Capítulo

06

Arreglos

Ania Cravero Leal

Departamento de Ingeniería de Sistemas
Facultad de Ingeniería, Ciencias y Administración

"Proyecto financiado por el Fondo de Desarrollo Educativo de la Facultad de Ingeniería, Ciencias y Administración de la Universidad de La Frontera"

Versión

0.9

TEMARIO

Arreglos

Usamos un arreglo para procesar una colección de datos del mismo tipo, como una lista de temperaturas registradas durante un día o una lista de nombres obtenida desde un archivo de alumnos de un curso. En este capítulo te presentaremos los fundamentos de la definición y uso de arreglos en los lenguajes C, C++ y Java, y muchas de las técnicas básicas que se emplean para diseñar algoritmos y programas que usan arreglos.

6.1 Conceptos Básico

Supongamos que deseamos obtener las temperaturas del día viernes cada una hora, con el fin de determinar cuál fue la temperatura más alta, cuál fue la más baja, y cuál es el promedio de las mismas. No sabremos cuál es la temperatura más alta o más baja hasta que leamos todas las temperaturas, por tanto, debemos mantener en la memoria las 24 temperaturas para que, una vez determinada la temperatura más alta, cada temperatura se pueda comparar con ella.

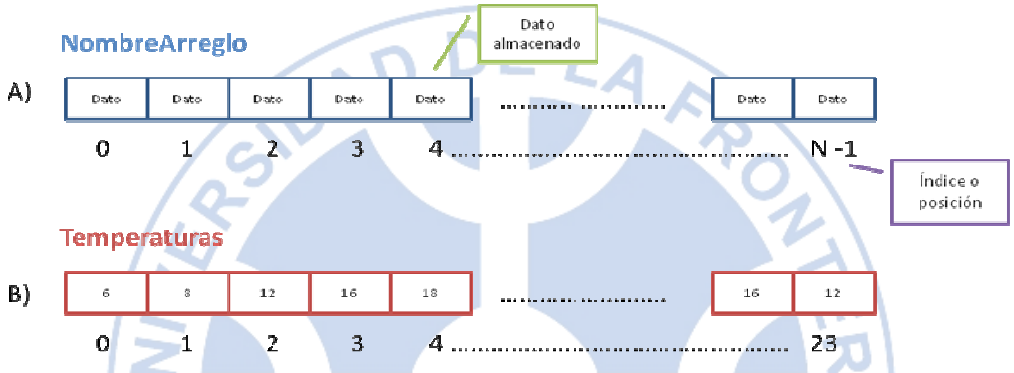
Para retener las 24 temperaturas requerimos algo equivalente a 24 variables de tipo int. Podríamos usar 24 variables individuales, pero no es fácil seguir la pista a 24 variables, y probablemente más adelante quisiéramos cambiar nuestro programa para registrar temperaturas cada media hora, o en cada minuto. Si n lugar a dudas 48 o más variables será poco práctico. Un arreglo es la solución perfecta para este tipo de situaciones.

a. Arreglo:

Un arreglo es una zona de memoria (variable) que puede almacenar un conjunto de N datos del mismo tipo.

Por lo tanto, un arreglo es una variable que dispone de una gran zona de memoria separada en N celdas del mismo tamaño. La figura 6.1 a) presenta una representación de la zona de memoria de un arreglo.

Figura 6.1: Representación de un arreglo



Observa que en la figura 6.1 b) está representado un arreglo con nombre Temperaturas que almacena las 24 temperaturas registradas del día viernes. Así tenemos que en la celda número 2 (índice 2) se ha registrado una temperatura de 12 ºC.

OJO: No debemos confundir ***índice*** con **dato almacenado**. El índice indica sólo una posición de cada celda del arreglo, las que están numeradas desde el número cero hasta N-1. Así por ejemplo, si necesitamos un arreglo con 24 celdas, entonces estas serán automáticamente numeradas desde el cero hasta 23. Por otro lado, el dato almacenado puede ser de cualquiera de los tipos de datos que ya conocimos en el capítulo 5 de este libro. Por ejemplo, pude ser un número entero, un número con punto flotante, una letra, un carácter, etc.

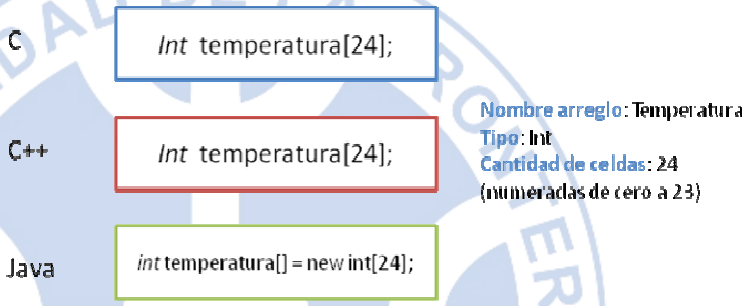
6.2 Arreglos Unidimensionales

Un arreglo unidimensional es capaz de almacenar N datos del mismo tipo, tal y como hemos mostrado en la figura 6.1

6.2.1 Declaración de arreglos

Como toda variable, antes de utilizar un arreglo debemos declararlo. Es decir, debemos definir un nombre que lo representa, un tipo de dato, y la cantidad de celdas que tendrá. La figura 6.2 presenta un ejemplo de declaración en los lenguajes C, C++ y Java.

Figura 6.2: Ejemplo declaración de un arreglo unidimensional



Observa que en los lenguajes C y C++ la forma general para declarar un arreglo es la siguiente:

```
Tipo  nombre[numCeldas];
```

En cambio en lenguaje Java es completamente diferente:

```
Tipo nombre[] = new tipo[numCeldas];
```

Nota: El operador **new** de Java sirve para realizar las siguientes acciones:

- Separar memoria para el nuevo objeto que se almacenará. En la figura 6.2 new separa memoria del computador para almacenar un arreglo de 24 celdas de tipo entero.

- Invocar el método (función) de inicio de la clase llamado constructor. Esto debido a que en ocasiones Java requerirá crear un gran espacio de memoria, es decir, se debe construir el espacio.

- Retornar la referencia a un nuevo objeto. En Java los objetos almacenan referencias, a diferencia de las variables primitivas que almacenan datos.

Entonces, ¿podemos decir que un arreglo es un objeto en Java?

Ejemplo 6.1:

Algunos ejemplos de declaración de arreglos unidimensionales para los lenguajes C y C++ son los siguientes:

```
char apellido[50];
```

//declaramos un arreglo que puede almacenar una secuencia de hasta 50 caracteres.

```
float  peso[20];
```

//declaramos un arreglo que puede almacenar 20 números reales.

En Java sería:

```
char  apellido[]  =  new char[50];

float  peso[]  =  new float[20];
```

El ejemplo 6.1 muestra arreglos declarados utilizando varios tipos de datos y tamaños, tanto para los lenguajes C y C++ como Java.

OJO!!: En C++ y Java existe un tipo **string** que crea de manera automática un arreglo que puede almacenar hasta 256 caracteres. La forma de declarar el arreglo en C++ y Java es así respectivamente:

```
string nombre;
```

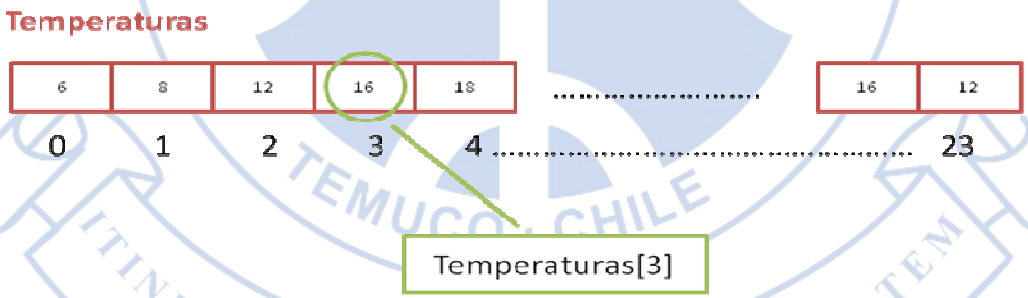
```
String nombre[];
```

Observe que no se debe utilizar corchetes [] para el caso de C++.

6.2.2 Acceso a un arreglo

Para acceder al contenido de un arreglo debemos utilizar el nombre del arreglo y el índice o posición en donde está ubicado dicho dato. La figura 6.3 muestra un ejemplo de acceso al arreglo Temperaturas para cualquiera de los lenguajes de programación revisados en este libro.

Figura nº 6.3: Ejemplo de acceso a un arreglo unidimensional

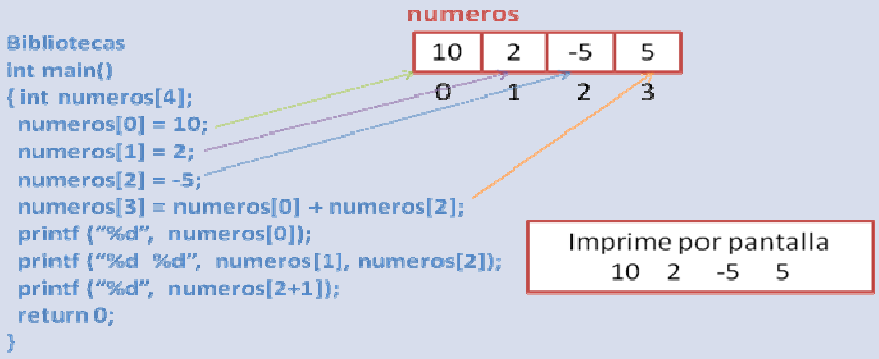


La figura anterior muestra el arreglo ***Temperaturas*** con 24 datos almacenados. El recuadro verde accede específicamente al dato ubicado en la posición 3, es decir, la temperatura 16 ºC.

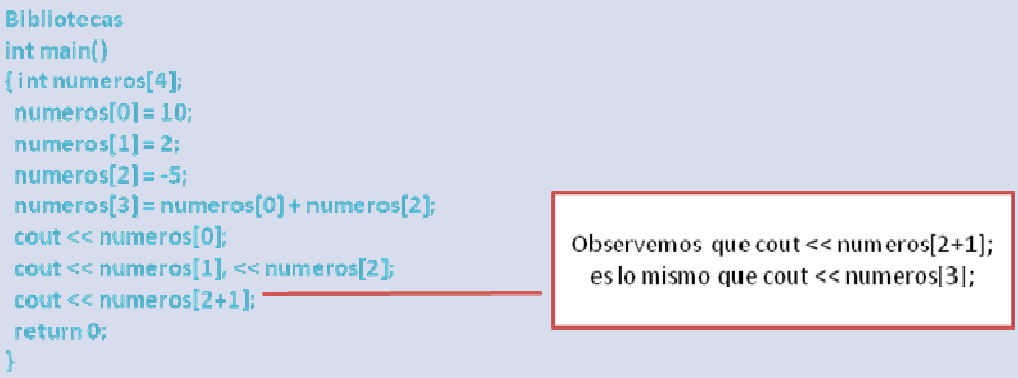
Ejemplo 6.2:

Este ejemplo muestra un programa en los lenguajes C, C++ y Java, que accede a cada celda de un arreglo, tanto para almacenar los datos como para procesarlos.

Lenguaje C:



Lenguaje C++:



Lenguaje Java:

```
Bibliotecas
public class arreglo{
    public static void main (String [] args) {
      int numeros[] = new int[4];
 numeros[0] = 10;
 numeros[1] = 2;
 numeros[2] = -5;
 numeros[3] = numeros[0] + numeros[2];
 System.out.println( numero[0]);
 System.out.println( numero[1], numero[2]);
 System.out.println( numero[2+1]);
                          }

}
```

Observamos que para almacenar un dato en una celda específica del arreglo necesitamos el nombre del arreglo y la ubicación de la celda (índice). De tal manera que

```
numeros[2] = -5
```

le asigna el número -5 a la celda en posición dos, y que

```
numeros[3] = numeros[0] + numeros[2];
```

le asigna la suma del contenido de las celdas cero y dos a la celda número tres, es decir, asigna 10 -5 que es 5.

También podemos ingresar los números desde el teclado. En tal caso, tendríamos lo siguiente para C, C++ y Java:

Lenguaje C

```
scanf ("%d", &numeros[0]);
```

Lenguaje C++

```
cin >> numeros[0];
```

Lenguaje Java:

```
numeros[0] = Integer.parseInt(In.readLine());
```

6.2.3 Inicialización de arreglos

Podemos inicializar un arreglo al declararlo. Al hacerlo, los datos que se almacenarán en cada celda se encierran en llaves y se separan con comas. Por ejemplo:

Lenguaje C y C++:

```
int pesos[3] = { 2, 12, 20};
```

Lenguaje Java:

```
int[] numeros={2, -4, 15};
```

el número encerrado en los corchetes especifica cuántas celdas de memoria tiene el arreglo. Cuando se usan en otro lugar el programa, el número encerrado en los corchetes indica cuál celda es la que se está utilizando o accediendo.

