UNIDAD III - ARRAYS

OBJETIVOS DE LA SEMANA 8va (cuatr.)

Clase teórica

- DECLARACIÓN DE ARRAYS. SUBINDICACIÓN. TAMAÑO DE ARRAYS. VERIFICACIÓN DEL RANGO DE ÍNDICES. INICIALIZACIÓN DE UN ARRAY.
- CONCEPTO DE CADENA. LECTURA. ARRAYS y CADENAS COMO PARÁMETROS DE FUNCIONES. USO DE PUNTEROS PARA PASAR UNA CADENA. ASIGNACIÓN DE CADENAS. La función strlen()

Clase práctica

- IMPLEMENTACIÓN DE UN CONJUNTO.

OBJETIVOS DE LA SEMANA 9na (cuatr.)

Clase teórica

- Las funciones strcat y strncat, strcmp y strncmp. Una clase tratando cadenas. BÚSQUEDA EN ARRAYS. UNA CLASE ITEMS. BÚSQUEDA SECUENCIAL Y BINARIA. COMPARACIÓN ENTRE BÚSQUEDAS.
- ORDENAMIENTO - Introducción. Algoritmos de ordenamiento básicos y mejorados. Ordenamiento por Método Burbuja. Ordenamiento por Método "sacudida". Ordenamiento por Método "Peinado". COMPARANDO TIEMPOS DE ORDENAMIENTO.

Clase práctica

- class MisCadenas. class Items. class BusqSeq. class BusqBin.
- class OrdBurb. class OrdSac. class OrdPein. class CalcTime.

OBJETIVOS DE LA SEMANA 10ma (cuatr.)

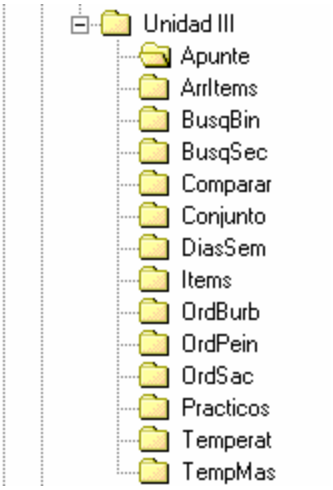
Clase teórica

- ARRAYS MULTIDIMENSIONALES. INICIALIZACIÓN, ACCESO. LECTURA. USO DE BUCLES. ARRAYS DE MAS DE DOS DIMENSIONES. Caso practico: matriz de temperaturas.

Clase práctica

- class Temperat. class DiasSem. class TempMas

Material disponible en el sitio labsys.



Autor: Tymoschuk, Jorge
Compaginación: Guzmán, Analía

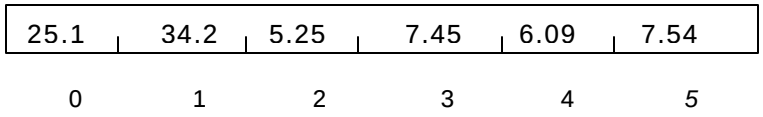
ARRAYS

En C, un *array* (lista o tabla) es una secuencia de elementos del mismo tipo. Los elementos se numeran consecutivamente 0, 1, 2, 3... El tipo de elementos almacenados en el array puede ser cualquier tipo de dato de C++, incluyendo estructuras u objetos definidos por el usuario, como se describirá más tarde.

Un array puede contener, por ejemplo, la edad de los alumnos de una clase, las temperaturas de cada día de un mes en una ciudad determinada, o el número de personas que residen en cada una de nuestras provincias. Los elementos de un array se numeran, consecutivamente 0, 1, 2, 3... Estos números se denominan *valores índice* o *subíndice* del array. El término *subíndice* se utiliza ya que se especifica igual que en matemáticas como una secuencia tal como a_0, a_1, a_2... Estos números localizan la posición del elemento dentro del array, proporcionando *acceso directo* al array.

Si la identificación del array es a, entonces a [0] es el elemento que está en la posición 0, a [1] es el elemento que está en la posición 1, etc.

Abajo la representación gráficamente de un array a con seis elementos. Su almacenamiento se realiza en memoria contigua



Declaración de un array

Al igual que con cualquier tipo de variable, se debe declarar un array antes de utilizarlo. Esto se hace de modo similar a otros tipos de datos, especificando al compilador el *tamaño* o *longitud* del array. Para indicar al compilador el *tamaño* o *longitud* del array se debe hacer seguir al nombre, el tamaño encerrado entre corchetes. La *sintaxis* para declarar un array de una dimensión determinada es:

tipo nombreArray[numeroDeElementos];

Por ejemplo, para, se escribe:

```
int numeros[1O];        //        crear un array de diez elementos enteros
```

Esta declaración hace que el compilador reserve espacio suficiente para contener diez valores enteros. En C++ los enteros ocupan, normalmente, 2 bytes, de modo que un array de diez enteros ocupa 20 bytes de memoria

Se accede a cada elemento del array sub indicando el elemento. Por ejemplo,

```
cout « numeros[4] « endl;       // visualiza el valor del quinto elemento del array
```

Subíndices de un array

El índice de un array se denomina, con frecuencia, *subíndice del array.* El término procede de las matemáticas, en las que un subíndice se utiliza para representar un elemento determinado.

$numeros_0$ *equivale a* numeros [0]
$numeros_3$ *equivale a* numeros [3]

El método de numeración del elemento i-ésimo con subíndice i-1 se denomina indexación basada en cero. Su uso tiene el efecto de que el índice de un elemento del array es siempre el mismo que el número de «pasos» desde el elemento inicial a [O] a ese elemento. Por ejemplo, a [3] está a 3 pasos o posiciones del elemento a [O] . Cuando veamos punteros apreciaremos las ventajas de este método. *Ejemplos:*

```
int pesos[25], longitudes [100]; // Declara 2 arrays de enteros.
```

```
float salarios[25]                    // Declara un array de 25 elementos float.
```

Un elemento de un array se subindica con tipo entero. El valor de esta subindicación puede provenir de una constante, variable, expresión matemática, retorno de función ...

Los arrays de caracteres funcionan de igual forma que los arrays numéricos, partiendo de la base de que cada carácter ocupa normalmente un byte. Así, por ejemplo, un array llamado nombre se puede representar en la Figura 7.4.

```
char nornbre[]    = "Mortimer";
char nombre[1O] = "Mackoy":
```

El tamaño de los arrays (sizeof)

La función sizeof() devuelve el número de bytes necesarios para contener su argumento. Si se usa sizeof() para solicitar el tamaño de un array, esta función devuelve el número de bytes reservados para el array completo.
Por ejemplo, supongamos que se declara un array de enteros de 100 elementos denominado codigos; si se desea producir el tamaño del array se puede utilizar:

```
n = sizeof(codigos);    //        donde n tomará el valor 200

n = sizeof(codigos[2]); // n almacenará el valor 2
```

Verificación del rango del índice de un array

C++ al contrario que otros lenguajes de programación -por ejemplo, Pascal - no verifica "motu propio" los límites de la subindicación. Podemos excedernos para abajo o para arriba de la memoria que contiene el array, C++ deja esto librado a la responsabilidad del programador . Esto es así por ser C un lenguaje "orientado eficiencia". De hecho, el programador puede tratar eficientemente los desbordes de subindicación, por ejemplo, con la metodología de Tratamiento de Excepciones.

Nota. Hablando de tratamiento de arrays por otros lenguajes, es conveniente decir que varían bastante de lenguaje a lenguaje. Pascal, por ejemplo, naturalmente permite que el programador especifique los límites de subindicación, por ejemplo vector[3,7], Clipper y otros XBase permiten arrays de elementos no uniformes, etc.

INICIALIZACIÓN DE UN ARRAY

Se deben asignar valores a los elementos del array antes de utilizarlos, tal como se asignan valores a variables. Para asignar valores a cada elemento del array de enteros precios, se puede escribir:

```
precios[0] = 10;
precios[1] = 20;
precios[3] = 30;
precios[4] = 40;
```

Estas sentencias fijan precios [0] al valor 10, precios [1] al valor 20, etc. Sin embargo, este método no es práctico cuando el array contiene muchos elementos. El método utilizado normalmente es *inicializar* el array completo en un ciclo.

