



PROGRAMACION ESTRUCTURADA BASICA

UNIDAD 1.

Algoritmos con Colecciones y

Manejo de Texto (strings)

INDICE

1. ALGORITMOS CON COLECCIONES.....	3
1.1 MÁXIMOS Y MÍNIMOS MÚLTIPLES	3
1.2 BÚSQUEDA SECUENCIAL EN VECTORES.....	6
1.3 CARGA DE UN VECTOR SIN ADMITIR VALORES REPETIDOS.....	7
1.4 ORDENAMIENTO DE VECTORES	8
1.4.1 Ordenamiento por selección	8
1.4.2 Ordenamiento por Burbujeo	11
1.4.3 Ordenamiento por Burbujeo optimizado	13
1.5 ORDENAMIENTO DE VECTORES PARALELOS.....	14
2. MANEJO DE TEXTO (STRINGS)	14
INTRODUCCIÓN	14
1. LECTURA POR TECLADO DE STRINGS	15
1.1 SCANF	15
1.2 GETS	16
1.3 FGETS.....	16
2. MOSTRAR STRINGS POR PANTALLA	17
3. INICIALIZACIÓN DE STRINGS.....	18
4. BIBLIOTECA PARA EL MANEJO DE TEXTO (STRING.H)	18
5. VECTOR DE STRINGS	21
6. FUNCIONES SOBRE VECTORES DE STRING	21
6.1 CARGA	21
6.2 MOSTRAR	22
6.3 BÚSQUEDA SECUENCIAL.....	22
6.4 ORDEN	22
7. EJEMPLO DE APLICACIÓN	23

1. Algoritmos con Colecciones

OBJETIVOS: *Realizar un manejo avanzado de las colecciones de datos aplicando distintos algoritmos sobre las colecciones y combinándolas para permitir resolver problemas complejos.*

Las colecciones de datos (vectores y matrices) permiten manejar grandes volúmenes de datos de una forma rápida y sencilla. Sobre dichas colecciones es posible realizar distintos algoritmos para trabajar sobre los datos como por ejemplo buscar un dato, ordenar los valores que contienen, etc. En esta unidad se verán algunos algoritmos sobre las colecciones y se combinará su uso para permitir la resolución de problemas más complejos.

1.1 Máximos y Mínimos Múltiples

Dado un conjunto de datos, muchas veces es necesario recuperar el valor más grande o el más chico del lote, para ello se realizará un algoritmo donde el primer elemento del conjunto es tomado como referencia y luego comparado con el resto. Como todos los datos ya están todos almacenados en una colección se podrá no solo determinar el mayor valor del conjunto, sino también determinar cuántas veces se repite y en que posiciones de la colección se encuentra.

El procedimiento consta de dos pasos:

1. Determinar el máximo o el mínimo valor del conjunto de datos
2. Volver a recorrer el conjunto de datos para ver cuántas veces se repite y donde se encuentra

El siguiente programa permite ingresar por teclado un vector de 10 elementos utilizando una función de carga completa, luego determina el valor máximo mediante una función y por último informa mediante otra función, la cantidad de repeticiones y las posiciones del máximo.

```
#include <stdio.h>
void Carga(int[], int);
int Maximo(int[], int);
void MostrarMaximo(int[],int,int);

int main()
{
    int vec[10], max;
    Carga(vec,10);
    max = Maximo(vec,10);
    MostrarMaximo(vec,max,10);
    return 0;
}
```

```

void Carga(int v[], int N)
{
    int i;
    for (i=0;i<N;i++)
    {
        printf("Ingrese un numero (%d): ",i+1);
        scanf("%d",&v[i]);
    }
}

int Maximo(int v[], int N)
{
    int max = v[0], i;
    /*se comienza en 1 ya que el primero
    se tomó como referencia*/
    for (i=1;i<N;i++)
    {
        if (v[i]>max)
            max = v[i];
    }
    return max;
}

void MostrarMaximo(int v[],int max,int N)
{
    int cont=0,i;
    printf("El valor maximo es: %d y se encuentra en las \
    siguientes posiciones del vector:\n",max);
    for (i=0;i<N;i++)
    {
        if (v[i]==max)
        {
            printf("%d\n", i+1);
            cont++;
        }
    }
    printf ("El valor maximo se repite %d veces", cont);
}

```

Si en lugar del máximo se desea calcular el mínimo el procedimiento es similar. El primero se toma como referencia y luego se compara si alguno de los siguientes es menor al guardado como referencia.