OJO!! El índice entre corchetes no tiene que ser siempre una constante entera. Podemos usar cualquier expresión en los corchetes en tanto la evaluación de la expresión dé uno de los enteros de cero hasta uno menos del tamaño del arreglo. Por ejemplo, lo que sigue asigna el dato 23 a pesos[2]:

```
int n =1;
pesos[n+ 1] =23;
```

Aunque se vea diferentes, pesos[n+ 1] es lo mismo que pesos[2]. La razón es que n + 1 es igual a 2.

La identidad de una celda específica, como pesos[i], la determina el valor de su índice, que en este caso es i. Así, podemos escribir programas que digan cosas como "hacer esto y lo otro con la celda i-ésima del arreglo", donde el programa calcula el valor de i. Esto es muy utilizado cuando disponemos de muchas celdas en el arreglo y necesitamos realizar varias acciones con las mismas. En la siguiente sección te mostraremos ejemplos de ello.

Nota: El uso de ***i*** como índice lo usaremos debido a que es muy común que cualquier programador, o incluso en los libros, utilicen como índices las letras i, j o k.

Recuerda que el declarar variables con nombres inapropiados no es una buena práctica de programación, por tanto, esta sería una excepción.

6.2.4 Utilizando ciclos para acceder a un arreglo

Cuando un arreglo dispone de varias celdas de memoria es muy común utilizar ciclos for para acceder a cada una de las celdas, ya sea para asignar datos, para leer datos desde el teclado, para procesar los datos o para mostrar cada dato almacenado en las distintas celdas de memoria.

Veamos algunos ejemplos prácticos.

Ejemplo 6.3:

Este ejemplo muestra el promedio de 10 números ingresados en un arreglo de enteros, tanto en C, C++ y Java.

Lenguaje C:

```
Bibliotecas
int main()
{ int numeros[10], i, suma =0;
  float promedio;
  for (i=0; i < 10; i++)
     { printf ("Ingrese un número para la celda nº %d", i);
       scanf ("%d", &numeros[i]);
       suma = suma + numeros[i];
     }
   promedio = (float ) suma /10;
   printf ("El promedio es %f ", promedio);
   return 0;
}
```

El índice i va obteniendo los valores desde el cero hasta 9 a medida que el ciclo for se ejecuta

Leemos un dato para la celda i Cuando i es cero, el dato queda almacenado en numeros[0]

Lenguaje C++:

```
Bibliotecas
int main()
{ int numeros[10], i, suma =0;
  float promedio;
  for (i=0; i < 10; i++)
     { cout << "Ingrese un número para la celda nº " << i ;
       cin >> numeros[i];
       suma = suma + numeros[i];
     }
   promedio = (float ) suma /10;
    cout << "El promedio es " << promedio ;
   return 0;
}
```

Para calcular el promedio debemos utilizar el conversor (float) ya que suma/10 entrega como resultado un entero y no un número real.
(float) convertirá el resultado en un número real.

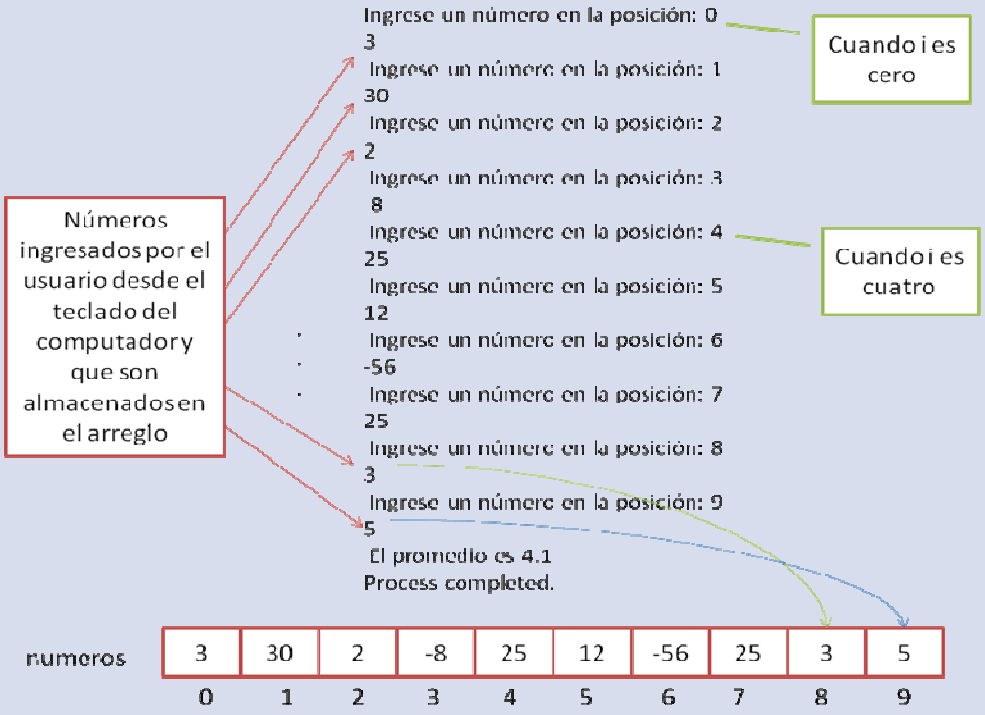
También podríamos haber utilizado una constante para el tamaño del arreglo
`#define N 10`
Así podemos declarar el arreglo así:
`int numeros[N];`
Y el ciclo for sería:
`for (i=0; i < N; i++)`

La ventaja es que si necesitamos cambiar el tamaño del arreglo en cualquier momento, basta con modificar el valor de la constante N

Lenguaje Java:

```
Bibliotecas
public class Promedio  {
    public static void main (String [] args) {
      int numeros[] = new int[10];
      int suma =0, i;
      float prom =0;
      BufferedReader in =new BufferedReader (new InputStreamReader(System.in));

            try{  for(i=0; i<10; i++)
                  {System.out.println(" Ingrese un número en la posicion: " + i);
                    numeros[i] = Integer.parseInt(in.readLine());
                      suma=suma + numeros[i];
                  }
                  prom =(float) suma / 10;
                System.out.println(" El promedio es " + prom);

            }
            catch (Exception e){
                    e.printStackTrace();
                  }
          }
}
```

La figura siguiente muestra un ejemplo de ejecución del programa escrito en Java.



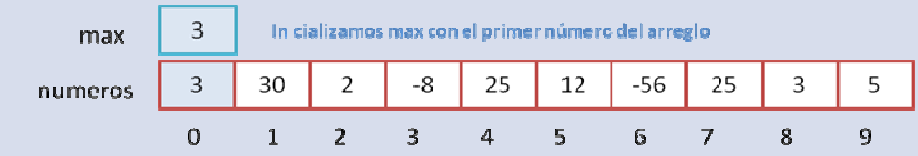
Ejemplo 6.4:

El siguiente programa escrito en C++ muestra el mayor de 10 números ingresados por el usuario.

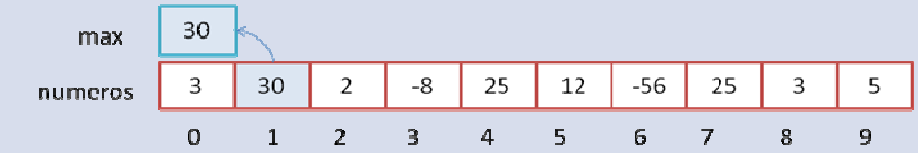
```
Bibliotecas
# define N 10
int main()
{ int numeros[N], i, max;

 //primero ingresamos los números
 for (i=0; i < N; i++)
      { cout << "Ingrese un número para la celda nº " << i ;
        cin >> numeros[i];
        }
 //después buscamos el mayor
  max = numeros[0];
  for (i=1; i < N; i++)
     { if ( max < numeros[i])
          // actualizamos max con el mayor encontrado hasta ahora
          max = numeros[i];
       }
    cout << "El mayor de los números es " << max;
   return 0;
}
```

En primer lugar, debemos ingresar los números al arreglo. Para ello utilizamos un ciclo ***for*** con un índice ***i*** que tomará los valores desde el cero hasta N-1 (es decir 9) a medida que el ciclo se va ejecutando.

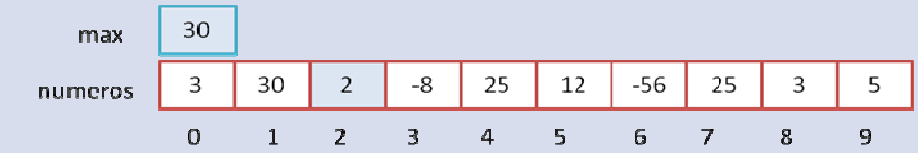
Después de ello, podemos buscar el número mayor. Como no sabemos en qué posición se encuentra exactamente utilizaremos una variable auxiliar (variable ***max***) que se inicializará con el primer número almacenado en el arreglo (es decir *numero[0]*). A medida que avanza el ciclo *for*, debemos comparar el contenido de la variable *max* con la siguiente celda de memoria (la primera vez compara con *numero[1]*); en caso de que *max* contiene un número menor a la de la celda que actualmente está comparando (*if (max < numero[i]*) quiere decir que debemos actualizar *max* con el nuevo número que hasta ahora es el mayor. El proceso finaliza cuando comparamos *max* con la última celda del arreglo (*numeros[9]*), si el contenido de ésta es mayor que *max*, entonces *max* se vuelve a actualizar con el valor almacenado en *numero[9]*, en otro caso, *max* mantiene su valor actual. La siguiente figura muestra el proceso cuando i varía desde cero hasta N-1.



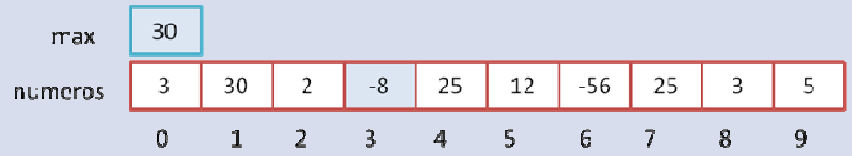
Cuando i= 1, comparamos max con la celda numeros[1]. Como 3 es menor que 30, entonces max se actualiza con el valor 30



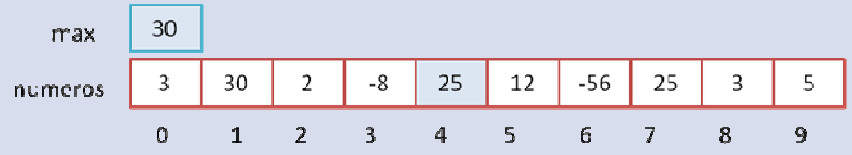
Cuando i= 2, comparamos max con la celda numeros[2]. Como 30 no es menor que 2, entonces max mantiene su valor en 30



Cuando i= 3, comparamos max con la celda numeros[3]. Como 30 no es menor que -8, entonces max mantiene su valor en 30

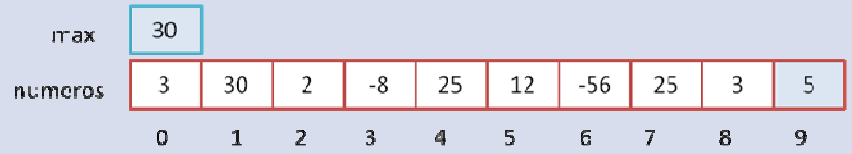


Cuando i= 4, comparamos max con la celda numeros[4]. Como 30 no es menor que 25, entonces max mantiene su valor en 30



.....

Cuando i= 9, comparamos max con la celda numeros[9]. Como 30 no es menor que 5, entonces max mantiene su valor en 30



Observamos que finalmente *max* almacena el dato 30, que justamente es el número mayor de los ingresados en el arreglo.

¿Qué debemos modificar del programa del ejemplo 6.4 para que muestre el número menor?

¿Es posible encontrar el número menor sin utilizar una variable auxiliar? Comenta con tus compañeros.

6.2.5 Búsquedas

Existen varios algoritmos para buscar un elemento en una lista, siendo los más conocidos, la búsqueda secuencial y la búsqueda binaria. Estos algoritmos permiten solucionar problemas del tipo, “encontrar los datos de un cliente”, “encontrar las características de un producto”, “encontrar un determinado número”, etc.

El siguiente ejemplo muestra un programa escrito en C++ que permite buscar un número específico en una lista de números. El algoritmo que hemos aplicado es la búsqueda secuencial.