Cuando se inicializa un array, el tamaño del array se puede determinar automáticamente por las constantes de inicialización. Estas constantes se separan por comas y se encierran entre llaves, como en los siguientes ejemplos:

```
int numeros[6] =  {10, 20, 30, 40, 50, 60};   //  Array, 6 elementos enteros
int n[] =  {3, 4, 5};                         //  Array, 3 elementos enteros
char c [] = {'L', 'u', 'i', 's'};             //  Array, 4 elementos caracter
```

Los arrays de caracteres se pueden inicializar con una constante de cadena, como en

```
char s[] =  {"Mortimer"};
```

El método de inicializar arrays mediante valores constantes después de su definición es adecuado cuando el número de elementos del array es pequeño. Por ejemplo, para inicializar un array (lista) de 10 enteros a los valores 10 a 1, y a continuación visualizar dichos valores en un orden inverso, se puede escribir:

```
int cuenta[10] = {10,  9, 8, 7, 6, 5, 4, 3, 2, 1, 0};
for (int i =9; i > 0; i--) cout << "\n cuenta descendente" << i << ' = '<< cuenta[i];
```

Se pueden asignar constantes simbólicas como valores numéricos, de modo que las sentencias siguientes son válidas:

```
const int ENE = 31, FEB = 28, MAR = 31, ABR = 30, MAY = 31,
          JUN = 30, JUL = 31,  AGO = 31, SEP = 30, OCT =      31,
          NOV = 30, DIC = 31;

int meses[12] = {ENE, FEB, MAR, ABR, MAY, JUN, JUL, AGO, SEP, OCT, NOV,DIC};
```

Se pueden asignar valores a un array utilizando un bucle for o while/do-while, y éste suele ser el sistema más empleado normalmente. Por ejemplo, para inicializar todos los valores del array numeros al valor 0, se puede utilizar la siguiente sentencia:

```
for (int i= 0; i <6; i++) numeros[i] = 0;
```

EJEMPLO

El siguiente programa asigna ocho enteros a un array nums, mediante lectura y los acumula; al salir del ciclo, visualiza el total.

```
#include <iostream.h>
const NUN 8

main (){
        int nums [NUN];
        int total = 0;
        for (int i = 0; i < NUN; i++){
                cout << "Por favor, introduzca el numero";
                cin   >>nums[i];
                total +=nums[i];
      }
        cout << "El total de números es" << total < endí;
        return 0;
}
```

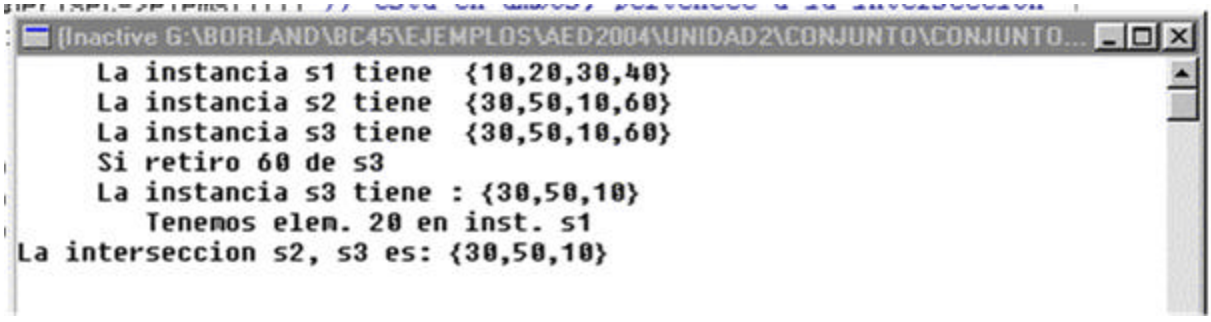
Las variables globales que representan arrays se inicializan al NULL del tipo por defecto. El valor Null para enteros es 0, carácter nulo ('\0') para caracteres, ...

Sería bueno ver un caso concreto. Un ejemplo que use arrays en un caso sencillo, claro ...

*Y si ... tengo uno. Un array de números enteros usado para implementar un **Conjunto**. En un conjunto los elementos no pueden estar duplicados, podemos agregar, quitarlos, generar nuevos conjuntos uniendo o intersectando otros, etc, etc... Sólo necesitamos el array y un atributo cardinalidad, que dice cuantos elementos tiene el conjunto actualmente*

```
// Implementación de un conjunto.

#include <iostream.h>
const int maxCard = 16;              //  Máxima cardinalidad del conjunto
enum Bool {false,true };             //  Definiendo mi tipo Bool
enum ErrCode {noErr, overflow};      //  y ErrCode

class Set{
private:
        int  elems[maxCard];         //  Definimos el array implementador del conjunto
        int card;                    // cardinalidad, o sea cuantos elementos tenemos
public:
        Set(){card=0;}               //  Construimos conjunto vacío
        int Card(){return card;}     //  Retorna cardinalidad
        Bool Member(int);            //  Es miembro el número pasado como argumento ?
        ErrCode AddElem(int);        //  Agregamos un elemento
        void RmvElem(int);           //  Excluimos un elemento
        void Copy(Set*);             //  Copiamos un conjunto
        Bool Equal(Set*);            //  Comparamos conjuntos
        void toString();             //  Exhibe el conjunto
        void Intersect(Set*, Set*);  //  Generamos un conjunto intersección
        ErrCode Union(Set*, Set*);   //  Generamos un conjunto union
};

Bool Set::Member(int elem){          //  Verifica si elem pertenece al conjunto nativo
        for (int i = 0;i < card; ++i)
                if(elems[i] == elem)     return true;
        return false;
};

ErrCode Set::AddElem (int  elem){
        for(int i = 0;i < card; ++i)
                if (elems[i] == elem) return noErr;     // Estaba, listo ...
                if (card < maxCard){                    //  No estaba, hay lugar
                        elems[card++] = elem; return noErr;}
                else return overflow;
};

void Set::RmvElem (int       elem){
        for (int i = 0; i < card; ++i)
                if (elems[i] == elem){
                        for (; i < card-1; ++i)                 // desplaza los elementos
                                elems[i] = elems[i+1];          // a la izquierda
                        --card;                                 //  decrementamos cardinalidad
                }
};

void Set::Copy(Set* set){
  for(int i = 0; i < card; i++)
        set->elems[ i ] = elems[ i ];
        set->card = card;
};

Bool Set::Equal (Set* set){          //  Verifica si dos conjuntos son iguales
        if(card != set->card) return false;
        for (int  i = 0; i < card; ++i)  // Recorremos el conjunto implicito
                if ( !set->Member(elems[i]))  // Cualquiera elemento del conj. parámetro
                        return false;                                   // que no esté, false
        return true;
};

void Set::toString(){
```

```
        cout << '{';
        for (int i = 0; i < card - 1; ++i)
                cout << elems[i] << ',';
        if (card > 0)
                cout <<elems[card-1] << "}\n";
};

void Set::Intersect (Set* set, Set* res){
        for (int i = 0; i < set->card; ++i)
                if (Member(set->elems[i])) // está en ambos, pertenece a la inte rsección
                        res->AddElem(set->elems[i]);  // incluirlo ...
};

/*  En Intersect estamos trabajando con tres conjuntos:
        1ro)    El nativo, objeto invocante de este método,  (s2.Intersect(...) en el main());
        cuando en Intersect preguntamos if(Member (...) estamos preguntando si elementos
        del conjunto set pasado como parámetro son miembros de s2.

        2do)    set, el conjunto cuyos elementos usamos para verificar intersección con s2. En
        if (Member(set->elems[i])).

        3ro)    res, el conjunto resultante de la intersección. Inicialmente vacío, va recibiendo
        los elementos de set que son miembros de s2. res->AddElem(set->elems[i])        */

void main(){
        Set s1, s2, s3; // Instancias de la clase Set
        s1.AddElem(10);s1.AddElem(20);s1.AddElem(30);s1.AddElem(40);
        s2.AddElem(30);s2.AddElem(50);s2.AddElem(10);s2.AddElem(60);
        s3.AddElem(30);s3.AddElem(50);s3.AddElem(10);s3.AddElem(60);
        cout << "    La instancia s1 tiene  ";s1.toString();
        cout << "    La instancia s2 tiene  ";s2.toString();
        cout << "    La instancia s3 tiene  ";s3.toString();
        s3.RmvElem(60);
        cout << "    Si retiro 60 de s3 \n";
        cout << "    La instancia s3 tiene : ";s3.toString();
        if (s1.Member(20))
                cout << "       Tenemos elem. 20 en inst. s1\n";
        else
                cout << "    No existe elem. 20 en inst. s1\n";
        Set s4;
        s2.Intersect(&s3,&s4);
        cout << "La interseccion s2, s3 es: ";s4.toString();
}
```



Concepto de cadena

Una cadena de caracteres es un array conteniendo caracteres, tales como «ABCDEFG», donde el elemento siguiente al último carácter es un carácter nulo. Este carácter nulo es el '\0'. La sentencia de asignación siguiente almacena una cadena en cadena[]; El octavo elemento, cadena[7], contiene '\0'

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

Es importante comprender la diferencia entre un array dc caracteres y una cadena de caracteres. Las cadenas contienen un carácter nulo al final del array de caracteres. Esto permite su tratamiento como tales por medio de la biblioteca string.h

Lectura de cadenas.

El método recomendado es usar la función miembro getline(), asociado al objeto cin, de la clase iostream, contenidas en la biblioteca iostream.h; Esta función permite leer cadenas conteniendo espacios en blanco.

La función **getline()** utiliza tres argumentos. El primer argumento es el identificador de la variable cadena (nombre de la cadena). El segundo argumento es la longitud máxima de cadena (el número máximo de caracteres que se leerán), que debe ser al menos dos caracteres mayor que la cadena real, para permitir el carácter nulo ' \ o ' y el ' \n'. Por último, el carácter separador se lee y almacena como el siguiente al último carácter de la cadena. La función getline () inserta_automáticamente el carácter nulo como indicador de fin de cadena.

```
#include <iostream.h>
void main(){
    char Nombre[80];
    cout << "Introduzca su nombre ";
    cin.getline(Nombre, sizeof(Nombre)-2);
    cout <<"Hola "<< Nombre <<", como está Ud. ? \n";
}
```

La función cin. get() se utiliza para leer carácter a carácter. La llamada Cin.get (car) copia el carácter siguiente del flujo de entrada cin en la variable car y devuelve 1, a menos que se detecte el final del archivo, en cuyo caso se devuelve 0.

