sentencia scanf no se usa el símbolo & delante de nombre. No hace falta porque se trata de un *array*, de modo que escribir el nombre del *array* es equivalente a poner &nombre[0].

También puede llamar la atención sobre la forma de imprimir el *array*. Con sólo usar %s ya se imprime su totalidad. Ya veremos esto más adelante.

Muchas veces suele ser útil saber el **tamaño de la cadena**, para eso se utiliza:

printf("Tamaño de la cadena: %i bytes\n", sizeof nombre);

Para conocer el número exacto de caracteres:

longitud = strlen(texto);

Así como antes queríamos darle a un entero otro valor de un entero ya existente, puede que nos interese darle a una cadena el valor de otra cadena, para esto se utiliza la función strcpy:

```
char texto[] = "Éste es un curso de C.";

char destino[50];

strcpy( destino, texto );

printf( "Valor final: %s\n", destino );
```

Para unir, o concatenar, cadenas debemos hacer:

```
char nombre_completo[50];

char nombre[]="Gandalf";

char apellido[]="el Gris";


strcpy( nombre_completo, nombre );

strcat( nombre_completo, " " );
```

```
strcat( nombre_completo, apellido );

printf( "El nombre completo es: %s.\n", nombre_completo );
```

Otra cosa importante que podemos necesitar es comparar cadenas, es decir verificar que son iguales. Para ello usaremos:

```
char nombre1[]="nicolas";

char nombre2[]="jorge";


printf( "Comparación con strcmp: %i\n", strcmp(nombre1,nombre2));
```

Esa comparación nos devuelve un 1, puesto que son distintas las cadenas, si fueran iguales nos darían un 0.

Ejemplo: A modo de ejemplo integrador, escribamos un texto en una cadena y generemos otra cadena con el texto invertido.

```
char cadena[50];

char cadenaInvertida[50];


printf("Ingresa tu nombre: \n");

scanf("%s", cadena);


strcpy(cadenaInvertida, cadena);
```

```
int largoCadena = strlen(cadena);


for (int indice = 0; indice<=largoCadena-1; indice++){
cadenaInvertida[largoCadena-1-indice]= cadena[indice];


}
printf("%s",cadenaInvertida);
```

Ejercicio integrador: WHILE – FOR – IF – CASE – printf – scanf

Se quiere pedir por consola dos números (numero1 y numero2),

ambos enteros... Que se muestre un menú donde:

-si se ingresa el 1, ambos números se suman y se muestra por pantalla

-si se ingresa el 2, ambos números se multiplican

-si se ingresa el 3, se muestra por pantalla el número más chico de ellos

-si se ingresa el 4 se muestran todos los números enteros entre esos dos

números, ejemplo, si se ingrese el número 10 y 15, se muestran el 10,11,12,13,14 y 15

-si se ingresa el 0 se sale del programa

-si se aprieta cualquier otra cosa se lanza un error

¿Cómo podríamos resolverlo?

El programa se cierra al ingresar el numero CERO, por lo que todas las funciones pedidas deben realizarse mientras NO SE INGRESE EL CERO. Esto se haría con un

WHILE, es decir, supongamos que ingresamos opciones por teclado, opciones que serán numéricas (int opción), es decir, Mientras (opción !=0) { realizar el programa…}.

El resto de las opciones, opción = 1, opción =2, opción = 3, opción = 4 podrían hacerse con un IF, pero seria muy repetitivo, por lo que propongo realizarlas con un SWITCH CASE.

Como solo trabajaremos con DOS números, y queremos saber cuál es el más grande y cual el más chico, podríamos realizar un IF ya que solo se compararían ambos. En caso de que sean más los números tendríamos que realizar otro procedimiento, que veremos en las próximas clases.

Y como ya hemos hecho en otras clases, mostrar todos los enteros entre dos números se realizaría por medio de un FOR.

La resolución, de este ejercicio esta a continuación, pero les proponemos que lo ejecuten, lo lean paso a paso, le pongan comentarios y le agreguen una funcionalidad más, es decir opción=5, que sea editar los números numeros1 y numero2, volver al menú y tener las mismas opciones, pero con los nuevos números ingresados. ¿Se animan?

Presten atención a mis comentarios, y estamos a la espera de todas sus consultas.

```
#include <stdio.h>

#include <stdlib.h>



////////////////////////////////////////////////////////////

//Programa: Nuestro primer menu para implementar IF, CASE, FOR

//en un mismo ejercicio
```

```
//////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////
/* Se quiere pedir por consola dos numeros (numero1 y numero2),
ambos enteros... Que se muestre un menu donde:
-si se ingresa el 1, ambos numeros se suman y se muestra por pantalla
-si se ingresa el 2, ambos numeros se multiplican
-si se ingresa el 3, se muestra por pantalla el numero mas chico de ellos
-si se ingresa el 4 se muestran todos los numeros enteros entre esos dos
numeros, ejemplo, si se ingrese el numero 10 y 15, se muestran el 10,11,12,13,14 y 15
-si se ingresa el 0 se sale del programa
-si se aprieta cualquier otra cosa se lanza un error
*/
//////////////////////////////////////////////////////////
int main()
{

    //Defino las variables que voy a usar, respetando el enuncia
    int numero1;
    int numero2;
    int opcionMenuPrincipal;
```

```
    int numeroMasChico;
    int numeroMasGrande;
    int suma;
    int producto;

    //Mostramos una bienvenida
    printf ("---------------------------------------------- \n");
    printf("Bienvenidos a nuestro primer menu con opciones!!!\n");
    printf ("---------------------------------------------- \n");

    //Pedimos por pantalla que se ingresen los dos numeros
    printf ("Primer ingrese el valor del numero 1: \n");
    scanf("%d", &numero1);
    printf ("Primer ingrese el valor del numero 1: \n");
    scanf("%d", &numero2);

    while (opcionMenuPrincipal!=0){ //el programa se mantiene abierto siempre que no sea un 0

    //Pedimos al usuario que elija una opcion del menu
    printf ("Seleccione que quiere realizar \n ");
```

```
    printf ("1- Para sumar \n ");
    printf ("2- Para multiplicar \n ");
    printf ("3- Ver el mas chico \n ");
    printf ("4- Numeros entre \n ");
    printf ("0- Para SALIR \n ");
    printf ("No me gusta que hagan copiar y pegar sin leer y entender el código u.u \n ");



    //Guardamos la opcion ingresa por el usuario
    scanf("%d", &opcionMenuPrincipal);



    //Antes de ingresar a las opciones calculamos cual es el numero mas grande y
    //cual el mas chico
    if (numero1>=numero2) { //Abre el if
                numeroMasChico = numero2;
                numeroMasGrande = numero1;

        } //Cierra el if

    if (numero1<numero2) { //Abre el if
                numeroMasChico = numero1;
                numeroMasGrande = numero2;
```

```
        } //Cierra el if



    //Entramos a los casos con un switch
    switch (opcionMenuPrincipal)    {

     case 1:  //Caso que suma

         suma = numero1 + numero2;
         printf("La suma me da: %d \n ", suma);
         break;

     case 2:  //Caso que multiplica
         producto = numero1*numero2;
         printf("El producto me da: %d \n ", producto);
         break;

     case 3: //El mas chico de todos
        printf("EL numero mas chico es el: %d \n", numeroMasChico);
        break;
```

```
    case 4: //Mostramos los numeros del mas chico al mas grande
        printf("Los numeros entre el mas chico y el mas grande son: \n");
        int i = numeroMasChico;
        for (i=numeroMasChico; i<=numeroMasGrande; i++){
            printf("---->  %d", i);
        };
        break;

    case 0:
        return 0;
        break;

    default:
        printf("\n La opcion no es correcta, lo siento U___U \n");

   }//Cierro el SWITCH

   }//Cierra el while

   return 0;
}
```

¿Has comprendido el ejemplo del menú? Para saberlo te dejamos este ejercicio para que practiques.

Ejercicio de practica integrador: Realicen un menú semejante al anterior, donde se salga con el cero nuevamente, donde las opciones del 1 a 5 sean el cuadrado, triangulo, triangulo invertido, pirámide y pirámide invertida. Pero el tamaño de las figuras sea elegido por pantalla, es decir se tiene que poder elegir la cantidad de renglones y de columnas de las figuras. Las figuras no deben ser estáticas. Si se ingresa por pantalla un numero negativo debe salir la frase "No se puede dibujar ninguna figura así". Una vez realizado hagan una prueba de escritorio.

Arrays, vectores o arreglos

Repitiendo, y casi aburriendo al lector, recordemos los principales tipos de datos que estuvimos manejando hasta ahora. Vimos primero los int, que son números enteros que van desde los negativos a los positivos, los famosos números naturales con el signo mas y menos. Luego vimos los float, o flotantes, así relacionan la materia con Organización de Computadoras, que son números tanto positivos como negativos, pero con números decimales. Y los Doubles que si bien no los manejamos mucho son como los float solo que permiten números mas grandes y mas pequeños.

Luego vimos los char, que son para guardar o trabajar con letras o simbolos, por ejemplo, char letra = 'a'. Fue entonces cuando nos dimos cuenta que si queríamos trabajar con palabras necesitábamos MUCHAS letras, entonces en ese instante fue cuando les mencionamos la existencia de los char[], es decir colección de letras en algún orden especifico. Esto que por ahora era casi un mandato divino ahora lo estudiaremos con profundidad y diremos que esto es un ARRAY, VECTOR o ARREGLO. Por ejemplo, lo que ya trabajamos, char palabra[50] no es otra cosa mas que un arreglo de 50 caracteres para escribir una palabra, o sea una palabra compuesta por COMO MUCHO 50 símbolos.

Arrays, vectores o arreglos

Repitiendo, y casi aburriendo al lector, recordemos los principales tipos de datos que estuvimos manejando hasta ahora. Vimos primero los int, que son números enteros que van desde los negativos a los positivos, los famosos números naturales con el signo mas y menos. Luego vimos los float, o flotantes, así relacionan la materia con Organización de Computadoras, que son números tanto positivos como negativos, pero con números decimales. Y los Doubles que si bien no los manejamos mucho son como los float solo que permiten números mas grandes y mas pequeños.

Luego vimos los char, que son para guardar o trabajar con letras o simbolos, por ejemplo, char letra = 'a'. Fue entonces cuando nos dimos cuenta que si queríamos trabajar con palabras necesitábamos MUCHAS letras, entonces en ese instante fue cuando les mencionamos la existencia de los char[], es decir colección de letras en algún orden especifico. Esto que por ahora era casi un mandato divino ahora lo estudiaremos con profundidad y diremos que esto es un ARRAY, VECTOR o ARREGLO. Por ejemplo, lo que ya trabajamos, char palabra[50] no es otra cosa mas que un arreglo de 50 caracteres para escribir una palabra, o sea una palabra compuesta por COMO MUCHO 50 símbolos.

Por ejemplo: int palabra[10] = "Nicolas";

Esta es una palabra formada como mucho con 10 caracteres. En memoria guardamos el valor "Nicolas".

¿Cómo hacemos para conseguir, o mostrar por pantalla la letra N, es decir la primera letra de mi arreglo? Muy fácil, las posiciones van desde el 0 al 9, es decir 10 posiciones.