De la misma forma este procedimiento puede aplicarse a una matriz. Se toma como referencia el primer valor (posición 0,0) y luego se recorre toda la matriz con un doble for comparando con dicho valor de referencia. Una vez obtenido el valor deseado se vuelve a recorrer la matriz para ver donde se repite dicho valor.

```
#include <stdio.h>
void CargaMatriz(int[][5], int, int);
int ValorMinimo(int[][5], int, int);
void MostrarValor(int[][5],int,int,int);
void MostrarMatriz(int[][5], int, int);

int main()
{
    int mat[3][5], min;
    CargaMatriz(mat,3,5);
    min = ValorMinimo(mat,3,5);
    MostrarMatriz(mat,3,5);
    MostrarValor(mat,min,3,5);
    return 0;
}

void CargaMatriz(int m[][5], int cf, int cc)
{
    int f,c;
    for (f=0;f<cf;f++)
        for (c=0;c<cc;c++)
        {
            printf("Ingrese un numero \
                (en fila %d columna %d):", f+1,c+1);
            scanf("%d",&m[f][c]);
        }
}

void MostrarMatriz(int m[][5], int cf, int cc)
{
    int f,c;
    for (f=0;f<cf;f++)
    {
        for (c=0;c<cc;c++)
            printf("%4d", m[f][c]);
        printf("\n");
    }
}

int ValorMinimo(int m[][5], int cf, int cc)
{
    int min = m[0][0], f,c;
    for (f=0;f<cf;f++)
        for (c=0;c<cc;c++)
            if (m[f][c]<min)
                min = m[f][c];
    return min;
}
```

```
void MostrarValor(int m[][5],int valor, int cf, int cc)
{
    int cont=0,i;
    printf("El valor es: %d y se encuentra en las \
    siguientes posiciones de la matriz:\n",valor);

    int min = m[0][0], f,c;
    for (f=0;f<cf;f++)
        for (c=0;c<cc;c++)
            if (m[f][c]==valor)
            {
                printf("fila %d columna %d\n", f+1,c+1);
                cont++;
            }

    printf ("El valor se repite %d veces", cont);
}
```

En ambos ejemplos a las posiciones que se muestran se les suma 1 para que sea más amigable y natural para el usuario.

1.2 Búsqueda secuencial en vectores

Frecuentemente es necesario buscar un determinado elemento en un vector, ya sea para determinar si el mismo está en el vector o para recuperar la posición en donde se encuentra.

Los datos del problema son:

- El vector.
- El elemento a buscar.
- La cantidad de elementos sobre los cuales se va a realizar la búsqueda

Las salidas son:

- Un indicador de la existencia del elemento en la colección
- La posición donde se encuentra.

Como la función solo puede retornar un valor, ambas salidas se unifican en una que represente, si existe, la posición donde se encuentra ($0 \leq \text{posición} \leq \text{cantidad de elementos}$) y si no existe retornará un -1 (valor inválido como posición).

Si bien existen distintos métodos de búsqueda, veremos el más sencillo de ellos que consiste en ir recorriendo uno a uno los elementos del vector hasta que encontremos el buscado. Este método se denomina **Búsqueda Secuencial**.

```
int Buscar(int vector[], int datoABuscar, int cantElem)
{
    int i=0,pos=-1;
    while (pos==-1 && i<cantElem)
    {
        if (vector[i]==datoABuscar)
            pos = i;
        else
            i++;
    }
    return pos;
}
```

Este algoritmo inicia con dos variables la variable i utilizada como índice para moverse por las posiciones del vector y la variable pos que indica la posición donde se encuentra el dato que se está buscando. De entrada, se inicia con un valor -1 que indica que el dato aún no se ha encontrado.

Se realiza un ciclo while ya que no siempre se debe recorrer el vector completo, una vez que se encuentra el dato se sale y no se sigue recorriendo es por ello que el ciclo while tiene dos condiciones:

pos == -1 indica que aún el dato no se ha encontrado

i < cantElem indica que aún quedan datos sobre los cuales seguir realizando la búsqueda

Como ambas condiciones están unidas con el operador AND ambas deben ser verdaderas para ingresar al ciclo entonces ingresa mientras no se haya encontrado el dato buscado y aún quedan datos sobre los cuales realizar la búsqueda.

El ciclo termina si se encuentra el dato al cambiar el valor de pos o cuando no se encuentra pero se llegó al final del vector porque el índice i es igual a la cantidad de elementos y por lo tanto se recorrió todo el vector ya que el primer índice válido es el 0.

La función retorna el contenido de la variable pos que quedará con la posición donde se encontró el dato buscado o -1 si nunca se lo encuentra.

1.3 Carga de un vector sin admitir valores repetidos

Utilizando la función de búsqueda es posible modificar la carga de un vector asegurándonos de que todos los valores ingresados sean diferentes. Para ello por cada vez que se ingrese un dato se buscará si el mismo ya está en el vector y si está se solicitará un dato diferente.

```
void CargarSinDuplicados(int vector[], int cant)
{
    int i, pos, aux;
    for (i=0; i<cant; i++)
    {
        do
        {
            printf("Ingrese un número:");
            scanf("%d",&aux);
            pos = Buscar(vector, aux, i);
            if (pos!=-1)
                printf("Dato duplicado. Ingrese otro\n");
        }while(pos!=-1);
        vector[i] = aux;
    }
}
```

Puede verse que cada vez que se ingresa un dato, se busca si ya estaba en el vector y si estaba se vuelve a solicitar otro dato que se guarda en la misma posición del vector. No se avanza de posición del vector hasta que no se cargue un dato diferente al resto. La función de búsqueda se llama enviando la variable i como cantidad de datos del vector que indica cuantos se cargaron previamente. La primera vez se envía 0 como tamaño del vector por lo tanto dentro de la función de búsqueda ni siquiera ingresa al ciclo y retorna directamente -1, lo que es correcto ya que al no existir datos previos no puede haber duplicados. La segunda vez ya comparará si el segundo dato ingresado coincide con el primero, y así sucesivamente cada vez se hará la búsqueda con más posiciones del vector que ya fueron cargadas previamente.

