Manejo de cadenas de caracteres

DEFINICIÓN

Una cadena de caracteres es un conjunto de caracteres que se almacenan en un arreglo unidimensional. Los diferentes lenguajes de programación proveen una serie de funciones y procedimientos que permiten manejar textos en un programa de manera sencilla a través de las cadenas de caracteres. En el caso del lenguaje C++, una cadena de caracteres se define de igual manera cómo se define cualquier arreglo en el lenguaje, no hay diferencias. La siguiente orden: `char arr[20];` define una cadena de caracteres que puede contener 20 caracteres y sus índices van del 0 al 19.

Sin embargo, a pesar que las cadenas de caracteres son simplemente arreglos, se requieren otros elementos para poderlos manejar. A continuación, explicamos esto.

Si, por ejemplo, en el arreglo que definimos anteriormente colocamos los caracteres que formen la palabra "Ana Roncal", como se muestra a continuación:

'A'	'n'	'a'	' '	'R'	'o'	'n'	'c'	'a'	'l'										
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9

Podemos darnos cuenta que colocarlos es una tarea sencilla, pero a la hora de querer recuperarlos, por ejemplo, imprimirlos, ¿qué hacemos?, como se ve en la figura, la frase "Ana Roncal" no tiene 20 caracteres, sólo tiene 10, entonces, ¿qué hacemos para imprimir sólo 10 caracteres y no los 20 con los que fue definido el arreglo?

Alguien podría decir que, ya que estamos trabajando con arreglos, por qué no definimos una variable auxiliar para manejar su tamaño como se hace con los arreglos comunes, no obstante, esto se vuelve poco práctico sobre todo cuando hablamos de varias cadenas de caracteres, como por ejemplo cuando se quiere trabajar con un conjunto de nombres, en este caso se tendría que definir una variable auxiliar por cada nombre.

Por esta razón muchos lenguajes de programación definen reglas y modos para declarar un variable como cadena de caracteres para que su manejo sea más práctico.

En el caso del lenguaje C++ se ha encontrado una forma muy práctica y sencilla de controlar la cantidad de caracteres válidos colocados en el arreglo, se hace mediante un ***'caracter de terminación'***, esto es, se ha elegido un caracter dentro de la tabla de caracteres que no se utilice dentro de cualquier texto y se coloca en la posición del arreglo inmediata al lado del último carácter válido. Este caracter es el caracter cuyo código ASCII es cero, conocido como ***'carácter nulo'***. Entonces, en el ejemplo anterior el sistema colocará los caracteres en el arreglo de la siguiente manera:

'A'	'n'	'a'	' '	'R'	'o'	'n'	'c'	'a'	'l'	'\0'									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9

'\0' representa al caracter cuyo código ASCII es cero

Ahora entonces cuando queramos manipular esta cadena no tendremos por qué saber cuántos caracteres son válidos, solo tendremos que recorrer, desde el inicio, uno a uno los elementos del arreglo hasta encontrar este ***'carácter nulo'***. Esta forma de definir

las cadenas de caracteres permite que prácticamente no haya límites en cuanto a la cantidad de caracteres que se pueda manejar en una variable.

Sin embargo, debemos tener muy en claro que en la definición de la cadena: `char arr[20];` solo se podrán colocar 19 caracteres válidos porque siempre debemos reservar un espacio para el ***'caracter de terminación'***. Además, si no olvidamos de colocar el ***'caracter de terminación'***, no podremos manipular correctamente la cadena.

Ahora es bueno que aclaremos un detalle en cuanto al carácter de terminación. Se ha dicho anteriormente que un tipo de dato ***char*** es un tipo de dato entero y que a una variable definida como: ***char m;*** podemos asignarle tanto un valor entero (p. e.: ***m = 65;)*** como un valor de tipo carácter (p. e.: ***m = 'A';***) y que en ambos casos se le asignó el mismo valor a la variable ***m***. Pues bien, lo mismo pasa con el carácter de terminación ***'\0'***, este carácter tiene como código cero (***0***) así que para simplificar la nomenclatura a partir de ahora hablaremos del carácter de terminación como el valor cero. Es importante que entienda que ***'0'*** ≠ ***'\0'*** ya que el código ASCII de ***'0'*** es ***48*** y el código ASCII de ***'\0'*** en ***cero***.

DECLARACIÓN DE UNA CADENA DE CARACTERES

Como hemos indicado esto se hace como cualquier arreglo, esto es una cadena de caracteres se puede definir de manera estática o dinámica, como se muestra a continuación:

```
char cadenaEstatica[20];
char *cadenaDinamica;   // Pero habrá que inicializar el puntero
                        // antes de usar el arreglo.
…
cadenaDinamica = new char[20];
```

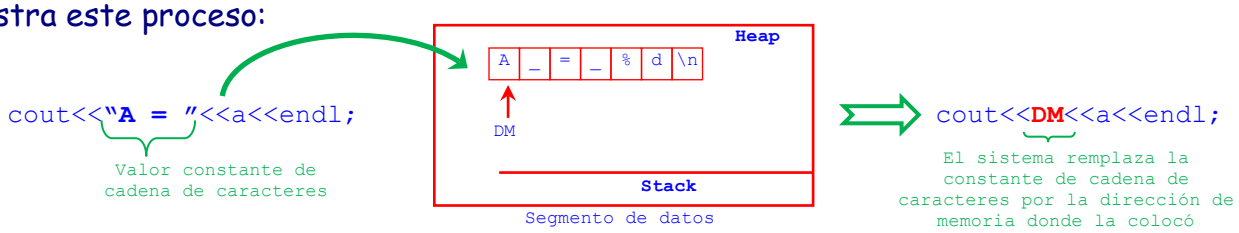
VALORES CONSTANTES EMPLEADOS COMO CADENAS DE CARACTERES

Recordemos que un valor constante es la representación de un tipo de dato, por ejemplo, en la ecuación: `area = 3.1416 * radio /2;` el valor 3.1436 es un valor constante del tipo `double` mientras que el valor 2 un valor constante del tipo `int`.

Pues bien, en un programa en C++ podemos definir valores constantes de tipo cadena de caracteres, estos se representan como una secuencia de caracteres encerrados entre comillas dobles, por ejemplo `"Hola"`, ejemplo del uso de valores constantes se dan en el uso del `cout`, cuando colocamos etiquetas en los valores como: `cout<< "A= "<<a;` donde `"A= "` es una cadena constante.

En este punto se debe tener en cuenta que cuando se coloca un valor constante como cadena de caracteres en un programa, estos no serán tratados de igual manera que los valores constantes numéricos. El compilador, cuando encuentre un valor constante como cadena de caracteres, extrae los caracteres del código del programa y los pone en el segmento de datos, en la zona del `heap`. Luego toma la dirección de memoria del

primero de estos caracteres y la coloca en el código del programa. A continuación, se muestra este proceso:



La explicación anterior tiene por finalidad que usted comprenda por qué el siguiente código está errado:

```
char nombre[20];
/*1*/ nombre = "Juan";
/*2*/ if (nombre == "Juan") {…
```

La línea 1 está mal porque la constante simbólica ***"Juan"*** será remplazada por una dirección de memoria, entonces la instrucción tratará de asignarle una dirección de memoria a la variable ***nombre*** que es un arreglo y por lo tanto un puntero que apunta a una dirección constante, por lo que no se puede modificar.

La línea 2 está mal porque el arreglo nombre apunta a un espacio de memoria que nunca será igual a la dirección de memoria en que el sistema colocó a la constante ***"Juan"***.

Sin embargo, en el siguiente código podemos ver algunas diferencias:

```
char *nombre;
/*1*/ nombre = "Juan";
/*2*/ if (nombre == "Juan") {…
```

La línea 1 es correcta, ya que como la variable ***nombre*** es un puntero variable, la dirección a la que apunta puede cambiar por lo tanto la variable ***nombre*** está recibiendo de manera correcta la dirección de la constante ***"Juan"***. Sin embargo, como ***"Juan"*** se trata de una constante no se podrá modificar, por ejemplo, si se quisiera pasar todos los caracteres a mayúsculas.