Si queremos mostrar por pantalla la N, necesitamos decir que quiero el char de la primera posición de mi palabra. Esto se escribe así: **palabra[0];**

Entonces para mostrarlo: printf ("La primer letra es %c: ", palabra[0]);

Si quisiéramos conseguir la "o" de nuestra palabra, printf ("%c", palabra[2]);

Ejemplo: Mostraremos todas las letras de mi palabra, una por una mostrando además en que posición esta.

```
char palabra[25] = "Nico es un mal docente";


  for (int i = 0; i<=25; i ++){

     printf("En la posicion %d, tenemos al char: %c", i, palabra[i]);
```

```
        printf("\n");
    }
```

Obtenemos por pantalla lo siguiente:

En la posicion 0, tenemos al char: N

En la posicion 1, tenemos al char: i

En la posicion 2, tenemos al char: c

En la posicion 3, tenemos al char: o

En la posicion 4, tenemos al char:

En la posicion 5, tenemos al char: e

En la posicion 6, tenemos al char: s

En la posicion 7, tenemos al char:

En la posicion 8, tenemos al char: u

En la posicion 9, tenemos al char: n

En la posicion 10, tenemos al char:

En la posicion 11, tenemos al char: m

En la posicion 12, tenemos al char: a

En la posicion 13, tenemos al char: l

En la posicion 14, tenemos al char:

En la posicion 15, tenemos al char: d

En la posicion 16, tenemos al char: o

En la posicion 17, tenemos al char: c

En la posicion 18, tenemos al char: e

En la posicion 19, tenemos al char: n

En la posicion 20, tenemos al char: t

En la posicion 21, tenemos al char: e

En la posicion 22, tenemos al char:

En la posicion 23, tenemos al char:

En la posicion 24, tenemos al char:

En la posicion 25, tenemos al char: \0

Notar que en la posición i = 25, se muestra un \0 es decir FINAL DE LA CADENA, así que desde ahora para evitar problemas hagamos los for hasta i<25.

Operaciones con vectores

Ahora que ya sabemos recorrer los vectores o arreglos, (¿Sabemos?), hagamos unos ejercicios para manipularlos bien y entrar en confianza, para esto hagamos ejemplos de vectores, donde los vectores son colecciones de números enteros.

Los vectores no son solo un invento en el ámbito de la informática y programación, los vectores son un tipo de magnitud, por ejemplo, el viento es un vector, la velocidad es un vector, es decir tienen coordenadas. Los vectores, así como los números se pueden sumar, restar, multiplicar y tienen otras operaciones. Usemos estas operaciones para entrenarnos en la utilización de arreglos.

Vamos a definir dos vectores de 5 coordenadas, y vamos a sumarlos.

int vector1[5] = {1,2,6,7,-2};

int vector2[5] = {2,2,2,5,6};

Ejemplo: ¿Cómo es la suma de vectores? Es muy simple, es coordenada a coordenada, y el resultado da otro vector, es decir la suma nos va a devolver un vector3.

El vector 3 será {1+2,2+2,6+2,7+5,-2+6}. ¿Cómo escribimos esto en C?. De paso utilicemos While, así seguimos repasándolo (Luego le proponemos que lo hagan ustedes con un FOR).

```
int vector1[5] = {1,2,6,7,-2};
```

```
int vector2[5] = {2,2,2,5,6};
int vector3[5];
int coordenada = 0;

while(coordenada<5){
    //Cuando la coordenada es 0, guardamos en la primer posicion de
    //vector3 la suma de las primeras coordenada de vector1 y vector2
    //esto se realiza mientras la coordenada sea menor que 5
    vector3[coordenada] = vector1[coordenada] +vector2[coordenada];
    coordenada = coordenada + 1;
}

//ya que estamos mostremos el resultado del vector3 para verificar que todo está bien
for (int i = 0; i<5; i++){
    printf("%d   ", vector3[i]);
}
```

Ejercicio: Realicen la resta de vectores, sabiendo que se obtiene un vector4 que es la resta coordenada a coordenada. Dados V1 = (1,2,3,4,5) y V2 = (-2,-3,4,7,0). Encuentre el producto escalar programando el algoritmo.

Ejemplo: También existe el producto, que se lo llama producto escalar, se multiplica coordenada a coordenada, pero luego todos los productos se suman. Es decir, el resultado no es un vector vector3, es un valor entero.

Si bien esto es matemática y todos ustedes se que la aman, la idea es que solo nos enfoquemos en el algoritmo que lo resuelve, así que aquí tienen un ejemplo de un producto escalar de vectores. Véanlo, compréndanlo, traten de resolverlo y luego miremos el código de continuación.

Ejemplo:

$$\vec{u} = (3,\ 0) \qquad \vec{v} = (5,\ 5)$$

$$\vec{u} \cdot \vec{v} = 3 \cdot 5 + 0 \cdot 5 = 15$$

Solución:

```
int v1[5] = {1,2,3,4,5};

int v2[5] = {-2,-3,4,7,0};

int resultado = 0;


int coordenada = 0;



while(coordenada<5){

 resultado = resultado + (v1[coordenada]*v2[coordenada]);
```

```
    coordenada = coordenada + 1;

    }


    printf("El resultado es:  %d ", resultado);
```

Teoría útil: El producto escalar da una relación entre el ángulo de los vectores, se sabe que si el producto escalar da cero los vectores son perpendiculares, es decir su ángulo es de 90°. O mejor dicho perpendiculares, porque se aplica a cualquier dimensión.

Ejercicio 2: Este podría ser un ejercicio de parcial, traten de resolverlo, con lo que sabemos hasta ahora no debería ser complicado. Se cargan 4 vectores, int vector1, int vector2, int vector3 e int vector4. El usuario ingresa por consola el largo de los vectores, por ejemplo, si el usuario ingresa un 3, los vectores tendrán 3 coordenadas. Una vez que el usuario ingresa el tamaño, debe cargar por teclado cada uno de los 4 vectores. Cuando ya tenemos los 4 vectores cargados en memoria, devolver por pantalla que vectores son perpendiculares, y si ninguno de ellos son perpendiculares devolver por pantalla “Ninguno es perpendicular”. Estamos a su disposición para consultas.

Ejercicio 3a: Se ingresa por teclado tu nombre, tu apellido, tu dni y el nombre de un docente. Con todo esto se pide formar y mostrar por pantalla una patente de un vehículo argentino, es decir, de la forma WX 123 YZ, donde la W es la primer letra de tu nombre, la X la primer nombre de tu apellido, el 123 son los 3 últimos números de tu dni, y el YZ son las dos primeras letras del docente.

Ejemplo: Nicolas, Perez, 123456, Jorge. Patente: NP 123 JO

Matrices: Vectores de Vectores

Como vimos anteriormente se puede guardar en memoria una colección de enteros, char, float, doubles en un mismo tipo de dato, a estos datos se los llaman vectores o array. Por ejemplo, int vect[5] es una colección de 5 valores enteros en la variable vect; char palabra[40] es una colección de 40 caracteres guardados en la variable palabra. O también float costos[10] son 10 números decimales guardados en el dato "costos".

Ahora bien, estamos guardando los datos en una única dimensión, ¿qué pasa si hacemos vectores de vectores?, esto es trabajar en dos o más dimensiones, y a dicho trabajo se lo llama Matrices o arreglos multidimensionales.

Por ejemplo int matriz[2][3] es una variable llamada matriz que guarda un total de 6 datos enteros, la matriz seria de la pinta:

2 3

1 0

0 9

Si prestamos atención, no es más que tres vectores de vectores de dos números, donde cada renglón sería un vec[2] y tenemos 3 renglones, por eso se le agrega el [3] a la matriz.

Ejemplo 1:

A modo de ejemplo ilustrativo, Vamos a definir una matriz donde queremos almacenar el número de alumnos con que cuenta una academia ordenados en función del nivel y del idioma que se estudia. Tendremos 3 filas que representarán Nivel básico, medio o de perfeccionamiento y 4 columnas que representarán los idiomas (0 = Inglés, 1 = Francés, 2 = Alemán y 3 = Ruso). La declaración de dicha matriz sería:

int alumnosfxniveleidioma [3] [4];

Podríamos asignar contenidos de la siguiente manera:

alumnosfxniveleidioma[0] [0] = 7; alumnosfxniveleidioma[0] [1] = 14;

alumnosfxniveleidioma[0] [2] = 8; alumnosfxniveleidioma[0] [3] = 3;

alumnosfxniveleidioma[1] [0] = 6; alumnosfxniveleidioma[1] [1] = 19;

alumnosfxniveleidioma[1] [2] = 7; alumnosfxniveleidioma[1] [3] = 2;

alumnosfxniveleidioma[2] [0] = 3; alumnosfxniveleidioma[2] [1] = 13;

alumnosfxniveleidioma[2] [2] = 4; alumnosfxniveleidioma[2] [3] = 1;

La representación gráfica que podríamos asociar a esta asignación de datos sería esta matriz:

$$\begin{pmatrix} 7 & 14 & 8 & 3 \\ 6 & 19 & 7 & 2 \\ 3 & 13 & 4 & 1 \end{pmatrix}$$

Ejemplo 2: Guardar un único valor en una ubicación determinada.

```
#include <stdio.h>
#include <stdlib.h>

int main() {
int habitantesVivienda[5][25];
```

```
habitantesVivienda[3][24] = 4;

printf("El numero de personas que viven en la vivienda 24 del piso 3 es %d\n",
habitantesVivienda[3][24]);

return 0;
}
```

Ejercicio 3 – Clase Práctica y Teórica: Se tienen 6 artículos y 4 trimestres, se pide guardar en una matriz la cantidad de artículos vendidos por trimestres. El ejercicio será separado en módulos para irlo atacando paso a paso.

- Declarar las variables a utilizar
- Cargar los datos en la matriz, por consola con printf y scanf
- Calcular los totales por trimestres y por artículos, guardándolos en vectores.
- Mostrar datos
- Mostrar totales
- Mostrar promedios trimestrales y por artículos
- Mostrar articulo más vendido y menos vendido
- Mostrar artículos más y menos vendidos por trimestres.

Este ejercicio es muy integrador y complejo, se pretende que ustedes traten de resolverlo por sus propios medios y calculen muchas más cosas de las que se piden en el enunciado. A continuación, se adjunta una posible solución, no se queden con ella, traten de entenderla y hacer si propia solución. Además, traten de comentar esta solución.

Respuesta:

```
// Carga de matrices y cálculo de totales

#include <stdio.h>

#include <stdlib.h>


int main()

{

// Declaración de variables

    int i = 0;

    int j = 0;


    int vec[4][6];



    int ttrim[4]= {0,0,0,0};

    int tart[6] = {0,0,0,0,0,0};


// Carga de Datos

    fflush(stdin);

    printf("Ingrese los numeros para la matriz: \n");

    for(i=0;i<4;i++){

        for(j=0;j<6;j++){

            printf("Ingrese el trimestre %d y articulo %d: ", i, j);

            scanf("%d",&vec[i][j]);
```

```
        }
    }

// Calculo los totales
    for(i=0;i<4;i++){
        for(j=0;j<6;j++){
            tart[j] = tart[j] + vec[i][j];
            ttrim[i] = ttrim[i] + vec[i][j];
        }
        printf("\n");
    }

// Muestro datos
    system("CLS");

    printf("\n Muestro la matriz con los totales: \n");

    for(i=0;i<4;i++){
        for(j=0;j<6;j++){
            printf(" %d ",vec[i][j]);
        }
        printf(" %d \n",ttrim[i]);
    }
```

```
// Muestro el total de articulos
    for(j=0;j<6;j++){
        printf(" %d ",tart[j]);
    }

// Muestro el promedio de los articulos
    printf("\nPromedio de los articulos:\n");
    for(j=0;j<6;j++){
        printf("\nEl promedio del articulo %d es %d ",j, tart[j]/4);
    }
    printf("\n");

// Muestro el promedio de los trimestres
    printf("\nPromedio de los trimestres:\n");

    for(i=0;i<4;i++){
        printf("\nEl promedio del trimestre %d es %d ",i, ttrim[i]/6);
    }

//Calculo el articulo mas vendido (total)
    int masvendido = -9999;
    for(j=0;j<6;j++){
```

```
        if(tart[j]>masvendido){
            masvendido = tart[j];
        }
    }
    printf("\nEl articulo mas vendido (total) es %d ",masvendido);

//Calculo el articulo menos vendido (total)
    int menosvendido = 9999;
    for(j=0;j<6;j++){
        if(tart[j]<menosvendido){
            menosvendido = tart[j];
        }
    }
    printf("\nEl articulo menos vendido (total) es %d ",menosvendido);

//Calculo el articulo mas vendido en cualquier trimestre
    int masvendidotodos = -9999;
    for(i=0;i<4;i++){
        for(j=0;j<6;j++){
            if(vec[i][j]>masvendidotodos){
                masvendidotodos = vec[i][j];
            }
        }
```

```
    }
    printf("\nEl articulo mas vendido de cualquier trimestre es %d ",masvendidotodos);

    return 0;
}
```

Ejercicio 4: Propuesto

I. Desarrolle un programa que rellene una matriz 5 * 3. Luego, en otra matriz del mismo tamaño, guarde los valores almacenados en la primera matriz elevando al cubo los almacenados en columnas pares y elevando al cuadrado los almacenados en posiciones impares.