1.4 Ordenamiento de vectores

Los métodos de ordenamiento son numerosos. Podemos considerar dos grandes grupos:

- Directos: Burbujeo, selección e inserción -
- Indirectos (avanzados): Shell, ordenación por mezcla.

En listas pequeñas, los métodos directos se comportan en forma eficiente, mientras que, en vectores con gran cantidad de elementos, se debe recurrir a métodos avanzados.

1.4.1 Ordenamiento por selección

Tomando el siguiente vector de punto partida el objetivo será ordenarlo de menor a mayor.

10	6	2	8	3
----	---	---	---	---

¿Como podríamos ordenarlo haciendo que en la primera posición se ubique el número más chico luego el siguiente y así hasta que queden orden creciente?

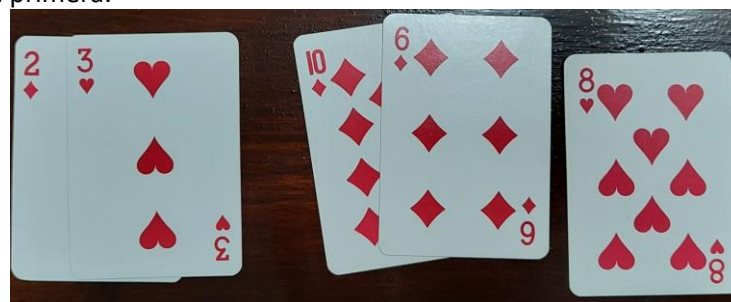
¿Para ayudarnos a pensar como lo haríamos imaginemos como haríamos para ordenar el siguiente conjunto de cartas?



Una forma sería buscar la carta más chica y ponerla primero.



Luego de las que quedaron desordenadas nuevamente buscar la mas chica y ponerla a continuación de la primera.



Y así seguir con el mismo procedimiento



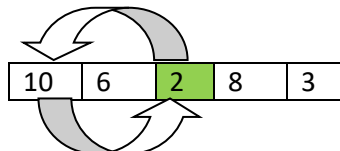
Hasta que queden todas ordenadas



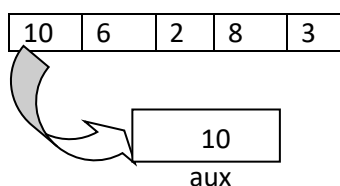
Volvamos ahora al vector que planteamos al comienzo y apliquemos el mismo algoritmo, el primer paso será buscar el menor del vector. En este caso el menor valor es el 2

10	6	2	8	3
----	---	---	---	---

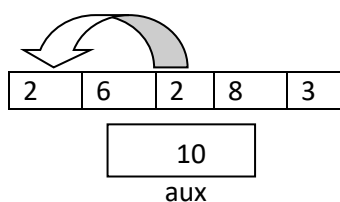
Entonces ese valor tenemos que llevarlo a la primera posición del vector, pero como el valor que se encuentra en la primera posición no tenemos que perderlo lo que vamos a hacer es intercambiarlo con el 2.



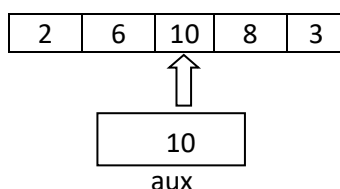
Pero recordemos que la asignación es destructiva no puedo guardar el 2 en la primera posición sin perder el 10, por lo tanto antes hay que copiar el 10 para no perderlo en una variable auxiliar.



Una vez resguardado el 10 podemos copiar el 2 en la primera posición del vector



Y luego donde estaba el 2 asignar lo que tenemos en la variable auxiliar para completar el intercambio



De esta forma ahora el 2 quedó en la primera posición y ya quedó ordenado. Ahora hay que repetir el procedimiento para ordenar el resto de los números, buscando el menor valor entre los desordenados es decir que ahora consideramos el vector para la búsqueda del menor partiendo desde el segundo valor ya que el primero está ordenado y se repite el mismo procedimiento en el segundo paso quedará ordenado el 3 y será intercambiado por el 6

2	3	10	8	6
---	---	----	---	---

Así se debe seguir el procedimiento siempre buscando el menor en un conjunto de datos que cada vez será más chico hasta lograr tener todo ordenado. La cantidad de repeticiones total es decir la cantidad de veces que debemos buscar el mínimo va a depender del tamaño del vector si el vector es de 5 elementos se debe buscar el mínimo 4 veces ya que cuando queden solo dos elementos al recuperar el mínimo el último quedará automáticamente ordenado.

