Metodología de la Programación

Tema 1. Arrays, cadenas estilo C y matrices

Departamento de Ciencias de la Computación e I.A.

Curso 2015-16

Índice

1. Motivación
2. Objetivos
3. Arrays de bajo nivel
4. Funciones y arrays de bajo nivel
5. Cadenas de caracteres estilo C
6. Matrices
7. Resumen

Motivación

Objetivo: conocer formas de almacenar colecciones de elementos relacionados entre sí (enteros, dobles, objetos, ...)

Puede ser cualquier tipo de relación lógica: conjuntos de notas, grupo de objetos de clase Estudiante, ...

Motivación

¿Podemos resolver esto con variables simples? Imaginemos que debemos procesar las notas de 500 alumnos... ¿Cómo se gestiona?

Índice

1. Motivación
2. Objetivos
3. Arrays de bajo nivel
4. Funciones y arrays de bajo nivel
5. Cadenas de caracteres estilo C
6. Matrices
7. Resumen

Objetivos

Objetivos:

- conocer los mecanismos disponibles en C++ para almacenar colecciones de valores de forma secuencial (no obligatoriamente como datos miembro de una clase)
- conocer los conceptos básicos del uso de cadenas de caracteres estilo C (por compatibilidad entre C y C++).
- comprender las matrices y saber realizar operaciones sobre ellas

Importante: comprensión de los ejemplos de código y realización de los ejercicios que se vayan indicando

Objetivos

Formas manejar colecciones en C++:

- arrays de bajo nivel
- clase vector
- clase array

Índice

1. Motivación
2. Objetivos
3. Arrays de bajo nivel
4. Funciones y arrays de bajo nivel
5. Cadenas de caracteres estilo C
6. Matrices
7. Resumen

Arrays de bajo nivel

Colección con número fijo de componentes (elementos), todos ellos del mismo tipo. Cada una de las componentes es accesible mediante un índice.

Los elementos necesarios para la declaración son:

- tipo de datos
- nombre de la colección
- máxima capacidad

Arrays de bajo nivel

Ejemplos:

```
double notas[500];

Estudiante listado[50];
```

Arrays de bajo nivel

```
1  int main(){
2      const int NUM_REACTORES=20;
3      const int TOTAL_ALUMNOS = 100;
4      int longitud=50;
5
6      double notas[TOTAL_ALUMNOS];
7      int TemperaturasReactores[NUM_REACTORES];
8      // Evitar la creacion de arrays asi
9      bool casados[40];
10     char  NIF[9];
11     ..........................
12 }
```

Arrays de bajo nivel

A tener en cuenta:

- número de componentes debe fijarse al crear la colección (y no puede modificarse posteriormente)
- tamaño especificado mediante literales o constantes (lo más adecuado), pero no mediante variables

Arrays de bajo nivel

Consideraciones:

- las posiciones de memoria son sucesivas
- los elementos se acceden mediante un índice

Arrays de bajo nivel I

```cpp
#include <iostream>
using namespace std;
int main(){
   // Se define constante para el numero de notas a leer
   const int NUM_NOTAS = 5;

   // Declaracion del array e inicializacion de valor de media
   double notas[NUM_NOTAS], media=0;

   // Bucle de lectura
   for (int i=0; i<NUM_NOTAS; i++){
      cout << "Nota del alumno " << i << ": ";
      cin >> notas[i];
   }

   // Bucle de suma
   for (int i=0; i<NUM_NOTAS; i++){
      media = media+notas[i];
   }

   // Calculo de la media
   media = media/NUM_NOTAS;

   // Se muestran los resultados
   cout << "\nMedia = " << media << endl;
}
```

Arrays de bajo nivel

A tener en cuenta:

- ventaja de usar constantes en la declaración
- diferencia entre capacidad máxima y espacio realmente usado (número de elementos almacenados)

Arrays de bajo nivel: control posiciones usadas I

```
1  #include<iostream>
2  using namespace std;
3
4  int main(){
5     // Maximo numero de notas a manejar
6     const int DIM_NOTAS = 100;
7
8     // Array de almacenamiento de notas
9     double notas[DIM_NOTAS];
10
11    // Indica posiciones usadas del array
12    int util_notas;
13
14    // Variable para calculo de la media
15    double media=0;
16
17    // Bucle de lectura de numero de alumnos: no puede ser negativo
18    // ni exceder la capacidad del array
19    do{
20      cout<<"Introduzca num. alumnos (entre 1 y " << DIM_NOTAS << "): ";
21      cin >> util_notas;
22    }while (util_notas < 1 || util_notas > DIM_NOTAS);
23
24
25
26
```