El siguiente programa cuenta las ocurrencias de la letra 'l' en el flujo de entrada.

```
#include <iostream.h>
void main(){
       char car=' ';
       int cuenta = 0;
       cout<<"Introduzca caracteres, salida: *\n";
       while (car!='*'){
           cin.get(car);
           if (car == 'l') ++cuenta;
       }
       cout<<"He contabilizado "<< cuenta << " letras l\n";
};
```

ARRAYS y CADENAS COMO PARÁMETROS DE FUNCIONES

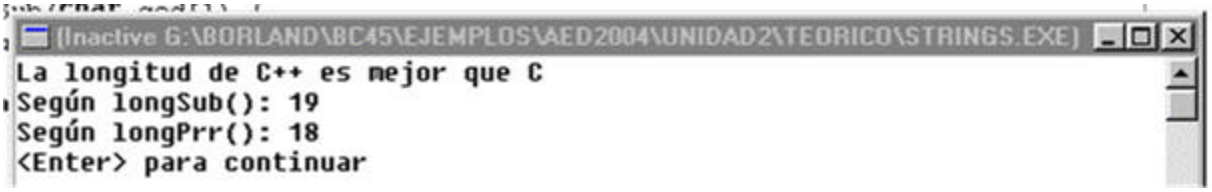
Los arrays y cadenas se pueden pasar sólo *por referencia,* no por valor. En la función, las referencias a los elementos individuales se hacen por indirección de la dirección del objeto. Considérese el programa C++ que implementa una función Longitud () que calcula la longitud de una cadena terminada en nulo.

El parámetro cad se puede declarar como un array de caracteres de tamaño desconocido.

```
#include <iostream.h>
int longSub(char cad[]) {
        int cuenta = 0;
        while (cad[cuenta++]!='\0');
        return cuenta;
}
```

El parámetro cad se puede declarar como un array de caracteres de tamaño desconocido.

```
int longPtr(char *cad) {
        int cuenta = 0;
        while (*cad++!='\0')cuenta++;
        return cuenta;
}

void main(void){
        char arr[] = "C++ es mejor que C";
        cout << "La longitud de " << arr << endl;
        cout << "Según longSub(): " << longSub(arr) << endl;
        cout << "Según longPrr():  " << longPtr(arr) << endl;
        cout << "<Enter> para continuar";
}
```



Cual de las dos dice la verdad ? longSub() o longPtr(). Por que ?

Otra duda. Recién decimos que arrays y cadenas se pueden pasar sólo *por referencia,* no por valor. Sin embargo, en la declaración de longSub(char cad[]) no aparece el & imprescindible a las declaraciones por referencia. Cual será la verdad ?

- Lo estamos pasando por valor
- Lo estamos pasando por referencia

Esto amerita una prueba

```
#include <iostream.h>

typedef char car20[20];

void prueRef1(char cad[]) {
      int cuenta  = 0;
      while (cad[cuenta]!='\0')cad[cuenta++]='A';
}

void prueRef2(car20 &cad) {
      int cuenta  = 0;
      while (cad[cuenta]!='\0')cad[cuenta++]='B';
```

```
}


void main(void){
	car20 cad = "C++ es mejor que C";
	cout << "cad contiene: " << cad << endl;
	cout << "Procesamos prueRef1() \n";
	prueRef1(cad);
	cout << "cad contiene: " << cad << endl;
	cout << "Procesamos prueRef2() \n";
	prueRef2(cad);
	cout << "cad contiene: " << cad << endl;
	cout << "<Enter> para continuar";
}
```

... no quedan dudas. Se lo pasa por referencia, siempre

Uso de punteros para pasar una cadena

Los punteros se pueden utilizar para pasar arrays a funciones.

```
#include <iostream.h>

//	Función extraer copia num_cars caracteres
//	de la cadena fuente a la cadena destino

int extraer(char *dest, char *fuente, int num_cars){
	int cuenta;
	for(cuenta = 1; cuenta <= num_cars; cuenta++)
		*dest++ = *fuente++;
	*dest   = '\0';
	return cuenta; // devuelve número de caracteres
}

void main(void){
	char s1[40] =  "Sierras de Cordoba";
	char s2[40];
	cout << "Iniciando prueba "<<endl;
	cout << "s1 conteniendo: "<< s1 <<endl;
	extraer(&s2[0], &s1[0], 15);
	cout <<"proceso extraer() " << endl;
	cout <<"s2 conteniendo: " << s2 <<endl;
}
```

Observe que en las declaraciones de parámetros, ningún array está definido, sino punteros de tipo char. En la línea *dest++ = *fuente++; los punteros se utilizan para acceder a las cadenas fuente y destino, respectivamente. En la llamada a la función extraer se utiliza el operador & para obtener la dirección de las cadenas fuente y destino.

ASIGNACIÓN DE CADENAS

C++ soporta dos métodos para asignar cadenas. Uno de ellos ya se ha visto anteriormente cuando se inicializaban las variables de cadena.

El segundo método para asignación de una cadena a otra es utilizar la función strcpy. La función strcpy copia los caracteres de la cadena fuente a la cadena destino. La función supone que la cadena destino tiene espacio suficiente para contener toda la cadena fuente. El prototipo de la función es:

char* strcpy(char* *destino,* const char* *fuente)*

EJEMPLO 11.7

```
char nombre[41J;
strcpy(nombre, "Cadena a copiar");
```

La función strcpy() copia "Cadena a copiar" en la cadena nombre y añade un carácter nulo al final de la cadena resultante. El siguiente programa muestra una aplicación de strcpy ()

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

void main (){
      char s[100] = "Buenos días Mr. Mackoy", t[100];
      strcpy(t, s);
      strcpy(t+12, "Mr. C++");
      cout << s << endl << t << endl;
}
```

La función **strncpy,** su prototipo:

char* strncpy(char* *destino,* const char* *fuente,* size_t num*);*

y su propósito es copiar *num* caracteres de la cadena fuente a la cadena destino. La función realiza truncamiento o rellenado de caracteres si es necesario.

La función strlen()

La función strlen calcuta el número de caracteres del parámetro cadena, excluyendo el carácter nulo de terminación de la cadena. Ejemplo:

```
#include <string.h>
#include <iostream.h>
void main(){
          char a[] = "ABCBEFG";
          cout << "strlen(a) "<< a << " es "<< strlen(a) << endl;
};
```

Las funciones strcat y strncat

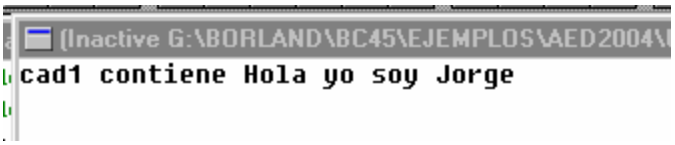
En muchas ocasiones se necesita construir una cadena, añadiendo una cadena a otra cadena, operación que se conoce corno *concatenación.* Las funciones strcat() y strncat() realizan operaciones de concatenación. strcat anade el contenido de la cadena fuente a la cadena destino, devolviendo un puntero a la cadena destino. Ejemplo:

```
char cadena[81];
strcpy)cadena, "Borland")
strcat)cadena, " C++")
```

La variable *cadena* contiene ahora "Borland C++".

Es posible limitar el número de caracteres a copiar utilizando la función strncat. La función strncat añade *nun* caracteres de la cadena fuente a la cadena destino y devuelve el puntero a la cadena destino. Ejemplo:

```
#include <iostream.h>
#include <string.h>
void main(){
        char cad1[40] = "Hola yo soy ";
        char cad2[40] = "Jorge Pablo ";
        strncat(cad1, cad2, 5);
        cout << "cad1 contiene " << cad1 << endl;
};
```



Comparación de cadenas

La biblioteca string.h proporciona un conjunto de funciones que comparan cadenas. Estas funciones comparan los caracteres de dos cadenas utilizando el valor ASCII de cada carácter. Las funciones de comparación son: strcmp, stricmp, strncmp y strnicmp.

11.7.1.La función strcmp

Cuando se desea determinar si una cadena es igual, mayor o menor que otra, se debe utilizar la función strcmp (). strcmp () compara su primer parámetro con su segundo. y devuelve 0 si las dos cadenas son idénticas, un valor menor que cero si la primer cadena es menor que la segunda, o un valor mayor que cero si la primera es mavor que la segunda (los términos *«mayor que»* y *«menor que»* se refieren a la ordenación alfabética de las cadenas). Por ejemplo, Arequito es menor que Santisimo. Ejemplo:

```
#include <string.h>
#include <iostream.h>
void main(){
        char cad1[] = "Nicrosoft C++";
        char cad2[] = "Nicrosoft Visual C++" ;
        cout  << "retorno strcmp(cad1,cad2) es " << strcmp(cad1,cad2) << endl;
        cout  << "retorno strcmp(cad2,cad1) es " << strcmp(cad2,cad1) << endl;
        cout  << "retorno strcmp(cad1,cad1) es " << strcmp(cad1,cad1) << endl;
}
```

```
(Inactive G:\BORLAND\BC45\EJEMPLOS\AED2004\UNIDAD2\CADENAS\NONAME01....
retorno strcmp(cad1,cad2) es -19
retorno strcmp(cad2,cad1) es 19
retorno strcmp(cad1,cad1) es 0
```

La comparación se realiza examinando los primeros caracteres dc cadi y cadi; a continuación, los siguientes caracteres, y así sucesivamente. Este proceso termina cuando:

- Se encuentran dos caracteres distintos del mismo orden
- Se encuentra el caracter nulo en una de las cadenas.

Vamos a suponer que no existe string.h, la biblioteca C++ que se ocupa del todo lo relacionado con el comportamiento de cadenas, o hileras, como Uds quieran llamarlas ...

Entonces tenemos que crear este comportamiento. No todo, hay algo hecho. Concretamente, lo que le pedimos es codificar las funciones miembro ***invierto()****, para invertir un objeto cadena dado, y* ***contiene(cadena)****, que cuenta cuantas veces cadena parámetro está contenida en el objeto cadena invocante.*

Ud. tiene los prototipos y el main. Falta esa codificación ...