Nota: imprimir las dos matrices.

II. Queremos almacenar en una matriz las notas de informática de los alumnos de secundaria de una escuela. Suponiendo que hay 4 grados distintos, 15 alumnos por grado, se pide:

a. Ingresar las notas que ha sacado cada alumno de cada grado.

b. Imprimir cuál es la nota promedio de cada grado.

c. Imprimir la mayor nota en general.

d. Imprimir todas las notas.

e. Imprimir cuántos alumnos aprobaron y cuántos reprobaron de cada grado.

Ejercicio 5: Realizar un programa que solicite una matriz A y una matriz B por teclado y almacene sus valores, teniendo en cuenta que las matrices han de ser de 3×4.

El programa ha de presentar el resultado de la suma, resta y traspuestas de las matrices.

Ayuda sobre operaciones en matrices:

$$A=\begin{pmatrix}2 & 0 & 1\\ 3 & 0 & 0\\ 5 & 1 & 1\end{pmatrix} \quad B=\begin{pmatrix}1 & 0 & 1\\ 1 & 2 & 1\\ 1 & 1 & 0\end{pmatrix}$$

$$A+B=\begin{pmatrix}2+1 & 0+0 & 1+1\\ 3+1 & 0+2 & 0+1\\ 5+1 & 1+1 & 1+0\end{pmatrix}=\begin{pmatrix}3 & 0 & 2\\ 4 & 2 & 1\\ 6 & 2 & 1\end{pmatrix}$$

$$A-B=\begin{pmatrix}2-1 & 0-0 & 1-1\\ 3-1 & 0-2 & 0-1\\ 5-1 & 1-1 & 1-0\end{pmatrix}=\begin{pmatrix}1 & 0 & 0\\ 2 & -2 & -1\\ 4 & 0 & 1\end{pmatrix}$$

$$A_{2\times 3}=\begin{pmatrix}-1 & 2 & 1\\ 3 & 3 & 5\end{pmatrix} \Rightarrow A^T=\begin{pmatrix}-1 & 3\\ 2 & 3\\ 1 & 5\end{pmatrix}$$

Modificación: Hagamos lo mismo, pero utilizando el concepto de funciones y procedimientos explicados en la unidad 2.

Respuesta:

Para resolver esto comprendamos tres cosas más, **VOID es una palabra que se utiliza para NO RETORNAR NADA**, es una característica de los procedimientos, es decir son funciones que realizan alguna operación pero no devuelven nada. Y de paso hagamos algo nuevo, definimos las funciones arriba del main, como siempre; pero

damos la funcionalidad abajo del main, **ustedes elijan que prefieren hacer de ahora en más.** Y por último y quizás el dato menos relevante es una nueva forma de definir, cada vez que tengamos algún valor que no va a cambiar, podemos definirlo de ante mano, por ejemplo con define filas 3, definimos que cada vez que escribimos filas en realidad ponemos un 3, pueden usarlo o no, pero es bueno verlo una vez aunque sea.

```
#include <stdio.h>
#include <stdlib.h>

#define filas 3
#define columnas 4

// Predefinición de funciones
void SoliciteArray(int [][columnas]);
void VisualArray(int [][columnas]);
void Sum(int [][columnas],int [][columnas]);
void Res(int [][columnas],int [][columnas]);
void Tras(int [][columnas]);


int main()
{
int MatrizA[filas][columnas];
int MatrizB[filas][columnas];
```

```
// Peticion de datos para la matriz a

printf("Solicitando datos para la primera matriz");

SoliciteArray(MatrizA);

// Peticion de datos para la matriz b

printf("Solicitando datos para la segunda matriz");

SoliciteArray(MatrizB);

// Visualizar los datos de la matriz a

VisualArray(MatrizA);

// Visualizar los datos de la matriz b

VisualArray(MatrizB);


// Sumar matrices

printf("\nLa suma de las matrices es la siguiente:\n");

Sum(MatrizA,MatrizB);

// Restar matrices

printf("\nLa resta de las matrices es la siguiente:\n");

Res(MatrizA,MatrizB);

// Traspuestas

printf("\nTraspuesta de la matriz a\n");

Tras(MatrizA);

printf("\nTraspuesta de la matriz b\n");

Tras(MatrizB);
```

```
return 0;


}

// Funcion para solicitar los datos del array
void SoliciteArray(int Matriz[][columnas]) {
int fila;
int columna;

printf("\n");
for(fila=0;fila<filas;fila++) {
for(columna=0;columna<columnas;columna++) {
printf("Posicion %d - %d :",fila,columna);
scanf(" %d",&Matriz[fila][columna]);
}
}
printf("\n");
}

// Funcion para visualizar los datos de la matriz
void VisualArray(int Matriz[][columnas]) {
int fila;
int columna;
```

```
printf("\n");

for(fila=0;fila<filas;fila++) {

for(columna=0;columna<columnas;columna++) {

printf("%5d",Matriz[fila][columna]);

}

printf("\n");

}


}


// Funcion para sumar matrices

void Sum(int Matriz1[][columnas],int Matriz2[][columnas]) {

int fila;

int columna;

int resultado[filas][columnas];


for(fila=0;fila<filas;fila++) {

for(columna=0;columna<columnas;columna++) {

resultado[fila][columna]=Matriz1[fila][columna]+Matriz2[fila][columna];

}

}
```

```
// Visualizar matriz de resultado

VisualArray(resultado);

}


// Funcion para restar matrices

void Res(int Matriz1[][columnas],int Matriz2[][columnas]) {

int fila;

int columna;

int resultado[filas][columnas];


for(fila=0;fila<filas;fila++) {

for(columna=0;columna<columnas;columna++) {

resultado[fila][columna]=Matriz1[fila][columna]-Matriz2[fila][columna];

}

}


// Visualizar matriz de resultado

VisualArray(resultado);

}


void Tras(int Matriz[][columnas]) {

int fila;

int columna;
```

```
printf("\nVisualizar la matriz normal\n");
VisualArray(Matriz);
for(fila=0;fila<columnas;fila++) {
for(columna=0;columna<filas;columna++) {
printf("%5d",Matriz[columna][fila]);
}
printf("\n");
}

}
```

Traten de comprender el ejercicio y resuelvan los dos siguientes:

Ejercicio 9: *Realizar un programa* que muestre un menú con las opciones sumar, restar, multiplicar y dividir, el programa solicitará una opción y realizará la tarea elegida, se debe usar un procedimiento para mostrar el menú, pedir los datos en el main y después usar funciones para realizar los cálculos.

Ejercicio 10: Que pida por pantalla una temperatura en grados Celsius, muestre un menú para convertirlos a Fahrenheit o Kelvin y muestre el equivalente por pantalla, utilizar funciones.

Ejercicio 11 – Sacado de parcial:

Que implemente la lógica de un juego de adivinar un número, para ello se seguirán las siguientes instrucciones:

Los números se almacenarán en un array de 10 posiciones.

Uno de los jugadores debe introducir dichos números.

El otro jugador debe adivinar el número escribiéndolo por pantalla, el programa le dirá si el número secreto es mayor o menor al introducido.

En caso de acertar se mostrará por pantalla una felicitación y el número de intentos realizados.

Ejercicio 12 – A modo de resumen integrador de los temas: El siguiente programa solicita al usuario que ingrese 10 números de tipo entero y los almacena en un arreglo; después le pide que introduzca un número para que busque su posición dentro del arreglo.

El programa utiliza una función llamada BuscaNumero que recibe como parámetros el arreglo con los 10 números capturados, el número de elementos del arreglo (en este caso 10) y el número del cual se desea saber su posición dentro del arreglo.. La función regresa -1 si el número que se busca no se encuentra en el arreglo y en caso contrario, regresa la primera posición del arreglo que contiene dicho número.

El programa también utiliza una función llamada MuestraArreglo que no regresa valor alguno, sólo recibe como parámetros el arreglo y el número de elementos. Esta función imprime en pantalla los elementos del arreglo separados por un tabulador.

Solución propuesta:

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

```
//Programa: Buscar numero en arreglo

//Pre: Tener un arreglo de 10 numeros enteros

//Post: Nos dice si un numero entero existe o no en nuestro arreglo y en que posicion esta.




//DEFINO FUNCIONES

//Buscar un numero

//ingreso con el vector de numeros, con su tamanio y con el numero que quiero buscar

//Retorno la posiicion o -1 si no existe

int BuscaNumero(int valores[], int tamanio, int busca)

{

int i=0, posicion=-1;



//Lo hare con un do while, solo porque no lo vimos

//traten de adaptarlo a un while o for ustedes

do

  {

   if (valores[i]==busca)

       posicion=i;
```

```
   else
       ++i;
   }
while(i<tamanio && posicion<0);

//Se devuelve la posicion  del numero
return posicion;
}

//Mostrar el vector
//Muestro todo el vector de los diez numeros, necesito el tamaño del vector
//El retorno es del tipo void, puesto que no retorna nada, es decir es un PROCEDIMIENTO
void MuestraArreglo(int valores[], int tamanio)
{
 int i=0;

 for (i=0; i<tamanio; ++i)
      printf("%d\t",valores[i]);
 printf("\n");
}
```

```
//Inicia el main


int main()

{

 int x=0, numero=0, posicion=0;

 int ar_numeros[10] = {0};


 printf("Introduzca los 10 numeros enteros que se almacenaran en el arreglo\n");

 for (x=0; x<10; ++x)

     {

      printf("Valor para el elemento [%d]: ", x);

      scanf("%d",&ar_numeros[x]);

     }

 printf("\n");


 printf("----------UNLA SISTEMA  PROG 2018---- \n");

 printf("Introduzca el número que desea buscar en el arreglo\n");

 scanf("%d",&numero);

 printf("\n");
```

```
//muestro al arreglo completo

 MuestraArreglo(ar_numeros,10);



//busco mi numero en el vector, llamando a mi funcion

 posicion=BuscaNumero(ar_numeros,10,numero);

 printf("----------No hagan copiar y pegar U_U ---- \n");

 if (posicion != -1)

     printf("El número %d está en la posición %d del arreglo\n",numero, posicion);

 else

     printf("El número %d no está en el arreglo\n",numero);



 return 0;

}
```

Hasta ahora hemos resuelto ya algunos problemas bastante complejos, como el de los artículos y trimestres, o la semana pasada el problema del TATETI. Ambos programas notamos que quedaron muy extensos, complejos, y difíciles de entender. Además, arreglar cada error que se le encontraba al programa significaba un esfuerzo muy grande para el programador.

Es por eso, para conseguir programas mucho más fáciles de entender, mucho más simples a la hora de corregir y notoriamente más aptos para el crecimiento de la lógica y complejidad del software surge la necesidad de realizar los programas con funciones y procedimientos.

De ahora en más vamos a tratar que todos nuestros programas cumplan con los siguientes ítems:

1- En la cabecera se debe detallar como se llama el programa y para qué sirve.
2- Las líneas deben estar comentadas casi en su totalidad
3- Dentro del MAIN solo debemos inicializar las variables y llamar a las funciones y procedimientos, debemos intentar que no tengamos lógica dentro del main.
4- Se debe dividir al problema en pequeños problemas aislados, los mismos deben ser resueltos por funciones y/o procedimientos
5- Las constantes deben estar si o si en mayúsculas, las variables en minúsculas y en caso de que la variable tenga mas de una palabra las mismas deben estar separadas por guiones bajos o en su defecto solo la primera letra de la segunda palabra en mayúscula.