La línea 2, como en el primer ejemplo, no es correcta, ni si quiera si se ejecutó luego de la línea 1. Esto se debe a que el sistema colocará cada ***"valor constante"*** en una posición de memoria diferente, así las cadenas sean idénticas. Por esta razón la comparación de la línea 2 siempre dará falso.

INICIALIZACIÓN DE UNA CADENAS DE CARACTERES

Una de las cosas que si permite el lenguaje C++ es que si la cadena es definida como un arreglo (no como un puntero), esta inicialización solo se puede dar en el momento de declarar la variable, una vez declarada la variable la cadena de caracteres solo podrá manipularse carácter por carácter, esto es elemento por elemento como cualquier arreglo. El siguiente ejemplo muestra este aspecto:

```
char nombre[20] = "Ana"; /* Correcto*
char apellido[20];
apellido = "Roncal";     /*Incorrecto*/
```

Afortunadamente el Lenguaje C++ posee un gran número de funciones, agrupadas en diferentes bibliotecas que permiten una manipulación adecuada y sencilla.

INGRESO Y SALIDA DE CADENAS DE CARACTERES

Los objetos: ***cout***, ***cin y los objetos de archivo*** de las bibliotecas ***iostream y fstream*** estudiadas en capítulos anteriores están adaptadas para trabajar con cadenas de caracteres, la forma de emplearlas será muy similar a la forma cómo se leen o escriben los datos numéricos en un programa y el hecho que aquí si se trabajará con el arreglo completo como se ve en el ejemplo siguiente:

```
char nombre[50];
cin >> nombre; /* Observe que se va a trabajar con todo el arreglo*/
cout <<left<<setw(10)<< "Nombre ="<<setw(10)<<nombre;
```

La combinación de cin con >> permitirá leer una secuencia de caracteres y colocarlos uno a uno en el arreglo desde el primer elemento, al terminar el sistema colocará automáticamente el ***'caracter de terminación'*** en el elemento inmediato al que se colocó el último carácter leído. Por eso es muy importante que la cadena de caracteres deba tener suficiente espacio para recibir los caracteres leídos y para el ***'caracter de terminación'***. Por eso si se pretende leer una cadena como "Ana" declarar una cadena como: ***char nombre[3];*** es un grave error de concepto porque no hay sitio para el ***'caracter de terminación'***. La definición ´ correcta será: ***char nombre[4];***.

Otro detalle que se debe tener en cuenta al usar ***cin*** es que ésta no cambia el modo como trabajan con respecto al tratamiento de los valores numéricos.

Recordemos cómo trabaja n estas funciones, cuando se ejecuta una orden como:

```
int a,b,c;
cinn >> a >> b >> c;
```

Cuando se ejecuta la ***cin*** se detiene la ejecución del programa para permitir el ingreso de datos al programa. Luego que el usuario escriba los datos y presione la tecla ***ENTER cin*** empieza a explorar los caracteres colocados en el buffer de entrada. Si los primeros caracteres corresponden a espacios, tabuladores o cambios de línea (a los que denominamos ***"caracteres de separación de datos"*** o simplemente ***"separadores"***), la función los descarta, apenas se detecte un carácter diferente a estos, la función extrae el caracter, lo convierte en un número y con él empieza a armar en número. Este proceso se repite hasta que se encuentre un ***"separador"***, en ese momento la función da por terminada la exploración de del primer dato y lo asigna a la variable correspondiente, en este caso a la variable ***"a"***, continuando la lectura del siguiente dato, para lo cual se repetirá de manera exacta el proceso descrito.

Cuando se trata de leer una cadena de caracteres, el proceso se realiza de la misma forma, con la única salvedad que los caracteres no se transforman en números, sino que se colocan tal cual, en el arreglo, desde el inicio. Este proceso nos podría causar una sensación de extrañeza, cuando en la práctica intentemos leer una cadena de caracteres, veamos el siguiente caso para el código:

```
char nombre[50];
cin >> nombre;
```

Si a la hora de ingresar la cadena, escribimos lo siguiente:

```
Ana   Cecilia   Roncal   Neyra
```

La ***cin*** solo colocará en el arreglo la primera palabra (Ana), porque habrá descartado todos los ***"separadores"*** antes del carácter 'A' y terminará su tarea cuando se encuentre el primer ***"separador"*** inmediatamente después del carácter 'a'. El resto de caracteres quedará en el buffer de entrada para ser utilizados en la siguiente operación de lectura.

Usted se preguntará entonces cómo hacemos para leer todos los caracteres y que se coloquen en una sola cadena de caracteres en una única operación de lectura. Pues eso lo veremos más adelante, ya que ***cin >>*** solo podrá leer palabras aisladas, por ahora veamos las ventajas de esta forma de trabajo.

Esta función es ideal para el caso que tuviéramos que procesar las palabras de un texto que se encuentre en un archivo. En otros lenguajes de programación tendríamos que leer línea por línea el archivo y luego por cada línea, empleando diversas funciones apropiadas, separar cada una de las palabras. Esto es impensable en el lenguaje C++.

Si quisiéramos leer de un archivo datos de diverso tipo, como, por ejemplo:

```
20192030  Roncal  2389.34 Informática
```

Solo tendríamos que escribir (luego de abrir el archivo) el siguiente código:

```
char apellidoPat[50], especialidad[60];
int codigo;
double cuota,
arch >> codigo >> apellidoPat >> cuota >> especialidad;
```

Esto no se podría realizar de esta forma tan sencilla en otros lenguajes de programación.

En el caso de ***cout***, lo que harán es tomar uno a unos los caracteres del arreglo y serán colocados en el medio de salida, este proceso se realizará hasta que se encuentre el ***'carácter nulo'***. Es importante que se entienda que aquí no se trabaja por palabras, la cadena se imprimirá completamente incluyendo los ***"espacios"***.

El siguiente código muestra estas características:

```
#include <iostream>
#include <iomanip>
using namespace std;

int main(int argc, char** argv) {
    char nomb1[50], apel1[50], nomb2[50], apel2[50];
    int a1 = 23, a2 = 1245;
    double b1 = 568.3, b2 = 8.7537;

    cout.precision(2);
    cout<<fixed;

    cout<<"Ingeres el nombre de una persona:  ";
    cin >> nomb1 >> apel1;
    cout<<"Ingeres el nombre de otra persona: ";
    cin >> nomb2 >> apel2;
```

```
    cout<<endl;
    cout<<"Usando solo el formato simple:"<<endl;
    cout << a1 << nomb1 << apel1 << b1 << endl;
    cout << a2 << nomb2 << apel2 << b2 << endl;
    cout<<endl;
    cout <<"Agregando un minimo de formato:"<<endl;
    cout<<setw(15)<<nomb1<<setw(15)<<apel1<<setw(6)<<a1<<setw(10)<<b1<<endl;
    cout<<setw(15)<<nomb2<<setw(15)<<apel2<<setw(6)<<a2<<setw(10)<<b2<<endl;
    cout<<endl;
    cout<<"Alineando a la izquierda:"<<endl;
    cout<<left<<setw(15)<<nomb1<<setw(15)<<apel1<<setw(6)<<a1<<setw(10)<<b1<<endl;
    cout<<left<<setw(15)<<nomb2<<setw(15)<<apel2<<setw(6)<<a2<<setw(10)<<b2<<endl;
    cout<<endl;
    cout<<"Alineando correctamente:"<<endl;
    cout<<left<<setw(15)<<nomb1<<setw(15)<<apel1<<right<<setw(6)<<a1
        <<setw(10)<<b1<<endl;
    cout<<left<<setw(15)<<nomb2<<setw(15)<<apel2<<right<<setw(6)<<a2
        <<setw(10)<<b2<<endl;

    return 0;
}
```

Al ejecutar el programa obtendremos algo similar a lo siguiente:

```
Ingeres el nombre de una persona:  Ana Roncal
Ingeres el nombre de otra persona: Valentina Rodriguez

Usando solo el formato simple:
23AnaRoncal568.30
1245ValentinaRodriguez8.75

Agregando un minimo de formato:
            Ana         Roncal    23    568.30
      Valentina      Rodriguez  1245      8.75

Alineando a la izquierda:
Ana             Roncal          23    568.30
Valentina       Rodriguez       1245  8.75

Alineando correctamente:
Ana             Roncal             23    568.30
Valentina       Rodriguez        1245      8.75
```

Métodos *getline(…)* y *get(…) con argumentos*

Si se necesita leer una cadena completa las métodos ***getline(…)*** y ***get(…)*** con argumentos, definidas para ***cin*** o archivos del tipo ***ifstream***, nos pueden ayudar en esta tarea. Ambas trabajan de manera similar pero también tienen algunas diferencias que se deben tener en consideración.