```
#include <iostream.h>
class MisCadenas{
private:
        char cadena[80];
public:
        MisCadenas();                   // Constructor sin argumentos
        int longCadena();               // Semejante a strlen(...) de string.h
        void invierto();                // Invierte cadena
        int MenIgMay(MisCadenas);       // Semejante a strcmp(...) de string.h
        void toString();                // Exhibiendo cadena
        int esCadena();                 // Verifica si el array contiene cadena
        int contiene(MisCadenas);       // Cuantas veces está contenida
        int indConten(MisCadenas);      // Indice donde inicia contención
};

MisCadenas::MisCadenas(){
        cout <<"Tipee cadena, por favor ";
        cin.getline(cadena, 78);    // Inserta '\0' al fin de la cadena
}

void MisCadenas::toString(){
        cout << "Objeto contiene: "<<cadena<<endl;
}

int MisCadenas::esCadena(){    // Un array de caracteres será cadena si
        int cad=0;              // termina en nulo
        int tamanio=sizeof(cadena);
        for(int ind=0;ind<=tamanio;ind++)if(cadena[ind]=='\0'){
                cad = 1;
                break;
        }
        return cad;
}

int MisCadenas::longCadena(){
```

```
        int longitud=0;
        for(int ind=0;cadena[ind]!='\0';ind++)longitud++;
        return longitud;
}

void main(void){
        MisCadenas cad1, cad2;
        cad1.toString();
        if(cad1.esCadena())     cout <<"(es cadena)"<<endl;
        else                    cout <<"(!es cadena)"<<endl;
        cout <<"Su longitud: " << cad1.longCadena() << endl;;
        cad2.toString();
        if(cad1.esCadena())     cout <<"(es cadena)"<<endl;
        else                    cout <<"(!es cadena)"<<endl;
        cout <<"Su longitud: " << cad2.longCadena()<<endl;
        cout <<"Segunda cadena contenida: "<<cad1.contiene(cad2)<<" veces \n";
        cad2.invierto();
        cout <<"Invertida "; cad2.toString();
}
```

BÚSQUEDA EN ARRAYS

Con frecuencia es necesario determinar si un elemento de un array contiene un valor que coincide con un determinado *valor clave.* El proceso de determinar si este elemento existe se denomina búsqueda. Existen distintas técnicas de búsqueda, pero podemos decir que esencialmente son dos: secuencial y binaria.

Un ejemplo de la vida cotidiana. Estamos comprando en un supermercado. El operador de caja presenta el producto al lector de código de barras. Y el sistema le devuelve el valor que se imprime en el ticket.
Normalmente el valor no es lo que está codificado en las barras. El valor puede variar, el producto puede ponerse en oferta, o encarecerse. Lo que viene en el código de barras es el código del producto. (perdón por la redundancia).
El código de producto está asociado a su valor. Hay muchas maneras de almacenar esta asociación, pero como estamos estudiando arrays vamos a suponer tenemos un array de items conteniendo asociaciones asociaciones código /valor. Entonces tenemos que ***buscar el código*** *en el array para obtener su valor asociado.*

Antes que me olvide. Si vamos a tener un array de items código / valor, necesitamos, primero de una clase con lo relacionado con el comportamiento mínimo de los items. Inicializar, leer. Será nuestra class Item*. Y bueno, si esos items estarán almacenados en un array de items, tendremos que preocuparnos por la obtención de área, carga de items en el array, mostrarlos ... definiremos todo esto en la* ***class ArrItems****. Una vez que tengamos todo esto funcionando comenzaremos a aprovecharlo en las aplicaciones clásicas de los arrays: buscar, ordenar, actualizar, etc, etc .....*

```
#include <stdlib.h>
#include <iostream.h>
class Item {                    // Una clase de claves asociadas a un valor ...
private:                        // parte privada
  int    codigo; float valor;
public:
  Item(){};                     // Constructor sin argumentos
  Item(int, float);             //  Otro
  void  toString();             // Exhibimos
```

```
  int   getCodigo(){return codigo;}
  float getValor(){return valor;}
  void  setCodigo(int);
  void  setValor(float);
  int    esMayor(Item);          //  Es mayor el obj. invocante que el parámetro ?
  int    esMenor(Item);          //  Es menor el obj. invocante que el parámetro ?
  void  intercambio(Item &);    //  Intercambiamos objetos invocante/parámetro
};

Item::Item(int cod, float val){         //  Constructor de la clase
  codigo=cod;
  valor=val;
}

void Item::setCodigo(int cod){codigo=cod;};
void Item::setValor (float val){valor=val;};
void Item::toString(){
  cout << codigo << " - "<< valor << endl;
}

int Item::esMayor(Item item){        // Es mayor el obj. invocante que el parámetro ?
return(codigo > item.codigo?1:0);
                        // Si clave del objeto invocante > clave objeto parámetro,
}                       // retorno 1, caso contrario 0;

int Item::esMenor(Item item){        //  Es menor el obj. invocante que el parámetro ?
return(codigo < item.codigo?1:0);
                        // Si clave del objeto invocante < clave objeto parámetro,
}                       // retorno 1, caso contrario 0;

void Item::intercambio(Item &item){         //  Intercambiamos objetos invocante/parámetro
  Item burb=item;               // Guardamos el objeto parámetro en burb
  item = *this;                 // Asignamos el invocante al objeto parámetro
  *this = burb;                 // Terminamos la inversión
}
```

La clase ArrItems, con comportamiento mínimo para manipular un array de items:

```
# include <G:\Borland\BC45\Ejemplos\AED2004\Unidad3\Items\Items.cpp>
class ArrItems{          // Una clase de implementación de un
protected:               //  array de items, comportamiento mínimo ...
  Item *ptrItem;         //  Un puntero a objetos de la clase Item
  int talle;             //  Tamaño del array de objetos Item
public:
  ArrItems(int, char);    //      Un constructor
  virtual void toString();// Mostrar items, los 10 primeros
};

ArrItems::ArrItems(int tam, char tipo='R') {// Constructor de un array de Item's
  int    auxCod;              // Una variable auxiliar
  talle=tam;                                  // inicializamos talle (tamaño)
  ptrItem = new Item[talle];    // Generamos el array
  for(int i=0;i<talle;i++){      // la llenaremos de Item's, dependiendo
        switch(tipo){           // del tipo de llenado requerido
                case 'A':{       // los haremos en secuencia ascendente
                        auxCod = i;
                        break;
                }
                case 'D':{       // o descendente ...
                        auxCod = talle - i;
                        break;
                }
```

```
                case 'R':{      // o bien randómicamentente (Al azar)
                        auxCod = random(talle);
                }
        }
        ptrItem[i].setCodigo(auxCod);
        ptrItem[i].setValor((float)random(talle));
  }
}

void ArrItems::toString(){
int ctos = (talle < 10 ? talle : 10);
cout << ctos << " Primeros elementos \n";
for(int i=0;i<ctos;i++)
  ptrItem[i].toString();
}
```

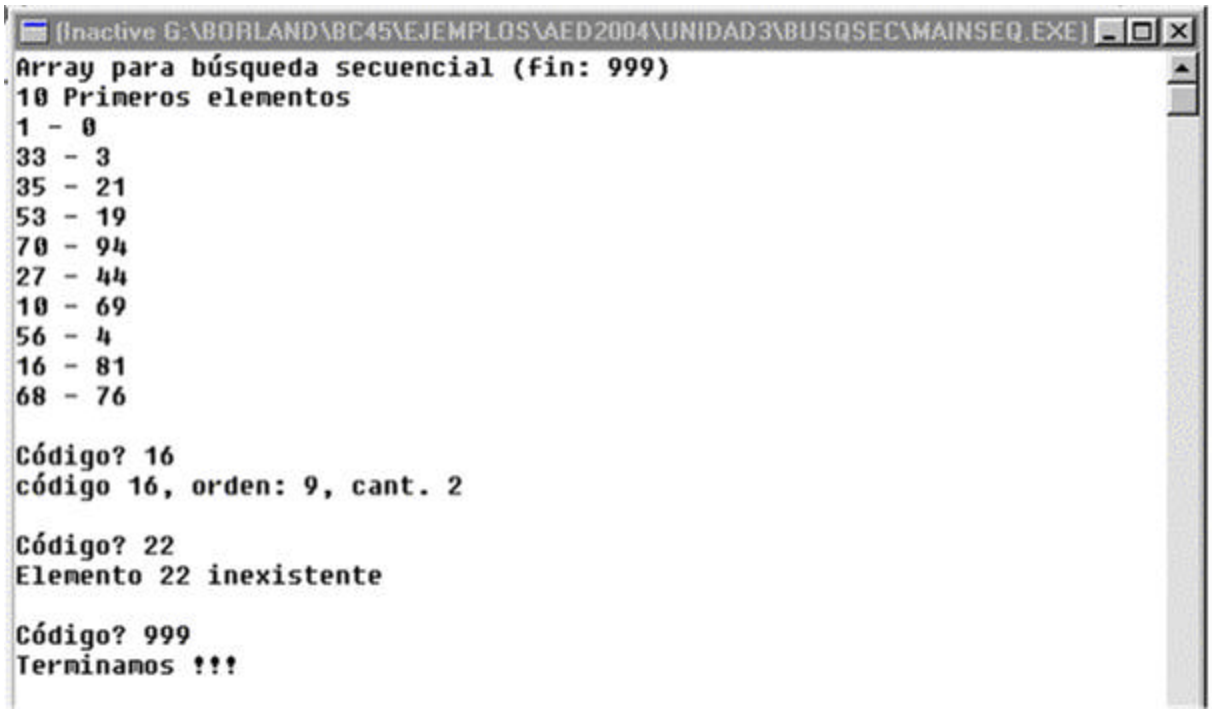
Búsqueda secuencial

La búsqueda secuencial verifica la existencia de un valor denominado clave en un array. En una búsqueda secuencial los elementos de una lista o vector se exploran (se examinan) en secuencia, uno después de otro. Es el único método de búsqueda cuando el array no está ordenado.
El algoritmo de búsqueda secuencial compara cada elemento del array con la *clave* de búsqueda. Dado que el array no está en un orden prefijado, es probable que el elemento a buscar pueda ser el primero el último o cualquier otro. De promedio, al menos el programa tendrá que comparar la clave de búsqueda con la mitad de los elementos del array. Este método de búsqueda, cuyo algoritmo es sencillo, es adecuado con arrays pequeños o no ordenados.

En la implementación que sigue heredamos de class ArrItems. Hacemos esto porque el array en el que buscaremos algunas claves "es un" array de items, y entonces nos conviene implementar esta relación mediante herencia. Haciendo esto el constructor de BusqSec() invoca al constructor de la clase base, quien hace todo el trabajo. También aprovechamos el método toString().