Para comprender la importancia de esta separación en pequeños módulos de un problema difícil de atacar resolvamos el problema del TATETI.

Ejercicio Integrador: Realizar un programa que permita jugar al TATETI.

Claramente este problema involucra una matriz de 3x3 donde cada una de sus posiciones se podría pensar como un char X o O según el jugador. Entonces ahí tenemos nuestra primer variable, ***char tablero[3][3];***

Aquí tenemos nuestro primer módulo, iniciar dicho tablero, yo optare por iniciarlo con números del 1 a 9, para que el jugador luego elija donde quiere jugar.

IniciarTablero(tablero); Este será un procedimiento que simplemente inicia al tablero con dichos números.

Luego realizare otro procedimiento que muestra al tablero por pantalla, ***MostrarTablero(tablero);***

Ahora estamos en condiciones de arrancar el juego, donde lo toca a un determinado jugador (numerodejugador) jugar en el tablero, por eso creare el procedimiento ***Jugar(tablero, numerodejugador);*** Este procedimiento le pregunta al jugador en qué posición quiere jugar , **verifica que el numero sea entre 1 y 9, y verifica que la posición no este ocupada, con dos funciones.**

Una vez que se sabe que la posición existe y que no esta ocupada, se actualiza el tablero con una X o O según que jugador sea. Al terminar de jugar se cambia al jugador siguiente, del 1 se para al 2, y del 2 al 1, en la función **SiguienteJugador(numerodejugador);**

Este procese se repite hasta que tengamos un ganador (While), ¿como sabemos si hay un ganador?, lo sabemos gracias a la función ***ComprobarGanador(tablero);***

De esta forma vemos como el programa quedo resuelto con un main de menos de 10 renglones, y con solo 7 funciones. Muy fácil de entender y sobre todo fácil de modificar y mejorar.

Veamos como quedo mi solución propuesta arriba.

```
#include <stdio.h>

#include <stdlib.h>


#define TAMANIO  3 //defino una constante


////////////////////////////////////////////////////

//Programa de TATETI - Catedra UNLA - Sistemas 2018
```

```
//Este sw es el programa del clasico juego tateti,
//diseñado para la clase del  7-5-2018
//////////////////////////////////////////////////////



//////////////////////////////////////////////////////
//Tablero inicializado con las posiciones de 1  a 9
void IniciarTablero(char tablero[TAMANIO][TAMANIO]){

	tablero[0][0] = '1';
	tablero[0][1] = '2';
	tablero[0][2] = '3';
	tablero[1][0] = '4';
	tablero[1][1] = '5';
	tablero[1][2] = '6';
	tablero[2][0] = '7';
	tablero[2][1] = '8';
	tablero[2][2] = '9';

}
//////////////////////////////////////////////////////

//////////////////////////////////////////////////////
```

```
//Se muestra por consola nuestro tablero
void MostrarTablero(char tablero[TAMANIO][TAMANIO]){
    for (int i =0; i<TAMANIO;i++){
        for (int j=0; j<TAMANIO;j++){
            printf("  %c  |  ",tablero[i][j]) ;
        }//cierro el for de columna
        printf("\n");
    }//Cierro el for de renglon



}//Cierra mostrar tablero
//////////////////////////////////////////////////////

//////////////////////////////////////////////////////
//Se verifica que la posicion que quiere usar un jugador no esta
//ocupada con una X o O
int VerificarPosicion(char tablero[TAMANIO][TAMANIO], int posicion){

    int retorno = 0; //Devuelvo 0 si esta libre, 1 si esta ocupada

    switch (posicion){
```

```
case 1: if(tablero[0][0] =='O'||tablero[0][0] =='X'){printf("Posicion Ocupada!!!\n\a");
retorno = 1;}; break;

case 2: if(tablero[0][1] =='O'||tablero[0][1] =='X'){printf("Posicion Ocupada!!!\n\a");
retorno = 1;}; break;

case 3: if(tablero[0][2] =='O'||tablero[0][2] =='X'){printf("Posicion Ocupada!!!\n\a");
retorno = 1;}; break;

case 4: if(tablero[1][0] =='O'||tablero[1][0] =='X'){printf("Posicion Ocupada!!!\n\a");
retorno = 1;}; break;

case 5: if(tablero[1][1] =='O'||tablero[1][1] =='X'){printf("Posicion Ocupada!!!\n\a");
retorno = 1;}; break;

case 6: if(tablero[1][2] =='O'||tablero[1][2] =='X'){printf("Posicion Ocupada!!!\n\a");
retorno = 1;}; break;

case 7: if(tablero[2][0] =='O'||tablero[2][0] =='X'){printf("Posicion Ocupada!!!\n\a");
retorno = 1;}; break;

case 8: if(tablero[2][1] =='O'||tablero[2][1] =='X'){printf("Posicion Ocupada!!!\n\a");
retorno = 1;}; break;

case 9: if(tablero[2][2] =='O'||tablero[2][2] =='X'){printf("Posicion Ocupada!!!\n\a");
retorno = 1;}; break;


   }//cierra el switch


return retorno;

}

//////////////////////////////////////////////////////

//////////////////////////////////////////////////////
```

```
//Se verifica que la posicion que se quiere jugar existe en la matriz

int VerificarNumero(int posicion){

int retorno = 0;


if(posicion<1|| posicion>9){

    retorno = 1;

}



return retorno;

}


//////////////////////////////////////////////////////


//////////////////////////////////////////////////////

//El jugador elige donde jugar y se pone su marca en el tablero

void Jugar(char tablero[TAMANIO][TAMANIO], int numerodejugador){


int posicion = 0;


printf("Turno del jugador %d, ¿en que posicion quiere jugar? \n", numerodejugador);

scanf("%d", &posicion);
```

```
int numeroincorrecto = VerificarNumero(posicion);

if (numeroincorrecto ==1){

      printf("\aPOSICION INCORRECTA. \n");

        Jugar(tablero, numerodejugador);

}



int posicionocupada = VerificarPosicion(tablero, posicion);



if (posicionocupada==0){

if (numerodejugador ==2){

    switch (posicion){

case 1: tablero[0][0] ='X'; break;

case 2: tablero[0][1] ='X'; break;

case 3: tablero[0][2] ='X'; break;

case 4: tablero[1][0] ='X'; break;

case 5: tablero[1][1] ='X'; break;

case 6: tablero[1][2] ='X'; break;

case 7: tablero[2][0] ='X'; break;

case 8: tablero[2][1] ='X'; break;

case 9: tablero[2][2] ='X'; break;
```

```
    }//cierra el switch

}

if (numerodejugador ==1){
    switch (posicion){
case 1: tablero[0][0] ='O'; break;
case 2: tablero[0][1] ='O'; break;
case 3: tablero[0][2] ='O'; break;
case 4: tablero[1][0] ='O'; break;
case 5: tablero[1][1] ='O'; break;
case 6: tablero[1][2] ='O'; break;
case 7: tablero[2][0] ='O'; break;
case 8: tablero[2][1] ='O'; break;
case 9: tablero[2][2] ='O'; break;

    }//cierra el switch

}

}

if(posicionocupada==1){
```

```
    Jugar(tablero, numerodejugador);
}



}//Cierra el jugar
//////////////////////////////////////////////////////

//////////////////////////////////////////////////////
//Una vez que se jugo se pasa al jugador siguiente
int SiguienteJugador(int numerodejugador){

int retorno = 0;

if (numerodejugador==1){retorno =2;};
if (numerodejugador==2){retorno =1;};

return retorno;

}
//////////////////////////////////////////////////////

//////////////////////////////////////////////////////
//Se verifica si ya hay un ganador, esto se hara cada vez que termine de jugar un jugador
```

```
int ComprobarGanador(char tablero[TAMANIO][TAMANIO]){
    int retorno = 0; //Si no hay ganador se queda en 0


    //verifico si hay ganador por renglon
    for (int i = 0; i<3; i++){

        if(tablero[i][0]=='O'&&tablero[i][1]=='O'&&tablero[i][2]=='O'){
            retorno= 1;
            printf("Felicidades, gano el jugador: %d \n", retorno);
        }//cierra el if

        if(tablero[i][0]=='X'&&tablero[i][1]=='X'&&tablero[i][2]=='X'){
            retorno= 2;
            printf("Felicidades, gano el jugador: %d \n", retorno);
        }//cierra el if


        }//Cierro renglon



    //verifico si hay ganador por columna
    for (int i = 0; i<3; i++){
```

```
        if(tablero[0][i]=='O'&&tablero[1][i]=='O'&&tablero[2][i]=='O'){
            retorno= 1;
            printf("Felicidades, gano el jugador: %d \n", retorno);
        }//cierra el if

        if(tablero[1][i]=='X'&&tablero[1][i]=='X'&&tablero[2][i]=='X'){
            retorno= 2;
            printf("Felicidades, gano el jugador: %d \n", retorno);
        }//cierra el if

        }//Cierro columna


    //verifico  las diagonales

    if(tablero[0][0]=='O'&&tablero[1][1]=='O'&&tablero[2][2]=='O'){
        retorno= 1;
        printf("Felicidades, gano el jugador: %d \n", retorno);
    }
     if(tablero[0][0]=='X'&&tablero[1][1]=='X'&&tablero[2][2]=='X'){
        retorno= 2;
        printf("Felicidades, gano el jugador: %d \n", retorno);
```

```
    }



    if(tablero[0][2]=='O'&&tablero[1][1]=='O'&&tablero[2][0]=='O'){

        retorno= 1;

        printf("Felicidades, gano el jugador: %d \n", retorno);

    }

     if(tablero[0][2]=='X'&&tablero[1][1]=='X'&&tablero[2][0]=='X'){

        retorno= 2;

        printf("Felicidades, gano el jugador: %d \n", retorno);

    }



return retorno;

}//Cierro comprobar
```

```
int main(void)
{
   char tablero[TAMANIO][TAMANIO]; //Tablero de 9 char en una matriz de 3 x 3
   int numerodejugador = 1; //Arranca el jugador 1
   int ganador = 0; //No hay Ganador

   //inicio el tablero de 3 x 3
   IniciarTablero(tablero); //pongo los numeros del 1 al 9

   //Muestro el tablero
   MostrarTablero(tablero); //Muestro el tablero

   while (ganador==0){ //Mientras no tengamos ganador
   Jugar(tablero, numerodejugador); //El jugador juega
   numerodejugador = SiguienteJugador(numerodejugador); //Se cambia de jugador
   MostrarTablero(tablero); //Se muetras se progreso
   ganador = ComprobarGanador(tablero); //Se verifica si termino el juego

   }
```

```
    return 0;
}
```

Ejercicio 2: Este programa se trató de hacer de una forma muy entendible para ustedes, los invitamos a que traten de mejorarlo agregándole nuevas funciones. Por ejemplo, validar que no se tipee un numero con coma, una función que limpie la pantalla y solo deje lo importante, un menú que permita volver a jugar, y la carga de los jugadores con nombre y apellido. Todo esto por medio de nuevas funciones y/o procedimientos.

Ejercicios propuestos de funciones y procedimientos, más simples para practicar:

Ejercicio 3: Que muestre un menú con las opciones sumar, restar, multiplicar y dividir, el programa solicitará una opción y realizará la tarea elegida, se debe usar un procedimiento para mostrar el menú, pedir los datos en el main y después usar funciones para realizar los cálculos.

Ejercicio 4: Que pida por pantalla un número del 1 al 10 y mediante un procedimiento muestre por pantalla el número escrito en letras.

Ejercicio 5: Que pida por pantalla una temperatura en grados Celsius, muestre un menú para convertirlos a Fahrenheit o Kelvin y muestre el equivalente por pantalla, utilizar funciones.

Ejercicio 6: Que transforme un número del 0 al 999 a números romanos, utilizar funciones.