Llevemos ahora dicho algoritmo a una función donde para la determinar la posición del menor valor del conjunto se utiliza una segunda función.

```
void OrdenSeleccion (int v[], int tam)
{
    int i, pmin, aux;
    for (i=0; i<tam-1; i++)
    {
        pmin = PosMinimo (v, i, tam);
        if (pmin!=i)
        {
            aux = v[i];
            v[i] = v[pmin];
            v[pmin] = aux;
        }
    }
}

int PosMinimo (int v[], int ini, int tam)
{
    int i, posMin;
    posMin = ini;

    for (i=ini+1; i<tam; i++)
        if (v[i]<v[posMin])
            posMin = i;

    return posMin;
}
```

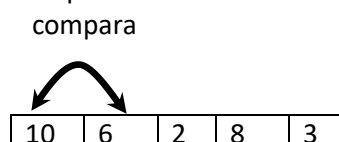
En el código de la función puede verse que el ciclo se hace restando 1 ya que último quedará ordenado como se mencionó anteriormente. La determinación de la posición del mínimo dentro del vector a ordenar se delega en una función que recibe el vector, desde donde buscar el mínimo (para saltar los ya ordenados) y la cantidad de datos del vector. También puede notarse que solo se hace el intercambio si en la primera posición del vector desordenado (dado por la variable i) no se encuentra ya el menor valor.

En el caso de que se desee ordenar en forma descendente (de mayor a menor) el único cambio será utilizar una función que devuelva la posición del máximo dentro del vector en lugar de la posición del mínimo.

1.4.2 Ordenamiento por Burbujeo

Otro método de ordenamiento fácil de recordar es el ordenamiento por burbujeo que lo que hace es ir comparando posiciones adyacentes y haciendo intercambios entre dichas posiciones.

Por ejemplo para ordenar de menor a mayor el mismo vector visto anteriormente, el algoritmo comienza comparando las dos primeras posiciones

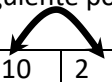


Y como el 10 es más grande que el 6 entonces hace el intercambio ya que buscamos que quede siempre el mas chico primero.

6	10	2	8	3
---	----	---	---	---

Luego sigue comparando, pero desde la siguiente posición es decir compara ahora el 10 con el 2

6	10	2	8	3
---	----	---	---	---

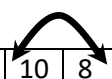


Y como también el 2 es menor entonces intercambia

6	2	10	8	3
---	---	----	---	---

Se sigue con el mismo procedimiento, pero comparando ahora la 3er posición del vector con la cuarta

6	2	10	8	3
---	---	----	---	---

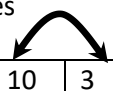


Y como el 10 es mayor que el 8 se debe realizar el intercambio

6	2	8	10	3
---	---	---	----	---

Se sigue comparando ahora las dos últimas posiciones

6	2	8	10	3
---	---	---	----	---



Y también se debe intercambiar quedando este vector

6	2	8	3	10
---	---	---	---	----

Como puede apreciarse el algoritmo luego de recorrer el vector una vez comparando posiciones adyacentes lo que hace es llevar a la última posición el valor más grande. Es decir que con este recorrido solo se ordenó un valor. Por lo tanto, hay que repetir el procedimiento para ordenar los 4 primeros valores ya que el último quedó ordenado.

El código de algoritmo es el siguiente:

```
void OrdenBurbujeo (int v[], int tam)
{
    int i,j, aux;
    for (i=0;i<tam-1;i++)
    {
        for (j=0;j<tam-1-i;j++)
            if (v[j]>=v[j+1])
            {
                aux = v[j];
                v[j] = v[j+1];
                v[j+1] = aux;
            }
    }
}
```

El ciclo externo indica la cantidad de veces que hay que recorrer el vector para que quede ordenado y el interno es el que controla la cantidad de comparaciones que hay hacer en cada recorrido. Nótese que al ciclo interno se le resta el índice del ciclo externo, esto se debe a que en cada recorrido hay un dato más que ya queda ordenado en las últimas posiciones del vector y por lo tanto cada vez se hacen menos comparaciones ya que la parte desordenada del vector es más chica.

En el caso de que se desee ordenar en forma descendente (de mayor a menor) el único cambio será cambiar el signo de comparación para realizar el intercambio si el dato de la posición j es menor al dato de la posición $j+1$.

1.4.3 Ordenamiento por Burbujeo optimizado

El algoritmo visto anteriormente puede mejorarse de dos formas:

- Detectando si el vector se llega a ordenar antes de realizar todos los recorridos
- Mirando hasta donde está ordenado el vector ya que nos pueden entregar un vector parcialmente ordenado.

Para utilizaremos el siguiente algoritmo:

```
void OrdenBurbujeoOptimizado (int v[], int tam)
{
    int i, desordenado, aux, limite=tam-1;
    do
    {
        desordenado = 0;
        for (i=0; i<limite; i++)
        {
            if (v[i]>v[i+1])
            {
                aux = v[i];
                v[i] = v[i+1];
                v[i+1] = aux;
                desordenado = i;
            }
        }
        limite = desordenado;
    } while (desordenado);
}
```

Este algoritmo implementa las dos mejoras, en lugar de un doble ciclo for el ciclo externo se maneja con un ciclo do while, para recorrer el vector y si no se realizaron intercambios en todo el recorrido quiere decir que ya estaba ordenado y por lo tanto no hace falta volver a recorrerlo.