```
# include <G:\Borland\BC45\Ejemplos\AED2004\Unidad3\ArrItems\ArrItems.cpp>
class BusqSeq:public ArrItems{      // Una clase de busqueda seccuencial en un
public:                             // array de items
  BusqSeq(int, char);               // Un constructor
  virtual int existe(int);          // Existe la clave parámetro ?
  int cuantas(int);                 // Cuantas veces existe la clave ?
  virtual void implementar();       // Implementando el demo
};

BusqSeq::BusqSeq(int cant, char tipo = 'R'):ArrItems(cant,tipo){};
        // El constructor de la clase base hace todo el trabajo

int BusqSeq::existe(int clave){
  for(int i=0;i<talle;i++)
        if(clave==ptrItem[i].getCodigo())return i+1;
  return 0;
}

int BusqSeq::cuantas(int clave){    //  Queremos saber cuantos son igual a clave
  int igual=0;                      //  Cuantos iguales ...
  for(int i=0;i<talle;i++)
        if(clave==ptrItem[i].getCodigo())igual++;
  return igual;
};

void BusqSeq::implementar(){        //  Implementando el demo de la búsqueda secuencial
  int cProd, orden,cant;
  cout  << "Array para búsqueda secuencial (fin: 999)\n";
```

```
  toString();
  cout  << "\nCódigo? ";
  cin   >> cProd;
  while (cProd != 999){
        orden=existe(cProd);
        cant =cuantas(cProd);
        if(orden)
              cout<<"código "<< cProd <<", orden: "<<orden<<", cant. "<< cant <<endl;
        else
              cout<<"Elemento "<< cProd <<" inexistente \n";
        cout    << "\nCódigo? ";
        cin     >> cProd;
  }
cout<<"Terminamos !!!\n";
}

# include <G:\Borland\BC45\Ejemplos\AED2004\Unidad3\BusqSec\BusqSeq.cpp>
void main(){
  BusqSeq busqSeq(100,'R'); // Generamos un array de 100 objetos item random
  busqSeq.implementar();
};
```



Búsqueda binaria

La búsqueda secuencial se aplica a cualquier array. Si está ordenado, la *búsqueda binaria* proporciona una técnica de búsqueda mejorada. Una búsqueda binaria típica es la búsqueda de un número en un directorio telefónico o de una palabra en un diccionario. Dado la palabra, se abre el libro cerca del principio, del centro o del final dependiendo de la primera letra de la palabra que se busca. Se puede tener suerte y acertar con la página correcta; pero, normalmente, no será así y el lector deberá probar en páginas anteriores o posteriores.
La misma idea se aplica en la búsqueda en un array ordenado. Nos posicionamos en el centro del array y se comprueba si nuestra clave coincide. Si no, tenemos dos situaciones:

clave mayor: debemos buscar en el tramo superior.
clave menor: la búsqueda será en el tramo inferior.

y este razonamiento se aplicará las veces necesarias hasta encontrar igual o definir que la clave no existe.

Se desea verificar si el elemento 225 se encuentra en el siguiente array:

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
13	44	75	100	120	275	325	510

El punto central del a lista es el elemento a [3] (100). El valor que se busca es 225 que es mayor que 100; por consiguiente, la búsqueda continúa en la mitad superior.

a[4]	a[5]	a[6]	a(7]
120	275	325	510

Ahora el último valor de la primera mitad de esta secuencia es 275; por consiguiente, el valor debe estar localizado en la secuencia

a[4]	a(S)
120	275

El último valor de la primera mitad que buscamos es 120, que es mas pequeño que el valor que se está buscando, de modo que se busca en la segunda mitad

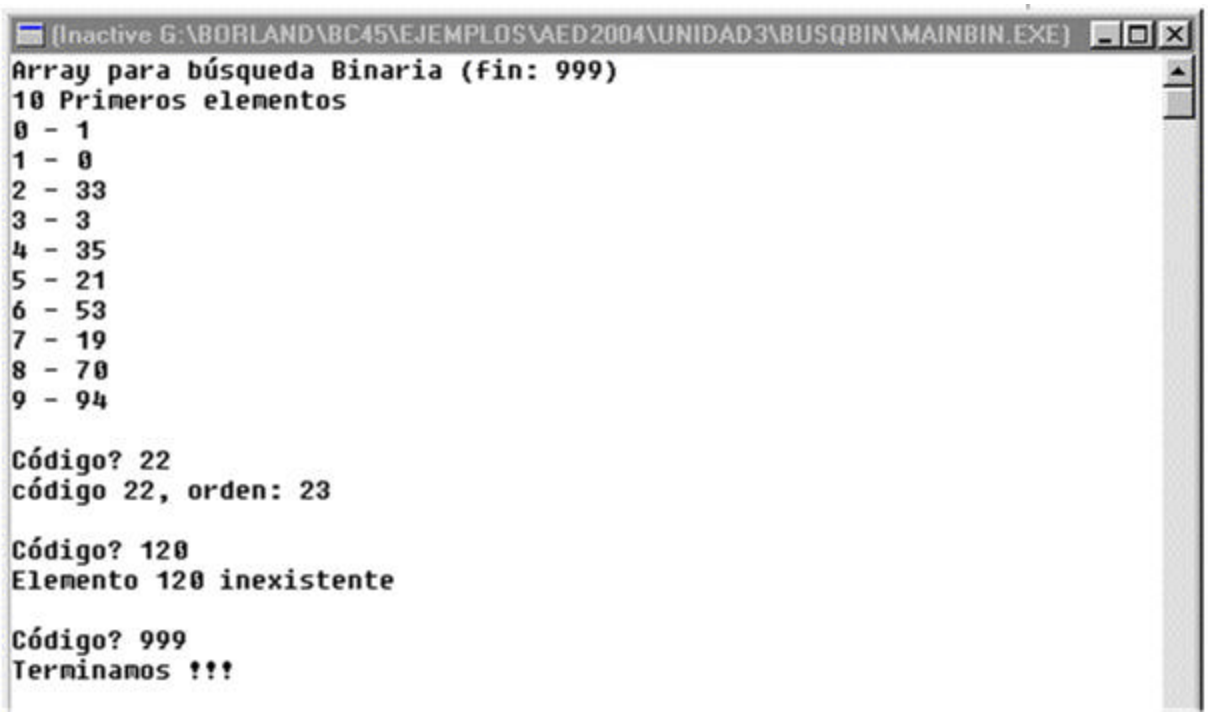
a[5]
275

Se observa, por último, que no ha habido éxito en la búsqueda ya que
225 <>275.

Por las mismas razones que en la búsqueda binaria, nos conviene heredar de class ArrItems.