Ejercicio 7: Que detecte si un número es primo.

Repaso, funciones para el manejo de los char y char[]

La clase pasada nos planteamos el problema de como generar un char[] que surja de la unión de otros dos char[], es decir, si tenemos char char1[] = “Nico”, y otro char char2[] =”---Perez”, obtener un char3 que sea “Nico---Perez, para esto ***basta con verificar que el area (array o bloque de memoria) tenga espacio suficiente para almacenar la cadena resultante, esta se genera utilizando la función strcat, esta agrega (concatena) la segunda cadena al final de la primera.***

Quedaría algo así: **(Usaremos la funcion puts que espara mostrar string)**

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


int main(void)

{

   char cadena1[] = " Concatenando cadenas ";

   char cadena2[] = "con la funcion strcmp";

   char cadena3[] =". No es dificil.";

   char cadena_destino[200] =" ";
```

```
   strcat(cadena_destino, cadena1);

    strcat(cadena_destino, cadena2);

     strcat(cadena_destino, cadena3);


   puts(cadena_destino);


   return EXIT_SUCCESS;

}
```

Además, vimos en el pizarrón otras funciones Asociadas al Manejo de cadenas de caracteres con la librería string.h

Aunque C no incorpora en su definición operadores para el manejo de cadenas de caracteres, todo compilador de C proporciona una librera estándar (string.h) con funciones para facilitar su utilización. Destacar algunas de ellas:

strcpy: La función strcpy se encuentra en la biblioteca <string.h> y se utiliza para copiar una cadena de caracteres (fuente) en el lugar que ocupaba otra (destino). Esta copia es destructiva, o sea, que todos los caracteres que estaban en la cadena destino desaparecen, aunque la cadena destino fuera más larga que la cadena fuente. La cadena destino se pone como primer argumento de la función y la cadena fuente como segundo. Vamos a verlo con un ejemplo.

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

int main()
{
   char texto1[]="corta";
```

```
    char texto2[]="mediana";
    char texto3[]="larguisima";

    strcpy(texto2,texto1);
    printf("%s\n",texto2);
    strcpy(texto2,texto3);
    printf("%s\n",texto2);
    getch();
return 0;
}
```

strcat: En el programa anterior vimos que la función strcpy es desctructiva, pero hay otra función en la librería <string.h> que copia una cadena (fuente) en otra (destino) sin destruir ésta, es decir, que copia una cadena detrás de la otra esta función es conocida como strcat. Vamos a hacer un ejemplo:

```
#include<stdio.h>
#include<string.h>
int main()
{
    char texto1[]="Don Pepito";
    char texto2[]=" y ";
    char texto3[]="Don Jose";printf("%s\n",texto1);

    strcat(texto1,texto2);
    printf("%s\n",texto2);
    strcat(texto1,texto3);
    printf("%s\n",texto2);
    getchar();
return 0;
}
```

strlen: esta función devuelve el total (entero) de caracteres que conforman una cadena (excluyendo el caracter nulo \0). Vamos a hacer un ejemplo:

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
#define MAXLON 80

int main(void)
{
   char a[MAXLON+1];
   int longitud;

   clrscr();
   printf ("Introduce una cadena (max. %d caracteres): ", MAXLON);
   scanf("%s",&a);
   longitud = strlen(a);
   printf ("\nLongitud de la cadena: %d\n", longitud);
   getch();
return 0;
}
```

strcmp: strcmp (abreviatura de ((string comparison))). La función strcmp recibe dos cadenas, a y b, devuelve un entero. El entero que resulta de efectuar la llamada strcmp(a, b) codifica el resultado de la comparación:

es menor que cero si la cadena a es menor que b,

es 0 si la cadena a es igual que b, y

es mayor que cero si la cadena a es mayor que b.

Naturalmente, menor significa que va delante en orden alfabético, y mayor que va detrás.

```
#include <stdio.h>
#include <string.h>
int main()
```

```
{
   char s1[6] = "Abeja";
   char s2[6] = "abeja";
   int i;

   printf( "s1=%s\t", s1 );
   printf( "s2=%s\n", s2 );

   i = strcmp( s1, s2 );
   printf( "s1 es " );
   if( i < 0 ) printf( "menor que" );
   else if( i > 0 ) printf( "mayor que" );
   else printf( "igual a" );
   printf( " s2\n" );

return 0;
}
```

Ordenamiento

Supongamos que trabajamos un vector de solo 5 posiciones, el vector1 (1,5,-1,2,3), si les pido crear un nuevo vector que tenga los mismos números que el vector1 pero que dichos números este de menor a mayor, ¿cómo quedaría? Pues simple, quedaría así: (-1,1,2,3,5). Ahora bien, ¿se animan a programarlo? Si en verdad se animan y prueban van a ver que no es para nada sencillo lograrlo ni con solo 5 posiciones.

Por fortuna este problema ya está resuelto gracias a los **algoritmos para lograr ordenamiento es decir Algoritmos de Ordenamiento.** En las próximas clases iremos usándolos y viéndolos uno por uno. Pero vayamos viéndolos ahora y usándolos,

En la catedra solo trabajaremos con algunos.

El ordenamiento se efectúa con base en el valor de algún campo en un registro. El propósito principal de un ordenamiento es el de **facilitar las búsquedas de los miembros del conjunto ordenado.**

El ordenar un grupo de datos significa mover los datos o sus referencias para que queden en una secuencia tal que represente un orden, el cual puede ser numérico, alfabético o incluso alfanumérico, ascendente o descendente.

- ✓ ***Métodos de Ordenamiento Elementales:***
  1. Inserción
  2. Selección
  3. Burbujeo

- ✓ ***Métodos de Ordenamiento no Elementales:***
  1. Shell
  2. Quick Sort

- ❖ ***Tipos de Ordenamiento:***
  - Ordenamiento Interno → Ordenamiento de datos en Memoria Principal. (La lectura y grabación se hacen en registros)
  - Ordenamiento Externo → Ordenamiento de datos en Disco.

- ❖ ***Tipos de Algoritmo***
  - Algoritmo Sensible: Modifica su tiempo de ejecución según el tipo de entrada.
  - Algoritmo No Sensible: Su tiempo de ejecución es independiente al tipo de entrada.
  - Algoritmo Estable: Aquellos que, teniendo clave repetida, mantiene su posición inicial igual a la final.

- Algoritmo No Estable: Aquello que no respetan la posición inicial igual que la final teniendo claves repetidas

✓ ***Métodos Elementales:***

1. ORDENAMIENTO POR SELECCIÓN

DESCRIPCIÓN.

- Buscas el elemento más pequeño de la lista.
- Lo intercambias con el elemento ubicado en la primera posición de la lista.
- Buscas el segundo elemento más pequeño de la lista.
- Lo intercambias con el elemento que ocupa la segunda posición en la lista.
- Repites este proceso hasta que hayas ordenado toda la lista.

ANÁLISIS DEL ALGORITMO.

- Requerimientos de Memoria: Al igual que el ordenamiento burbuja, este algoritmo sólo necesita una variable adicional para realizar los intercambios.
- Tiempo de Ejecución: El ciclo externo se ejecuta n veces para una lista de n elementos. Cada búsqueda requiere comparar todos los elementos no clasificados.

Ventajas:

1. Fácil implementación.
2. No requiere memoria adicional.
3. Rendimiento constante: poca diferencia entre el peor y el mejor caso.

Desventajas:

1. Lento.
2. Realiza numerosas comparaciones.

2. ORDENAMIENTO POR INSERCIÓN DIRECTA

DESCRIPCIÓN.

El algoritmo de ordenación por el método de inserción directa es un algoritmo relativamente sencillo y se comporta razonablemente bien en gran cantidad de situaciones.

Completa la tripleta de los algoritmos de ordenación más básicos y de orden de complejidad cuadrático, junto con SelectionSort y BubbleSort.

Se basa en intentar construir una lista ordenada en el interior del array a ordenar.

De estos tres algoritmos es el que mejor resultado da a efectos prácticos. Realiza una cantidad de comparaciones bastante equilibrada con respecto a los intercambios, y tiene un par de características que lo hacen aventajar a los otros dos en la mayor parte de las situaciones.

Este algoritmo se basa en hacer comparaciones, así que para que realice su trabajo de ordenación son imprescindibles dos cosas: un array o estructura similar de elementos comparables y un criterio claro de comparación, tal que dados dos elementos nos diga si están en orden o no.

En cada iteración del ciclo externo los elementos 0 a i forman una lista ordenada.

ANÁLISIS DEL ALGORITMO.

- Estabilidad: Este algoritmo nunca intercambia registros con claves iguales. Por lo tanto es estable.
- Requerimientos de Memoria: Una variable adicional para realizar los intercambios.

- Ø Tiempo de Ejecución: Para una lista de n elementos el ciclo externo se ejecuta n1 veces. El ciclo interno se ejecuta como máximo una vez en la primera iteración, 2 veces en la segunda, 3 veces en la tercera, etc.

Ventajas:

1. Fácil implementación.
2. Requerimientos mínimos de memoria.

Desventajas:

1. Lento.
2. Realiza numerosas comparaciones.

Este también es un algoritmo lento, pero puede ser de utilidad para listas que están ordenadas o semiordenadas, porque en ese caso realiza muy pocos desplazamientos.

3. METODO DE LA BURBUJA

DESCRIPCIÓN

La idea básica del ordenamiento de la burbuja es recorrer el conjunto de elementos en forma secuencial varias veces. Cada paso compara un elemento del conjunto con su sucesor (x[i] con x[i+i]), e intercambia los dos elementos si no están en el orden adecuado.

El algoritmo utiliza una bandera que cambia cuando se realiza algún intercambio de valores, y permanece intacta cuando no se intercambia ningún valor, pudiendo así detener el ciclo y terminar el proceso de ordenamiento cuando no se realicen intercambios, lo que indica que este ya está ordenado.

Este algoritmo es de fácil comprensión y programación, pero es poco eficiente puesto que existen n-1 pasos y n-i comprobaciones en cada paso, aunque es mejor que el algoritmo de ordenamiento por intercambio.

En el peor de los casos cuando los elementos están en el orden inverso, el número máximo de recorridos es n-1 y el número de intercambios o comparaciones está dado por (n-1) * (n-1) = n^2 - 2n + 1.En el mejor de los casos cuando los elementos están en su orden, el número de recorridos es mínimo 1 y el ciclo de comparaciones es n-1.

La complejidad del algoritmo de la burbuja es O(n) en el mejor de los casos y O(n^2) en el peor de los casos, siendo su complejidad total O(n^2).

- ***Métodos No Elementales:***

4. ORDENAMIENTO POR EL MÉTODO DE SHELL

DESCRIPCIÓN

El método Shell es una versión mejorada del método de inserción directa. Este método también se conoce con el nombre de inserción con incrementos crecientes. En el método de ordenación por inserción directa cada elemento se compara para su ubicación correcta en el arreglo, con los elementos que se encuentran en la parte izquierda del mismo. Si el elemento a insertar es más pequeño que el grupo de elementos que se encuentran a su izquierda, es necesario efectuar entonces varias comparaciones antes de su ubicación.

Shell propone que las comparaciones entre elementos se efectúen con saltos de mayor tamaño pero con incrementos decrecientes, así, los elementos quedarán ordenados en el arreglo más rápidamente.

El Shell sort es una generalización del ordenamiento por inserción, teniendo en cuenta dos observaciones:

1. El ordenamiento por inserción es eficiente si la entrada está "casi ordenada".

2. El ordenamiento por inserción es ineficiente, en general, porque mueve los valores sólo una posición cada vez.

El algoritmo Shell sort mejora el ordenamiento por inserción comparando elementos separados por un espacio de varias posiciones. Esto permite que un elemento haga "pasos más grandes" hacia su posición esperada. Los pasos múltiples sobre los datos se hacen con tamaños de espacio cada vez más pequeños. El último paso del Shell sort es un simple ordenamiento por inserción, pero para entonces, ya está garantizado que los datos del vector están casi ordenados.

5. ORDENAMIENTO QUICK SORT

DESCRIPCIÓN

El ordenamiento por partición (Quick Sort) se puede definir en una forma más conveniente como un procedimiento recursivo.

Tiene aparentemente la propiedad de trabajar mejor para elementos de entrada desordenados completamente, que para elementos semiordenados. Esta situación es precisamente la opuesta al ordenamiento de burbuja.

Este tipo de algoritmos se basa en la técnica "divide y vencerás", o sea es más rápido y fácil ordenar dos arreglos o listas de datos pequeños, que un arreglo o lista grande.

Normalmente al inicio de la ordenación se escoge un elemento aproximadamente en la mitad del arreglo, así al empezar a ordenar, se debe llegar a que el arreglo este ordenado respecto al punto de división o la mitad del arreglo.

Se podrá garantizar que los elementos a la izquierda de la mitad son los menores y los elementos a la derecha son los mayores.

Los siguientes pasos son llamados recursivos con el propósito de efectuar la ordenación por partición al arreglo izquierdo y al arreglo derecho, que se obtienen de la primera fase. El tamaño de esos arreglos en promedio se reduce a la mitad.

Así se continúa hasta que el tamaño de los arreglos a ordenar es 1, es decir, todos los elementos ya están ordenados.

En promedio para todos los elementos de entrada de tamaño n, el método hace O(n log n) comparaciones, el cual es relativamente eficiente.

COMPLEJIDAD

Cada algoritmo de ordenamiento por definición tiene operaciones y cálculos mínimos y máximos que realiza (complejidad), a continuación, una tabla que indica la cantidad de cálculos que corresponden a cada método de ordenamiento:

Algoritmo	**Operaciones máximas**
Burbuja	$\Omega (n^2)$
Inserción	$\Omega (n^2/4)$
Selección	$\Omega (n^2)$
Shell	$\Omega (n \log^2 n)$
Quick	$\Omega(n \log n)$ en el promedio de los casos.

Métodos de ordenamiento funcionando, de ahora en mas puede utilizar dichas funciones ustedes mismos, y mejorarlas.