Arrays de bajo nivel: control posiciones usadas II

```
27      // Bucle de lectura de las notas
28      for (int i=0; i < util_notas; i++){
29        cout << "nota[" << i << "]: ";
30        cin >> notas[i];
31      }
32
33      // Bucle de calculo de la media
34      for (int i=0; i < util_notas; i++){
35        media += notas[i];
36      }
37
38      // Calculo de la media
39      media /= util_notas;
40      cout << "\nMedia: " << media << endl;
41  }
```

Arrays de bajo nivel: control posiciones usadas I

Otra variante: controlar el número de posiciones ocupadas mediante una marca final (centinela) que indica el final de los datos de interés

Arrays de bajo nivel: centinela I

```
1  #include<iostream>
2  using namespace std;
3
4  int main(){
5     // Maximo numero de notas
6     const int DIM_NOTAS = 10;
7
8     // Declracion del array y de variable para calculo
9     // de media
10    double notas[DIM_NOTAS], media=0;
11    int i=0;
12
13    // Se leen las notas hasta introducir centinela
14    do{
15      cout << "nota[" << i << "]: (-1 para terminar): ";
16      cin >> notas[i];
17      i++;
18    }while(notas[i-1] != -1 && i < DIM_NOTAS);
19
20    //Aseguramos valor -1 ultima posicion
21    if (i == DIM_NOTAS){
22      notas[i-1] = -1;
23    }
24
25
26
```

Arrays de bajo nivel: centinela II

```
27      // Se calcula la media
28      for (i=0; notas[i] != -1; i++){
29          media += notas[i];
30      }
31
32      // Presentacion de resultados
33      if (i == 0){
34          cout << "No se introdujo ninguna nota\n";
35      }
36      else{
37          media /= i;
38          cout << "\nMedia: " << media << endl;
39      }
40  }
```

Índice

1. Motivación
2. Objetivos
3. Arrays de bajo nivel
4. Funciones y arrays de bajo nivel
5. Cadenas de caracteres estilo C
6. Matrices
7. Resumen

Funciones y arrays de bajo nivel

Consideramos ahora aspectos relacionados con el paso de arrays de bajo nivel como argumentos. A tener en cuenta:

- las funciones (métodos) son esenciales para descomponer el problema
- los tipos de parámetro formal y actual deben coincidir (no basta con que sean compatibles)

Funciones y arrays de bajo nivel: impresión array I

```
1  #include <iostream>
2  using namespace std;
3
4  // Funcion para impresion de array de 5 caracteres
5  void imprime_array (char v[5]){
6     for (int i=0; i < 5; i++)
7        cout << v[i] << " ";
8  }
9
10 // Main para probar la funcion indicada
11 int main(){
12    char vocales[5]={'a','e','i','o','u'};
13    imprime_array(vocales);
14 }
```

Funciones y arrays de bajo nivel: impresión array I

Problema: necesidad de método específico para cada tipo de datos

Sí que es posible pasar argumento tipo array de bajo nivel sin indicar tamaño. ¿Qué información habrá que proporcionar al método/función?

Funciones y arrays de bajo nivel: impresión array (sin dimensión) I

```
1  #include <iostream>
2  using namespace std;
3
4  // Funcion generica para imprimer arrays de caracteres
5  // Como la primera dimension queda sin especificar hay
6  // que pasar como argumento un parametro que indique el
7  // numero de posiciones usadas
8  void imprime_array(char v[], int util){
9     for (int i=0; i < util; i++)
10       cout << v[i] << " ";
11    cout << endl;
12 }
13
14 // Metodo main para probar
15 int main(){
16    char vocales[5]={'a','e','i','o','u'};
17    char digitos[10]={'0','1','2','3','4',
18                      '5','6','7','8','9'};
19
20    imprime_array(vocales, 5); cout<<endl;
21    imprime_array(digitos, 10); cout<<endl;
22    imprime_array(digitos, 5); cout<<endl;
23    imprime_array(vocales, 100); cout<<endl;
24 }
```

Funciones y arrays de bajo nivel: impresión array I

Otro ejemplo sobre clase SecuenciaEnteros:

- método asignarValores para fijar los valores almacenados por el objeto
- argumentos:
  - array con valores a almacenar en el dato miembro
  - indicador del número de elementos almacenados en el array (primer argumento)
- dovolución: valor booleano (operación finaliza correctamente o no)

Funciones y arrays de bajo nivel: métodos SecuenciaEnteros I

```
1  #include <iostream>
2  #include <cmath>
3  #include<sstream>
4  using namespace std;
5
6  /**
7   * Clase SecuenciaEnteros
8   */
9  class SecuenciaEnteros {
10  private:
11    static const int TAMANIO = 50;
12    int vectorPrivado[TAMANIO];
13    int totalUtilizados;
14
15  public:
16    /**
17     * Constructor de la clase
18     */
19    SecuenciaEnteros() {
20       totalUtilizados = 0;
21    }
22
23
24
25
26
```