El límite del ciclo interna se maneja con una variable que inicial recorre hasta el final del vector, pero para ciclos siguientes dicho límite será la posición del último intercambio ya que si al recorrer una vez no se hicieron intercambios luego de dicha posición era porque el resto del vector ya estaba ordenado.

1.5 Ordenamiento de vectores paralelos

Cuando se tienen vectores paralelos al ordenar un vector se debe también realizar el intercambio en el o los vectores que guardan información relacionada. Por ejemplo, si se tienen dos vectores paralelos uno con dni de alumnos y otro con las nota y si busca mostrar un listado ordenado por nota, si los intercambios solo se realizan sobre el vector de notas quedarían mezclados los dni de los alumnos con una nota que no les corresponde. Al ordenar vectores en paralelo el algoritmo se aplica sobre el vector que se quiere ordenar, pero al momento de realizar el intercambio se lo hace también, sobre los vectores relacionados.

```
void Ordenar (int vo[], int vi[], int ce)
{
    int i,j, aux;
    for (i=0;i<ce-1;i++)
        for (j=0;j<ce-1-i;j++)
            if (vo[j] < vo[j+1])
            {
                aux = vo[j];
                vo[j] = vo[j+1];
                vo[j+1] = aux;

                aux = vi[j];
                vi[j] = vi[j+1];
                vi[j+1] = aux;
            }
}
```

En este caso se necesita una única variable auxiliar ya que los vectores son del distinto tipo, si fueran de distinto tipo de dato entonces se necesitarían dos variables como auxiliares para realizar el intercambio.

2. Manejo de Texto (strings)

OBJETIVOS: *Comprender la forma para el manejo de texto en el lenguaje C. Utilizar funciones para trabajar con texto. Incorporar el manejo de texto a los programas existentes.*

Introducción

El lenguaje C, a diferencia de otros lenguajes, hace un manejo muy particular de las variables para almacenar texto. Un texto no es más que una sucesión de letras, por lo tanto, se necesita almacenar varios caracteres uno a continuación del otro.

Una variable del tipo char permite almacenar un único carácter, por lo tanto, para almacenar un texto se debe definir un vector de caracteres con tantas posiciones como letras pueda tener el texto que se quiere almacenar. Entonces, en un principio se podría decir que una variable del tipo string o cadena de caracteres, no es más que un vector de char. Sin embargo, esta afirmación no es

del todo cierta ya que hay una característica que las diferencia. Toda cadena de caracteres define su terminación con un carácter de control '\0'. Este carácter especial indica el fin de la cadena.

Por ejemplo, si se quiere ingresar un nombre, no se sabe exactamente el largo del mismo por lo tanto se define un vector con una cantidad suficiente de acuerdo al nombre más largo que se quiera guardar por ejemplo:

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

Declara un vector de char de 20 posiciones. Si en dicho vector se almacena un nombre corto, por ejemplo "JUAN", dicho nombre solo ocupará 4 de los 20 caracteres, y por lo tanto, luego de la letra N se debe poner el carácter de fin de cadena indicando hasta dónde llega dicho nombre.

J	U	A	N	\0															
---	---	---	---	----	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Entonces la diferencia principal entre un vector de caracteres y un string es la presencia de ese carácter especial de fin de cadena.

1. Lectura por teclado de strings

Ahora bien, como se indicó, un string es un vector de caracteres, pero al momento de solicitar el ingreso por teclado del dato no se le va a pedir al usuario que ingrese una a una las letras en cada posición del vector. Para la lectura de string el lenguaje dispone de distintas alternativas:

1.1 scanf

Tradicionalmente todos los ingresos desde teclado hasta ahora se realizaron utilizando esta función. Donde se pone como primer parámetro el formato de los datos a ingresar y como segundo parámetro la dirección de memoria donde almacenar el dato. El formato para los strings es %s. Veamos el siguiente ejemplo:

```
#include <stdio.h>
int main()
{
    char nombre[20];
    printf ("Ingrese un nombre: ");
    scanf("%s", nombre);
    printf ("El nombre ingresado es: %s", nombre);
    return 0;
}
```

Note que en el scanf no fue necesario poner el & para obtener la dirección de memoria ya que al ser nombre un vector contiene la dirección de inicio del mismo. Por lo tanto, el & NO debe ponerse ya que contiene la dirección de memoria que se necesita para indicarle donde guardar el dato.

Si se ejecuta y se prueba este programa e ingresa por teclado JUAN y luego presiona enter, automáticamente se completa el vector guardando la J en la posición 0, la U en la posición 1, y así sucesivamente, hasta la última letra y automáticamente en la posición siguiente a la letra N el scanf agrega el carácter de fin de cadena \0 ya que se le indica que está leyendo un string con el formato %s.

Hasta aquí no hay problemas y todo funciona normalmente. Pero el scanf con formato de string tiene un problema. Si prueba nuevamente el ejemplo y en el nombre ingresa JUAN CARLOS, en pantalla mostrará:

```
El nombre ingresado es: JUAN
```

Internamente en el vector solo guardará JUAN y a continuación el carácter de fin de cadena. Esto se debe a que el espacio corta el ingreso del scanf almacenando solo JUAN y haciendo que se pierdan el resto de los datos. Recordemos que cuando con scanf se leía más de una variable una forma de separarlas era con espacio y por ello no funciona correctamente en la lectura de string.