```
#include <stdio.h>

#include <stdlib.h> //COMENTAR SI SE CIERRA LA VENTANA
```

```
#define TAMANIO 10 //Es una constante


void SelectionSort(int array[10], int n) {
  int x, y, min, tmp;



  for(x = 0; x < n; x++) {
    min = x;
    for(y = x + 1; y < n; y++) {
      if(array[min] > array[y]) {
        min = y;
      }
    }
    tmp = array[min];
    array[min] = array[x];
    array[x] = tmp;
  }
}


void Shell(int array[10], int n) {
  int i,x,y,tmp;
```

```
  for(i = 1; i < n; i = i*3+1) {} //la idea es que SOLO se incremente i a razón de i*3+1 por repetición


    while (i > 0) {
      for(x = i; x < n; x++) {
        y = x;
        tmp = array[x];
        while (y >= i && array[y - i] > tmp) {
          array[y] = array[y - i];
          y = y - i;
        }
        array[y] = tmp;
      }
      i = i / 2;
    }
}


void IBinaria(int array[10], int n) {
  int x,y,m,tmp,izq,der;
  for(x = 1; x < n; x++) {
    tmp = array[x];
    izq = 0;
    der = x - 1;
```

```
    while (izq <= der) {
      m = (izq + der) / 2;
      if(tmp < array[m]) {
        der = m - 1;
      } else {
        izq = m + 1;
      }
    }
    y = x - 1;
    while (y >= izq) {
      array[y + 1] = array[y];
      y = y - 1;
    }
    array[izq] = tmp;
  }
}

void IDirecta(int array[10], int n) {
  int x,val,y;
  for(x = 1; x < n; x++) {
    val = array[x];
    y = x - 1;
    while (y >= 0 && array[y] > val) {
```

```
      array[y + 1] = array[y];
      y--;
    }
   array[y + 1] = val;
  }
}

void QuickSort(int array[10], int inicio, int final) {
  int i = inicio, f = final, tmp;
  int x = array[(inicio + final) / 2];
  do {
    while(array[i] < x && f <= final) {
      i++;
    }
    while(x < array[f] && f > inicio) {
      f--;
    }
    if(i <= f) {
      tmp = array[i];
      array[i] = array[f];
      array[f] = tmp;
      i++; f--;
    }
```

```
  } while(i <= f);


  if(inicio < f) {
    QuickSort(array,inicio,f);
  }


  if(i < final){
    QuickSort(array,i,final);
  }


}


void Burbuja(int array[10], int n) {
  int x,y,tmp;
  for(x = 1; x < n; x++) {
    for(y = 0; y < n - x; y++) {
      if(array[y] > array[y + 1]) {
        tmp = array[y];
        array[y] = array[y + 1];
        array[y + 1] = tmp;
      }
    }
  }
```

```
}

void Print(char function[100],int array[10], int n) {
  int x;
  printf("%s:",function);
  for(x = 0; x < n; x++) {
    printf(" %i",array[x]);
  }
  printf("\n");
}

int main() {

  int vector[TAMANIO] = {1,2,5,71,99,0,5,16,26,3};
  Print("Vector Original:", vector, TAMANIO);

  SelectionSort(vector,TAMANIO);
  Print("SelectionSort",vector,TAMANIO);

  Shell(vector,TAMANIO);
  Print("Shell",vector,TAMANIO);

  IBinaria(vector,TAMANIO);
```

```
  Print("Isercion Binaria",vector,TAMANIO);


  IDirecta(vector,TAMANIO);

  Print("Isercion Directa",vector,TAMANIO);


  QuickSort(vector,0,TAMANIO - 1);

  Print("QuickSort",vector,TAMANIO);


  Burbuja(vector,TAMANIO);

  Print("Burbuja",vector,TAMANIO);


  //system("PAUSE"); DESCOMENTAR SI SE CIERRA LA VENTANA


  return 1;

}
```

Repaso general

Repaso de tipos de datos y funciones de C

Si por un instante nos detenemos y nos preguntamos qué cosas hemos aprendido y utilizado hasta ahora debería venirse a nuestras mentes que, hemos aprendido los ***tipos de datos***, algunos, como por ejemplo, los enteros **int**; los flotantes **float**, los caracteres **char**, los vectores de números **int[]** los arreglos de caracteres es decir los que se utilizan para formar palabras y oraciones, los **char[].**

Luego aprendimos algunas **funciones básicas de C, el printf** que es para mostrar por consola lo que queramos, por ejemplo textos entre comillas (“Esto es lo que muestro”), mostrar valores enteros (“%d”, valor), mostrar caracteres (“%c”, carácter), mostrar varios caracteres (“%s”, palabra) y combinarlas (“Voy a combinar %d, %c y %s”, valor, carácter, palabra).

Vimos otra función embebida en C, denominada **scanf** que la utilizamos para guardar los valores que se ingresan por teclado, por ejemplo guardar enteros (scanf(“%d”, &valor)), guardar caracteres, (scanf(“%c”, &caracter)), o guardar palabras (scanf(“%s”, palabra)).

Vale la pena aclarar que muchas veces que guardamos por teclado una cadena de char, es decir un String o una palabra**, nos interesa dejar espacios**. Por ejemplo si queremos guardar “Unla Sistemas” en la variable char carrera[40] y utilizamos el scanf(“%S”, carrera) podríamos encontrarnos en un problema **y perder todo lo que se ha tipeado luego del “espacio”**, para evitar esto aparece la función **gets (gets(carrera)) que nos evita el problema de los espacios**, pero como contra partida no pone limite al tipeo, ergo se podría perder el control de los límites de caracteres, para esto tenemos una nueva función que nos permite a nosotros limitar la cantidad de caracteres a tipear, es la función fgets (fgets(carrera, 40, stdin)). Nosotros a nuestro nivel de programación y los problemas que resolveremos nos alcanzara con scanf y gets.

Luego, y casi sin querer, aprendimos el uso de otras funciones, por ejemplo **system(“cls”) para borrar el contenido de la consola**; **system(“pause”) para poner una pausa al proceso en ejecución**. Otra muy importante y útil nos ha **sido fflush(stdin) para limpiar el buffer del teclado**, cosa muy recomendable luego de los scanf.

Otras funciones que vimos relacionadas a los arreglos de char son strlen, toupper, getchar, etc. Strlen devuelve la longitud del texto, toupper transforma a mayúsculas y getchar espera un ingreso por teclado.

Aquí hay un resumen de otras funciones que nos pueden simplificar la vida con las cadenas de caracteres:

Función	Significado y ejemplo
strcpy (arg1, arg2)	Copia arg2 en arg1. Ejemplo: strcpy (cadena, “control”);
strlen (arg1)	Devuelve la longitud del texto representado por arg1. Ejemplo: strlen(cadena1)
strcat (arg1, arg2)	Concatena las cadenas representadas por arg1 y arg2. Ejemplo: strcat(cadena1, “ unidades”)
strcmp (arg1, arg2)	Devuelve 0 si las cadenas representadas por arg1 y arg2 son iguales, o un valor menor que cero si arg1 precede alfabéticamente a arg2. Ejemplo: resComparacion = strcmp (cadena4, cadena2);

Otra operación importante que podríamos llegar a necesitar es la de “resto de dividir por” que ya la hemos utilizado en otros momentos. Por ejemplo esta operación:

```
int valor = 5; int resultado = valor%2;
```

En este caso valor nos quedaría con un resultado de 1, puesto que el **resto de dividir a 5 por 2 es 1.**

Repaso de Char y String con ejemplos:

Para comprender mejor estas funciones y operaciones básicas descriptas arriba hagamos algunos ejemplos refrescando y aceitando los conceptos de char y string o cadenas de char utilizando funciones y procedimientos.

Ejercicio 1: Crear una función que reciba una cadena y una letra, y devuelva la cantidad de veces que dicha letra aparece en la cadena. Por ejemplo, si la cadena es "Universidad" y la letra es 'a', debería devolver 1.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
//Programa que devuelve la cantidad de veces que
//tenemos una letra en una palabra



//Pre: Una palabra de 20 caracteres, y una letra a comparar
//Post: devuelve el número de veces que tenemos esa letra
int numVecesApareceLetra (char cadena[20], char letra)
{
   int i, num=0;

   for (i=0; i<strlen(cadena); i++)
```

```
   {
      if (cadena[i]==(letra) || cadena[i]==toupper(letra))
         num++;
   }

   return num;
}

//inicia el main
int main()
{
   char caden[20];
   char letr;
   int i;

   printf("Escriba una palabra: ");
   gets(caden);
   fflush(stdin);

   printf("Escriba una letra: ");
   scanf("%c", &letr);

   printf("La letra %c se encuentra %d veces en %s",
```

```
    letr, numVecesApareceLetra(caden, letr), caden);
  getchar();
  getchar();
  return 0;
}
```

Ejercicio 2: ¿Se acuerdan de los vectores capicúas? Les propongo que realicemos lo mismo, pero con palabras, es decir, veamos si un arreglo de char es o no un palíndromo.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

//Palabra palindromo

int main ()
{

  char palabra[100];
  int i, a, c;              /*//contadores*/

  printf("Escriba la palabra: ");
```

```
    gets(palabra);

    a = c = strlen(palabra);

    for (i=0;i<a;i++,c--) {

        if (palabra[i] != palabra[c-1]) {
            printf("La palabra %s no es palíndromo", palabra);
            return EXIT_SUCCESS;
        }
    }

    printf("La palabra %s es un palindromo", palabra);

    return 1;
}
```

Repaso de vectores y matrices

Sigamos repasando alguna de las funciones antes mencionadas, pero análogo al tratamiento que realizamos con los char, podemos pensar en una colección de números enteros o flotantes, estos arrays tienen el nombre particular de vectores. Resolvamos unos ejercicios:

Ejercicio 3: Que lea 10 números por teclado, los almacene en un array y muestre la media.