Funciones y arrays de bajo nivel: métodos SecuenciaEnteros II

```
27      /**
28       * Metodo para agregar un entero a la secuencia
29       */
30      void agregarEntero(int nuevo) {
31         // Se comprueba si se puede realizar la operacion
32         if (totalUtilizados < TAMANIO) {
33            vectorPrivado[totalUtilizados] = nuevo;
34            totalUtilizados++;
35         }
36      }
37
38      /**
39       * Metodo para asignar al dato miembro vectorPrivado
40       * un conjunto de valores
41       */
42      bool asignarValores(int valores[], int numeroValores) {
43         bool result = true;
44         // Se comprueba en primer lugar si se pueden asignar
45         // todos los valores: es decir, si hay espacio suficiente
46         if (numeroValores < TAMANIO) {
47            // copia de valores
48            for (int i = 0; i < numeroValores; i++) {
49               vectorPrivado[i] = valores[i];
50            }
51            // Se modifica el valor de totalUtilizados
52            totalUtilizados = numeroValores;
53         }
```

Funciones y arrays de bajo nivel: métodos SecuenciaEnteros III

```
54          else {
55              // La operacion no es viable
56              result = false;
57          }
58          // Se devuelve el resultado de la operacion
59          return result;
60      }
61
62      /**
63       * Metodo para devolver una cadena (objecto de la clase
64       * string) con la informacion del contenido del objeto
65       */
66      string mostrarContenido() {
67          stringstream oss;
68
69          // Se usa el objeto de la clase stringstream para
70          // volcar en el los datos almacenados en el vector
71          // privado
72          oss << "----------------------------------------" << endl;
73          oss << "Valores almacenados: " << totalUtilizados << endl;
74          for (int i = 0; i < totalUtilizados; i++) {
75              oss << vectorPrivado[i] << " ";
76          }
77          oss << endl;
78          oss << "----------------------------------------" << endl;
```

Funciones y arrays de bajo nivel: métodos SecuenciaEnteros IV

```
79
80          // Se devuelve la cadena con la informacion del objeto
81          return (oss.str());
82      }
83
84      /**
85       * Metodo para obtener el valor almacenado en una
86       * posicion
87       */
88      int obtenerElemento(int indice) {
89          int valor = -1;
90
91          if (indice >= 0 && indice < totalUtilizados) {
92              valor = vectorPrivado[indice];
93          }
94
95          return valor;
96      }
97
98      /**
99       * Metodo para devolver el valor de totalUtilizados
100      */
101     int obtenerTotalUtilizados() {
102         return totalUtilizados;
103     }
```

Funciones y arrays de bajo nivel: métodos SecuenciaEnteros V

```
104 };
105
106
107
108 // Metodo main para probar
109 int main() {
110    // Se crea objeto de la clase secuencia de enteros
111    SecuenciaEnteros indices;
112
113    // Se asignan los valores al array
114    int array[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};
115
116    // Se llama al metodo que permite asignar los valores
117    bool resultado = indices.asignarValores(array, 15);
118
119    // Se muestra el resultado de la operacion
120    cout << "Resultado operacion asignacion valores: "
121                                    << resultado << endl;
122
123    // Se agrega un elemento adicional
124    indices.agregarEntero(23);
125
126    // se muestra el contenido del objeto indices
127    cout << indices.mostrarContenido() << endl;
128 }
```

Funciones y arrays de bajo nivel I

Importante: los arrays no se pasan por valor; los elementos pueden modificarse como resultado de las operaciones de la función (método). Se considera un ejemplo sobre la clase **SecuenciaEnteros**: método que devuelve (en un argumento) los valores almacenados en el objeto

Funciones y arrays de bajo nivel: métodos SecuenciaEnteros I

```
1  #include <iostream>
2  #include <cmath>
3  #include<sstream>
4  using namespace std;
5
6  // Se define la constante fuera de la clase para que pueda
7  // consultarse desde cualquier sitio
8  const int TAMANIO = 50;
9
10 /**
11  * Clase SecuenciaEnteros
12  */
13 class SecuenciaEnteros{
14  private:
15    int vectorPrivado[TAMANIO];
16    int totalUtilizados;
17
18  public:
19
20    /**
21     * Constructor de la clase
22     */
23    SecuenciaEnteros(){
24       totalUtilizados=0;
25    }
26
```