1.2 gets

En la biblioteca stdio.h hay una función llamada gets que permite solucionar el problema del scanf al leer un string. Reemplazando dicha función en el programa anterior quedaría:

```
#include <stdio.h>
int main()
{
    char nombre[20];
    printf ("Ingrese un nombre: ");
    gets(nombre);
    printf ("El nombre ingresado es: %s", nombre);
    return 0;
}
```

De esta forma se elimina el problema del scanf y si se ingresa por teclado JUAN CARLOS en el vector los datos quedarán almacenados de la siguiente forma:

J	U	A	N		C	A	R	L	O	S	\0								
---	---	---	---	--	---	---	---	---	---	---	----	--	--	--	--	--	--	--	--

Y en pantalla se mostrará:

```
El nombre ingresado es: JUAN CARLOS
```

1.3 fgets

Tanto el scanf como gets NO CHEQUEAN LIMITES, por lo tanto, si se define el vector para guardar 20 caracteres y se ingresa un nombre más largo la variable se guarda en las posiciones siguientes que NO tiene reservadas haciendo que probablemente se pierdan datos de otras variables, y por lo tanto, haciendo que el programa funcione mal. Algunos compiladores darán una advertencia diciendo que es peligroso utilizar el gets debido a que se puede escribir memoria no reservada.

Para solucionar esto se puede acudir a otra función que se encuentra también en la biblioteca stdio.h. Esta función es fgets y tiene 3 parámetros, el primero es el vector donde se va a guardar el dato, el segundo la cantidad de caracteres que se pueden almacenar en dicho vector y el tercero es de donde se leen los datos que en este caso siempre será desde teclado. La siguiente línea:

```
fgets(nombre,20, stdin);
```

Indica que se va a leer desde teclado (indicado por stdin), como máximo 20 caracteres y se va a guardar a partir de la dirección de memoria indicada por nombre.

Reemplazando en el programa anterior:

```
#include <stdio.h>
int main()
{
    char nombre[20];
    printf ("Ingrese un nombre: ");
    fgets(nombre,20,stdin);
    printf ("El nombre ingresado es: %s", nombre);
    return 0;
}
```


Al probar este programa si se ingresa un texto de más de 20 caracteres, guardará en el vector los 19 primeros y en la última posición pondrá el carácter de fin de cadena \0, el resto de caracteres no los guarda, y por lo tanto, no sobrescribe memoria no reservada.

Al parecer esta función soluciona todo los problemas pero si se prueba nuevamente ingresando por teclado JUAN CARLOS y luego la tecla enter, se verá que el vector almacena lo siguiente:

J	U	A	N		C	A	R	L	O	S	\n	\0							
---	---	---	---	--	---	---	---	---	---	---	----	----	--	--	--	--	--	--	--

Es decir, que la función fgets almacena también el enter ingresado. Por lo tanto, cuando se muestra la cadena luego de JUAN CARLOS dejará un salto de línea. Dependerá del uso que quiera darle al string si ese salto de línea molesta o no. Una posible solución luego de leer el string con fgets, sería buscar si quedó almacenado el \n y eliminarlo poniendo en su lugar el \0. En el siguiente ejemplo ingreso del string se realiza mediante una función LeerTexto que lee mediante fgets y luego busca y elimina el salto de línea almacenado:

```
#include <stdio.h>
void LeerTexto (char[], int);
int main()
{
    char nombre[20];
    printf ("Ingrese un nombre: ");
    LeerTexto(nombre,20);
    printf ("El nombre ingresado es: %s", nombre);
    return 0;
}

void LeerTexto (char texto[], int largo)
{
    int i;
    fgets(texto, largo, stdin);
    i=0;
    while (texto[i]!='\0')
    {
        if (texto[i]=='\n')
            texto[i]='\0';
        else
            i++;
    }
}
```

2. Mostrar strings por pantalla

En los ejemplos anteriores se vio que es posible mostrar los string mediante un printf con formato %s, esta es la forma más difundida y es la que más utilizará ya que permite darle formato, mostrar otras variables, etc.

Otra forma para mostrar un string es mediante la función puts disponible en la biblioteca stdio.h. Esta función recibe como parámetro el string y lo muestra por pantalla y además automáticamente luego de mostrar el texto hace un salto de línea.

Es decir, que la siguiente línea de código (definiendo nombre como un string):

```
puts(nombre);
```

es equivalente a:

```
printf ("%s\n", nombre);
```

3. Inicialización de Strings

Al declarar la memoria del string es posible asignar un valor, para ello se le asigna una constante del tipo string que se escribe entre comillas. El carácter de fin de cadena se agrega automáticamente luego de la última letra. A continuación, observe algunos ejemplos de inicialización y como queda el vector resultante:

```
char nombre[20] = "ANA MARIA";
```

A	N	A		M	A	R	I	A	\0										
---	---	---	--	---	---	---	---	---	----	--	--	--	--	--	--	--	--	--	--

```
char nombre[] = "ANA MARIA";
```

A	N	A		M	A	R	I	A	\0
---	---	---	--	---	---	---	---	---	----

Si no se especifica el tamaño automáticamente reserva la memoria mínima suficiente para almacenar el texto y el carácter de fin de cadena.