Ejemplo 6.5:

El próximo programa escrito en C++ permite buscar un número en el arreglo y decir si está o no está en el arreglo.

```
Bibliotecas
# define N 10
int main()
{ int numeros[N], i, esta =0 , NumBusca;
 /*esta = 0 significa que supondremos que el número que desea buscar el usuario no está en
el arreglo */

 //primero ingresamos los números
 for (i=0; i < N; i++)
     { cout << "Ingrese un número para la celda nº " << i ;
       cin >> numeros[i];
       }

// después el usuario ingresa el número que desea buscar
 cout << "Ingrese el número que desea buscar";
 cin >> NumBuscar;

  //después buscamos el número ingresado en el arreglo
  for (i=0; i < N; i++)
     { if ( NumBuscar == numeros[i])
          { // cambiamos la  variable esta a 1, que significa verdadero
               esta = 1;
               break;
           }
      }
   if (esta == 0)  //si mantuvo el valor cero quiere decir que el número no está
         cout << "El " << NumBuscar << "NO está en el arreglo";
   else
         cout << "El " << NumBuscar << "SI está en el arreglo" ;
   return 0;
}
```

La variable "esta" trabajará como una bandera. De esta forma cuando almacene 0 significa falso y cuando almacene 1 significa verdadero.
Nota: en C++ existen las variables boolenas, así que podríamos utilizar la variable de la siguiente manera:
bool esta =false;
Y dentro del if sería:
esta = true;

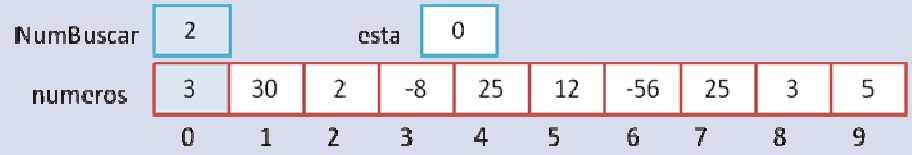
En primer lugar debemos ingresar los números al arreglo. Para ello utilizamos un ciclo *for* con la variable de control *i*, que hace de índice en cada celda del arreglo.

En segundo lugar, el usuario debe ingresar el número que desea buscar para saber si está o no está en el arreglo. Para ello, utilizamos la variable ***NumBuscar*** que almacena dicho número.

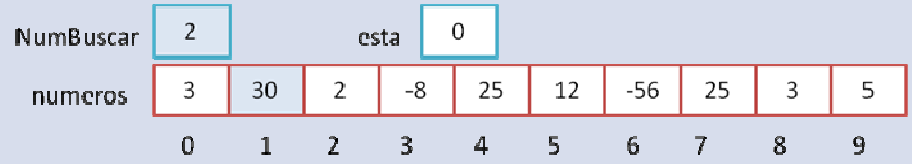
Para buscar el número (variable *NumBuscar*) en el arreglo, necesitaremos comparar dicho número con cada número almacenado en la celdas del arreglo. Así que nos conviene nuevamente utilizar un ciclo *for* con la variable de control *i* para recorrer todas las celdas del arreglo *numeros*.

La siguiente figura muestra el proceso de búsqueda cuando *i* varía de cero a N-1. Supondremos que el usuario ya ha ingresado los números al arreglo y que NumBuscar es 2.

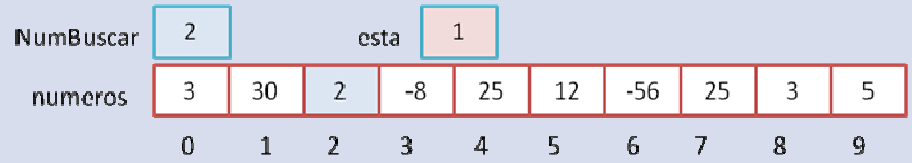
Cuando i= 0, comparamos NumBuscar con la celda numeros[0]. Como 2 no es igual a 3, entonces la variable esta mantiene su valor en 0



Cuando i= 1, comparamos NumBuscar con la celda numeros[1]. Como 2 no es igual a 30, entonces la variable esta mantiene su valor en 0



Cuando i= 2, comparamos NumBuscar con la celda numeros[2]. Como 2 es igual a 2, entonces la variable esta cambia su valor a 1, y finaliza el ciclo al ejecutar break.



Por lo tanto If (esta ==0) será falso e imprimirá "El 2 SI está en el arreglo".

OJO!! Cuando escribimos **if (esta ==0)** es lo mismo que decir si ***esta*** es falso. Algunos programadores prefieren escribir la condición de la siguiente manera.

if (!esta)

Que también significa si ***esta*** es falso.

RIESGO índice de arreglo fuera de intervalo: El error de programación más común que se comete al usar arreglos es tratar de hacer referencia a un elemento inexistente del arreglo. Por ejemplo, considera la siguiente declaración de un arreglo

int notas[6];

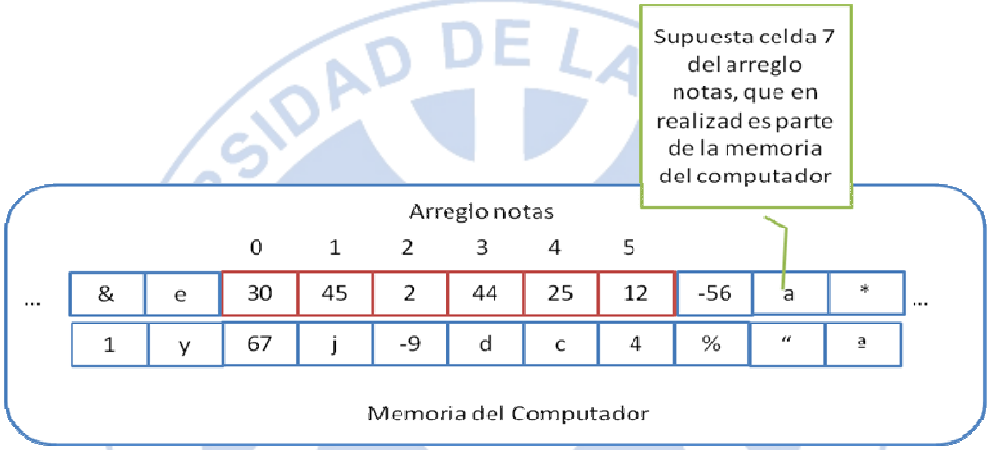
Al utilizar el arreglo ***notas***, toda expresión que se use como índice deberá dar uno de los valores enteros 0 a 5 al evaluarse. Por ejemplo, si el programa contiene la sentencia *notas[i],* la evaluación de i debe dar alguno de los enteros de 0 a 5. Si la evaluación de i da alguna otra cosa, será un error. En este caso, decimos que el índice está fuera del intervalo o simplemente que no es válido.

Un ejemplo de error sería si ejecutamos la siguiente instrucción

```
notas[7] =220;
```

En este caso, estaríamos almacenado el dato 220 en una zona de memoria ubicada en la supuesta posición 7 del arreglo, y borraríamos lo que estaba almacenado allí, quizás por otra variable o por el propio sistema.

La siguiente figura representa la situación.



6.2.6 Ordenamiento

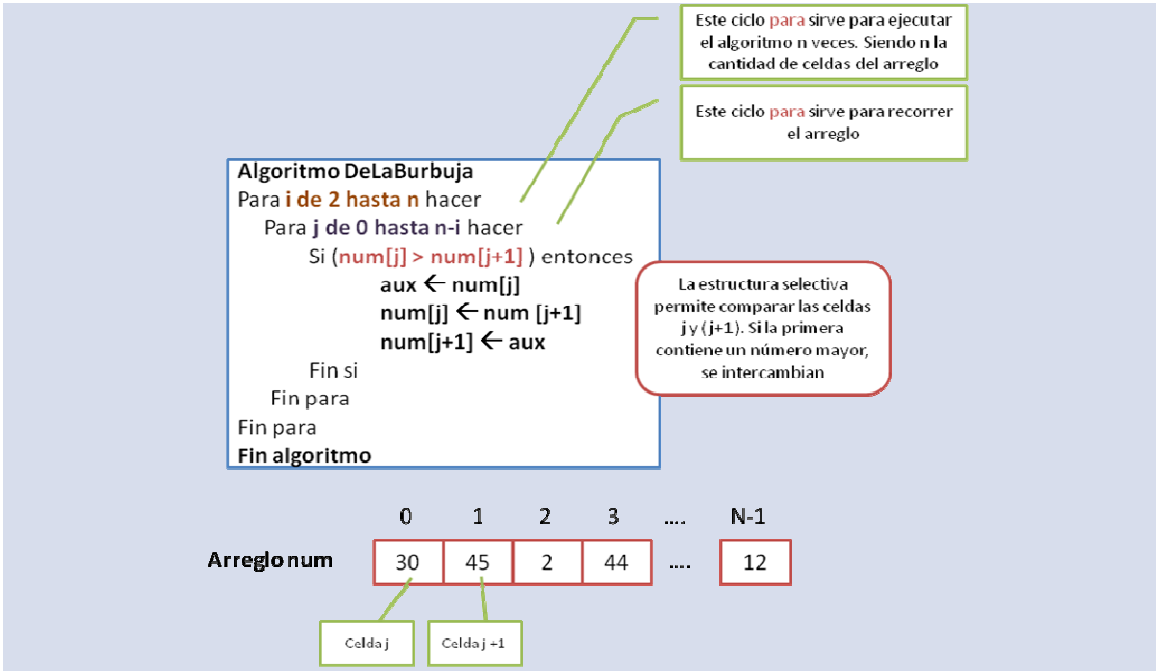
También existen algoritmos que permiten ordenar una lista de elementos. Algunos de los algoritmos más conocidos son, el de la burbuja, por selección, por inserción, por intercambio y el quicksort. Estos algoritmos permiten resolver problemas del tipo “mostrar los datos de los clientes en orden alfabético”, “mostrar las ventas de la semana de manera ascendente”, etc.

A continuación te mostramos un programa que permite ordenar números utilizando el famoso algoritmo de la burbuja.

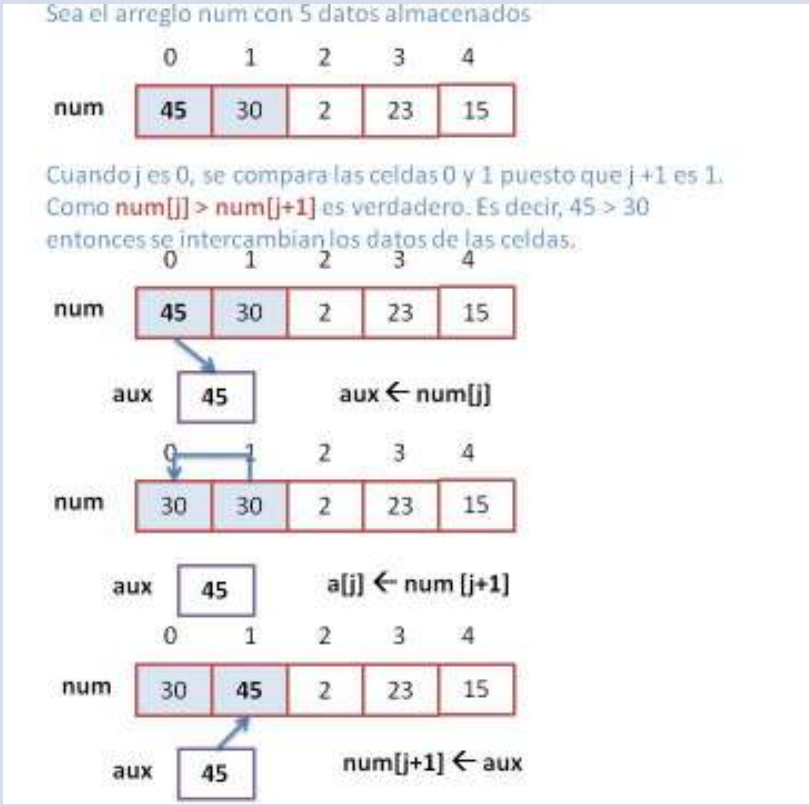
Ejemplo 6.6: Ordenamiento de la Burbuja

La **Ordenación de burbuja** (**Bubble Sort** en inglés) es un sencillo algoritmo de ordenamiento. Funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el orden equivocado. Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada. Este algoritmo obtiene su nombre de la forma con la que suben por la lista los elementos durante los intercambios, como si fueran pequeñas "burbujas". También es conocido como el **método del intercambio directo**. Dado que solo usa comparaciones para operar elementos, se lo considera un algoritmo de comparación, siendo el más sencillo de implementar.