Los métodos ***getline(…)*** y ***get(…)*** se emplean de la siguiente manera:

a) cin.getline(cadena,n); **o** ***b) cin.getline(cadena,n,delimitador)***
a) cin.getl(cadena,n); **o** ***b) cin.getl(cadena,n,delimitador)***

Ambos métodos leen los caracteres del buffer de entrada y los colocan desde el inicio del arreglo ***cadena***, incluyendo los espacios y tabuladores. El proceso se detiene cuando se encuentre el cambio de línea en el caso ***a),*** o el delimitador (que es un carácter cualquiera) en el caso ***b),*** al final ambas colocan el ***'carácter nulo'*** inmediatamente después de colocar el último carácter en la cadena, o cuando se

extraen ***n-1*** caracteres del buffer antes de encontrar el cambio de línea en el caso ***a)*** o el delimitador en el caso ***b)***.

La diferencia entre ambos es que en primer lugar ***getline(…)*** lee los caracteres del buffer de entrada y extrae y descarta el cambio de línea o el delimitador, mientras que ***get(…)*** deja en el buffer de entrad el cambio de línea o el delimitador. En segundo lugar, si el método ***getline(…)*** llega a leer ***n-1*** caracteres, levanta la bandera de error, por lo que ya no se podrá leer hasta hacer un ***clear()***, a diferencie de ***get(…)*** que no levanta la bandera.

Los siguientes programas nos pueden servir para apreciar estas similitudes y diferencias:

```
#include <iostream>
#include <iomanip>
using namespace std;
int main(int argc, char** argv) {
    char cadena1[50],cadena2[50];
    cout<<"Ingrese una linea de texto: "<<endl;
    cin.getline(cadena1, 50);
    cout<<"Cadena leida: "<<cadena1<<endl;
    cout<<endl;
    cout<<"Ingrese nuevamente la linea de texto: "<<endl;
    cin.getline(cadena1, 50,'/');
    cout<<"Cadena leida: "<<cadena1<<endl;
    cout<<endl;
    cin.getline(cadena2, 50);
    cout<<"Cadena restante: "<<cadena2<<endl;
    return 0;
}
```

Al ejecutar el programa obtendremos algo similar a lo siguiente:

```
Ingrese una linea de texto: Ana Cecilia/Roncal Neyra
Cadena leida: Ana Cecilia/Roncal Neyra

Ingrese nuevamente la linea de texto: Ana Cecilia/Roncal Neyra
Cadena leida: Ana Cecilia

Cadena restante: Roncal Neyra
```

Como se aprecia, primero se han leído los caracteres del nombre hasta extraer el cambio de línea, luego se ha leído la misma cadena, pero hasta extraer el delimitador '/'.

```
#include <iostream>
#include <iomanip>
using namespace std;
int main(int argc, char** argv) {
    char cadena1[50];
    cout<<"Ingrese una linea de texto: ";
    cin.get(cadena1, 50);
    cout<<"Cadena leida: "<<cadena1<<endl;
    cin.get(); // Al no tener argumentos lee solo un caracter: el cambio de línea
    cout<<endl;
    cout<<"Ingrese nuevamente la linea de texto: ";
    cin.get(cadena1, 50);
    cout<<"Cadena leida: "<<cadena1<<endl;
    cout<<endl;
    return 0;
}
```

Al ejecutar el programa obtendremos algo similar a lo siguiente:

```
Ingrese una linea de texto: Ana Cecilia/Roncal Neyra
Cadena leida: Ana Cecilia/Roncal Neyra

Ingrese nuevamente la linea de texto: Ana Cecilia/Roncal Neyra
Cadena leida: Ana Cecilia/Roncal Neyra
```

Como se aprecia, luego de usar ***get(…)*** con argumentos hubo usar ***get()*** sin argumentos ya que el cambio de línea se quedó en el buffer de entrada. De no haberlo hecho, la segunda vez que se usó ***get(…)*** hubiera leído una cadena en blanco ya que el método hubiera encontrado el cambio de línea por lo que allí se hubiera terminado la lectura.

```
#include <iostream>
#include <iomanip>
using namespace std;

int main(int argc, char** argv) {
    char cadena1[50];
    cout<<"Ingrese una linea de texto: ";
    cin.get(cadena1, 50,'/');
    cout<<"Cadena leida: "<<cadena1<<endl;
    cout<<endl;
    cin.get(cadena1, 50);
    cout<<"Cadena leida: "<<cadena1<<endl;
    cout<<endl;

    return 0;
}
```

Al ejecutar el programa obtendremos algo similar a lo siguiente:

```
Ingrese una linea de texto: Ana Cecilia/Roncal Neyra
Cadena leida: Ana Cecil

Cadena leida: ia/Roncal Neyra
```

Como se aprecia, luego de usar ***get(…)*** con el delimitador '/', el método deja el delimitador en el buffer de entrada, por eso que en la segunda lectura se lee el delimitador junto con los restantes caracteres.

```
#include <iostream>
#include <iomanip>
using namespace std;

int main(int argc, char** argv) {
    char cadena1[10],cadena2[50];
    cout<<"Ingrese una linea de texto: ";
    cin.getline(cadena1, 10);
    cout<<"Cadena leida: "<<cadena1<<endl;
    cout<<endl;

    cin.clear();

    cin.getline(cadena2, 50);
    cout<<"Cadena leida: "<<cadena2<<endl;

    return 0;
}
```

Al ejecutar el programa obtendremos algo similar a lo siguiente:

```
Ingrese una linea de texto: Ana Cecilia/Roncal Neyra
Cadena leida: Ana Cecilia

Cadena leida: /Roncal Neyra
```

Como se aprecia, como la cadena de entrada sobrepasa el valor ***n-1*** (10) de caracteres luego de usar ***getline(…)*** se levanta la bandera de error, por lo que si no se usa el método ***clear()*** no se podrá seguir leyendo.

```
#include <iostream>
#include <iomanip>
using namespace std;

int main(int argc, char** argv) {
    char cadena1[10],cadena2[50];
    cout<<"Ingrese una linea de texto: ";
    cin.get(cadena1, 10);
    cout<<"Cadena leida: "<<cadena1<<endl;
    cout<<endl;

    cin.getline(cadena2, 50);
    cout<<"Cadena leida: "<<cadena2<<endl;

    return 0;
}
```

Al ejecutar el programa obtendremos algo similar a lo siguiente:

```
Ingrese una linea de texto: Ana Cecilia/Roncal Neyra
Cadena leida: Ana Cecilia

Cadena leida: /Roncal Neyra
```

Como se aprecia, aquí no se requiere usar ***clear()***.

FUNCIONES QUE MANEJAN CADENAS:

El lenguaje C++ proporciona un conjunto de funciones que permiten el manejo de caracteres, estas funciones se encuentran definidas e implementadas en la biblioteca ***cstring***, en esta sección estudiaremos el uso de muchas de estas funciones, sin embargo algo muy importante para un desarrollador de software es entender cómo realizan sus operaciones estas funciones y no conformarse simplemente con utilizar estas ***'cajas negras'***, esto le ayudará a no cometer errores en la programación que le pueden hacer perder mucho tiempo. Por esto, a continuación, desarrollaremos una serie de funciones que permitirán manejar las cadenas de caracteres y, en el caso que estás tengan una similar en la biblioteca ***cstring*** se lo haremos notar.

A continuación, mostraremos una lista de funciones para el procesamiento de textos en un programa.