```
#include <stdio.h>

#include <stdlib.h>


int main()

{

  float sum, numeros1[10];

  int i;


  sum=0;

  for (i=0;i<10;i++){

                printf("Escriba un número");

                scanf("%f",&numeros1[i]);

     }


  for(i=0;i<10;i++)

  {

   sum=sum+numeros1[i];

  }


  printf("%f\n",sum/10);
```

```
  system("PAUSE");

  return 0;

}
```

Ejercicio 4: Que rellene una matriz de 3x3 y muestre su traspuesta (la traspuesta se consigue intercambiando filas por columnas).

```
#include <stdio.h>

#include <stdlib.h>


int main()

{


    int x,y,num=0, numeros[4][4];


    for (x=0;x<3;x++)

    {

        for (y=0;y<3;y++)

        {

            numeros[x][y]=num;

            num++;

        }
```

```
    }

    printf("El array original es: \n\n\n");

    for(x = 0;x < 3;x++)
    {
      for(y = 0;y < 3;y++)
      {
          printf("    %d    ", numeros[x][y]);
      }
      printf("\n\n\n");
    }

    printf("La traspuesta es: \n\n\n");

    for(x = 0;x < 3;x++)
    {
      for(y = 0;y < 3;y++)
      {
          printf("    %d    ", numeros[y][x]);
      }
      printf("\n\n\n");
    }
```

```
    system("PAUSE");
    return 0;
}
```

Repaso de generación de menús

La semana pasada además hicimos algo nuevo para ver una aplicación de las funciones y los procedimientos, que era la creación de un menú, para que el usuario elija que desea hacer, y en base a su elección pueda realizar distintas operaciones.

Ejercicio 5: Muestre un menú con 4 opciones. Calcular el doble de un número entero. Calcular la mitad de un número entero. Calcular el cuadrado de un número entero. Salir.

En esta solución tienen el esqueleto y el llamado a las funciones, realicen ustedes mismos dichas funciones para terminar de entender su funcionamiento.

```
#include <math.h>
#include <stdio.h>
//Libreria math facilita algunas operaciones como las que tienen
//exponentes, ver el comando pow

int main()
{
```

```
   int opcion;



   do
   {
      printf( "\n   1. Calcular el doble de un numero entero.");
      printf( "\n   2. Calcular la mitad de un numero entero.");
      printf( "\n   3. Calcular el cuadrado de un numero entero.");
      printf( "\n   4. Salir." );
      printf( "\n\n   Introduzca opciocn (1-4): ");

      scanf( "%d", &opcion );

      /* Inicio del anidamiento */

      switch ( opcion )
      {
         case 1: calcularElDoble();
                 break;

         case 2: calcularLaMitad();
                 break;
```

```
            case 3: calcularCuadrado();
                        break;
        }

        /* Fin del anidamiento */

    } while ( opcion != 4 );

    return 0;
}
```

Ejercicio 6: El resuelto en clase. Se propone generar un menú que permita crear alumnos hasta 10, con 4 notas enteras y nombre de los alumnos de hasta 15 caracteres. Las opciones del menú deben ser las clásicas y denominadas ABM, altas bajas y modificaciones, donde además le agregaremos la opción de consulta. Con este enunciado trabajaremos varias clases.

```
#include <stdio.h>
#include <stdlib.h>

int opcion = 99;
int ultimo = 0;
```

```
int nroalumno[10];

int n1[10];

int n2[10];

int n3[10];

int n4[10];

char nombres[10][15];


void Altas(){

    PedirDatos();

}


void PedirDatos(){

    system("CLS");

    printf("ALTAS\n");


    printf("Numero: ");

    scanf("%d", &nroalumno[ultimo]);

    printf("\Nombre: ");

    scanf("%s", nombres[ultimo]);

    printf("Nota 1: ");

    scanf("%d",&n1[ultimo]);

    printf("Nota 2: ");
```

```
    scanf("%d",&n2[ultimo]);
    printf("Nota 3: ");
    scanf("%d",&n3[ultimo]);
    printf("Nota 4: ");
    scanf("%d",&n4[ultimo]);

    ultimo++;
};

void Consultas(){
    int i;
    for(i=0;i<10;i++){
        printf("%s", nombres[i]);
        printf(" %d %d %d %d", nroalumno[i], n1[i], n2[i], n3[i], n4[i]);
    }
}
int main()
{
    while(opcion!=0){
        opcion = 99;
        while(opcion > 4 || opcion < 0){
            system("CLS");
            printf("Menu de alumnos\n");
```

```
        printf("------------------\n");
        printf("1 - Altas\n");
        printf("2 - Bajas\n");
        printf("3 - Modificaciones\n");
        printf("4 - Consultas\n");
        printf("0 - Salir del programa\n");
        printf("Opcion seleccionada: ");

        scanf("%d", &opcion);

        /* Ya nombro como invalida la opcion en el SWITCH
        if(opcion > 4 || opcion < 0){
            printf("\aDebe reingresar la opcion!\n");
            system("PAUSE");
        }
        */

        switch (opcion){
            case 0:
                printf("Hasta Pronto!\n");
                break;
            case 1:
                Altas();
```

```
                break;
            case 2:
                printf("BAJA\n");
                break;
            case 3:
                printf("MODIFICACION\n");
                break;
            case 4:
                Consultas();
                break;
            default:
                printf("La opcion elegida no es valida. Debe volver a elegir! \n");
        }
        system("PAUSE");
    }


    }



    return 0;
}
```

Menú con funciones y procedimientos ABM

Como ya venimos trabajando en los últimos apuntes y clases, seguimos trabajando con el ejercicio de generar 10 alumnos, con 4 notas cada uno, donde los alumnos tienen un número y su nombre.

Habíamos arrancado el ejercicio creando un menú que nos mostraba solo descripciones por medio de "printf", luego fuimos reemplazando esos "printf" con llamados a funciones o procedimientos. Estas funciones y procedimientos eran las que nosotros denominamos ABM, altas, bajas y modificaciones. Altas es crear un alumno nuevo, modificación, es alterar algún campo o valor de un alumno ya existente y bajas es quitar a un alumno de nuestros registros. Además, le agregamos una opción mas denominada consultas, donde veíamos a todos los alumnos cargados en memoria.

La primera semana que trabajamos con este ejercicio simplemente generamos las altas y veíamos por pantalla las altas realizadas por medio de la consulta.

Ahora, en esta parte de la lectura nos enfocaremos en las bajas y modificaciones.

Las bajas, serán un procedimiento en el cual, una vez que se elige que renglón se quiere borrar, todos los renglones que estaban debajo suben una posición, como el ultimo renglón quedará dos veces será borrado.

De esta forma si queremos borrar el renglón 2 de:

Nico 1 1 1 1

Perez 2 2 2 2

Jorge 2 3 4 5

Brenda 1 1 3 4

Debemos conseguir algo como:

Nico 1 1 1 1

Jorge 2 3 4 5 ← Renglón a borrar reemplazado por el registro siguiente

Brenda 1 1 3 4

Brenda 1 1 3 4

Como se aprecia arriba, el ultimo alumno queda repetido, por lo que lo borramos así queda:

Nico 1 1 1 1

Jorge 2 3 4 5

Brenda 1 1 3 4

Esta operación se la puede realizar así:

```
void Bajas(){

   int i=0;

   system("CLS");
   printf("Bajas\n");
   printf("--------------------\n");
```

```
//para entrar si o si pongo -5
int nro = -5;
while(nro > ultimo || nro < 0){
   printf("Ingrese el numero a dar de baja (99- Volver al menu): ");
   scanf("%d", &nro);
}

for(i=nro;i<ultimo;i++){

   strcpy(nombres[i], nombres[i+1]);
   nroalumno[i] = nroalumno[i+1];
   n1[i] = n1[i+1];
   n2[i] = n2[i+1];
   n3[i] = n3[i+1];
   n4[i] = n4[i+1];
}

strcpy(nombres[ultimo], "");
nroalumno[ultimo] = 0;
n1[ultimo] = 0;
n2[ultimo] = 0;
n3[ultimo] = 0;
n4[ultimo] = 0;
```

```
    ultimo--;


    Consultas();
}
```

Mientras que para terminar este segmente, las modificaciones son elegir uno de los alumnos, mostrar por pantalla opciones que nos permiten elegir “que es lo que se quiere cambiar”, una vez elegida una de las variables se la cambia por un nuevo valor.

Dicho procedimiento podemos concluirlo con el siguiente código fuente:

```
void Modif(int nro){
    opcion = 99;


    system("CLS");
    printf("Que desea modificar?\n");
    printf("--------------------\n");
    printf("1 - Nro. de Alumno (%d)\n", nroalumno[nro]);
    printf("2 - Nombre (%s)\n", nombres[nro]);
    printf("3 - Nota 1 (%d)\n", n1[nro]);
    printf("4 - Nota 2 (%d)\n", n2[nro]);
```

```
printf("5 - Nota 3 (%d)\n", n3[nro]);

printf("6 - Nota 4 (%d)\n", n4[nro]);

printf("0 - Volver a la pantalla anterior\n");


while(opcion > 6 || opcion < 0){

    printf("Ingrese el numero de orden a modificar: ");

    scanf("%d", &opcion);


    switch (opcion){

            case 0:

                return;

                break;

            case 1:

                printf("Ingrese el numero de alumno nuevo: ");

                scanf("%d", &nroalumno[nro]);

                break;

            case 2:

                printf("Ingrese el nombre nuevo: ");

                scanf("%s", nombres[nro]);

                break;

            case 3:

                printf("Ingrese la Nota 1 nueva: ");

                scanf("%d",&n1[nro]);
```

```
                break;
            case 4:
                printf("Ingrese la Nota 2 nueva: ");
                scanf("%d",&n2[nro]);
                break;
            case 5:
                printf("Ingrese la Nota 3 nueva: ");
                scanf("%d",&n3[nro]);
                break;
            case 6:
                printf("Ingrese la Nota 4 nueva: ");
                scanf("%d",&n4[nro]);
            default:
                printf("La opcion elegida no es valida. Debe volver a elegir! \n");
        }
        system("PAUSE");
    }
}

void Modificaciones(){
    while(1){
        int nro = 99;
        system("CLS");
```

```
        while(nro > ultimo || nro < 0){
            printf("Ingrese el numero a modificar (99- Volver al menu): ");
            scanf("%d", &nro);

            if(nro != 99){
                Modif(nro);
            }else{
                return;
            }
        }