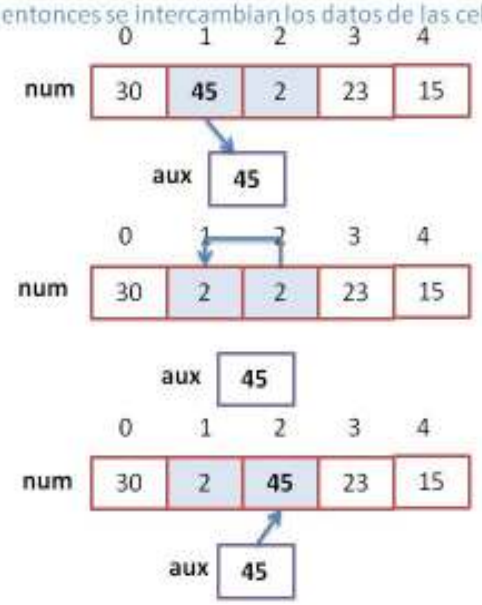
El algoritmo básico es el siguiente:



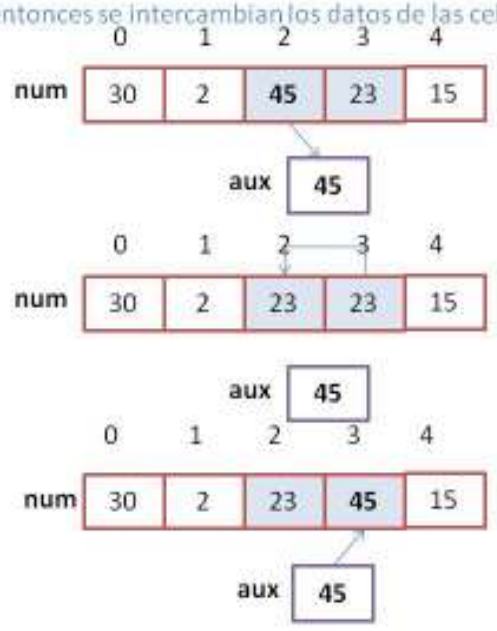
Un ejemplo de ejecución del algoritmo se muestra a continuación.



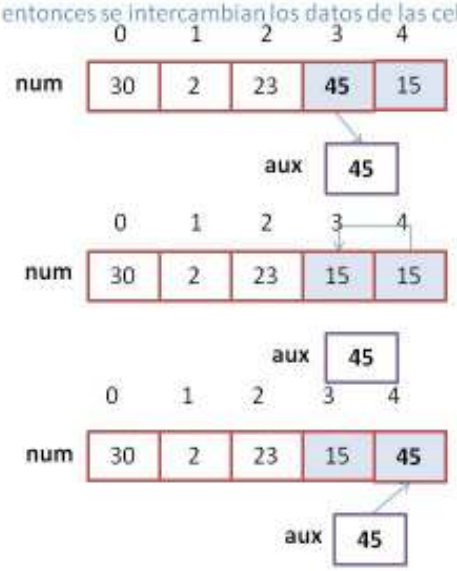
Cuando j es 1, se compara las celdas 1 y 2 puesto que j +1 es 2.
Como num[j] > num[j+1] es verdadero. Es decir, 45 > 2
entonces se intercambian los datos de las celdas.



Cuando j es 2, se compara las celdas 2 y 3 puesto que j +1 es 3.
Como num[j] > num[j+1] es verdadero. Es decir, 45 > 23
entonces se intercambian los datos de las celdas.



Cuando j es 3, se compara las celdas 3 y 4 puesto que j +1 es 4. Como **num[j] > num[j+1]** es verdadero. Es decir, 45 > 15 entonces se intercambian los datos de las celdas.



Observemos que el número **45**, que es el mayor de todo el arreglo, ha quedado almacenado en la última posición. Es como si una burbuja subiera.

Ahora quedará ejecutar el mismo proceso 4 veces más para que finalmente el arreglo quede totalmente ordenado.

El próximo programa escrito en C++ permite ordenar los números del un arreglo de manera ascendente de acuerdo al método de ordenamiento de la Burbuja.

```
Bibliotecas
# define N 10
int main()
{ int numeros[N], i,j, aux;

 //primero ingresamos los números
 for (i=0; i < N; i++)
      { cout << "Ingrese un número para la celda nº " <<i ;
        cin >> numeros[i];
        }

 //después ordenamos los números
  for (i=2; i <= N; i++)
     for (j =0; j < N-i , j++)
      { if ( numeros[j] > numeros[j-1])
          { // intercambiamos las  variables
               aux = numeros [j];
               numeros [j] = numeros [j+1];
               numeros [j+1] = aux;
           }
       }

 //Finalmente mostramos los números ordenados
 for (i=0; i < N; i++)
      { cout << "a [" <<j<<"] ="<< a[j] << endl;
        }
 return 0;
 }
```

¿Cómo podemos ordenar de mayor a menor?

Modifica el programa anterior para solucionar el problema.

6.3 Arreglos Multidimensionales

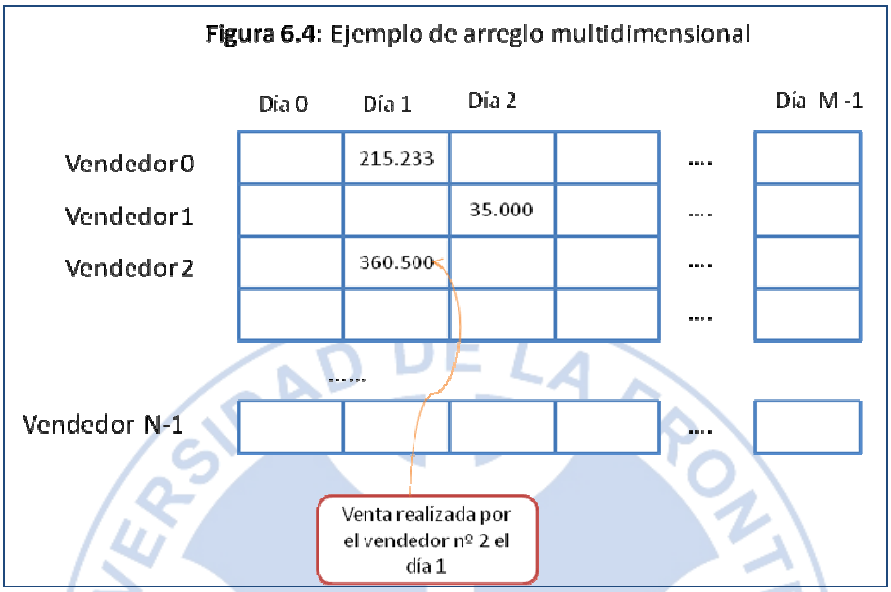
Los lenguajes C, C++ y Java permiten declarar arreglos con más de un índice. En esta sección describiremos estos arreglos multidimensionales.

6.3.1 Fundamentos de arreglos multidimensionales

A veces es útil disponer de un arreglo con más de un índice, ya que nos da la posibilidad de manipular información que requiere dos o más dimensiones. Por ejemplo, si una empresa desea registrar las ventas diarias, por un mes, de cada vendedor de la tienda. En este caso necesitamos dos dimensiones: día y vendedor, y los datos que almacenaríamos en el arreglo son las ventas.

La figura 6.4 muestra un esquema que representa el arreglo bidimensional para registrar las ventas de los vendedores de la empresa.

Figura 6.4: Ejemplo de arreglo multidimensional



Observa que la figura 6.4 muestra un arreglo con dos dimensiones, el número del vendedor para las filas y el número del día del mes para las columnas. En general la empresa dispone de N vendedores y el mes es de M días; y las ventas realizadas, que son los datos, son almacenados en las celdas. Así tenemos que la venta del vendedor 2 el día 1 es de $360.500.

A este tipo de arreglo se le llama comúnmente matriz, ya que para acceder a los valores almacenados necesitamos conocer el índice de la fila y el índice de la columna de cada celda específica.

Ejemplo 6.6:

Ahora te mostraremos algunos ejemplos de declaraciones de matrices utilizando distintos tipos de datos y dimensiones.

En C y C++

```
float promedios[10][25];

long int pesos[5][33];
```

En Java

```
float promedios[][] = new float[10][25];

long pesos[][] = new long [5][33];
```

6.3.2 Matrices

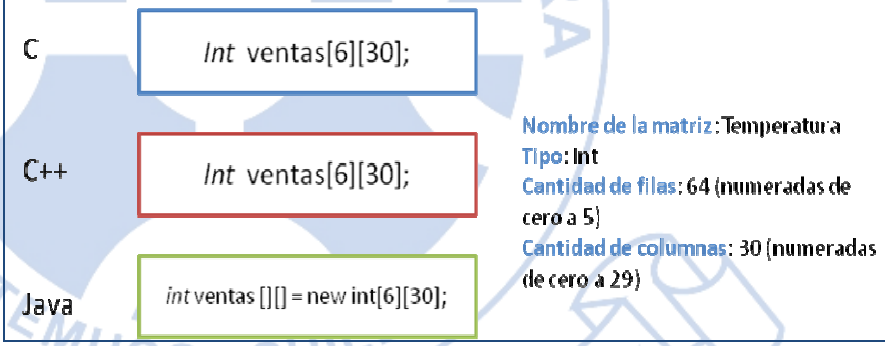
a. Matriz:

Una matriz es un arreglo bidimensional que almacena de manera general N x M datos del mismo tipo. N corresponde al número de filas y M corresponde al número de columnas.

6.3.3 Declaración de matrices

Para declarar una matriz necesitamos un tipo de dato, un nombre que la represente, y la definición de su tamaño. El tamaño de la matriz está determinado por la cantidad de filas y columnas. La figura 6.5 muestra un ejemplo de declaración en C, C++ y Java.

Figura 6.5: Ejemplos de declaración de una matriz



Observa que en los lenguajes C y C++ la declaración de una matriz es de la misma forma

```
Tipo nombreMatriz [numFilas][numColumnas];
```

En cambio para Java, las matrices se declaran de la siguiente forma general

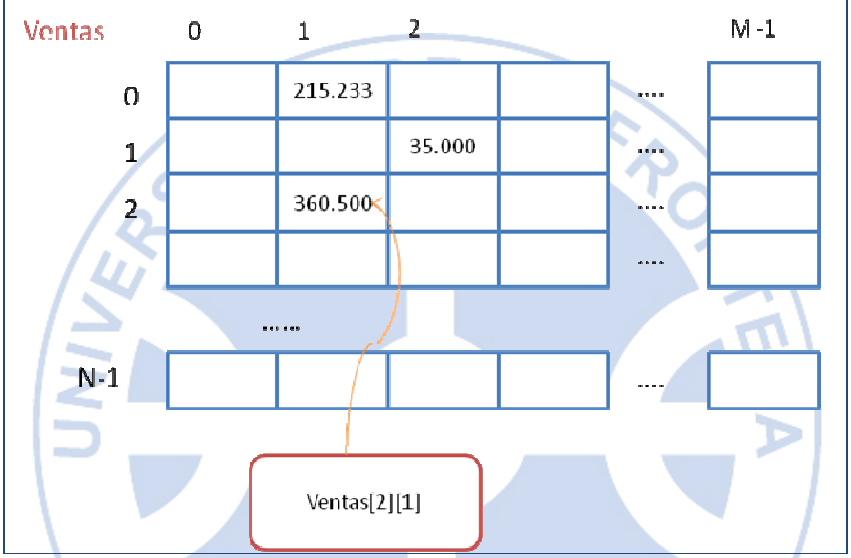
```
Tipo nombreMatriz [][] = new tipo [numFilas][numColumnas];
```

El ejemplo 6.6 presenta algunos ejemplos de declaraciones de matrices en los tres lenguajes.

6.3.4 Acceso a una matriz

Para acceder al contenido de una celda específica de una matriz debemos hacerlo mediante su nombre, posición de la fila, y posición de la columna. La figura 6.6 muestra un ejemplo de acceso.

Figura 6.6: Ejemplo de acceso a un dato en una matriz



Observa que Ventas[2][1] representa el contenido de la celda ubicada en la fila 2 y columna 1, es decir, los 360.500.

Nuevamente, y al igual que los arreglos unidimensionales, se deberá utilizar índices (por lo general, i, j y k) para representar la posición en la que se encuentra la celda que se está accediendo desde un programa. La única diferencia, es que ahora necesitaremos dos índices, el primer para determinar la posición de la fila y el segundo para determinar la posición de la columna.

El ejemplo siguiente muestra un programa (en los tres lenguajes) que declara y accede a una matriz.