Función para determinar el número de caracteres válidos en una cadena:

Esta función tiene su equivalente en ***csting*** en la función ***strlen***.

```
int strlen(const char* cadena){
    int nCar;
    for (nCar=0; cadena[nCar]; nCar++);
    return nCar;
}
```

Lo que hemos hacho es simplemente recorrer la cadena contando uno a uno los caracteres y terminamos cuando se encuentre el ***'carácter nulo'***, observe como hacemos está detección, recuerde que el ***'carácter nulo'*** vale cero y cero significa falso, luego cuando ***cadena[i]*** corresponda al ***'carácter nulo'*** el ***for*** termina. Este carácter no se considera en la cuenta de caracteres.

Es importante que entienda este proceso y reflexione sobre su uso, entienda que cada vez que llame a esta función se recorrerá toda la cadena. Por eso si por ejemplo en un programa tenemos que trabajar con todos los caracteres de una cadena pasándolos a mayúsculas, verá lo altamente ineficiente que resulta la siguiente función:

```
void pasaAMayusculasMALA(char* cadena){
    for(int i=0; i<strlen(cadena); i++)
        cadena[i] -= ((cadena[i]>='a'&&cadena[i]<='z') ? 'a'-'A' : 0);
}
```

Está función está muy mal implementada porque por cada carácter de la cadena el ***for*** llamará a la función ***strlen*** y esta recorrerá toda la cadena, y se puede apreciar que, si la cadena tuviera 10 caracteres, por cada caracter que se analiza para convertirlo en mayúsculas se recorren 10 veces la cadena para contar el número de caracteres, en total habremos procesado 100 caracteres (n^2), por eso es altamente ineficiente y nunca se debería emplear la función ***strlen*** así.

La siguiente implementación si es eficiente porque solo se recorre la cadena una vez.

```
void pasaAMayusculas(char* cadena){
    for(int i=0; cadena[i]; i++)
        cadena[i] -= ((cadena[i]>='a'&&cadena[i]<='z') ? 'a'-'A' : 0);
}
```

Nuevamente aquí hemos aprovechado la propiedad del ***'carácter nulo'*** para salir del ***for***.

Función para corregir el problema de leer datos mixtos (números y cadenas):

Existe un problema cuando, luego de haber leído un número, se quiere leer una cadena de caracteres. Este problema se presenta cuando se emplea tanto el método ***getline(…)*** como con ***get(…)***, pero, por su forma de trabajar, no pasa cuando se usa ***cin >>***. Observe el siguiente código:

```
#include <iostream>
#include <iomanip>
using namespace std;

int main(int argc, char** argv) {
    int numero;
    char cadena[200];

    cout << "Ingrese un numero: ";
    cin >> numero;
    cout << "Ingrese una oracion: ";
    cin.getline(cadena,200);
    cout <<left<<setw(10)<<"Numero ="<<right<<setw(8)<<numero<<endl;
    cout <<left<<setw(10)<<"Cadena ="<<setw(10)<<cadena<<endl;

    return 0;
}
```

Al ejecutar el programa observamos lo siguiente:

```
Ingrese un numero: 30547032
Ingrese una oracion: Numero = 30547032
Cadena =
```

Luego de ingresar el número, el programa continúa sin dejarnos ingresar la cadena de caracteres y, como se ve, no se imprime una cadena. La razón para esto es que, por su forma de trabajar, ***cin*** luego de leer un dato, en este caso el número, deja los ***'separadores'*** que le siguen en el buffer, en este caso lo que se queda en el buffer es el carácter de cambio de línea (***'\n'***). Luego, cuando se quiere leer la cadena, ***getline(…)*** verá que el buffer no está vacío, intentará leer los caracteres y se encontrará con el cambio de línea, lo sacará del buffer de entrada y terminará el proceso de lectura, el método finalmente colocará el ***cero*** en la cadena, resultando una cadena ***'vacía'***. De allí en resultado que se aprecia. Este problema nunca sucederá si leemos la cadena con ***cin>>***, pero hay que recordar que solo se podrá leer una palabra.

Por esta razón deberíamos implementar una función que limpie el buffer de entrada, de modo que el método ***getline(…)*** o ***get(…)*** encuentre el buffer vacío y se vea obligado a detener la ejecución del programa para que el usuario ingrese la cadena.

A continuación, veremos el código para realizar la limpieza del buffer:

```
while(cin.get()!='\n'); o while(arch.get()!='\n');
```

La función leerá uno a uno los caracteres del buffer de entrada y se detendrá luego de leer el cambio de línea.

Finalmente, en código siguiente:

```
#include <iostream>
#include <iomanip>
using namespace std;

int main(int argc, char** argv) {
    int numero;
    char cadena[200];

    cout << "Ingrese un numero: ";
    cin >> numero;
    while(cin.get()!='\n');
    cout << "Ingrese una oracion: ";
    cin.getline(cadena,200);
    cout <<left<<setw(10)<<"Numero ="<<right<<setw(8)<<numero<<endl;
    cout <<left<<setw(10)<<"Cadena ="<<setw(10)<<cadena<<endl;

    return 0;
}
```

Se ejecutará correctamente de la manera siguiente:

```
Ingrese un numero: 305470764
Ingrese una oracion: Este programa si funciona
Numero = 305470764
Cadena = Este programa si funciona
```

Función para copiar una cadena en otra:

Lo que se busca es poder asignar cadenas de caracteres a las variable, como las cadenas son arreglos, ya lo explicamos, no podemos emplear el operador =. Es muy importante que entienda esto y que la única forma que asignar una cadena a otra es trabajando carácter por carácter.

Esta función tiene su equivalente en ***csting*** en la función ***strcpy***.

```
void strcpy(char* destino, const char* fuente){
    int i=0;
    while(1){
        destino[i] = fuente[i];
        if(fuente[i]==0) break;
        i++;
    }
}
```

Observe que el ***'carácter de terminación'*** también se está pasando a la cadena ***'destino'***.

Luego la forma de utilizar esta función sería la siguiente:

```
#include <cstring>
…
char cad1[30], cad2[30];
strcpy(cad1,"Ana Roncal");
strcpy(cad2,cad1);
```

Función para comparar dos cadenas:

En el procesamiento de cadenas de caracteres, una tarea que se nos presentará con frecuencia es el hecho de tener que comparar dos cadenas, por ejemplo, si tenemos un conjunto de nombres de personas y queremos ordenarlos. De igual manera que la función ***strcpy***, para realizar esta tarea no podemos utilizar los operadores de relación (==, >, <, etc.), por esto requerimos de una función que realice esta tarea. Afortunadamente en la biblioteca ***csting*** se define la función ***strcmp***, que permite realizar esta operación. El diseño de esta función hace que no se requiera implementar una función por cada operador debido a que la función devolverá un valor entero, si el valor devuelto es cero querrá decir que las dos cadenas que se están comparando son iguales, si este valor es positivo indicará que la primera cadena es mayor que la otra, hablando alfabéticamente, y si es negativo, la primera será menor que la otra.

A continuación, presentamos el código equivalente a la función ***strcmp***:

```
int strcmp(const char* cad1,const char* cad2){
    int i=0;
    while(cad1[i] == cad2[i]){
        if(cad1[i] == 0) return 0;
        i++;
    }
    return cad1[i] - cad2[i];
}
```

Como se aprecia, la función va recorriendo simultáneamente uno a uno los caracteres de ambas cadenas. Mientras los caracteres sean iguales no se podrá tomar una decisión sobre la relación de una cadena sobre la otra por lo que se sigue avanzando.

Si durante este proceso se determina que ya se evaluaron todos los caracteres (`cad1[i] == 0`) entonces se puede decir que ambas cadenas son iguales, por lo que se devuelve cero.

Si en el proceso se detecta un par de caracteres diferentes entonces el proceso se detiene y en la operación que se realiza a continuación se aprovecha la ubicación de los caracteres en la tabla ASCII, por lo que restando ambos caracteres se podrá determinar la relación que hay entre ambos y por lo tanto entre ambas cadenas, recuerde que lo que se guarda en una variable no es un carácter sino un valor entero que corresponde al código ASCII.