Funciones y arrays de bajo nivel: métodos SecuenciaEnteros II

```
27      /**
28       * Metodo para agregar un entero a la secuencia
29       */
30      void  agregarEntero(int  nuevo){
31         // Se comprueba si se puede realizar la operacion
32         if (totalUtilizados  <  TAMANIO){
33            vectorPrivado[totalUtilizados]  =  nuevo;
34            totalUtilizados++;
35         }
36      }
37
38      /**
39       * Metodo para asignar al dato miembro vectorPrivado
40       * un conjunto de valores
41       */
42      bool asignarValores(int valores[], int numeroValores){
43         bool result=true;
44
45         // Se comprueba en primer lugar si se pueden asignar
46         // todos los valores: es decir, si hay espacio suficiente
47         if (numeroValores < TAMANIO){
48            // Puede hacerse la operacion y se copian los valores
49            // del array al dato miembro
50            for(int i=0; i < numeroValores; i++){
51               vectorPrivado[i]=valores[i];
52            }
53
```

Funciones y arrays de bajo nivel: métodos SecuenciaEnteros III

```
54            // Se modifica el valor de totalUtilizados
55            totalUtilizados=numeroValores;
56         }else{
57            // La operacion no es viable
58            result=false;
59         }
60
61         // Se devuelve el resultado de la operacion
62         return result;
63      }
64
65      /**
66       * Metodo para recuperar los valores almacenados en el
67       * dato miembro vectorPrivado. El metodo devuelve el
68       * numero de valores recuperados
69       */
70      int obtenerValores(int valores[]){
71         // Copia los valores de vectroPrivado en valores
72         for(int i=0; i < totalUtilizados; i++){
73            valores[i]=vectorPrivado[i];
74         }
75
76         // Devuelve el numero de valores
77         return totalUtilizados;
78      }
```

Funciones y arrays de bajo nivel: métodos SecuenciaEnteros IV

```
79
80      /**
81       * Metodo para devolver una cadena (objecto de la clase
82       * string) con la informacion del contenido del objeto
83       */
84      string mostrarContenido(){
85         stringstream oss;
86
87            // Se usa el objeto de la clase stringstream para
88            // volcar en el los datos almacenados en el vetor
89            // privado
90            oss << "----------------------------------------" << endl;
91            oss << "Valores almacenados: " << totalUtilizados << endl;
92            for(int i=0; i < totalUtilizados; i++){
93                oss << vectorPrivado[i] << " ";
94            }
95            oss << endl;
96            oss << "----------------------------------------" << endl;
97
98            // Se devuelve la cadena con la informacion del objeto
99               return(oss.str());
100     }
101
102
103
```

Funciones y arrays de bajo nivel: métodos SecuenciaEnteros V

```
104     /**
105      * Metodo para obtener el valor almacenado en una posicion
106      */
107     int obtenerElemento(int indice){
108        int valor=-1;
109
110        if (indice >= 0 && indice < totalUtilizados){
111            valor=vectorPrivado[indice];
112        }
113
114        return valor;
115     }
116
117     /**
118      * Metodo para devolver el valor de totalUtilizados
119      */
120     int obtenerTotalUtilizados(){
121        return totalUtilizados;
122     }
123 };
124
125
126
127
128
129
130
```

Funciones y arrays de bajo nivel: métodos SecuenciaEnteros VI

```
131 /**
132  * Metodo main para probar
133  */
134 int main(){
135    // Se crea objeto de la clase secuencia de enteros
136    SecuenciaEnteros indices;
137 
138    // Se asignan los valores al array para inicializar el
139    // objeto
140    int array[]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
141 
142    // Se llama al metodo que permite asignar los valores
143    // Hay que pasar como argumento tanto el array como el
144    // numero de valores almacenados
145    bool resultado=indices.asignarValores(array,15);
146 
147    // Se muestra el resultado de la operacion
148    cout << "Resultado operacion asignacion valores: " <<
149                        resultado << endl;
150 
151    // Se agrega un elemento adicional
152    indices.agregarEntero(23);
153 
154    // se muestra el contenido del objeto indices
155    cout << indices.mostrarContenido() << endl;
```

Funciones y arrays de bajo nivel: métodos SecuenciaEnteros VII

```
156
157     // Se crea un array que nos servira para recuperar los datos
158     // almacenados en el objeto indices
159     int datosObjeto[TAMANIO];
160
161     // Se llama al metodo que hace la obtencion de los valores. Se
162     // le pasa como argumento el array datosObjeto recien creado.
163     // Al finalizar la operacion contendra los valores que estan
164     // almacenados en el dato miembro vectorPrivado del objeto
165     int recuperados=indices.obtenerValores(datosObjeto);
166
167     // Se muestran los datos almacenados en el array, para comprobar
168     // que realmente se recuperaron bien
169     cout << "\nDatos recuperados del objeto: " << recuperados << endl;
170     for(int i=0; i < recuperados; i++){
171        cout << datosObjeto[i] << " ";
172     }
173     cout << endl;
174 }
```

Funciones y arrays de bajo nivel: métodos SecuenciaEnteros I

Importante: el array de argumento del método obtenerValores es modificado y al final contendrá los valores almacenados en el datos miembro privado