Ejemplo 6.7:

El siguiente programa permite ingresar desde el teclado las ventas de una semana para los 10 vendedores de una empresa. Luego el programa calculará el promedio de ventas semanal par cada vendedor.

Lenguaje C

```
Bibliotecas
#define VENDEDORES 10
#define DIAS 7
int main()
{ int ventas[VENDEDORES][DIAS], i, j, suma ;
  float promedio;
  for (i=0; i < VENDEDORES; i++)
     {   printf  ("Ingrese  las ventas de la semana para el vendedor nº %d \n", i);
         for (j =0; j < DIAS ; j++)
            {  printf ("Ingrese venta para el dia  %d :", j);
               scanf ("%d", &ventas[i][j]);
              }
     }
   for (i=0; i < VENDEDORES; i++)
     {   suma = 0;
         for (j =0; j < DIAS ; j++)
               suma = suma + ventas[i][j];
         promedio = (float ) suma /DIAS;
         printf ("El promedio de ventas para el vendedor %d es  %f  \n", promedio);
     }
   return 0;
}
```

Cuando i es cero, calcula la suma dela fila cero, para luego calcular el promedio. Recuerda que i irá avanzado de cero a 9, y j de cero a 6.

Lenguaje C++

```
Bibliotecas
#define VENDEDORES 10
#define DIAS 7
int main()
{ int ventas[VENDEDORES][DIAS], i, j, suma ;
  float promedio;
  for (i=0; i < VENDEDORES; i++)
     {   cout  <<"Ingrese  las ventas de la semana para el vendedor nº", << i <<endl;
         for (j =0; j < DIAS ; j++)
            {  cout << "Ingrese venta para el dia :" << j;
               cin >> ventas[i][j];
              }
     }
   for (i=0; i < VENDEDORES; i++)
     {   suma = 0;
         for (j =0; j < DIAS ; j++)
               suma = suma + ventas[i][j];
         promedio = (float ) suma /DIAS;
         cout << "El promedio de ventas para el vendedor "<< i <<" es  "<< promedio<<endl;
     }
   return 0;
}
```

Lenguaje Java

```
Bibliotecas
public class Ventas  {
    public static void main (String [] args) {
      final int VENDEDORES =10;
      final int DIAS =7;
      int ventas[][]= new int[VENDEDORES][DIAS];
      int suma =0, i,j;
      float promedio =0;
      BufferedReader in =new BufferedReader (new InputStreamReader(System.in));

            try{ for (i=0; i<VENDEDORES; i++)
                  {System.out.println(" Ingrese las ventas de la semana para el vendedor: " + i);
                   for (j=0; j < DIAS; J++)
                       { System.out.println(" Ingrese ventas para el día: " + j);
                        ventas[i][j] = Integer.parseInt(in.readLine());
                       }
                  }
               }
            catch (Exception e){
                    e.printStackTrace();
                  }
       for (i=0; i < VENDEDORES; i++)
          {  suma = 0;
             for (j =0; j < DIAS ; j++)
                 suma = suma + ventas[i][j];
                 promedio = (float ) suma /DIAS;
          System.out.println(" El promedio de ventas para el vendedor " + i + "es " + promedio);
          }
      }
}
```

Modifica el programa anterior (escoge algún lenguaje) para responder las siguientes preguntas:

- Qué vendedor vende más en la semana
- Qué día de la semana se logra la mayor venta (considere la venta de un día la suma de ventas década vendedor para ese día)
- Qué vendedores lograron ventas superiores a $100.000. Indique el día en que lo logra.

6.4 Cadenas

Hasta ahora te hemos presentado ejemplos de arreglos y matrices que procesan números. En este apartado te mostraremos que los arreglos y matrices también pueden almacenar caracteres. A este tipo de vectores le llaman cadenas.

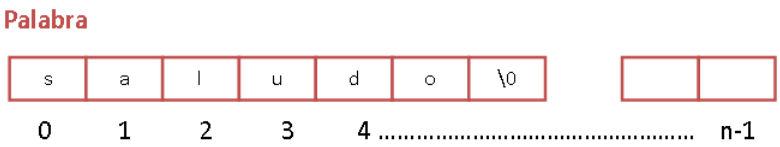
6.4.1 Fundamentos de cadenas

a. Cadena

Una cadena es un vector que almacena N caracteres, por lo tanto cada celda puede almacenar un dato de tipo char.

En C y C++ las cadenas se almacenan en un arreglo de caracteres, utilizando el marcador '\0' como fin de cadena. La figura 6.7 muestra una representación de un arreglo que almacena una cadena de caracteres

Figura 6.7: Ejemplo de cadena en C y C++



En la figura 6.7 '\0' es un marcador de fin de cadena que es necesario para determinar donde finaliza la cadena. De esta manera podemos mostrar por pantalla la cadena sólo imprimiendo el vector (en la figura es la variable Palabra). Lenguaje C o C++ automáticamente mostrarán por pantalla todos los caracteres (uno al lado del otro) hasta que se encuentra con este marcador (en la figura imprimirá saludo).

El carácter nulo, '\0': Este carácter se utiliza para marcar el final de una cadena que se almacena en un arreglo de caracteres. Aunque el carácter nulo '\0' se escribe con dos símbolos, es un solo carácter que puede almacenarse en un arreglo de tipo *char*.

Para manejar las cadenas contamos con bibliotecas que disponen de las funciones necesarias para su funcionamiento.

Dentro de las bibliotecas usadas en C y C++ está:

string.h: Que contiene tipos, macros y funciones para la manipulación de cadenas de caracteres.

En Java las cadenas son objetos de las clases predefinida String o StringBuffer, que están incluidas en el paquete java.lang.*

String: Java crea un arreglo de caracteres o una cadena de forma similar a como lo hace C++. A estas cadenas accedemos a través de las funciones que poseen esta clase.

6.4.2 Declaración de cadenas

- **Cadenas tipo *char para C, C++ y Java*:**

Una variable de cadena es sólo un arreglo de caracteres. Por tanto, la siguiente declaración de un arreglo en C y C++ nos proporciona una variable de cadena capaz de almacenar un valor de cadena de tipo char con nueve o menos caracteres.

```
char palabra[10];
```

El 10 es para las nueve letras de la cadena, más el carácter nulo '\0' para marcar el final de cadena. Observa la figura 7.7 que muestra la variable palabra que contiene la cadena "saludo" y a continuación el carácter nulo '\0'.

En Java la declaración de un arreglo de caracteres sería de la siguiente manera.

```
char [] palabra = new char[10];
```

Podemos inicializar una cadena al declararla, como se muestra a continuación:

```
char miMensaje[20]= "Hola alumnos";
```

En Java sería de la siguiente manera:

```
char miMensaje[] = {'H','o','l','a',' ','a','l','u','m','n','o','s'};
```

Observa que la cadena que se asigna a la variable miMensaje no necesita llenar todo el arreglo.

Cuando inicializamos una cadena se puede omitir el tamaño el arreglo. C, C++ se encargarán de manera automática que el tamaño de la variable de cadena sea uno más que la longitud de la cadena entre comillas. (La celda extra es para almacenar el símbolo '\0', pero sólo en C y C++).

```
char cadena[] = "hola amiguito";
```

OJO!!! Asegúrate de no confundir las siguientes inicializaciones para C y C++:

```
char cadena[] ="abc";
```

Y

```
char cadena[] = {'a', 'b', 'c'};
```

Estas dos inicializaciones no son equivalentes. La primera coloca el carácter nulo '\0' en el arreglo después de los caracteres 'a', 'b' y 'c'. La segunda no coloca el '\0' en ninguna parte del arreglo.

Una cadena es una variable, pero también es un arreglo, por lo tanto cada celda posee un índice y podemos utilizarlo como cualquier otro arreglo. Revisemos el siguiente ejemplo:

Nota: Debemos tener cuidado al asignar una cadena a un arreglo de caracteres, ya que el uso del símbolo = es **ilegal** para este caso. Es decir

```
char cadena[20];
cadena ="hola amigos";
```

Traerá un error ya que no será posible asignar cada letra en las celdas del arreglo.

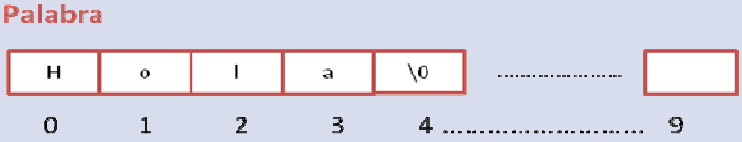
Esto se debe a que cadena es un puntero implícito y recibirá como dato sólo direcciones de memoria. Pero este tema no será abordado en este libro.

Ejemplo 6.8:

Supongamos que nuestro programa (en C y C++) contiene la siguiente declaración de una cadena:

```
char palabra[10]= "Hola";
```

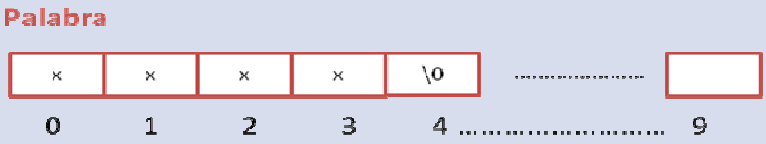
Al declarar palabra como se muestra arriba, nuestro programa tiene las siguientes celdas con datos almacenados: palabra[0], palabra[1], palabra[2], palabra[4], y palabra[5] que almacena '\0'.



Por ejemplo, el siguiente código modificará el valor actual de la cadena palabra para convertirla en una nueva cadena que contiene en todas las celdas ocupadas la letra 'x'.

```
Int índice =0;
while (palabra[indice] != '\0')
  {   palabra[indice] = 'x';
      Índice ++;
```

```
}
```



Java tendría que ser de la siguiente manera, ya que no existe el marcador '\0':

```
import java.io.*;
public class cadenaChar   {
   public static void main (String [] args) {
     char palabra[] = {'h','o','l','a'};
     int i;
                    for (i=0; i<4; i++)
                    palabra[i] = 'x';
        }
}
```

OJO!!! Al manipular las celdas de un arreglo que es una cadena, hay que tener cuidado de no reemplazar el caracter nulo '\0' con cualquier otro valor. Si el arreglo pierde el valor '\0', ya no se comportará como una variable de cadena de tipo char.

6.4.3 Lectura y escritura de cadenas

- **Cadenas tipo *char en C*:**

Primero revisaremos la lectura y escritura en lenguaje C. Para ello utilizaremos el siguiente ejemplo.

Ejemplo 6.9:

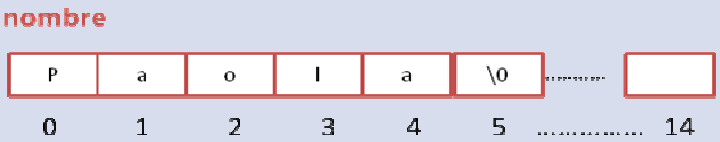
El siguiente programa permite ingresar por teclado en nombre y apellido de una persona.

```
Bibliotecas
main()
{
    char nombre[15], apellido[30];

    printf("Introduce tu nombre: ");
    scanf("%s",nombre);
    printf("Introduce tu apellido: ");
```

```
    scanf("%s",apellido);
    printf("Usted es %s %s\n",nombre,apellido);
}
```

Observa que para leer la cadena no utilizamos el ***&*** como lo hicimos con los datos de tipo numérico. Esto se debe a que un arreglo posee un *puntero implícito* (tema que no abordaremos en este libro) que permite almacenar cada carácter ingresado en cada celda del arreglo.

Entonces si ingresamos "Paola", scanf("%s",nombre) es capaz de leer toda la cadena y asignar las letras al arreglo de la siguiente manera:



Por lo tanto, printf("%s",nombre) permitirá imprimir por pantalla toda la cadena "Paola". Esto es posible gracias al marcador final '\0'. En otras palabras, la propia función printf se encargará de mostrar todas las letras, una al lado de la otra, hasta que se encuentre con '\0'.

OJO!!! La función `scanf` permitirá leer una secuencia de caracteres para almacenarlo en el arreglo. Sin embargo, no permitirá almacenar una frase, como por ejemplo "Hola a todos". Esto se debe a que `scanf` finaliza el ingreso de datos por medio de la tecla Enter, cuando la presiona el usuario del programa, o cuando se ingresa un espacio. Entonces si ingresamos por teclado "Hola a todos", `scanf` sólo almacenará "Hola".

Para solucionar este problema, podemos utilizar la función ***gets*** que también pertenece a la librería estándar de C, stdio.h. *Gets* permit ingresar una frase cualquiera a la cadena. Veamos un ejemplo.