El siguiente programa muestra cómo podemos emplear esta función:

```
#include <iostream>
#include <iomanip>
using namespace std;
#include <cstring>

int main(int argc, char** argv) {
    char nombre1[30], nombre2[30];
    int cmp;
    while(1){
        cout<<"Ingrese un nombre (FIN para terminar): ";
        cin.getline(nombre1,30);
        if (strcmp(nombre1, "FIN") == 0) break;
        cout<<"Ingrese otro nombre: ";
        cin.getline(nombre2,30);
        cmp = strcmp(nombre1, nombre2);
        if (cmp == 0)
            cout<<nombre1<<" es igual a "<<nombre2<<endl;
        else if (cmp > 0)
            cout<<nombre1<<" es mayor que "<<nombre2<<endl;
        else if (cmp < 0)
            cout<<nombre1<<" es menor que "<<nombre2<<endl;
        cout<<endl;
    }

    return 0;
}
```

La ejecución de este programa se muestra a continuación:

```
Ingrese un nombre (FIN para terminar): Naomi
Ingrese otro nombre: Naomi
Naomi es igual a Naomi

Ingrese un nombre (FIN para terminar): Valentina
Ingrese otro nombre: Naomi
Valentina es mayor que Naomi

Ingrese un nombre (FIN para terminar): Ana
Ingrese otro nombre: Naomi
Ana es menor que Naomi

Ingrese un nombre (FIN para terminar): FIN
```

En el programa se aprecia que en la primera comparación (`if (strcmp(nombre1, "FIN")…`) la función se emplea directamente, mientras que en la segunda se hace de manera indirecta (`cmp = strcmp(nombre1, nombre2);`), la razón es que se quiere realizar este proceso de manera eficiente. En el primer caso solo se requiere una comparación, en este caso para ver si son iguales, pero en el segundo se requiere de tres

comparaciones (`==. > y <`) y en los tres casos se debe realizar el mismo proceso, entonces para no tener que repetirlo lo hacemos solo una vez, colocando el resultado en una variable y usándola en las tres decisiones, de este modo el proceso se hace más eficiente.

Función para concatenar dos cadenas:

La idea de esta función es la de agregar a una cadena el contenido de otra, por ejemplo, si una cadena contiene ***"Ana"*** y otra contiene ***"stasia"*** podamos formar la cadena ***"Anastasia"***. En la biblioteca ***csting*** se define la función ***strcat***, que permite realizar esta operación.

A continuación, presentamos el código equivalente a la función ***strcat***:

```
#include <cstring>
…
void strcat(char* cadena1, const char* cadena2){
    int pos;
    pos = strlen(cadena1);
    strcpy(&cadena1[pos],cadena2);
}
```

Observe que lo único que se pretende hacer es copiar la cadena 2 en la cadena 1 pero a partir del último carácter de la cadena 1 por eso se coloca en el llamado a ***strcpy*** la dirección en la cadena 1 de la posición a partir de la cual se quiere colocar los caracteres de la cadena 2.

Debe tener en cuenta también que es importante que la cadena 1 tenga una cadena válida (inicializada) de lo contrario la función ***strlen*** dará un resultado incoherente y por lo tanto la cadena resultado también quedará con valores incoherentes.

Función para localizar una cadena dentro de otra:

De lo que se trata aquí es de poder determinar si una secuencia de caracteres se encuentra dentro de otra, por ejemplo, verificar si la cadena ***"mente"*** se encuentra en la cadena ***"Es inmensamente grande"***. En la biblioteca ***csting*** se define la función ***strstr***, que permite realizar esta operación. Esta función tratará de localizar una cadena dentro de otra, si la encuentra devuelve la dirección (un puntero) del elemento en el arreglo del primer carácter a partir del cual se encuentre la cadena buscada. Si no lo encuentra devuelve una dirección nula ***(nullptr)***.

A continuación, presentamos el código equivalente a la función ***strstr***:

```
char* strstr(char* cad1, const char* cad2){
    for(int i=0; cad1[i]; i++){
        if(cad1[i] == cad2[0])
            if(corresponde(&cad1[i],cad2))
                return &cad1[i]; /* dirección del elemento de inicio */
    }
    return nullptr;
}

bool corresponde(const char *cad1,const char *cad2){
    for(int i=0; cad2[i]; i++)
        if(cad1[i] != cad2[i])return false; //son diferentes
    return true;
}
```

Para hacer más claro el código se ha decidido desarrollarla en dos funciones y así se pueda entender mejor el algoritmo. Primero se trabaja solo con el primer carácter de la segunda cadena, cuando se encuentre una coincidencia se llama a la función ***'corresponde'*** que se encargará de verificar si allí se encuentra la cadena.

Es importante que entienda que esta forma de implementar la función no dará errores, por lo que, si usted desea implementarla de otra manera, compruebe que si por ejemplo se llama a la función de la manera siguiente:

```
ptr = strstr("La mamadera del bebe", madera");
```

No dé como resultado ***nullptr***, ya que la cadena ***"madera"*** si se encuentre en la cadena1.

Función para funciones para convertir una cadena de caracteres en un número:
Muchas veces nos encontraremos en la situación en que en una cadena de caracteres se encuentra representado un valor numérico como ***"2387"***, este valor no se puede operar como un número, por ejemplo:

```
int num;
char valor[50] = "5381";

num = 571 + "2387";
num = 739 + valor;
```

En ambas operaciones se está cometiendo un grave error de concepto.

Afortunadamente contamos con un par de funciones que nos pueden ayudar en problemas como estos. Se tratan de las funciones ***atoi*** y ***atof*** definidas, a diferencia de las otras, en la biblioteca ***cstdlib*** (no en ***cstring***). La primera función su utilizará cuando la cadena tenga la representación de un valor entero y la segunda cuando se trate de un valor de punto flotante.

A continuación, presentamos el código equivalente a ambas funciones:

```
#include <cctype >

int atoi(const char*cadena){
    int valor = 0, i=0, digito;

    while(isspace(cadena[i]))i++;

    while(isdigit(cadena[i])){
        digito = cadena[i] - '0';
        valor *= 10;
        valor += digito;
        i++;
    }
    return valor;
}

double atof(const char*cadena){
    double valor = 0, factor = 0.1;
    int i=0, digito;

    while(isspace(cadena[i]))i++;

    while(isdigit(cadena[i])){ /* parte entera */
        digito = cadena[i] - '0';
        valor *= 10;
        valor += digito;
        i++;
    }
```

```
    if(cadena[i] == '.'){ /* parte decimal */
        i++;
        while(isdigit(cadena[i])){
            digito = cadena[i] - '0';
            valor += digito * factor;
            factor *= 0.1;
            i++;
        }
}
```

El trabajo que hacen estas dos funciones es tomar cada uno de los caracteres de la cadena, eliminando los espacios al inicio y luego cada carácter es convertido a su valor numérico (`cadena[i] - '0'`) y con ese valor se construyendo el valor numérico.

Fíjese que se han empleado dos funciones especiales: ***isspace*** e ***isdigit***, estas funciones están definidas en la biblioteca ***cctype*** y son muy útiles cuando se trate de comparar caracteres, ahorrándonos mucho tiempo. En el caso de la función ***isspace*** devolverá 1 (verdadero) si el carácter analizado es un ***'separador'*** esto es, si es un espacio, un tabulador o un cambio de línea, y devolverá cero (falso) en caso contrario. La función devolverá 1, si el carácter analizado es un dígito decimal ('0', '1' , '2' , '3' , '4' , '5' , '6' , '7' , '8' o '9'). Sería bueno que le dé una mirada a esta biblioteca de modo que en el futuro pueda emplear otras funciones similares como ***isalpha***, ***islower***, ***isipper***, ***ispunct***, etc.

Por otro lado, la operación inversa, esto es pasar de número a cadena de caracteres, no está soportada en lenguaje C, salvo por algunos intentos, no estándar, que algunos compiladores tienen, los cuales no se recomienda su uso.

Otras funciones interesantes.