    }
}
```

Por lo que una versión actualizada de nuestro proyecto podría ser la siguiente:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int opcion = 99;
int ultimo = 0;
```

```
int nroalumno[10];

int n1[10];

int n2[10];

int n3[10];

int n4[10];

char nombres[10][15];


// Esta es una forma de resolverlo,

//llamando funciones dentro de funciones. No es necesario


void Modif(int nro){

    opcion = 99;


    system("CLS");

    printf("Que desea modificar?\n");

    printf("--------------------\n");

    printf("1 - Nro. de Alumno (%d)\n", nroalumno[nro]);

    printf("2 - Nombre (%s)\n", nombres[nro]);

    printf("3 - Nota 1 (%d)\n", n1[nro]);

    printf("4 - Nota 2 (%d)\n", n2[nro]);

    printf("5 - Nota 3 (%d)\n", n3[nro]);

    printf("6 - Nota 4 (%d)\n", n4[nro]);
```

```
    printf("0 - Volver a la pantalla anterior\n");


    while(opcion > 6 || opcion < 0){
        printf("Ingrese el numero de orden a modificar: ");
        scanf("%d", &opcion);


        switch (opcion){
                case 0:
                    return;
                    break;
                case 1:
                    printf("Ingrese el numero de alumno nuevo: ");
                    scanf("%d", &nroalumno[nro]);
                    break;
                case 2:
                    printf("Ingrese el nombre nuevo: ");
                    scanf("%s", nombres[nro]);
                    break;
                case 3:
                    printf("Ingrese la Nota 1 nueva: ");
                    scanf("%d",&n1[nro]);
                    break;
                case 4:
```

```
                printf("Ingrese la Nota 2 nueva: ");
                scanf("%d",&n2[nro]);
                break;
            case 5:
                printf("Ingrese la Nota 3 nueva: ");
                scanf("%d",&n3[nro]);
                break;
            case 6:
                printf("Ingrese la Nota 4 nueva: ");
                scanf("%d",&n4[nro]);
            default:
                printf("La opcion elegida no es valida. Debe volver a elegir! \n");
        }
        system("PAUSE");
    }
}

void Modificaciones(){
    while(1){
        int nro = 99;
        system("CLS");

        while(nro > ultimo || nro < 0){
```

```
            printf("Ingrese el numero a modificar (99- Volver al menu): ");
            scanf("%d", &nro);

            if(nro != 99){
                Modif(nro);
            }else{
                return;
            }
        }

    }
}


void Bajas(){

    int i=0;

    system("CLS");
    printf("Bajas\n");
    printf("--------------------\n");

    int nro = 0;
```

```
    while(nro > ultimo || nro < 0){
        printf("Ingrese el numero a dar de baja (99- Volver al menu): ");
        scanf("%d", &nro);
    }

    for(i=nro;i<ultimo;i++){

        strcpy(nombres[i], nombres[i+1]);
        nroalumno[i] = nroalumno[i+1];
        n1[i] = n1[i+1];
        n2[i] = n2[i+1];
        n3[i] = n3[i+1];
        n4[i] = n4[i+1];
    }

    strcpy(nombres[ultimo], "");
    nroalumno[ultimo] = 0;
    n1[ultimo] = 0;
    n2[ultimo] = 0;
    n3[ultimo] = 0;
    n4[ultimo] = 0;

    ultimo--;
```

```
    Consultas();
}


void Altas(){
    PedirDatos();
}

void PedirDatos(){
    system("CLS");
    printf("Altas\n");
    printf("--------------------\n");

    printf("Numero: ");
    scanf("%d", &nroalumno[ultimo]);
    printf("Nombre: ");
    scanf("%s", nombres[ultimo]);
    printf("Nota 1: ");
    scanf("%d",&n1[ultimo]);
    printf("Nota 2: ");
    scanf("%d",&n2[ultimo]);
    printf("Nota 3: ");
```

```
    scanf("%d",&n3[ultimo]);
    printf("Nota 4: ");
    scanf("%d",&n4[ultimo]);

    ultimo++;
};

void Consultas(){
    int i;
    for(i=0;i<10;i++){
        printf("%s", nombres[i]);
        printf(" %d %d %d %d %d\n", nroalumno[i], n1[i], n2[i], n3[i], n4[i]);
    }
}
int main()
{
    while(opcion!=0){
        opcion = 99;
        while(opcion > 4 || opcion < 0){
            system("CLS");
            printf("Menu de alumnos\n");
            printf("------------------\n");
            printf("1 - Altas\n");
```

```
        printf("2 - Bajas\n");
        printf("3 - Modificaciones\n");
        printf("4 - Consultas\n");
        printf("0 - Salir del programa\n");
        printf("Opcion seleccionada: ");

        scanf("%d", &opcion);

        switch (opcion){
            case 0:
                printf("Hasta Pronto!\n");
                break;
            case 1:
                Altas();
                break;
            case 2:
                Bajas();
                break;
            case 3:
                Modificaciones();
                break;
            case 4:
```

```
                Consultas();
                break;
            default:
                printf("La opcion elegida no es valida. Debe volver a elegir! \n");
        }
        system("PAUSE");
    }


    return 0;
}
```

Mas adelante seguiremos avanzando con este proyecto en el cual generaremos estadísticas, como el alumno con nota mas alta, el mejor promedio, el alumno con la peor 3er nota etc.

Modelo de Examen – Programación de Computadoras

En esta parte del apunte dejaremos un modelo de parcial, para que vayan practicando y viendo en donde están parados en cuanto a sus conocimientos. Traten de realizarlo como si fuera el examen.

1) Realizar un programa que lea de teclado un entero y un carácter. La salida en pantalla deber ser una figura como la que se muestra en el dibujo, compuesta por el carácter tecleado, y de altura dada por el número entero introducido. Por ejemplo, si el entero tecleado es 4 y el carácter el asterisco (*), la imagen debe ser:

```
*
**
***
****
***
**
*
```

2) Crear un vector donde el usuario elige que tan largo será el mismo, carga por teclado cada una de sus componentes enteras. Luego generar otro vector que tiene su mismo largo y reemplaza a los pares por la mitad y a los impares por el triple.

Ejemplo: Vector1 = (1,2,5,-3-6) Vector2 = (3,1,15,-5, -3)

3) Se comparan a tres goleadores del futbol, donde se muestran la cantidad de goles por semestre de estos 3 por goleadores. La distribución de los goles por semestre está dada por:

	Semestre1	Semestre2
jugador1	11	20
jugador2	10	9
jugador3	19	9

a-Conseguir los goles por semestre y los goles por jugador.

b-Realizar la prueba de escritorio.

4) Realizar una función/procedimiento en la que el usuario escriba una oración y una letra, entonces se muestre por pantalla la cantidad de veces que aparece esa letra en la oración. Llamar a la función o procedimiento desde un main simple.

Solución orientadora

A continuación, se ha realizado una posible respuesta en C a este modelo de examen, no vean esta respuesta hasta no tratar de resolverloustedes mismos, piensen que es para orientarlos en que tan bien o mal van con la materia. Evaluen la dificultad de este modelo para ver que tan listos están para el examen.

Además, se adjunta al final del código la prueba de escritorio del ejercicio 3, recuerden que en la prueba de escritorio uno debe ver la evolución de los parámetros claves que influyen en el programa. Por ejemplo, poner a la matriz en las columnas no aporta nada, puesto que la matriz no cambia en todo el ejercicio.

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>


//////////////////////////////////////////////////

//Este programa es una solucion simple

//y sin muchas validaciones del modelo de parcial

//traten de verlo, entenderlo y mejorarlo

//se tiene un menu simple para elegir las opciones

//////////////////////////////////////////////////



//Ejercicio1 Piramide espejada que

//dibuja el caracter que el usuario

//selecciona con el tamaño de piramide

//tambien dado por el usuario

//Procedimiento para el ejercicio1

void ejercicio1(){


int numero = 0;

char caracter = ' ';


printf("Elija el numero para la priramide espejada: \n");
```

```
scanf("%d", &numero);

fflush(stdin);


printf("Elija con que quiere dibujar: \n");

scanf("%c", &caracter);

fflush(stdin);


for (int i = 0; i<numero; i++){


        for (int j = 0; j<i+1; j++){

            printf("%c", caracter);

        }//cierro for j


        printf("\n");



}//cierra el for i


for (int i = numero; i<(numero*2)-1; i++){


    for(int j = numero-i+numero-1; j>0; j-- ){

        printf("%c", caracter);
```

```
    }//for j

    printf("\n");


}//cierra el for i


}//Fin ejercicio 1


//Este es el ejercicio 2, donde se carga un vector por teclado
//el usuario selecciona que tan largo sera el vector
//una vez cargdo se genera otro igual de largo pero reemplaza
//a los numeros pares por la mitad, y a los impares por el triple
void ejercicio2(){

  int largoVector = 0;

  printf("Ingrese el largo del vector a trabajar: \n");
  scanf("%d", &largoVector);
```

```
  //Declaro vectores
  int vector1[largoVector];
  int vector2[largoVector];

  //Pido por consola el vector1
  for (int i = 0; i <largoVector; i++){
    printf("Ingrese la posicion %d ---> ", i);
    scanf("%d", &vector1[i]);
  }//cierra el for

  //Genero el vector2 cumpliendo con el enunciado
  for (int i = 0; i<largoVector; i++){

    if((vector1[i]%2)==0){
        vector2[i]= vector1[i]/2;
    }
    else {
        vector2[i]= vector1[i]*3;
    }

  }//Cierra el for
```

```
  //Muestro los dos vectores
  for (int i = 0; i<largoVector; i++){
    printf("  %d   ", vector1[i]);
  }

  printf("\n");

  for (int i = 0; i<largoVector; i++){
    printf("  %d   ", vector2[i]);
  }

}//fin ejercicio 2

//Ejercicio 3, donde se carga la matriz de goles de
//jugadores por semestres, una vez cargada
//se generan dos vectores en donde se guarda la suma
//de goles por jugador y semesstre
//La prueba de escritorio esta mas adelante
void ejercicio3(){
```

```
int matriz[3][2] = {{11,20},{10,9},{19,9}};



int totalPorJugador[3] = {0,0,0};

int totalPorSemestre[2] = {0,0};

//muestro

for (int renglon = 0; renglon <3; renglon++){

	for(int columna = 0; columna<2; columna++){

		printf("%d ", matriz[renglon][columna]);

	}printf("\n");}



for (int renglon = 0; renglon <3; renglon++){

	for(int columna = 0; columna<2; columna++){
```

```
        totalPorJugador[renglon] = totalPorJugador[renglon] + matriz[renglon][columna];

        totalPorSemestre[columna] = totalPorSemestre[columna] + matriz[renglon][columna];


        }//cierra columna


}//cierra el for


//Muestro los totales

printf("Totales por jugadores: \n");

printf("El jugador 1 anoto: %d, el jugador 2 anoto: %d, y el jugador 3 anoto: %d", totalPorJugador[0],totalPorJugador[1],totalPorJugador[2]);


printf("\nTotales por semestre: \n");

printf("El semestre 1 se anotaron %d y el semestre 2 se anoto: %d", totalPorSemestre[0],totalPorSemestre[1]);




}//fin ejercicio 3


//ejercicio 4: Se escribe una frase, y se da una letra
```

```
//luego se busca cuantas veces esta la letra en esa

//oracion. Se plantea como una funcion.

int  ejercicio4(){


int total = 0;


char oracion[100];

char buscar;


printf("\nEscriba una oracion: \n");

gets(oracion);


printf("\nIngrese la letra a buscar: \n");

scanf("%c", &buscar);


int numero = strlen(oracion);




for (int i = 0; i<numero; i++){



    if (oracion[i]==buscar){

        total = total +1;
```

```
    }
}


printf("El retorno es: %d", total);

return total;
}


//Arranca el main
int main()
{

    //Menu ssimple para elegir el ejercicio a probar
    //ustedes saben hacer menues mucho mas complejos
    int opcion = 0;

    printf("Que ejercicio quiere probar: \n");
    scanf("%d", &opcion);

switch(opcion){
case 1: ejercicio1(); break;
```

```
case 2: ejercicio2(); break;

case 3: ejercicio3(); break;

case 4: ejercicio4(); break;

default: main();break;

}


    return 0;

}
```

Prueba de escritorio, ejercicio 3:

Operación en C	renglón	columna	totalPorJugador	totalPorSemestre	Consola
inicio	0	0	(0;0;0)	(0;0)	
for renglon	0				
for columna	0	0	(11;0;0)	(11;0)	
columna +1	0	1	(31;0;0)	(11;20)	
renglon +1	1	0	(31;10;0)	(21;20)	
columna +1	1	1	(31;19;0)	(21;29)	
renglon +1	2	0	(31;19;19)	(40; 29)	
columna + 1	2	1	(31;19;28)	(40; 38)	
Muestro goles por jugador					(31;19;28)
Muestro goles por semestre					(40; 38)

Notar que los vectores se los pueden ir viendo con todas sus coordenadas a la par, no es necesario que solo se muestre la coordenada que ha cambiado, esto lo hace mas entendible y se ve mejor la evolución de dicho vector.

Ejercicio 7: Se propone crear un menú que, 1 pida una letra y una palabra y diga cuantas veces esta esa letra en esa palabra, 2 dada una palabra diga si es o no palíndromo, 3 pida un entero y nos diga si es o no primo, 4 dibuje un cuadrado de a x b con a y b tamaños ingresados por el usuario y 5 salga del menú.