Ejemplo 6.10:

Supongamos que en ejemplo anterior se desea ingresar los dos apellidos de la persona. El siguiente programa utiliza la función gets para ello.

```
Bibliotecas
main()
{
    char nombre[15], apellidos[30];

    printf("Introduce tu nombre: ");
    scanf("%s",nombre);
    printf("Introduce tus apellidos: ");
    gets(apellidos);
    printf("Usted es %s %s\n",nombre,apellidos);
}
```

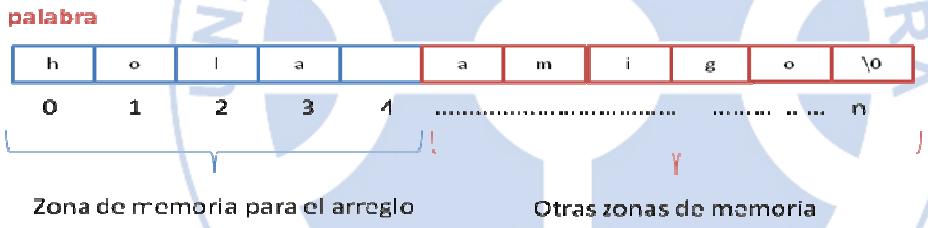
Observa que ingresamos los apellidos utilizando gets(apellidos), de esta manera el programa almacenará dos o más palabras, por ejemplo “Aravena Soto”.

OJO!!! La función gets permitirá ingresar al arreglo todos los caracteres que ingrese el usuario hasta que presione la tecla Enter, esto, sin importar la cantidad de caracteres totales que se escribieron. Entonces, puede ocurrir que se utilice más espacio en memoria de la que se definió originalmente para el arreglo.

Por ejemplo, si tenemos la siguiente declaración para una cadena.

```
char palabra[5];
```

y el usuario ingresa la siguiente frase “hola amigo” por medio de gets(palabra), entonces la cadena se almacenará como se muestra en la figura.



Las celdas azules son las definidas para el arreglo *palabra*, y las celdas rojas son zonas de memoria continuas al arreglo pero que son utilizadas por otros programas o archivos.

Para evitar este problema, podemos utilizar otra función de entrada que también pertenece a la biblioteca estándar de C. Es la función ***fgets***, que permite determinar la cantidad máxima de caracteres que se pueden almacenar en el arreglo. Observa el siguiente ejemplo.

Ejemplo 6.11:

El siguiente programa utiliza la función fgets para ingresar una frase.

```
Bibliotecas
main()
{
    char frase[30];
    printf("Ingrese una frase de hasta 30 caracteres: ");
    fgets(frase, 30, stdin);
    printf("La frase es %s \n",frase);
}
```

En este caso, si el usuario del programa ingresa más de 30 caracteres para la frase, entonces fgets solo almacenará en el arreglo los primeros 30 ingresados.

- **Cadenas tipo *char en C++*:**

Una forma de ingresar cadenas es utilizando la función cin que ya conoces.

Ejemplo 6.12:

El siguiente programa permite ingresar un email y luego valida si el email es correcto. Consideraremos un email correcto cuando contenga un @.

```
Bibliotecas
void main()
{ char email[30];
  int índice=0, bandera=0;
  cout<< "Ingrese un email";
  cin>> email;
   while (email[indice] != '\0')
   {  if (email[índice] == '@')
        { bandera =1;
          break;
         }
      indice++;
   }
   if (bandera)
     cout << "El email es válido";
   else
     cout << "El email no es válido";
   cout<< "El email ingresado fue "<< email;
  }
```

Observa que ingresamos una cadena de caracteres por medio de cin>>email, por lo tanto, la propia función cin se preocupará de almacenar cada caracter en las celdas definidas para el arreglo.

Modifica el programa anterior para que la validación incluya el punto que se ingresa en un email. Considera que tal punto debe ubicase antes de los dos últimos caracteres ingresados. Por ejemplo alex@ufro.cl sería un email válido, en cambio alex.@ufro sería un email no válido.

OJO!!! Al igual que en lenguaje C, la función cin permite ingresar solo palabras a un arreglo y no frases. Esto se debe a que la función finaliza la entrada de datos (por teclado) al presionar la tecla Enter, o al momento de ingresar un espacio.

Por ejemplo, supongamos que se dispone del siguiente arreglo:

```
char frase[30];
```

E ingresamos la cadena “Hola a todos” por medio de la siguiente instrucción.

```
cin>>frase;
```

Ejemplo 6.13:

El siguiente programa utiliza la función get para ingresar una frase.

```
Bibliotecas
void main()
{
    char frase[30];
    cout<<"Ingrese una
frase de hasta 30
caracteres: ";
    cin.get(frase,
30);
    cout<<"La frase es
"<< frase<<endl;
}
```

Ejemplo 6.14:

Supongamos que cadena1, cadena2 y cadena3 son variables de tipo ***string*** y que cadena2 y cadena3 ya tienen almacenado una cadena. Entonces a cadena1 le podemos asignar la concatenación de las cadenas cadena2 y cadena3 de la siguiente manera:

```
cadena1 = cadena2 + cadena3;
```

Lo que recibe frase es solo la palabra "Hola", debido al espacio que hay a continuación.

Para solucionar este problema, se debe utilizar otra función de entrada de datos que pertenece a la biblioteca iostream.h al igual que cin. Es la función get que se utiliza en conjunto con cin. Observemos el siguiente ejemplo.

En este caso, si el usuario del programa ingresa más de 29 caracteres para la frase, entonces get solo almacenará en el arreglo los primeros 29 ingresados y a continuación el caracter nulo '\0'. Observa el ejemplo 6.13.

Nota: La función get también permite determinar la forma cómo se desea finalizar la entrada de datos. Esto se logra utilizando un tercer argumento para get. Por ejemplo, cin.get (frase, 30, 't'), permite finalizar la entrada de datos una vez que el usuario ingrese el caracter 't'.

- **El tipo de datos string de C++:**

El estándar ANSI/ISO más reciente para C++ especifica que este lenguaje debe tener también un tipo de datos **string** que permita al programador tratar las cadenas como un tipo de datos básico, sin necesidad de preocuparse por los detalles de implementación. En esta sección presentamos este tipo **string**.

El tipo **string** se define en la biblioteca cuyo nombre también es **<string>**, y las definiciones se colocan en el espacio de nombres **std**. Por lo tanto, para poder usar el tipo **string** nuestro código debe contener lo siguiente:

```
#include <string>
using namespace std;
```

El tipo ***string*** nos permite tratar valores y expresiones ***string*** en forma muy parecida a los valores de un tipo de datos simple. Podemos utilizar el operador **=** para asignar un valor a una variable de tipo ***string*** y podemos usar el signo **+** para concatenar dos cadenas. Observa el ejemplo 6.14.

Como vimos anteriormente, las cadenas entre comillas son en realidad cadenas de tipo char. Sin embargo, C++ proporciona la conversión de tipo de manera automática, de la cadena entre comillas a tipo string. Por ello, podemos utilizar cadenas entre comillas como si fueran valores literales que se pueden asignar a las variables de tipo string. Observa el ejemplo 6.15.

Ejemplo 6.15:

El siguiente código fuente permite crear una variable de tipo string y luego asignarle una cadena.

```
#include <iostream>
#include <string>
using namespace std;
void main()
{ string cadena;
cadena =”hola mundo”;
cout << cadena;
}
```

Nota: En las variables de tipo ***string*** podemos asignar una cadena cualquiera. La variable almacena todos los caracteres de dicha cadena sin incluir el carácter nulo ‘\0’ como en los arreglos de tipo char.

Entonces, ¿cómo sabremos cuantos caracteres posee una cadena determinada? La respuesta está en la siguiente sección al utilizar algunas funciones de la ***biblioteca string***.

Para ingresar una cadena a una variable bastará con utilizar la función de entrada ***cin*** tal y como lo hemos realizado con la variables simples. Veamos el ejemplo 6.16.

Ejemplo 6.16:

El siguiente programa permite leer una palabra desde el teclado y luego la imprime por pantalla.

```
#include <iostream>
#include <string>
using namespace std;
void main()
{ string palabra;
cout<<”Ingrese una palabra”;
cin>>palabra;
cout << "Esta es la palabra ingresada "<< palabra << endl;
}
```

Ejecuta el programa ingresando algunas palabras como “hola”, “amigo”, “programación”, etc. Pero qué pasará si introducimos la frase “Amigo mío, tanto tiempo sin verte!”.

En este caso, el operador de extracción >> y ***cin*** funcionan de la misma manera para las variables de tipo ***string*** que para los arreglos de tipo ***char***. Pero hay que recordar que ***cin*** ignora los espacios, así que para la frase señalada la cadena sólo almacenará la palabra “Amigo”.

Si deseamos ingresar una frase a una variable de tipo ***string*** debemos utilizar la función ***getline***. La forma cómo utilizar esta función es un poco distinta que cuando utilizábamos ***cin.get***. Revisemos el ejemplo 6.17.

Ejemplo 6.17:

El siguiente programa permite ingresar a una frase a una variable de tipo string.

```
#include <iostream>
#include <string>
using namespace std;
void main()
{ string frase;
cout<<”Ingrese una frase”;
getline(cin, frase);
cout << “Esta es la frase ingresada “<< frase << endl;
}
```

OJO!!! No podemos utilizar **>>** con el operador ***cin*** para ingresar un carácter en blanco. Si deseamos ingresar un carácter a la vez podemos utilizar ***cin.get***, que vimos anteriormente. La función ***cin.get*** lee valores de tipo ***char***, no se tipo ***string***, pero puede ser útil cuando manejamos entradas de tipo ***string***.

Ejemplo 6.18:

El siguiente programa muestra un menú con varias opciones, “1. Ingrese sólo su nombre”, 2. Ingrese su nombre y apellido”.

```
#include <iostream>
#include <string>
using namespace std;
void main()
```

```
{ string nombre;
char opcion;
cout<< "MENU"<< endl;
cout<< "1. Ingrese su nombre"<<endl;
cout<< "2. Ingrese su nombre y apellido"<< endl;
cin.get(opcion);
if (opcion == '1')
    { cout<< "Ingrese su nombre:";
     cin>> nombre;
    }
else if (opcion =='2')
    { cout<< "Ingrese su nombre y apellido:";
     getline(cin, nombre);
     }
cout << "Su nombre es "<< nombre << endl;
}
```

Nota: Podemos acceder a los caracteres de una variable de tipo ***string*** tal y como lo hicimos con los arreglos de tipo ***char***, por lo que las variables de tipo ***string*** poseen todas las ventajas de los arreglos de caracteres, además de otras ventajas que los arreglos de caracteres no tienen.

Ejemplo 6.19:

El siguiente programa permite ingresar una frase y luego se mostrará carácter a carácter.

```
#include <iostream>
#include <string>
using namespace std;
void main()
{ string frase; int i;
cout<<"Ingrese una frase";
getline(cin, frase);
for (i=0; i <frase.length(); i++) //*length es una función de la biblioteca
string y que entrega la cantidad de caracteres de frase*//
    cout << frase[i]<<"  ";
}
```

Observamos que frase[i] es el elemento i de la cadena completa, por tanto podemos acceder a cada carácter al igual que los arreglos de tipo char.

- **La clase string de Java:**

En Java las cadenas se tratan de forma diferente a C y C++. En Java las cadenas son objetos de las clases predefinida ***String*** o ***StringBuffer***, que están incluidas en el paquete ***java.lang.****

Esto quiere decir que siempre que aparecen conjuntos de caracteres entre comillas dobles, el compilador de Java crea automáticamente un objeto *String*. El compilador es más eficiente y usa un objeto *StringBuffer* para construir cadenas a partir de las expresiones, creando el *String* final sólo cuando es necesario.

Los caracteres de las cadenas tienen un índice que indica su posición. El primer carácter de una cadena tiene el índice 0, el segundo el 1, el tercero el 2 y así sucesivamente; tal y cómo lo hemos visto con los arreglos numéricos, sólo que en esta caso cada celda almacena sólo un caracter.

Los strings u objetos de la clase **String** se pueden crear explícitamente o implícitamente. Para crear un string implícitamente basta poner una cadena de caracteres entre comillas dobles. Por ejemplo, cuando se escribe

```
System.out.println("El primer programa");
```

En este caso, Java crea un objeto de la clase ***String*** automáticamente.

Para crear un string explícitamente escribimos

```
String frase=new String("El primer programa");
```

También se puede escribir de manera alternativa

```
String frase="El primer programa";
```

Nota: También podemos asignar una cadena de manera directa a la variable, tal y como lo hacemos con las variables comunes (float, int, double, etc.)

```
String frase;

frase="El primer programa";
```

Para crear un string nulo se puede hacer de estas dos formas

String frase="";

String frase=new String();

Un string nulo es aquél que no contiene caracteres, pero es un objeto de la clase *String*. Sin embargo,

String str;

está declarando un objeto ***frase*** de la clase ***String***, pero aún no se ha creado ningún objeto de esta clase, es decir, no se ha construido el objeto en la memoria.