Funciones y arrays de bajo nivel: paso de arrays como argumentos I

Ventajas de esta forma de paso de argumentos:

- un método (función) puede devolver varios resultados
- esta forma de paso de argumentos se verá en el tema de punteros
- otra forma adicional: paso por referencia

Funciones y arrays de bajo nivel: paso por referencia I

En el paso por referencia:

- un argumento pasado por referencia es un alias del correspondiente parámetro actual
- para indicar que un argumento se pasa por referencia basta con incluir el símbolo **&** antes de su nombre

```
// Declaracion de la funcion
void function(int & contador, .....)
.....................................
// Llamada a la funcion
int x=0;
funcion(x, ....)
```

Funciones y arrays de bajo nivel: paso por referencia I

```
1  // Ejemplo de paso por valor y paso por referencia
2  #include <iostream>
3  using namespace std;
4
5  // Declaracion anticipada de las funciones a usar
6
7  // declaracion de la funcion con paso por valor
8  int calcularCuadradoPasoPorValor( int );
9
10 // declaracion de la funcion con paso por referencia
11 void calcularCuadradoPasoPorReferencia( int & );
12
13
14 // Metodo main para probar
15 int main() {
16    // valor a elevar al cuadrado con paso por valor
17    int x = 2;
18
19    // valor a elevar al cuadrado con paso por referencia
20    int z = 4;
21
22    // se hace uso de las funciones
23    cout << "Valor inicial de x = " << x << " antes de llamada\n";
24    cout << "Valor devuelto por calcularCuadradoPasoPorValor: "
25       << calcularCuadradoPasoPorValor( x ) << endl;
26    cout << "Valor final de x = " << x << " tras llamada\n" << endl;
```

Funciones y arrays de bajo nivel: paso por referencia II

```
27
28    // uso de emonstrate calcularCuadradoPasoPorReferencia
29    cout << "Valor inicial de z = " << z <<
30     x" antes de llamada a calcularCuadradoPasoPorReferencia" << endl;
31    calcularCuadradoPasoPorReferencia( z );
32    cout << "Valor final de z = " << z << " tras llamada" << endl;
33 } // end main
34
35 // calcularCuadradoPasoPorValor usa paso por valor y el
36 // resultado debe devolverse mediante sentencia return
37 int calcularCuadradoPasoPorValor( int valor ) {
38    return valor *= valor;
39 }
40
41 // calcularCuadradoPasoPorReferencia usa paso por referencia
42 // y el valor calculado se devuelve sobre el propio argumento
43 void calcularCuadradoPasoPorReferencia( int &valor ) {
44    valor *= valor;
45 }
```

Funciones y arrays de bajo nivel: uso de const

Se usa el modificador const para evitar que los métodos (funciones) puedan modificar los valores almacenados en un array pasado como argumento

Funciones y arrays de bajo nivel: uso de const I

```
1  #include <iostream>
2  using namespace std;
3
4  // Funcion para impresion de array de caracteres
5  void imprime_array (const char v[], int numeroCaracteres){
6     for (int i=0; i < 5; i++){
7        cout << v[i] << " ";
8     }
9     cout << endl;
10 }
11
12 // Main para probar la funcion indicada
13 int main(){
14    char vocales[5]={'a','e','i','o','u'};
15    imprime_array(vocales,5);
16 }
```

Funciones y arrays de bajo nivel: uso de const

Consecuencias del uso de const:

- si en el interior del método (función) se intenta modificar el contenido de un array pasado como argumento, se produce un error de compilación
- si no se usa const en la declaración del argumento entonces el compilador asume que puede modificarse, aunque no se haga. Esto hace que se produzca un error de compilación si el parámetro actual es un array constante

Funciones y arrays de bajo nivel: uso de const I

```
1  #include <iostream>
2  using namespace std;
3
4  // Funcion para impresion de array de caracteres
5  void imprime_array (char v[], int numeroCaracteres){
6     for (int i=0; i < 5; i++){
7        cout << v[i] << " ";
8     }
9     cout << endl;
10 }
11
12 // Main para probar la funcion indicada
13 int main(){
14    const char vocales[5]={'a','e','i','o','u'};
15    imprime_array(vocales,5);
16 }
```

Funciones y arrays de bajo nivel: uso de const I

Mensaje de error:

```
error: invalid conversion from 'const char*' to 'char*' [-fpermissive]
    imprime_array(vocales,5);
```

Funciones y arrays de bajo nivel: devolución de arrays

Cabe preguntarse: ¿es posible que un método (función) devuelva como resultado un array?