Cuidado!! Si se le pone un tamaño al vector y se le asignan más caracteres de los definidos NO guardará el \0, y por lo tanto, ya no puede tratarse como un string ya que la función para mostrar no encontrará el fin de cadena.

```
char nombre[3] = "MARCELO";
```

M	A	R
---	---	---

4. Biblioteca para el manejo de texto (string.h)

Debido a que los string son vectores con un carácter especial que indica el final de la cadena hay ciertas cosas como copiar uno a otro, comparar, etc que NO se pueden hacer directamente ya que no son un tipo de dato en sí mismos. Identificando el fin de cadena es posible armar distintas funciones para hacer esas tareas. Por ejemplo, para saber la cantidad de caracteres de un string se puede recorrer posición a posición el vector hasta encontrar el fin de cadena contando cuantas posiciones nos desplazamos. Para copiar un string en otro es similar a la copia de vectores donde se copia posición a posición en este caso hasta encontrar el fin de cadena. Todas estas funciones las podemos desarrollar pero dentro del lenguaje C ya existe una biblioteca que maneja distintas funciones relacionadas con cadenas de caracteres. La biblioteca string.h

Dentro de string.h hay muchas funciones, veremos algunas de ellas:

strlen	Determina la longitud de una cadena
strcpy	Copia una cadena a otra
strcat	Concatena dos cadenas dejando el resultado en la primera
strcmp	Compara dos cadenas
strcmpi	Compara dos cadenas ignorando si son mayúsculas o minúsculas

Función strlen:

Determina la longitud de una cadena, sin contabilizar el carácter nulo de terminación de la misma.

Sintaxis: strlen (cadena)

La función retorna un entero indicando la cantidad de caracteres.

Ejemplo:

```
#include <stdio.h>
#include <string.h>
int main ()
{
    char cadena [21];
    printf ("Ingrese una cadena de no más de 20 caracteres \n");
    gets (cadena);
    printf ("La cadena ingresada contiene: %d caracteres", strlen(cadena) );
    return 0;
}
```

Función strcpy:

Copia desde una cadena de origen hacia una cadena de destino.

Sintaxis: strcpy (cadena destino, cadena origen)

Ejemplo:

```
#include <stdio.h>
#include <string.h>
int main ()
{
    char original [15], copia [15];
    printf ("Ingrese una cadena que sera luego copiada \n");
    gets (original);
    strcpy (copia, original);
    printf ("La cadena original es: %s y la copia es: %s", original, copia );
    return 0;
}
```

También strcpy se puede usar para asignar una constante a un string cuando no se hace al momento de inicializar la variable.

```
#include <stdio.h>
#include <string.h>
int main ()
{
    char texto [25];
    strcpy(texto, "MENSAJE");
    puts(texto);
    return 0;
}
```

Función strcat:

Concatena (añade) una cadena detrás de otras quedando el resultado en la cadena que se encuentra en primer orden.

Sintaxis: strcat (cadena receptora, cadena a añadir)

Ejemplo:

```
#include <stdio.h>
#include <string.h>
int main ()
{
    char receptor [40] = "Se agrego lo siguiente", dador [] = "me agregue";
    printf ("Las cadenas por separado son: \n\t %s\n\t %s", receptor, dador);
}
```

```

    strcat (receptor, dador);
    printf ("\nLas cadenas unificadas son: \n\t %s ", receptor );
    return 0;
}

```

Es importante asegurarse de que la cadena receptora tenga el espacio suficiente para guardar la cadena a añadir caso contrario sobrescribe memoria no asignada

Función strcmp:

Esta función compara dos cadenas y devuelve el resultado de la comparación.

Sintaxis: strcmp (cadena 1, cadena 2)

El valor que devuelve que será el resultado de la comparación es el siguiente:

Si las cadenas son iguales	devolverá un cero (0)
Si la cadena 1 es mayor que la cadena 2	devolverá un valor positivo
Si la cadena 1 es menor que la cadena 2	devolverá un valor negativo

Ejemplo:

```

#include <stdio.h>
#include <string.h>
int main ()
{
    char cadena1[30], cadena2[30];
    printf ("Ingrese la primer cadena:");
    gets(cadena1);
    printf ("Ingrese la segunda cadena:");
    gets(cadena2);
    if (strcmp (cadena1, cadena2) == 0)
        printf ("\nAmbas cadenas son iguales " );
    else
        if (strcmp (cadena1, cadena2) > 0 )
            printf ("\nLa cadena1 es mayor que la cadena2");
        else
            printf ("\nLa cadena2 es mayor que la cadena1");
    return 0;
}

```

La comparación NO es por largo sino alfabética, comparando el peso en valor ASCII de cada una de las letras, posición a posición, hasta encontrar alguna diferencia. Hay que recordar que el ASCII de las minúsculas es mayor que el de las mayúsculas, por lo tanto, una palabra en minúsculas será mayor que una en mayúscula. A continuación se muestran algunos ejemplos:

Cadena1	Cadena2	Mayor / iguales
HOLA	HOLA	IGUALES
Hola	hola	Cadena2
hola mundo	hola	Cadena1
ana	anana	Cadena2
teXto	texto	Cadena2

Función strcmpi:

Es exactamente igual a la función strcmp pero ignora si son mayúsculas o minúsculas es decir que el string "ANA" es igual al string "ana" y también es igual al string "anA" o cualquiera de sus variantes.