Para ingresar una cadena en Java desde el Buffer, debemos utilizar el método readLine(). Observemos el siguiente ejemplo escrito en JCreator.

Ejemplo 6.20:

El siguiente programa permite al usuario ingresar una cadena desde el Buffer y luego la muestra por pantalla.

```
import java.io.*;

public class CadenaBuffer    {
    public static void main (String [] args) {
       String cadena;
       BufferedReader In =new BufferedReader (new InputStreamReader(System.in));

 try{
          System.out.print(" Ingrese una cadena:");
          cadena = In.readLine();
          System.out.println(" Su cadena es " + cadena);
 }
 catch (Exception e){
     e.printStackTrace();
 }

 }
}
```

Nota: Observemos que al leer una cadena desde el Buffer no debemos utilizar alguna función de conversión de datos como lo hacíamos con las variables de tipo int, float, double, etc. Esto se debe a que el Buffer siempre contendrá datos de tipo alfanuméricos.

Nota: Podemos acceder a los caracteres de una variable de tipo ***string*** tal y como lo hicimos con los arreglos de tipo ***char***, por lo que las variables de tipo ***string*** poseen todas las ventajas de los arreglos de caracteres, además de otras ventajas que los arreglos de caracteres no tienen.

Ejemplo 6.21:

El siguiente programa permite ingresar una frase y luego se mostrará carácter a carácter.

```
import java.io.*;

public class MostrarCarac    {
    public static void main (String [] args) {
      BufferedReader In =new BufferedReader (new InputStreamReader(System.in));
      String cadena;
      int i;

 try{
          System.out.print(" Ingrese una cadena:");
          cadena = In.readLine();
          for (i=0; i < cadena.length(); i++)
             System.out.println(" carater " + i + " es " + cadena.charAt(i));
 }
 catch (Exception e){
     e.printStackTrace();
 }

 }
}
```

Length() y charAt(i) son métodos (funciones) que describiremos en la siguiente sección.

Observamos que cadena.charAt(i) es el elemento i de la cadena completa, por tanto podemos acceder a cada carácter al igual que los arreglos de tipo char. Utilizamos el método charAt(i) ya que en Java no se permite el uso de cadena[i] como en el caso de los arreglos de tipo char.

6.4.4 Funciones propias para cadenas

Las bibliotecas nos aportan funciones para trabajar sobre las cadenas, y saber por ejemplos, la cantidad total de caracteres ingresados por un usuario, la cantidad de vocales que tiene una frase, entre otras que veremos a continuación.

Ejemplo 6.22:

El siguiente programa permite al usuario ingresar una palabra, luego muestra la palabra de manera inversa.

```
# include <stdio.h>
# include <string.h>
#include <stdlib.h>
int main()
{char palabra[15]="Hola";
 int  i;
for (i= strlen(palabra) - 1 ; i>=0; i--)
printf("%c", palabra[i]);
system("pause");
return 0;
}
```

Ejemplo 6.23:

El siguiente programa copia una cadena en otra cadena

```
# include <stdio.h>
# include <string.h>
#include <stdlib.h>
int main()
{ char cadena1[30]="hola amigo", cadena2[30];
strcpy (cadena1, cadena2);
printf ("la cadena2 contiene %s ", cadena2);
system("pause");
return 0;
}
```

En los lenguajes de C, C++ y Java, las funciones no cambian en gran cantidad, en lo único que quizás difieren es en la escritura. Revisaremos algunas funciones para los tres lenguajes de programación.

- **Funciones para C:**

La tabla 6.1 presenta algunas de las funciones que se pueden utilizar con arreglos de caracteres para el lenguaje C.

Tabla 6.1: Ejemplo de Funciones para manipulación de cadenas en C

Funciones	Descripción
strcat(destino, original)	Concatena la cadena original en la cadena de destino
strcmp(c1,c2)	Compara las cadenas c1 y c2
strcpy(c1,c2)	Copia los caracteres de la cadena c1 en la cadena c2
strlen(c1)	Cuenta cuantos caracteres tiene una cadena dada

Los ejemplos 6.22 y 6.23 utilizan estas funciones.

Ahora te mostramos cómo podemos comparar dos cadenas, ya que la comparación

```
if (cadena1 == cadena2)
```

siempre devolverá falso. Esto se debe a que **cadena1 == cadena2** compara los punteros implícitos que contienen que tienen asociados ambas cadenas, es decir, las posiciones de memoria en donde se inicia cada cadena; y como ambas son variables diferentes, entonces siempre se crearán en zonas de memorias distintas. Para solucionar este problema debemos utilizar la función strcmp(c1, c2) de la bibliteca string.h, que permite comparar dos cadenas si son iguales.

Ejemplo 6.24:

El programa a continuación permite comparar dos cadenas

```
# include <stdio.h>
# include <string.h>
#include <stdlib.h>
int main()
{char cadena1[30]="hola amigo", cadena2[30]="hola amigo";
If (strcmp (cadena1, cadena2) == 0)
  printf ("las cadenas son iguales ");
else
   printf ("las cadenas son distintas ");
system("pause");
return 0;
}
```

La función ***strcmp(c1, c2)*** devolverá 0 (cero) en caso de que c1 sea igual a c2, devolverá -1 cuando c1 sea menor que c1, y devolverá 1 cuando c1 sea mayor que c2. Observa el ejemplo 6.24.

Nota: Es posible comparar dos cadenas, o también caracteres, para averiguar si una cadena es menor que la otra. Suena raro para nosotros, pero en realidad lo que sucede es que simplemente para el computador la letra 'a' es menor que la letra 'b' ya que 'a' está antes que 'b' en el código ASSCII.

Así también sabe que el número 1 es menor que el número 2, ya que 1 está antes que 2 en el mismo código ASSCII.

- **Funciones para C++:**

En C++ podemos utilizar las mismas funciones que nos ofrece la biblioteca string.h de lenguaje C. Sin embargo, y a modo de ejemplo, en este capítulo utilizaremos las funciones de la biblioteca String que nos ofrece C++. La tabla 6.2 presenta algunas funciones que nos permitirán manipular cadenas en el lenguaje C++

Tabla 6.2: Ejemplo de Funciones para manipulación de cadenas en C++

Funciones	Descripción
c1.emtpy()	Devuelve true si c1 es una cadena vacía
c1.insert(pos, c2)	Inserta c2 en c1, empezando en la posición pos
c1.substr(posición, longitud)	Devuelve la subcadena de c1, comenzando desde *posición* y con *longitud* caracteres.
c1.remove(pos, longitud)	Elimina la subcadena de tamaño *longitud*, comenzando desde la posición *pos*
c1.length	Cuenta cuantos caracteres tiene una cadena dada
c1.at(i)	Devuelve una referencia de lectura/escritura al carácter i almacenado en c1. Al igual que hacemos con c1[i], pero esta versión comprueba índices ilegales

Ejemplo 6.25:

El siguiente programa permite al usuario ingresar una palabra, luego muestra la palabra de manera inversa.
En este caso, debemos utilizar la función length() que devuelve la cantidad de caracteres de una cadena. Esto par conocer la ubicación del último carácter asignado o ingresado.

```
#include <iostream>
#include <string>
using namespace std;
int main()
{string palabra="Hola";
 int  i;
for (i= palabra.length() -1 ;
i>=0; i--)
/*length() devolverá la cantidad
de caracteres de palabra */
   cout<< palabra[i];
system("pause");
return 0;
}
```

Recordemos que a diferencia de lenguaje C, el tipo string de C++ nos permite realizar acciones con las cadenas como si éstas fueran variables simples. Por ejemplo

```
cadena1 = cadena2
```

asigna el contenido de cadena2 en cadena1, y

```
cadena1 == cadena2
```

permite comparar si dos cadenas son iguales. Así también podemos verificar si una cadena es menor que otra, como en el caso de

```
cadena1 < cadena2
```

que compara si la cadena1 es menor que la cadena2.

Ejemplo 6.26:

El siguiente programa copia una cadena en otra cadena.

```
#include <iostream>
#include <string>
using namespace std;
int main()
{  string cadena1=”hola amigo”,
cadena2;
cadena2.insert(0, cadena1);

//inserta cadena1 en cadena2 a
partir de la posición 0
cout << "la cadena2 contiene
"<< cadena2;
system("pause");
return 0;
}
```

Nota: Es posible comparar dos cadenas, o también caracteres, para averiguar si una cadena es menor que la otra. Suena raro para nosotros, pero en realidad lo que sucede es que simplemente para el computador la letra

La tabla 6.2 presenta algunas de las funciones de la biblioteca String que nos permite manipular cadenas. Los ejemplos 6.25, 6.26 y 6.27 utilizan estas funciones.

Ejemplo 6.27:

El programa a continuación muestra parte de la cadena ingresada por el usuario.

```
#include <iostream>
#include <string>
using namespace std;
int main()
{ string cadena;
  cout<< "Ingrese una cadena";
  getline(cin, cadena);
  cout << "una subcadena de  "<< cadena << " es " << cadena.substr(5, 10);
// cadena.substr(5,10) muestra 10 caracteres a partir de la posición 5
system("pause");
return 0;
}
```

- **Métodos (funciones) para Java:**

Al igual que en C y C++, en Java existen métodos (o funciones) que permiten manipular cadenas. La tabla 6.3 presenta algunos ejemplos.

Tabla 6.3: Ejemplo de Métodos para manipulación de cadenas en Java

Funciones	Descripción
c1.concat(c2)	Concatena las cadenas c1 y c2
c1.equals(c2)	Devuelve true en caso de que las cadenas c1 y c2 son iguales
c1.equalsIgnoreCase(c2)	Devuelve true en caso de que las cadenas c1 y c2 son iguales, no importando si están escritas en minúsculas o mayúsculas
c1.trim()	Elimina los espacios del principio y fin de la cadena
c1.length()	Devuelve la cantidad de caracteres que hay en c1

Observemos los siguientes ejemplos que utilizan algunas de estos métodos.

Ejemplo 6.28:

El siguiente programa permite al usuario ingresar una cadena y luego la muestra al revés. Para ello utilizaremos el método ***length***() que devuelve la cantidad de caracteres de una cadena, así sabremos cuál es la última posición de la misma.

```
import java.io.*;

public class MostrarCaracReves    {
    public static void main (String [] args) {
      BufferedReader In =new BufferedReader (new InputStreamReader(System.in));
      String cadena;
      int i;

 try{
          System.out.print(" Ingrese una cadena:");
          cadena = In.readLine();
          for (i=cadena.length()-1; i>=0; i--)
             System.out.print(cadena.charAt(i));
 }
 catch (Exception e){
     e.printStackTrace();
 }

 }
}
```

Ejemplo 6.29:

El próximo programa compara dos cadenas ingresadas por el usuario. Para ello, utilizaremos el método ***equals()*** que devuelve verdadero si dos cadenas son iguales.

```
 import java.io.*;

 public class comparaCadenas    {
     public static void main (String [] args) {
       BufferedReader In =new BufferedReader (new InputStreamReader(System.in));
       String cadena1, cadena2;

  try{
          System.out.print(" Ingrese una cadena:");
          cadena1 = In.readLine();
          System.out.print(" Ingrese otra cadena:");
          cadena2 = In.readLine();
          if (cadena1.equals(cadena2))
             System.out.println("las cadenas son iguales");
          else
             System.out.println("las cadenas son distintas");
  }
  catch (Exception e){
      e.printStackTrace();
  }

  }
 }
```

6.5 Ejercicios Resueltos

Ejercicio 6.1:

Cree un programa que genere 20 números de Fibonacci. Para ello, debe almacenar los números en un arreglo.

Los números de Fibonacci se generan en forma sucesiva de acuerdo a la siguiente regla:

Fibo(1) = 1

Fibo(2) = 1

Fibo(3) = Fibo(1) + Fibo(2)

Fibo(4) = Fibo(2) + Fibo(3)

.....

Fibo(n) = Fibo(n-2) + Fibo(n-1)

Por lo tanto, necesitamos un arreglo de 20 celdas de tipo entero para almacenar dichos números. Las dos primeras celdas se inicializan en 1, y luego en las otras celdas (celda i) se almacena la suma de los números ubicados en las dos celdas anteriores, es decir, la celda i-1 y la celda i-2.