Funciones y arrays de bajo nivel: metodología

Nota metodológica en relación a la implementación de métodos (funciones):

- importancia del nombre (que contenga verbo y sea significativo)
- describir brevemente su funcionalidad
- indicar sus argumentos (tipos)
- indicar qué devuelve (tipo)

Índice

1. Motivación
2. Objetivos
3. Arrays de bajo nivel
4. Funciones y arrays de bajo nivel
5. Cadenas de caracteres estilo C
6. Matrices
7. Resumen

Cadenas de caractares estilo C

Cadenas estilo C:

- arrays de bajo nivel que almacenan caracteres
- el carácter especial
  0 se usa como centinela

Ejemplos:

```
char nombre[10] = {'J','a','v','i','e','r','\0'};

char otro[] = {"Javier"};
```

Cadenas de caractares estilo C

Importante:

- debe haber espacio suficiente para el centinela
- se aplica todo lo visto sobre paso de argumentos, devolución, etc

Cadenas de caractares estilo C

Ejemplo: función para concatener cadenas de caracteres. Diseño:

- nombre: concatenar
- argumentos: cadena1, cadena2, cadenaResultante
- salida: void
- descripción: se concatenan cadena1 y cadena2 y se produce como resultado cadenaResultante. La cadena de resultado debe tener espacio suficiente para la operación

Cadenas de caracteres estilo C I

```
1  #include<iostream>
2  using namespace std;
3  /**
4   * Funcion para concatenar dos cadenas
5   * @param primera cadena
6   * @param segunda cadena
7   * @param cadena resultante
8   */
9  void concatenar(const char cad1[], const char cad2[], char res[]){
10    int pos=0;
11
12    // Bucle para agregar todos los caracteres de la primera cadena.
13    // pos controla la posicion de escritura en la cadena de resultado
14    for (int i=0;cad1[i]!='\0';i++){
15       res[pos]=cad1[i];
16       pos++;
17    }
18    // Bucle de agregacion de caracteres de la segunda cadena.
19    for (int i=0;cad2[i]!='\0';i++){
20       res[pos]=cad2[i];
21       pos++;
22    }
23
24    // Se agrega el caracter de fin de cadena a res
25    res[pos]='\0';
26 }
```

Cadenas de caracteres estilo C II

```
27 // Metodo main para probar
28 int main(){
29    const int MAXTAM=300;
30    char cadena1[MAXTAM];
31    char cadena2[MAXTAM];
32    char cadena3[2*MAXTAM];
33
34    // Lectura de la primera cadena con getline (lee todos
35    // los caracteres introducidos hasta que se pulsa ENTER)
36    cout << "Introduce una cadena de caracteres: ";
37    cin.getline(cadena1,300);
38
39    // Lectura de la segunda cadena
40    cout << "Introduce otra cadena de caracteres: ";
41    cin.getline(cadena2,300);
42
43    // Llamada al metodo de concatenacion
44    concatenar(cadena1, cadena2, cadena3);
45    cout << "La concatencación de las cadenas es:" << cadena3 << endl;
46 }
```

Cadenas de caractares estilo C

cin y cout pueden usarse sobre cadenas de caracteres. A considerar:

- cin: salta separadores previos al dato a leer, lee y detiene la lectura al encontrar el siguiente separador
- la función getline lee hasta encontrar salto de línea

Cadenas de caractares estilo C

Ejemplo de lectura de datos:

```
Antonio Perez Rodriguez
18
Calle Periodista Daniel Saucedo Aranda s/n
.....................................................
char nombre[80], direccion[120];
int edad;
cout << "Introduzca nombre: ";
cin.getline(nombre, 80);
cin >> edad;
cin.getline(direccion, 120);
```

Cadenas de caractares estilo C

Ejercicio: función que evite este problema . Análisis:

- descripción: saltar todos los separadores iniciales no consumidos por operaciones previas y leer los datos tras ellos
- nombre: leerLinea
- argumentos:
  - array donde almacenar lo tecleado por el usuario
  - contador del número de caracteres máximo a leer
- devolución: contador de caracteres leídos

Cadenas de caracteres estilo C I

```
1  #include<iostream>
2  using namespace std;
3  /**
4   * Metodo leerLinea para saltar los posibles separadores
5   * no consumidos al ir a leer datos tecleados por el
6   * usuario
7   * @param c cadena donde almacenar los datos
8   * @param tamano numero maximo de caracteres a leer
9   * @return numero de caracteres leidos
10  */
11 int leerLinea(char c[], int tamano){
12     // Mientras se lee unicamente un separador (en este
13     // caso c solo contendra el marcador de fin de cadena)
14     do{
15         // se leen datos mediantes getline
16         cin.getline(c, tamano);
17     } while (c[0] == '\0');
18
19     // Cuando se han leido datos se obtiene el numero
20     // de caracteres leidos
21     int contador;
22     for(contador=0; c[contador] != '\0'; contador++);
23
24     // Se devuelve el valor de contador
25     return(contador);
26 }
```