5. Vector de strings

Un string es en realidad un vector de caracteres, ahora si quiere guardar varios string en un vector entonces tiene que definir varios vectores de caracteres y para ello se utiliza una matriz. Por lo tanto, un vector de strings es una matriz de caracteres donde en cada fila se guardará un string finalizado con \0. Si bien se define como una matriz se manejará tanto para la lectura como para mostrar como un vector ya que no se va a recorrer letra a letra sino que se va a leer y mostrar las palabras completas.

Para definir un vector de string entonces se define una matriz donde:

La cantidad de filas es la cantidad de string que se quiere guardar y la cantidad de columnas indica la cantidad máxima de caracteres de cada uno de los strings (contando el \0).

El siguiente esquema representa un vector de string, donde se pueden almacenar 5 nombres de hasta 10 caracteres cada uno (se define de 11 para el \0).

char nombres[5][11];

A	N	A	\0							
L	U	I	S	\0						
J	U	A	N		P	A	B	L	O	\0
L	U	C	I	A	N	O	\0			
M	A	R	I	A	\0					

Los nombres ingresados en cada fila pueden ser de cualquier largo siempre y cuando no pasen la cantidad máxima de caracteres reservada.

6. Funciones sobre vectores de string

Como se indicó anteriormente si bien un vector de string es una matriz se maneja como un vector. A continuación se desarrollan algunas de las funciones más usadas. Tomando cadenas de 10 caracteres máximo.

6.1 Carga

Esta función realiza el ingreso de 10 nombres en un vector.

```
void carga(char vn[][11],int cant)
{
    int i;
    for (i=0;i<cant;i++)
    {
        printf ("Ingrese el nombre numero %d: ", i+1);
        gets(vn[i]);
    }
}
```

6.2 Mostrar

Esta función muestra el contenido de un vector de strings uno debajo del otro.

```
void mostrar(char vn[][11],int cant)
{
    int i;
    for (i=0;i<cant;i++)
        puts(vn[i]);
}
```

En lugar del puts también es posible mostrar utilizando el printf :

```
printf("%s\n", vn[i]);
```

6.3 Búsqueda Secuencial

Esta función busca de forma secuencial un texto dentro de un vector de texto, al encontrarlo retorna la posición del vector donde lo encontró o -1 sino lo encuentro. Nótese que para comparar texto se utiliza la función strcmpi disponible en la biblioteca string.h

```
int buscar(char vn[][11],char buscado[], int cant)
{
    int i=0, pos=-1;
    while (pos==-1 && i<cant)
    {
        if (strcmpi(vn[i], buscado)==0)
            pos=i;
        else
            i++;
    }
    return pos;
}
```

6.4 Orden

Esta función ordena un vector de texto en forma alfabética utilizando el método de burbujeo.

```
void ordenar (char vn[][11],int cant)
{
    int i,j;
    char aux[11];
    for (i=0;i<cant-1;i++)
    {
        for (j=0;j<cant-1-i;j++)
        {
            if (strcmpi(vn[j],vn[j+1]) > 0)
            {
                strcpy(aux, vn[j]);
                strcpy(vn[j],vn[j+1]);
                strcpy(vn[j+1], aux);
            }
        }
    }
}
```

7. Ejemplo de aplicación

Utilizando las funciones definidas anteriormente se desea realizar el siguiente programa, cargar 5 nombres de hasta 10 caracteres cada uno en un vector de string. Luego ingresar nombres por teclado e indicar si se encuentran en el vector, la búsqueda finalizará con un nombre igual a "FIN". Luego mostrar el listado de nombres ordenado alfabéticamente de menor a mayor.

No se desarrollan las funciones ya que son las mismas del punto 7.

```
#include <stdio.h>
#include <string.h>
void carga(char[][11],int);
void mostrar(char[][11],int);
int buscar(char[][11],char [], int);
void ordenar (char[][11],int);
int main ()
{
    int pos;
    char nombres[5][11], abuscar[11];
    carga(nombres,5);
    printf("\nListado de nombres ingresado:\n");
    mostrar(nombres,5);
    printf ("\nIngrese un nombre a buscar (FIN para terminar): ");
    gets(abuscar);
    while (strcmpi(abuscar, "FIN")!=0)
    {
        pos = buscar(nombres,abuscar,5);
        if (pos!=-1)
            printf ("\nEl nombre buscado se encuentra en el vector");
        else
            printf ("\nEl nombre buscado NO se encuentra en el vector");

        printf ("\nIngrese un nombre a buscar (FIN para terminar): ");
        gets(abuscar);
    }
    ordenar(nombres,5);
    printf("\n\nListado de nombres ordenado:\n");
    mostrar(nombres,5);
    return 0;
}
```