```
# include <G:\Borland\BC45\Ejemplos\AED2004\Unidad3\ArrItems\ArrItems.cpp>

class BusqBin : public ArrItems{
public:
     BusqBin(int, char);          //        Un constructor
     virtual int existe(int);       // Existe la clave parámetro ?
     virtual void implementar();// Implementando el demo
};


BusqBin::BusqBin(int tam, char tipo='A'):ArrItems(tam,tipo){;}

int BusqBin::existe(int clave){
     int alt=talle-1,baj=0;
     int indCent, valCent;
     while (baj <= alt){
        indCent = (baj + alt)/2;                        // índice de elemento central
        valCent = ptrItem[indCent].getCodigo();         // valor del elemento central
        if (clave == valCent)                           // encontrado valor;
                return indCent+1;                       // devuelve orden
        else if (clave < valCent)
                        alt = indCent - 1;               //       ir a sublista inferior
                else  baj = indCent + 1;                // ir a sublista superior
     }
     return 0;                                          // elenento no encontrado
};

void BusqBin::implementar(){
     int cProd, orden;
     cout       << "Array para búsqueda Binaria (fin: 999)\n";
     toString();
     cout       << "\nCódigo? ";
     cin   >> cProd;
     while (cProd != 999){
```

```
        orden=existe(cProd);
        if(orden)
                cout<<"código "<< cProd <<", orden: "<<orden<<endl;
        else
                cout<<"Elemento "<< cProd <<" inexistente \n";
        cout    << "\nCódigo? ";
        cin     >> cProd;
    }
cout<<"Terminamos !!!\n";
}

# include <G:\Borland\BC45\Ejemplos\AED2004\Unidad3\BusqBin\BusqBin.cpp>
void main(){
    BusqBin buscar(100,'A'); // Generamos array c/100 objetos item ascend
    buscar.implementar();
};
```



Comparación de la búsqueda binaria y secuencial

La diferencia en tiempo entre los algoritmos de búsqueda secuencial y binaria se va haciendo espectacular a medida que crece el tamaño del array. Tengamos presente que:

En el caso de la búsqueda secuencial en el peor de los casos coincidirá el número de elementos examinados con el número de elementos del array. O sea que en este caso el tiempo de búsqueda es del orden de n y se expresa O(n)

En el caso de la búsqueda binaria, realizamos comparaciones con el elemento medio del array y subsarrays resultantes de la partición expuesta en el punto anterior. El array es partido en dos antes de cada comparación. Muy rápidamente llegamos al elemento buscado o a la conclusión de su inexistencia. La función matemática que nos da el número máximo de comparaciones (peor caso) resultante de esta mecánica de sucesivas particiones es $\log_2 n$. Entonces el tiempo de búsqueda es del orden $\log_2 n$ y se expresa $O(\log_2 n)$.

Números de comparaciones considerando el peor caso

Tamaño array	Búsqueda binaria	Búsqueda secuencial
1	1	1

10	4	10
1.000	11	1.000
5.000	14	5.000
100.000	18	100.000
1.000.000	21	1.000.000

Pero entonces, si la búsqueda binaria es tanto mejor, no es mucho mejor ordenar el array antes ?

Si y no ... ordenar el array ***da mucho mas trabajo*** *que buscar secuencialmente. Hay muchos métodos de ordenamiento, los mas simples, llamados básicos, comparan elementos de a pares y van intercambiando si el par no está en el orden deseado. Y hacen falta varias pasadas para que el array quede ordenado. Entonces, si vamos a buscar varias veces, conviene, si no no*

ORDENAMIENTO - Introducción

Muchas actividades humanas requieren que a diferentes colecciones de elementos utilizados se pongan en un orden específico. Las oficinas de correo y las empresas de mensajería ordenan el correo y los paquetes por códigos pos-tales con el objeto de conseguir una entrega eficiente; los anuarios o listines telefónicos se ordenan por orden alfabético de apellidos con el fin de encontrar fácilmente el número de teléfono deseado. Los estudiantes de una clase de la universidad se ordenan por sus apellidos o por los números de matrícula. Una de las tareas que realizan más frecuentemente las computadoras en el procesamiento de datos es la *ordenación.* (Después de todo, no olvidemos que a las computadoras se les llama también ordenadores).
El estudio de diferentes métodos de ordenación es una tarea muy interesante desde un punto de vista teórico y práctico. A continuación estudiaremos varios algoritmos de ordenamiento.

Algoritmos de ordenamiento básicos y mejorados.

Es usual dividir los algoritmos de ordenamiento en dos grupos. Los básicos, de codificación relativamente simple, y los mejorados, de muy alta eficiencia en su tarea de ordenar.
De los básicos estudiaremos el Ordenamiento por Burbuja, el mas simple (e ineficiente) que existe, y el Ordenamiento por Sacudida, algo mas complejo, pero bastante mejor.
Los llamados Mejorados, no son algo mejores que los básicos, son 10 o mas veces mas rápidos, de ellos veremos el Algoritmo de Peinado, de invención bastante reciente, muy simple y eficiente. Anteriores a él hay varios, citemos al Shell, HeapSort y QuicSort. El Quick es el mas rápido de todos, (solo levemente que el de Peinado). Todos ellos son mas complejos que el Peinado.

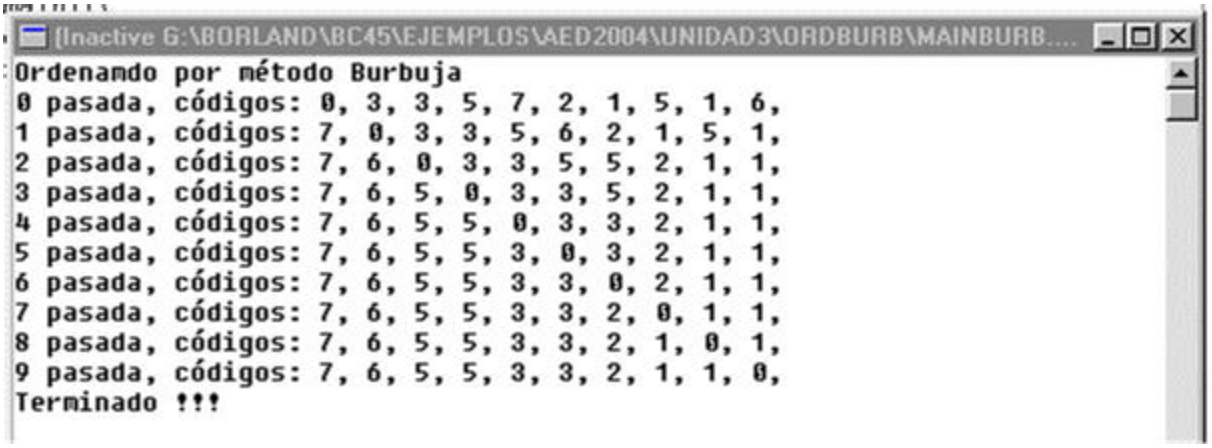
Ordenamiento por Método Burbuja.

El método de ***Ordenación por Burbuja***, muy popular y conocido por los estudiantes de programación, es ***el menos eficiente*** de todos.
La técnica utilizada se denomina ordenación por hundimiento debido a que los valores mayores burbujean (suben hacia la cima) del array. Se comparan elementos de a pares, y si el array tiene n elementos, el método realiza n-1 pasadas. Como en cada pasada el mayor "burbujea" hasta la cima, cada pasada sucesiva es de un elemento menos. El método no tiene capacidad de detectar si el array ya esta ordenado. Esto significa que si el array hace n-1 pasadas siempre, aunque el array esté ordenado de partida.

Como en lo visto para búsqueda, por las mismas razones, nos conviene definir class OrdBurb heredando de class ArrItems. Para trazar como va quedando el array tras sucesivas pasadas, vamos a incluir un método mostArr() que muestre en una línea los 10 primeros elementos del array. Y a este método lo usaremos opcionalmente dentro del burbuja, para ir trazando el avance del ordenamiento.

```
# include <G:\Borland\BC45\Ejemplos\AED2004\Unidad3\ArrItems\ArrItems.cpp>
```

```
class OrdBurb:public ArrItems{      //Una clase de ordenamiento de un array de items
private:                            // mediante el método Burbuja
        void mostArr(int);          // Mostrando los valores del array
public:
        OrdBurb(int, char);         //Un constructor
        void ordenar(int);          // El método de ordenamiento
        void implementar();         // Implementando el demo
};

OrdBurb::OrdBurb(int cant, char tipo = 'R'):ArrItems(cant,tipo){};
         // El constructor de la clase base ArrItems nos genera un array
         // cant items desordenado

void OrdBurb::mostArr(int pas){   // Mostramos el array, solo los códigos
        int cuant=(talle < 10 ? talle : 10);  // Lo que sea menor
        cout << pas <<" pasada, códigos: ";
        for (int i=0;i<cuant;i++)
         cout << ptrItem[i].getCodigo() <<", ";
        cout << endl;
}

void OrdBurb::ordenar(int trace=0){ // Si no pasamos parámetro, no
        int i,j;                    // trazamos el ordenamiento
        if (trace) mostArr(0);      // Mostramos el array antes de ordenarlo
        for (i = 1; i<talle; i++){  / Ordenando - 1 pasadas
           for (j = talle-1; j>=i; j--)     // realizando la pasada
             if (ptrItem[j].esMayor(ptrItem[j-1])) // Si invocante es mayor que parámetro
                 ptrItem[j].intercambio(ptrItem[j-1]); // intercambio
           if (trace) mostArr(i);           // Mostramos el array después de la pasada
        }
}

void OrdBurb::implementar(){
        cout     << "Ordenamdo por método Burbuja\n";
        ordenar(1);     // queremos ver el ordenamiento
        cout     << "Terminado !!!\n";
}