Lenguaje C:

```
# include <stdio.h>
#include <stdlib.h>
#define n 20
int main()
{  int fibonacci[20];
   int  i;
//primero inicializamos las dos primeras celdas con 1
fibonacci[0]=1;
fibonacci[1]=1;

// luego completamos las otras celdas
for (i= 2 ; i<n; i++)
     fibonacci[i]= fibonacci[i-2] + Fibonacci[i-1];

//finalmente mostramos los números por pantalla
for (i= 0 ; i<n; i++)
    printf (“fibonacci %d es %d \n”, i , Fibonacci[i]);
system("pause");
return 0;
}
```

Lenguaje C++:

```
#include <iostream>
```

```
#define n 20

using namespace std;
int main()
{  int fibonacci[20];
   int  i;
//primero inicializamos las dos primeras celdas con 1
fibonacci[0]=1;
fibonacci[1]=1;

// luego completamos las otras celdas
for (i= 2 ; i<n; i++)
     fibonacci[i]= fibonacci[i-2] + fibonacci[i-1];

// finalmente mostramos los números por pantalla
for (i= 0 ; i<n; i++)
    cout <<"Fibonacci " << i << "es "<< fibonacci[i] << endl;
system("pause");
return 0;
}
```

Lenguaje Java:

```
import java.io.*;

  public class Fibonacci   {
     public static void main (String [] args) {
       int fibonacci[] = new int[20];
       int i;

          // primero inicilaizamos las dos primeras celdas con 1
          fibonacci[0]= 1;
          fibonacci[1]= 1;

         // luego le asignamos los números a las otras celdas que quedan
          for (i=2; i<20; i++)
             fibonacci[i] = fibonacci[i-2] + fibonacci[i-1];

          // finalmente mostramos los números por pantalla
          for (i=2; i<20; i++)
              System.out.println(" Fibonacci  " + i + " es " + fibonacci[i]);
          }
 }
```

Ejercicio 6.2:

Una persona posee una cuenta de ahorro en el banco "Ahorra Feliz". Para mantener información detallada de su cuenta le solicita a usted crear un programa en C++ que permita por medio de **un arreglo**:

a) Ingresar los abonos mensuales a la cuenta de ahorro durante el año.
b) Mostrar el mes en que se ingresó el mayor abono a la cuenta.
c) Mostrar el promedio de su cuenta de ahorro en el año.
d) Mostrar la varianza de la cuenta de ahorro.
e) Mostrar la desviación estándar de su cuenta de ahorro anual.

Algunos antecedentes para calcular el **Promedio, Varianza y Desviación estándar**:

Esta medida nos permite determinar el promedio aritmético de fluctuación de los datos respecto a su punto central o media. La desviación estándar nos da como resultado un valor numérico que representa el promedio de diferencia que hay entre los datos y la media. Para calcular la desviación estándar basta con hallar la raíz cuadrada de la varianza, por lo tanto su ecuación sería: $S = \sqrt{S^2}$

Para comprender el concepto de las medidas de distribución vamos a suponer que la persona que solicita el programa desea saber qué tanto varían los abonos mensuales en su cuenta de ahorro. Como ejemplo, supongamos que los abonos en dólares para los primeros 5 meses son los que se muestran en la siguiente fórmula que calcula el promedio:

$$\overline{X} = \frac{490+500+510+515+520}{5} = \frac{2535}{5} = 507$$

La varianza sería:

$$S_X^2 = \frac{\sum_{i=1}^{n} \left(X_i - \overline{X}\right)^2}{n-1}$$

$$S^2 = \frac{(490-507)^2+(500-507)^2+(510-507)^2+(515-507)^2+(520-507)^2}{(5-1)}$$

$$S^2 = \frac{(-17)^2+(-7)^2+(3)^2+(8)^2+(13)^2}{4} = \frac{289+49+9+64+169}{4} = \frac{580}{4} = 145$$

Por lo tanto la desviación estándar de los 5 primeros abonos sería: $S = \sqrt{145} = 12.04 \cong 12$

Con lo que concluiríamos que el abono promedio de los 5 primeros meses es de 507 dólares, con una tendencia a variar por debajo o por encima de dicho abono en 12 dólares.

Una solución general del problema sería el siguiente:

```
Bibliotecas
Encabezado programa (puede ser en C , C++ o Java)
  int  ahorro[N];
  float prom, vari;
  int i, mayor, mes;

 // primero ingresamos el ahorro por teclado
 //así sería en C ++
// tú puedes modificarlo para C o para Java
  for (i=0; i<N; i++)
 {  cout<< "Ingrese abono para el mes "<< i+1 << endl;
    cin >> cuenta[i];
 }

// luego podemos encontrar el mes en que se ingresó el mayor ahorro
mayor = cuenta[0];
mes=0;
 for (i=1; i<N; i++)
   if (cuenta[i] > mayor)
     { mayor = cuenta[i];
       mes =i;
   }
  cout << "el mes en que se ingresó el mayor ahorro es " << mes;

//luego calculamos el promedio
  for (i=1; i<N; i++)
       suma = suma + cuenta[i];
 prom = (float) suma /N;
 cout << "el promedio de dinero en el cuenta de ahorro para el año  es "<< prom;

//calculamos la varianza
 for (i=0; i<N; i++)
     suma = suma + (cuenta[i]-(int) prom)* (cuenta[i]-(int) prom);
//o bien   suma = suma + (cuenta[i]-prom)* (cuenta[i]-prom)
 vari= (float) suma /N-1;
 cout << "la varianza es "<< vari;

//finalmente la desviación estándar
cout << "y la desviación estándar es "<< sqrt(vari);
```

Ejercicio 6.3:

Una empresa de servicios desea almacenar la cantidad de reclamos que realizan los clientes durante una semana. Para ello dispone de una tabla de valores como la que sigue:

Ejemplo de tabal de reclamos:

	Lunes	Martes	Miércoles	Jueves	Viernes	Sábado	Domingo
10:00-13:00	4	6	2	34			
13:00 – 15:00		10		7			
15:00-20:00			29				

La cantidad de reclamos son almacenados por día de la semana de acuerdo a la jornada laboral. Por ejemplo, para el día martes en la jornada laboral de 10:00 a 13:00 hrs se registraron 6 reclamos.
Con estos datos, el gerente general requiere que cree un programa que pueda realizar las siguientes actividades:

a) Ingresar la cantidad de reclamos por día y jornada laboral.
b) Mostrar el promedio de reclamos para la jornada laboral de 15:00 a 20:00 hrs.
c) Mostrar el día de la semana que tuvo más reclamos.

Una solución en lenguaje C++ sería la siguiente:

```
Bibliotecas
#define jornadas 3
#define días 7

int main()
{ int reclamos[jornadas][días];
 int i, j, suma, mayor, dia;

// primero ingresamos los reclamos por teclado
 for (j=0; j < días; j++)
  {  cout << "Ingrese los reclamos para el día "<< j+1<< endl;
    for (i=1; i< jornadas; i++)
     { cout << "Ingrese el reclamo para la jornada " << i+1 << endl;
       cin>> reclamos[i][j];
     }
  }

// luego calculamos el promedio de reclamos para la jornada de 15:00 a 20:00 hrs.
// es decir la jornada número 2
for (i=0; i< días; i++)
    suma =suma + reclamos[2][i];  //jornada =2
 cout << "El promedio de reclamos para la jornada de 15:00 a 20:00 hrs es"<< (float) suma/días;

//finalmente calculamos el día de la semana que tuvo más reclamos
for (j= 1; j <dias; j++)
{  suma =0;
   for (i= 0; i< jornadas; i++)
```

```
        suma= suma + reclamos[i][j];
   if (mayor < suma)
     { mayor = suma;
       dia = j;
     }
}
cout << "El dia de la semana que tuvo más reclamos fue "<< dia;
return 0;
}
```

6.6 Ejercicios Propuestos

Ejercicio 6.4:

Cree un programa, que permita a un usuario realizar las siguientes acciones con una matriz de 10 x 10

a). Ingresar sólo números positivos entre 0 y 250
b). Calcular el promedio de los números para una fila que ingresa el usuario
c). Sumar los números de la diagonal.

Ejercicio 6.5:

Un banco registra el saldo mensual de la cuenta corriente de un cliente en un arreglo. Se pide crear un programa para realizar las siguientes acciones:

a) Ingresar los saldos por mes
b) Mostrar los meses en que el saldo es negativo
c) Mostrar el mes en que se tiene el mayor saldo
d) Calcular un promedio de saldos para el año

Ejercicio 6.6:

En un arreglo de 30 celdas se almacena las ventas diarias logradas por un vendedor de artículos de cocina durante el mes de abril. La empresa requiere un programa en C++ que permita responder a los siguientes requerimientos por medio de funciones:

a) Ingresar las ventas diarias del vendedor
b) Mostrar el día del mes en que logró la mayor venta
c) Calcular el total de ventas del mes
d) Mostrar los días del mes en el que se logró ventas inferiores a los $10.000

e) Calcular la remuneración del empleado para el mes de abril si se sabe que se le paga una comisión del 1% de las ventas totales logradas.

Ejercicio 6.7:

Se ingresan a un arreglo las ventas diarias (total $ de cada día) que realiza un vendedor de una tienda de CDs, se pide crear un programa que permita:

a) Ingresar las ventas diarias del vendedor realizadas durante un mes (30 días) al arreglo. (Nota: recuerde validar que los números ingresados sean mayores o iguales a cero)
b) Calcular el promedio de ventas logradas durante los primeros 15 días del mes.
c) Mostrar las ventas diarias mayores a $345.000
d) Mostrar el día en que logró la mayor venta.

Ejercicio 6.8:

Un método clásico para identificar los números primos existentes en una secuencia de números que va desde 2 a N es la llamada Criba de Eratóstenes.

El algoritmo usado para este propósito va marcando (eliminando) todos los múltiplos de 2, 3, 4, 5 y así sucesivamente hasta que se encuentre el primer número no marcado que es mayor que la raíz cuadrada de N.

Ejemplo: suponga una secuencia de números desde 2 a 20.

a) Se toma el número 2, dado que es el primer número no marcado.
b) Marcar todos los múltiplos de 2 a partir de 2^2 .
c) Se toma al 3, dado que es el siguiente número no marcado.
d) Marcar todos los múltiplos de 3 a partir de 3^2.
e) Se toma el número 5 como el siguiente número. Pero 5^2 es mayor que 20. Entonces el algoritmo aquí se detiene y todos los números no marcados son los números primos.

Gráficamente : (E significa eliminado (marcado))

a)

2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

b)

2	3	E	5	E	7	E	9	E	11	E	13	E	15	E	17	E	19	E

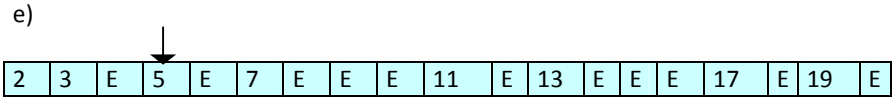
c)

2	3	E	5	E	7	E	9	E	11	E	13	E	15	E	17	E	19	E

d)

2	3	E	5	E	7	E	E	E	11	E	13	E	E	E	17	E	19	E

e)



Escriba un programa que permita solucionar el problema, para una secuencia que va desde 2 hasta cualquier N.

Ejercicio 6.9:

Dado un arreglo llamado PROM, mantiene los promedios (valores reales), de un curso que posee N alumnos, escriba un programa que entregue:

a) El promedio de las notas
b) El mayor y el menor promedio
c) La cantidad de promedios en [4.5 - 6.0]

Ejercicio 6.10:

Dado un mensaje se debe calcular su costo para enviarlo por telégrafo. Para esto se sabe que las letras cuestan, cada una, $10. Lo caracteres especiales que no sean letras cuestan $30 y los dígitos tienen un valor de $20 cada uno. Los espacios no tienen valor.

Restricciones:

- El mensaje es una cadena
- Las letras ñ,á,é,í,ó,ú se consideran caracteres especiales.

Un ejemplo de ejecución del programa es:

Entrada: Feliz cumpleaños
Salida: Valor del mensaje $ 170

Creer un programa que permita calcular el costo del mensaje dado un texto ingresado por el usuario.

6.7 Comentarios Finales

En este capítulo te hemos presentado los fundamentos de la definición y uso de arreglos en los lenguajes C, C++ y Java.

En este capítulo, los arreglos los hemos clasificado en arreglos unidimensionales, arreglos multidimensionales y cadenas. Para cada caso, hemos descrito las principales características con ejemplos básicos que te ayudan a comprender de mejor manera estos elementos.

Finalmente, te hemos entregado un set de ejercicios resueltos con el fin de reforzar el desarrollo de programas que utilizan arreglos. También hemos incorporado un set de ejercicios propuestos con el fin de que practiques y comentes las dudas con tu profesor y compañeros.