La biblioteca ***cstring*** cuenta con un gran número de funciones, por eso sería bueno que revise por su cuente el funcionamiento de algunas de ellas, les recomendamos que se fije en: ***strncpy***, ***strncat***, ***strchr***, ***strrchr***.

APLICACIONES QUE EMPLEAN CADENAS DE CARACTERES:

Veamos ahora algunas aplicaciones que permitan procesar textos.

Invertir una cadena de caracteres:

La siguiente función transformará una cadena de caracteres como "ABCDEFG" en "GFEDCBA".

```
void invertirCadena(char *a){
    char aux[200];
    int lon;

    lon = longCad(a);
    for(int i=0, j=lon-1; a[i]; i++,j--)
        aux[i] = a[j];

    aux[lon] = 0;
    strcpy(a,aux);
}
```

Observe que el proceso se hace caracteres por carácter como en las funciones que presentamos anteriormente, es la única forma de hacerlo.

Buscar una cadena en un texto y reemplazarla por otra:

La idea de este ejemplo es que por ejemplo se tiene una cadena como **"ABCDEFGHIJK"** y se quiere reemplazar la cadena **"EFGH"** por la cadena como **"efgh"** de modo que la cadena original quede de la siguiente manera: **"ABCDefghIJK"**. Pero además introduciendo un mayor grado de dificultad, permitiendo que la cadena de reemplazo no tenga relación con el tamaño de la cadena que será reemplazada, esto es que una cadena como **"ABCDEFGHIJK"** pueda quedar como **"ABCDIJK"**, **"ABCDxyIJK"** o, **"ABCDrstuvwxyzIJK"**.

```
#include <cstring>
…
void reemplazar(char *texto,const char *cadBus,
     const char *cadReem){
     char aux[500], *pt;
     int longitud;

     /* Primero ubicamos la cadBus en el texto*/
     pt = strstr(texto, cadBus);
     if(pt == NULL) return; /* Si no se encuentra, salimos */

     *pt = 0; // Luego colocamos en esa posición el 'carácter nulo' para
              // facilitar el trabajo
     strcpy(aux, texto); // Copiamoa solo la parte izquierda del texto
                         // que no se modificará */
     strcat(aux, cadReem); // Agregamos la cadena de reeemplazo

     longitud = strlen(cadBus);
     strcat(aux, &pt[longitud]); // Agregamos la parte derecah del texto
     strcpy(texto, aux);
}
```

A continuación, mostramos gráficamente cómo se realiza esta operación:

texto:	'A'	'n'	'a'	' '	'D'	'i'	'a'	'z'	' '	'N'	'e'	'y'	'r'	'a'	0					

cadBus:	'D'	'i'	'a'	'z'	0															

cadReem:	'R'	'o'	'n'	'c'	'a'	'l'	0													

Con la instrucción: `pt = strstr(texto, cadBus);` se obtiene:

texto:	'A'	'n'	'a'	' '	'D'	'i'	'a'	'z'	' '	'N'	'e'	'y'	'r'	'a'	0					

↑ pt

La instrucción: `*pt = 0;` deja la cadena ***'texto'*** de la siguiente manera:

texto:	'A'	'n'	'a'	' '	0	'i'	'a'	'z'	' '	'N'	'e'	'y'	'r'	'a'	0					

↑ pt

De esta manera cualquier operación sobre la cadena ***'texto'*** se realizará hasta el cero. Luego, la instrucción `strcpy(aux, texto);` copiará en la cadena ***'aux'*** lo siguiente:

aux:	'A'	'n'	'a'	' '	0															

A continuación, la instrucción `strcat(aux, cadReem);` agregará el contenido de la cadena ***'cadReem'***, quedando de la siguiente manera:

aux:	'A'	'n'	'a'	' '	'R'	'o'	'n'	'c'	'a'	'l'	0									

Luego la variable ***'longitud'*** tomará el valor de 4 ya que ***'Diaz'*** tiene 4 caracteres. Finalmente aprovechando las propiedades de los punteros se tiene que:

↓ pt

texto:	'A'	'n'	'a'	' '	'D'	'i'	'a'	'z'	' '	'N'	'e'	'y'	'r'	'a'	0					
					pt[0]	pt[1]	pt[2]	pt[3]	pt[4]	pt[5]	pt[6]	pt[7]	…							

Por esta razón, la orden `strcat(aux, &pt[longitud]);` copiará los caracteres desde el carácter que se encuentra en la posición ***p[4]*** hasta el final del texto, por lo que la cadena ***'aux'*** quedará como:

aux:	'A'	'n'	'a'	' '	'R'	'o'	'n'	'c'	'a'	'l'	' '	'N'	'e'	'y'	'r'	'a'	0			

MANEJO DINÁMICO DE LAS CADENAS DE CARACTERES:

Una cadena de caracteres también se puede manipular dinámicamente a través de un puntero ***(char *cadena)*** pero antes de poder manipularla debemos garantizad que, como cualquier puntero, debemos asegurarnos que el puntero esté apuntando a una dirección válida de la memoria donde podamos colocar allí los caracteres.

Las siguientes instrucciones:

```
#include <cstring>
…
char *cadena;
strcpy(cadena, "Hola amigos"); /* ERROR */
```

Constituirá un grave error de concepto ya que la variable ***'cadena'*** no está apuntando a un lugar en la memoria válido donde colocar los caracteres de la cadena ***"Hola amigos"***.

Para que este código sea válido deberemos hacer lo siguiente:

```
#include <cstring>
…
char *cadena;
cadena = new char[100];
strcpy(cadena, "Hola amigos");
```

Fíjese que no basta con separar un espacio, sino que ese espacio debe ser lo suficientemente grande como para poder contener la cadena que se le asignará.

Así como en este caso, cualquier operación o función que emplee un puntero a una cadena de caracteres quedará invalidada si antes no se le ha separado el espacio de memoria adecuado al puntero. Es importante que entienda que si usted obvia esto el programa probablemente continuará trabajando, pero llegará a un punto donde el programa no podrá seguir y se interrumpirá y si intenta depurar el programa verá que el programa se interrumpe en una instrucción que no tiene que ver con la cadena, esto es desconcertante, pero tiene una lógica que escapa a los alcances de este texto. Lo que pasará es que perderá muchísimo tiempo tratando de corregir este tipo de error ya que, en estos casos, no se esperará que el error se deba a que no asignó el espacio de memoria requerido para manejar el puntero.

Función para asignar un espacio de memoria exacto para contener una cadena:

La eficiencia de un programa no solo se mide en el algoritmo que se emplee para solucionar un problema, la eficiencia también se mide en cuanto a los recursos que utilicemos, esto es a los espacios de memoria que utilicemos y desperdiciemos en un programa. Cuando se trabaja con cadenas de caracteres es muy frecuente definir muchos arreglos en los cuales no son empleados la mayoría de sus elementos, por lo

que el desperdicio de espacios es muy grande. Por esta razón analizaremos la forma cómo podemos hacer eficiente este trabajo.

La pregunta es ¿Cómo podemos asignar a una cadena de caracteres (o a un arreglo en general) un espacio de memoria que pueda contener exactamente los caracteres (o datos en general) que necesitamos sin desperdiciar espacios? A esto se le puede agregar otra pregunta, ¿Si antes de utilizar una cadena de caracteres (o un arreglo en general) necesitamos haber definido el espacio de memoria, ¿cómo podemos predecir el tamaño de la cadena antes de haberla leído?

Para solucionar este problema, debemos recordar algunos conceptos. En primer lugar, debemos entender que una variable es colocada en la pila del sistema en el momento de su definición en la función que la va a utilizar, luego cuando la función termine esa variable es desapilada (destruida) y el espacio de memoria que la albergaba es reutilizado por el sistema en otras variables. Por otro lado, un espacio de memoria dinámico trasciende a la función que lo definió y solo se le puede reutilizar luego que se libere el espacio por medio del operador ***'delete'***.