# include <G:\Borland\BC45\Ejemplos\AED2004\Unidad3\ordBurb\ordBurb.cpp>
void main() {
      OrdBurb burb(10,'R'); // Generamos un array de 10 objetos item random
      burb.implementar();
};
```



Como puede apreciarse, el ordenamiento es en secuencia descendente.
Por la forma de trabajar, los valores altos fluyen rápidamente a la izquierda del array.

El 7, en la primera pasada. El 6, que estaba último, en la segunda. Pero el 0 necesita de todas las pasadas para ir a su lugar, a la extrema derecha.
De todas maneras, en el método burbuja esto no importa mucho ya que él ejecutará n-1 pasadas así sea que el array esté ordenado antes de comenzar.

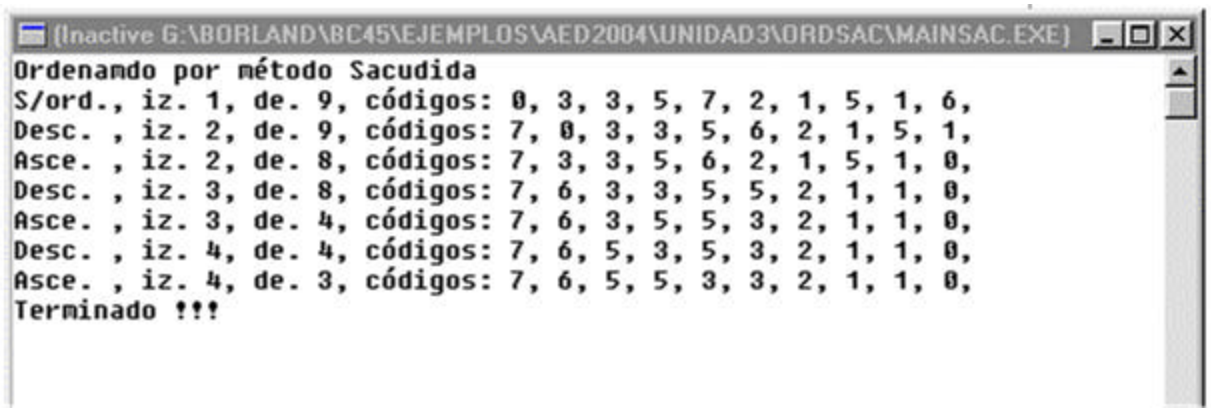
*El Método **"sacudida**" que veremos ahora incorpora un par de ventajas:*

- *Alterna pasadas de izquierda a derecha y viceversa, con lo que consigue que tanto menores fluyan a derecha como mayores a izquierda con igual velocidad*
- *Lleva un registro de donde fue la última inversión realizada en la pasada anterior, esto le posibilita no recorrer tramos de la pasada innecesariamente.*

Ordenamiento por Método "sacudida"

```
# include <G:\Borland\BC45\Ejemplos\AED2004\Unidad3\ArrItems\ArrItems.cpp>
class OrdSac:public ArrItems{//         Una clase de ordenamiento de un array de items
private:                        // mediante el método Sacudida
        void mostArr(int,int,int); // Mostrando los valores del array
public:
        OrdSac(int, char);      //      Un constructor
        void ordenar(int);      // El método de ordenamiento
        void implementar();     // Implementando el demo
};

OrdSac::OrdSac(int cant, char tipo = 'R'):ArrItems(cant,tipo){};
                // El constructor de la clase base ArrItems nos genera un array
                // cant items desordenado

void OrdSac::mostArr(int pas,int iz, int de){// Mostramos el array
        int cuant=(talle < 10 ? talle : 10);  // Lo que sea menor
        if(pas==0) cout  <<"S/ord., iz. "<<iz<<", de. "<<de<<", códigos: ";
        if(pas==1) cout  <<"Desc. , iz. "<<iz<<", de. "<<de<<", códigos: ";
        if(pas==2) cout  <<"Asce. , iz. "<<iz<<", de. "<<de<<", códigos: ";
        for (int i=0;i<cuant;i++)
                cout << ptrItem[i].getCodigo() <<", ";
        cout << endl;
}

void OrdSac::ordenar(int trace=0)
{       int j,k = talle-1,iz = 1,de = talle-1; Item aux;
        if (trace) mostArr(0,iz,de);  // Mostramos el array antes de su ordenamiento
        do {                            // Ciclo de control de pasadas
                for (j = de; j>=iz; j--) // Pasada descendente
                        if (ptrItem[j].esMayor(ptrItem[j-1])){
                                ptrItem[j].intercambio(ptrItem[j-1]);
                                k = j;      // Guardamos el lugar del último intercambio
                        }
                iz = k+1;
                if (trace) mostArr(1,iz,de);  // Mostramos el array despues de la pasada
                for (j = iz; j<=de; j++)   // Pasada ascendiente
                        if (ptrItem[j].esMayor(ptrItem[j-1])){
                                ptrItem[j].intercambio(ptrItem[j-1]);
                                k = j;              // Guardamos el lugar del último intercambio
                        }
                de = k-1;
                if (trace) mostArr(2,iz,de);  // Mostramos el array despues de la pasada
  } while (iz<=de);
```

```
}



void OrdSac::implementar(){
        cout    << "Ordenamdo por método Sacudida\n";
        ordenar(1);     // queremos ver el ordenamiento
        cout    << "Terminado !!!\n";
}

# include <G:\Borland\BC45\Ejemplos\AED2004\Unidad3\ordSac\ordSac.cpp>
void main(){
        OrdSac sac(10,'R'); // Generamos un array de 10 objetos item random
        sac.implementar();
};
```

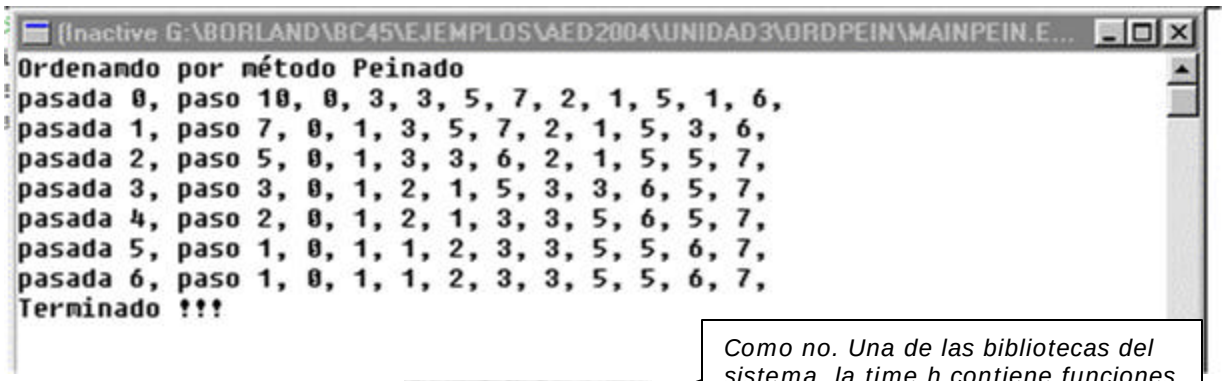


Ordenamiento por Método "Peinado"

Es uno de los métodos mejorados. Es muy veloz, pero no consigue destronar al Quick Sort ó rápido. Se variante principal respecto a los métodos ya vistos es el concepto de paso: lo usa para comparar elementos no contiguos y lo va ajustando pasada tras pasada. Codificación y demo a continuación.

```
# include <G:\Borland\BC45\Ejemplos\AED2004\Unidad3\ArrItems\ArrItems.cpp>
class OrdPein:public ArrItems{//    Una clase de ordenamiento de un array de items
private:                    // mediante el método Peinado
        void mostArr(int); // Mostrando los valores del array
public:
        OrdPein(int, char);//     Un constructor
        void ordenar(int); // El método de ordenamiento
        void implementar();// Implementando el demo
};

OrdPein::OrdPein(int cant, char tipo = 'R'):ArrItems(cant,tipo){};
                // El constructor de la clase base ArrItems nos genera un array
                // cant items desordenado

void OrdPein::mostArr(int paso){// Mostramos el array
        static int pasada=0;
        int cuant=(talle < 10 ? talle : 10);  // Lo que sea menor
        cout << "pasada "<<pasada++<<", paso "<< paso<<", ";
        for (int i=0;i<cuant;i++)
                cout << ptrItem[i].getCodigo() <<", ";
        cout << endl;
}

void OrdPein::ordenar(int trace=0)  //programa de ordenamiento por peinado
{ int i,paso = talle;
```

```
 int cambio;
 if (trace) mostArr(paso);  // Mostramos el array antes de su ordenamiento
 do {
        paso = (int)((float) paso/1.3);
        paso = paso>1 ? paso : 1;
        cambio = 0;
        for (i = 0; i<talle-paso; i++)
              if (ptrItem[i].esMayor(ptrItem[i+paso])){
                    ptrItem[i].intercambio(ptrItem[i+paso]);
                    cambio = 1;
              }
        if (trace) mostArr(paso);  // Mostramos el array luego de una pasada
 } while (cambio || paso>1);
}

void OrdPein::implementar(){
       cout    << "Ordenamdo por método Peinado\n";
       ordenar(1);     // queremos ver el ordenamiento
       cout    << "Terminado !!!\n";
}