Cadenas de caracteres estilo C II

```
27
28 /**
29  * Metodo main para probar
30  */
31 int main(){
32    char nombre[80],direccion[120];
33    int edad, caracteresLeidos;
34
35    // Se hace el mismo ejercicio de antes
36    cout << "Introduzca el nombre: ";
37    caracteresLeidos=leerLinea(nombre,80);
38
39    // Se muestra el numero de caracteres leidos
40    cout << "Numero de caracteres leidos: " << caracteresLeidos << endl;
41
42    // Se muestra el nombre tal y como se leyi
43    cout << "El nombre introducido es: " << nombre << endl;
44
45    // Se lee la edad con cin directamente
46    cout << "Introduzca la edad: ";
47    cin >> edad;
48
49    // Se muestra la edad leida
50    cout << "\nLa edad introducida es: " << edad << endl;
51
52    // Se introduce la direccion
53    cout << "Introduzca la direccion: ";
```

Cadenas de caracteres estilo C III

```
54      caracteresLeidos=leerLinea(direccion,120);
55
56      // Se muestra el numero de caracteres leidos
57      cout << "Numero de caracteres leidos: " << caracteresLeidos << endl;
58
59      // Se muestra la direccion leida
60      cout << "La direccion introducida es: " << direccion << endl;
61  }
```

Cadenas de caracteres estilo C I

La biblioteca cstring ofrece funciones para realizar tareas comunes:

- strcpy: copia una cadena en otra
- strlen: determina lalongitud, sin incluir la marca final en el conteo
- strcat: concatenación de cadenas
- strcmp: comparación de cadenas en orden lexicográfico

Índice

1. Motivación
2. Objetivos
3. Arrays de bajo nivel
4. Funciones y arrays de bajo nivel
5. Cadenas de caracteres estilo C
6. Matrices
7. Resumen

Matrices I

Se trata de arrays de más de una dimensión. La forma de declaración indica el número de dimensiones que se desea:

```
const int NUMCIUDADES=50;
double distancias[NUMCIUDADES][NUMCIUDADES];
```

Matrices I

Formas de inicialización:

```
int m[2][3]={{1,2,3}. {4,5,6}};
```

Importante:

- sin suficientes valores para la fila se inicializa a 0
- el compilador trata las matrices como arrays de arrays
- todos los elementos se almacenan en posicion contiguas

Matrices I

Operaciones usuales:

- acceso: se precisan tanto índices como dimensiones tenga la matriz. Los índices comienzan en 0
- asignación:

```
m[1][3]=3;
```

- lectura y escritura:

```
cin >> m[1][1];
cout << m[1][1];
```

Matrices I

Como ocurre con los arrays es importante distinguir entre:

- espacio reservado para la matriz
- número de posiciones realmente usadas
- tantos índices como dimensiones

Matrices I

Ejemplo donde:

- se introducen datos en una matriz
- se busca un determinado elemento y se indica la posición que ocupa

Matrices I

```
1  #include <iostream>
2  using namespace std;
3
4  // Programa de ejemplo de uso de matrices: el objetivo
5  // es buscar un elemento en una matriz bidimensional
6  int main(){
7    const int FIL=20, COL=30;
8
9    // Declaracion de la matriz
10   double m[FIL][COL];
11
12   // Indican filas y columnas usadas
13   int util_fil, util_col;
14
15   // Indican posicion valor buscado
16   int fil_enc, col_enc;
17
18   // Permite almacenar el valor a localizar
19   double buscado;
20
21   // Resultado de la busqueda
22   bool encontrado;
23
24
25
26
```

Matrices II

```
27    // Se lee el numero de filas: el bucle asegura la lectura
28    // de valor comprendido entre 1 y 20
29    do{
30      cout << "Introducir el numero de filas: ";
31      cin >> util_fil;
32    }while ((util_fil<1) || (util_fil>FIL));
33
34
35    // Se lee el numero de columnas: entre 1 y 30
36    do{
37      cout << "Introducir el numero de columnas: ";
38      cin >> util_col;
39    }while ((util_col<1) || (util_col>COL));
40
41    // Bucle doble (uno por dimension) para introduccion de valores
42    // en la matriz
43    for (int f=0 ; f<util_fil; f++) {
44      for (int c=0 ; c<util_col ; c++){
45        cout << "Introducir el elemento ("<< f << "," << c << "): ";
46        cin >> m[f][c];
47      }
48    }
49
50    // Se hace la lectura del valor a buscar
51    cout << "\nIntroduzca elemento a buscar: ";
52    cin >> buscado;
53
```