Con estos conceptos podemos plantear una respuesta a la segunda pregunta: no se puede predecir el espacio que requerirá el usuario, sin embargo, se puede estimar, en un caso extremo, qué tan grande podría ser ese espacio. Por ejemplo, si lo que se quiere guardar en una cadena es el nombre de pila de una persona, dudo mucho que se requieran más de 20 caracteres para contenerlo, si por el contrario se tratara de almacenar el nombre completo de una persona (nombres y apellidos), 100 caracteres podrían ser suficientes. Entonces, qué pasa si dentro de una función definimos un arreglo estático con el tamaño máximo estimado, allí colocamos los caracteres de la cadena y luego definimos un espacio de memoria dinámico con la cantidad de caracteres que están almacenados en el arreglo y allí copiamos la cadena, cuando la función termine el arreglo desaparecerá, pero el espacio dinámico se mantendrá.

A continuación, presentamos el código de una función que, siguiendo las recomendaciones anteriores solucionará la primera pregunta, entregándonos una cadena como un arreglo dinámico de caracteres con un espacio asignado exacto a la cantidad de caracteres que el usuario ingresó, el código de la función es el siguiente:

```
#include <fstream>
Using namespace std;
#include <cstring>
…
char *leeCadenaExacta(ifstream &arch){
    char buff[500],*cad;
    int longitud;

    arch.getline(buff,500);
    if(arch.eof()) return nullptr;

    longitud = strlen(buff);
    cad = new char[longitud + 1];
    strcpy(cad,buff);

    return cad;
}
```

En la función podemos observar: primero que se definen dos cadenas de caracteres, una dinámica (***cad**) y una estática (**buff**), a esta última se le ha dado un espacio de 500 caracteres, este tamaño depende de lo que queramos leer, pero si no estamos seguros, poner un número grande como ese no está mal ya que la variable solo existirá por un lapso muy pequeño de tiempo. En segundo lugar, la lectura se hace sobre el arreglo estático. Una vez que se verifique que se pudo hacer la lectura se determina la cantidad de caracteres que contiene el texto leído y con este valor se separa el espacio de memoria. Note que el espacio de memoria asignado es **longitud + 1**, esto porque debemos separar un espacio adicional para el carácter de terminación (0), de no hacerlo estaríamos cometiendo un grave error de concepto y el programa se caerá en cualquier momento.

La función retornará la dirección de memoria de la cadena dinámica que contiene los caracteres leídos sin desperdiciar los espacios.

ALMACENAMIENTO DE UN CONJUNTO DE CADENAS DE CARACTERES

Ahora veremos cómo podemos almacenar un conjunto de cadenas de caracteres en una única estructura, como podrían ser los nombres completo de todos los alumnos de un curso.

Para esto debemos introducir un nuevo concepto: ***"arreglo de punteros"***. Esto quiere decir que estamos hablando de un arreglo que, en lugar de almacenar números enteros, números de punto flotante o caracteres almacene direcciones de memoria. Como en este capítulo se está tratando el tema de "Cadenas de caracteres", no limitaremos a trabajar con lo que denominaremos ***"arreglos de cadenas de caracteres"***, entienda que se trata, por lo tanto, de un arreglo en el que cada elemento almacenará la dirección de memoria donde se encuentre una ***'cadena de caracteres dinámica'***. Los otros tipos de arreglos se revisarán en los siguientes capítulos.

Entonces un arreglo de cadenas de caracteres se definirá de la siguiente manera:

```
char* arreglo[20];
```

En donde cada elemento será un puntero del tipo `char*`. Un ejemplo de la forma cómo se puede utilizar este arreglo se muestra a continuación:

```
#include <cstring>
…
char* nombre[20];
…
nombre[0] = new char[30];
strcpy(nombre[0],"Ana Roncal");
…
nombre[2] = leeCadenaExacta(arch);
if(strcmp(nombre[0], nombre[2]) == 0)
    printf("Las cadenas son iguales\n");
…
```

Como puede apreciar su manejo es similar al de cualquier arreglo, salvo que como los elementos son punteros, se deberá reservar los espacios de memoria previamente a su uso.

El siguiente programa le permite leer los nombres de personas de un archivo de textos, colocarlos en un arreglo de cadenas de caracteres, luego se procederá a transformar cada nombre a mayúscula, ordenarlo y luego emitir un reporte con los nombres ordenados. El archivo será similar al siguiente:

```
Acuna Ayllon Blanca Luz
Urbina Antezano Valentina Paula
Gonzales Aguirre Emilio
Mori Suarez Edilberto Jesus
Arca Aquino Ana
Vargas Guevara Humberto Alberto Luis
Preciado Pinto Surami Heli
Jauregui Bravo Luz Sofia
Cabello Lam Alfonso Waldemar
Garcia Astete Victor Edgar
Mercado Caro Ofelia Carmen
…
```

Mostramos a continuación el programa que dará solución al programa que solucione este problema.

```
#include <iostream>
#include <iomanip>
using namespace std;
#include "FuncDeCadenas.h"

int main(int argc, char** argv) {
    char* persona[200]; // Arreglo de cadenas
    int numPer;
    leeNombres(persona,numPer,"Personas.txt");
    pasaAMayusculasTodo(persona,numPer);
    ordenaNombres(persona,numPer);
    imprimeNombres(persona,numPer,"reporte-Nombres.txt");
    // Si el programa continua y queremos eliminar los espacios que ya no usaremos:
    for(int i=0; i< numPer;i++)
        delete persona[i];

    return 0;
}
```

El archivo que contiene los encabezados (***FuncDeCadenas.h***) se presenta a continuación:

```
#ifndef FUNCDECADENAS_H
#define FUNCDECADENAS_H

    void leeNombres(char**,int &, const char*);
    void imprimeNombres(char**, int , const char*);
    void pasaAMayusculasTodo(char**,int);
    void pasaAMayusculas (char *);
    void ordenaNombres(char**,int );
    void cambiar(char*&, char *&);
    char *leeCadenaExacta(ifstream &arch);

#endif /* FUNCDECADENAS_H */
```

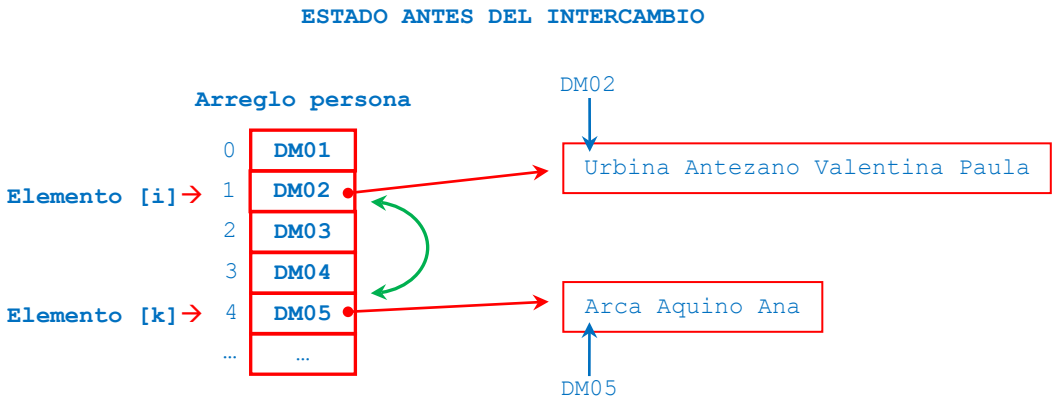
El archivo que contiene la implementación (***FuncDeCadenas.cpp***) se presenta a continuación:

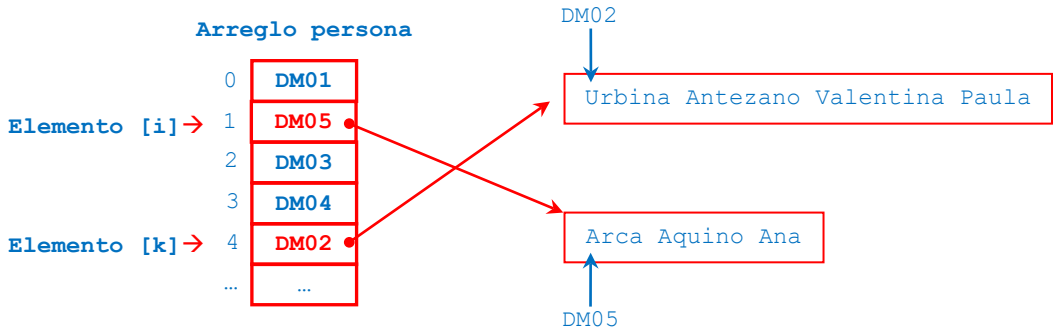
```
#include <iostream>
#include <fstream>
#include <iomanip>
using namespace std;
#include <cstring>
#include "FuncDeCadenas.h"
```

```
void leeNombres(char **persona, int &numNombs, const char *nombArch){
    ifstream arch(nombArch, ios::in);
    if(not arch.is_open()){
        cout<<"ERROR: No se pudo abrir el archivo "<<nombArch<<endl;
        exit(1);
    }
    char *nombre;
    numNombs = 0;
    while (1){
        nombre = leeCadenaExacta(arch);
        if(nombre == nullptr)break;
        persona[numNombs] = nombre;
        numNombs++;
    }
}

void imprimeNombres(char**persona, int numPer, const char*nombArch){
    ofstream arch(nombArch, ios::out);
    if(not arch.is_open()){
        cout<<"ERROR: No se pudo abrir el archivo "<<nombArch<<endl;
        exit(1);
    }
    for(int i=0; i< numPer;i++)
        arch <<left<<setw(45)<<persona[i]<<endl;
}

void pasaAMayusculasTodo(char**persona,int numPer){
    for(int i=0; i< numPer;i++)
        pasaAMayusculas(persona[i]);
}

void pasaAMayusculas (char *nombre){
    for(int i=0; nombre[i]; i++)
        nombre[i] -= ((nombre[i]>='a' && nombre[i]<='z') ? 'a'-'A': 0);
}

void ordenaNombres(char **persona,int numPer){
    for(int i=0; i< numPer -1;i++){
        for(int k=i+1; k<numPer; k++){
            if(strcmp(persona[i],persona[k])>0)
                cambiar(persona[i],persona[k]);
        }
    }
}

void cambiar(char *&personaI, char * &personaK){
    char*aux;
    aux = personaI;
    personaI = personaK;
    personaK = aux;
}
```

En el programa hemos tratado de manejar el arreglo de cadenas de caracteres como lo haríamos con cualquier otro tipo de arreglo (salvo excepciones propias del manejo de una cadena), por eso podrá apreciar la simplicidad del código, incluso en la ordenación de los datos. En cuanto al código de ordenación, también puede apreciar la eficiencia del proceso, esto porque la función 'cambiar' al intercambiar los datos no lo está realizando a nivel de cada carácter, sino que se están intercambiando las direcciones de memoria contenidas en cada elemento del arreglo. La figura siguiente muestra este proceso:

}





ARCHIVOS CSV (Comma-Separated Values)

Los archivos CSV (Comma-Separated Values) son generados por muchas aplicaciones comerciales como por ejemplo el MS Excel, se trata de archivos de texto que tiene por particularidad que los datos están separados por comas, por ejemplo: "73564207,Aldo Cabral,33,Profesor contratado,2361.50". Esta es una cadena típica que se puede encontrar en un archivo CSV, allí podemos observar que la cadena guarda el DNI, nombre, edad, cargo y sueldo de una persona. El lenguaje C++ proporciona herramientas adecuadas para poder leer estos archivos.

A continuación veamos un ejercicios que nos puedan ver leer y manipular un archivo CSV.

Ejercicio:

Se cuenta con un archivo CSV, similar al siguiente:

```
378708,SAENZ ARANDA WILMER,23455.6
140158,CALIXTO TORRES MIGUEL,12363.223
349213,ZARATE URBINA DELIA,2353.90
...
```

Aquí vemos que en cada línea tenemos el DNI, nombre y sueldo de un empleado. Lo que vamos a hacer con este archivo es separar los datos según el tipo, colocarlos en arreglos y luego ordenarlos de mayor a menor por el sueldo.

```
include <iostream>
#include <iomanip>
using namespace std;
#include "FuncionesAuxiliares.h"

int main(int argc, char** argv) {
    int dni[100], numEmp;
    double sueldo[100];
    char *empleado[100];

    leeDatosDelArchivo(dni,empleado,sueldo, numEmp,"personal.csv");
    ordenarEmpleados(dni, empleado, sueldo, numEmp);
    imprimirEmpleados(dni, empleado, sueldo, numEmp,
                      "ReporteOrdenadoDelPersonal.txt");

    return 0;
}

#ifndef FUNCIONESAUXILIARES_H
#define FUNCIONESAUXILIARES_H

    void leeDatosDelArchivo(int*, char**, double*, int&,const char*);
    void leerLinea(ifstream &,int &, char *&, double &);
    void ordenarEmpleados(int *, char **, double*, int );
    void imprimirEmpleados(int*, char**, double*, int, const char*);
    void cambiarDni(int &, int &);
    void cambiarNombre(char * &, char * &);
    void cambiarSueldo(double &, double &);

#endif /* FUNCIONESAUXILIARES_H */

/* Impementación de las funcione en le archivo: FuncDeCadenas.cpp */
#include <iostream>
#include <fstream>
#include <iomanip>
using namespace std;
#include <cstring>
#include "FuncionesAuxiliares.h"

void leeDatosDelArchivo(int *dni,char **empleado,double *sueldo,int &numEmp,
                        const char *nombArch){
    int dniArch, numPal;
    double sueldoArch;
    char *empleadoArch;
    ifstream arch(nombArch,ios::in);
    if(not arch.is_open()){
        cout << "ERROR: no se pudo abrir el archivo: "<<nombArch<<endl;
        exit(1);
    }
    numEmp = 0;
    while(1){
        // Leemos una línea del archivo CSV
        leerLinea(arch,dniArch,empleadoArch,sueldoArch);
        if(arch.eof())break;

        dni[numEmp] = dniArch;
        empleado[numEmp] = empleadoArch;
        sueldo [numEmp] = sueldoArch;
        numEmp++;
    }
}

void leerLinea(ifstream &arch,int &dni, char *&empleado, double &sueldo){
    char nombre[100];
    arch>>dni;  // La lectura se para en la coma pero asigna un valor correcto
    if(arch.eof()) return;
    arch.get(); // Sacamos la coma
    arch.getline(nombre, 100, ','); // Leemos el nombre hasta la coma
    empleado = new char[strlen(nombre)+1]; // Asignamos memoria exacta
    strcpy(empleado, nombre);
    arch>>sueldo;
}
```

```
void ordenarEmpleados(int *dni, char **empleado, double *sueldo, int numEmp){
    for(int i=0; i<numEmp-1;i++)
        for(int k=i+1; k<numEmp; k++)
            if(sueldo[i]<sueldo[k]){
                cambiarDni(dni[i],dni[k]);
                cambiarNombre(empleado[i],empleado[k]);
                cambiarSueldo(sueldo[i],sueldo[k]);
            }
}

void cambiarDni(int &dniI, int &dniK){
    int aux;
    aux = dniI;
    dniI = dniK;
    dniK = aux;
}

void cambiarNombre(char * &nombreI, char * &nombreK){
    char *aux;
    aux = nombreI;
    nombreI = nombreK;
    nombreK = aux;
}

void cambiarSueldo(double &sueldoI, double &sueldoK){
    double aux;
    aux = sueldoI;
    sueldoI = sueldoK;
    sueldoK = aux;
}

void imprimirEmpleados(int *dni, char **empleado, double*sueldo, int numEmp,
                       const char *nombArch){
    ofstream arch(nombArch,ios::out);
    if(not arch.is_open()){
        cout << "ERROR: no se pudo abrir el archivo: "<<nombArch<<endl;
        exit(1);
    }
    arch.precision(2);
    arch<<fixed;
    for(int i=0; i<numEmp;i++)
        arch<<left<<setw(10)<<dni[i]<<setw(35)<<empleado[i]
            <<right<<setw(10)<<sueldo[i]<<endl;
}
```