Matrices III

```
54    // Busqueda del valor
55    encontrado=false;
56
57    // Se recorre la matriz hasta encontrar el valor o final.
58    // Condiciones de parada: que se haya encontrado el valor
59    // buscado o que se alcance el final de la dimension
60    for (int f=0; !encontrado && (f < util_fil) ; f++){
61      for (int c=0; !encontrado && (c < util_col) ; c++){
62        if (m[f][c] == buscado){
63          encontrado = true;
64          fil_enc = f; col_enc = c;
65        }
66      }
67    }
68
69    // Se muestran los resultados
70    if (encontrado){
71      cout << "Encontrado en la posición " <<
72              fil_enc << "," << col_enc << endl;
73    }
74    else{
75      cout << "Elemento no encontrado\n";
76    }
77
78    // El programa principal devuelve 0: todo OK al finalizar
79    return 0;
80  }
```

Matrices I

Es posible definir matrices de tantas dimensiones como deseemos:

```
int cubo[3][3][3];
```

Matrices I

Al pasar una matriz como argumento es necesario especificar todas las dimensiones, menos la primera

```
void leerDatosMatriz(int m[][COL], int utilFil, int utilCol)
```

Matrices I

Ejemplo completo usando modularización:

- lectura de valores de la matriz
- lectura de valor entero: el valor a buscar
- búsqueda de valor en la matriz

Matrices I

```
1  #include <iostream>
2  using namespace std;
3  const int FIL=20, COL=30;
4
5  /**
6   * Metodo para leer el contenido de una matriz
7   * @param matriz a rellenar
8   * @param filas
9   * @param columnas
10  */
11 void leerMatriz(double m[][COL], int util_fil, int util_col){
12   // Recorrido de las posiciones
13   for (int f=0 ; f<util_fil; f++){
14     for (int c=0 ; c<util_col ; c++){
15       cout << "Introducir el elemento ("<< f << "," << c << "): ";
16       cin >> m[f][c];
17     }
18   }
19 }
20
21
22
23
24
25
26
```

Matrices II

```
27 /**
28  * Metodo para leer un valor entero
29  * @param mensaje a mostrar al hacer la lectura del valor
30  * @return valor introducido por el usurario
31  */
32 int leerValorEntero(const char mensaje[]){
33   int aux;
34
35   // Lectura del valor
36   cout << mensaje;
37   cin >> aux;
38
39   // Se devuelve el valor leido
40   return aux;
41 }
42
43
44
45
46
47
48
49
50
51
52
53
```

Matrices III

```
54
55 /**
56  * Metodo para buscar un valor en la matriz
57  * @param matriz a buscar
58  * @param filas
59  * @param columnas
60  * @param elemento a buscar
61  * @return flag booleano con resultado
62  */
63 bool buscarValor(const double m[][COL], int util_fil,
64         int util_col, double elemento){
65   bool encontrado=false;
66
67   // Bucle de recorrido de la matriz
68   for (int f=0; !encontrado && (f < util_fil) ; f++){
69     for (int c=0; !encontrado && (c < util_col) ; c++){
70       if (m[f][c] == elemento){
71         encontrado = true;
72       }
73     }
74   }
75
76   // Devuelve el valor de encontrado
77   return encontrado;
78 }
79
80
```

Matrices IV

```
81 /**
82  *  Metodo main para probar el codigo
83  */
84 int main(){
85   double m[FIL][COL];
86
87   // Filas y columnas de la matriz
88   int util_fil, util_col;
89
90   // Para almacenar el valor buscado
91   double buscado;
92
93   // Para almacenar el resultado de la busqueda
94   bool encontrado;
95
96   // Se lee el numero de filas deseado
97   util_fil =
98      leerValorEntero("Introducir el numero de filas (de 1 a 20): ");
99
100  // Se lee el numero de columnas: de 1 a 30
101  util_col =
102     leerValorEntero("Introducir el numero de columnas (de 1 a 30): ");
103
104  // Se leen los valores de la matriz
105  leerMatriz(m, util_fil, util_col);
106
107
```

Matrices V

```
108     // Se introduce el valor a buscar
109     buscado = leerValorEntero("Introduzca el valor a buscar: ");
110
111     // Se llama al metodo que busca el valor
112     encontrado=buscarValor(m, util_fil, util_col, buscado);
113
114     // Se muestra el resultado
115     cout << "Valor encontrado: " << encontrado << endl;
116
117     // Indicacion de finalizacion OK del programa
118     return 0;
119 }
```

Índice

1. Motivación
2. Objetivos
3. Arrays de bajo nivel
4. Funciones y arrays de bajo nivel
5. Cadenas de caracteres estilo C
6. Matrices
7. Resumen

Resumen:

Conceptos esenciales:

- array, como elemento independiente (sin necesidad de tener una clase como *envoltorio*)
- uso de la clase vector
- elementos diferenciales entre arrays de bajo nivel y clases de C++ (vector y array)
- cadenas estilo C (algo secundario, sólo son arrays de caracteres con marca final de fin de cadena)
- matrices: arrays de arrays