# include <G:\Borland\BC45\Ejemplos\AED2004\Unidad3\ordPein\ordPein.cpp>
void main(){
       OrdPein peinado(10,'R'); // Generamos un array de 10 objetos item random
       peinado.implementar();
};
```



Profe, en el apunte dice que los métodos mejorados son mucho, mas de 10 veces mas rápidos que los básicos. Podríamos ver como se comparan ?

Como no. Una de las bibliotecas del sistema, la time.h contiene funciones de calendario que nos permiten obtener fecha y hora del sistema. Lo que tenemos que hacer es registrar la hora de inicio y la de fin y obtener la diferencia. Ah, no olvidemos que los arrays a ordenar deben ser bastante grandes, los procesadores actuales son muy rápidos, sino la diferencia será de 0 segundos ...

COMPARANDO TIEMPOS DE ORDENAMIENTO

Las funciones de calendario (fecha y hora) permiten obtener la hora actual y convertirla de acuerdo a nuestras necesidades. La hora actual se toma siempre del sistema. Estas funciones están todas ellas en la biblioteca time.h y la que necesitamos es la `clock()`, justamente para determinar el intervalo de tiempo entre en inicio y fin del proceso de ordenamiento. Un ejemplo, tomado del Help del lenguaje.

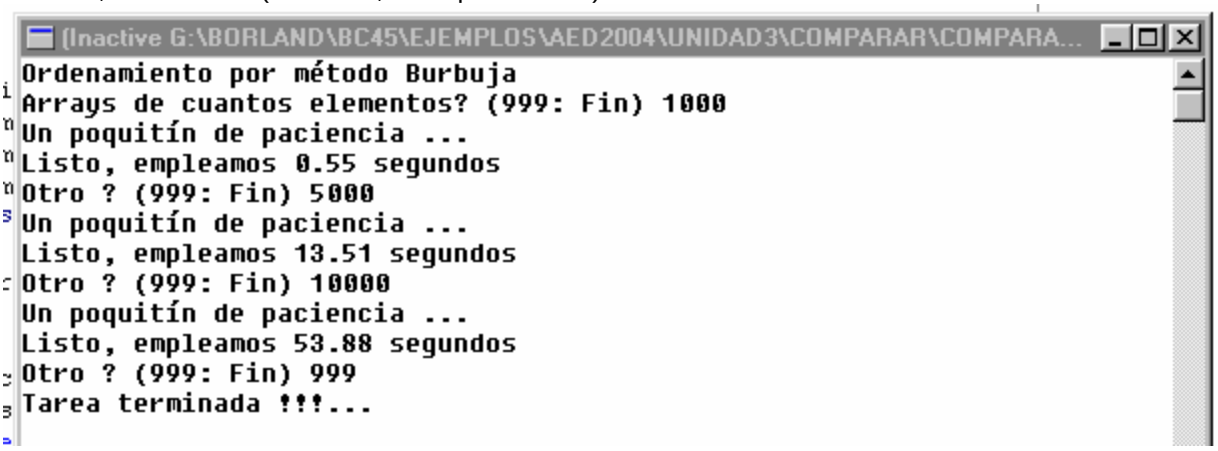
```
/* clock example */

#include <time.h>
#include <stdio.h>
#include <dos.h>

int main(void)
{
  clock_t start, end;
  start = clock();

  delay(2000);

  end = clock();
  printf("The time was: %f\n", (end - start) / CLK_TCK);

  return 0;
}
```

Para hacerlo sencillo, podemos definir una clase que incluya un objeto de una de las clases de ordenamiento ya vistas,(Relación tiene un) y en ella implementar el calculo de tiempo usado por el proceso de ordenamiento. Comencemos por el ordenamiento Burbuja.

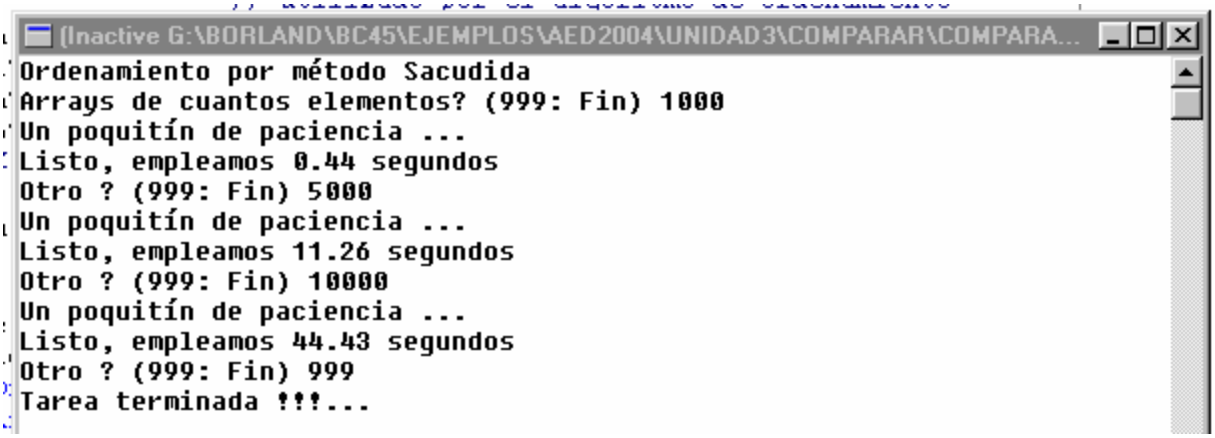
```
#include <time.h>
#include <dos.h>
# include <G:\Borland\BC45\Ejemplos\AED2004\Unidad3\ordBurb\ordBurb.cpp>

class CalcTime{                                 // clase implementadora del cálculo de tiempo
private:                                        // utilizado por el algoritmo de ordenamiento
        clock_t inicio, fin;        // clock_t es un tipo de dato definido en time.h
        void   iniTime(){inicio = clock();}     // Registramos tiempo de inicio
        void   finTime(){fin    = clock();}     // y de finalización
        double proTime(){return (fin - inicio)/CLK_TCK;}
        // CLK_TCK es una constante necesaria para obtener el tiempo en segundos
public:
        void calcular();                        // Implementando el cálculo de tiempos
};

void CalcTime::calcular(){
        int cantElems;
        cout << "Ordenamiento por método Burbuja \n";
        cout << "Arrays de cuantos elementos? (999: Fin) ";
        cin >> cantElems;
        while(cantEle ms!=999){
                cout << "Un poquitín de paciencia ...\n";
                OrdBurb v1(cantElems); // Construimos v1, objeto + array
                iniTime();          // Registramos tiempo de inicio;
                v1.ordenar();
                finTime();          // Registramos tiempo de finalización;
                cout << "Listo, empleamos "<<proTime()<<" segundos"<<endl;
                cout << "Otro ? (999: Fin) ";
                cin >> cantElems;
        }
        cout << "Tarea terminada !!!...\n";
}

void main(){
        CalcTime tiempo;
        tiempo.calcular();
};
```

En el capture abajo, vemos como se van incrementando los tiempos, mucho mas rápidamente que la cantidad de elementos. Para ordenar 5000 elementos, necesitamos 13.51 segundos. para ordenar 10000, el doble, se requieren 53.88 segundos. Al margen, estos tiempos dependen de la

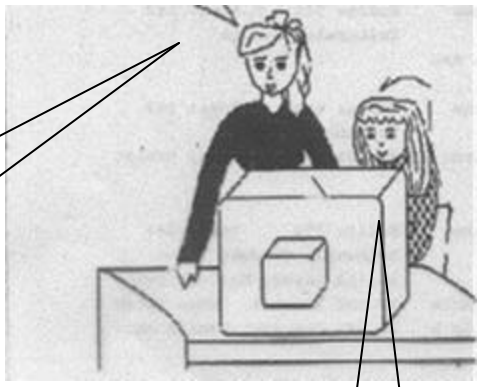
computadora, estos aquí están siendo obtenidos en una modesta Genuine Intel x86, Family 6, Model 8, 64 MB RAM (550 MHz, Monoprocesador).



Vamos al método Sacudida. La misma codificación anterior, adecuada, nos informa :



Sacudida no es tanto mejor como esperábamos. Veamos a Peinado:

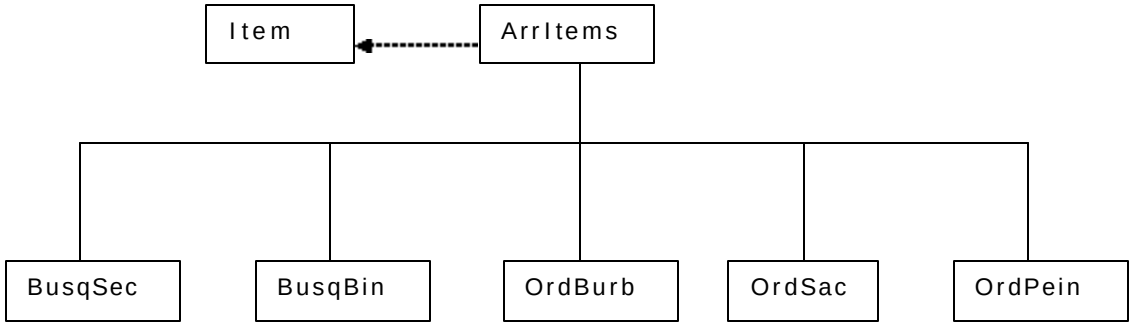


Ver para creer. Después de esto, quien mas usa Burbuja o Sacudida ?

Y si. Es mas, si contamos las líneas de código, Burbuja tiene 10, Sacudida 20 y Peinado 15. O sea que está en una complejidad en el medio de los otros dos. Pero para ordenar 10000 items es, a ver, 53.88/0.28 = ***192.43*** *veces mas rápido que Burbuja, o 44.43/0.28 =* ***158.68*** *que Sacudida ...*

```
(Inactive G:\BORLAND\BC45\EJEMPLOS\AED2004\UNIDAD3\COMPARAR\COMPARA...
Ordenamiento por método Peinado
Arrays de cuantos elementos? (999: Fin) 1000
Un poquitín de paciencia ...
Listo, empleamos 0.06 segundos
Otro ? (999: Fin) 5000
Un poquitín de paciencia ...
Listo, empleamos 0.11 segundos
Otro ? (999: Fin) 10000
Un poquitín de paciencia ...
Listo, empleamos 0.28 segundos
Otro ? (999: Fin) 999
Tarea terminada !!!...
```

Bueno, recapitulemos. Todo lo de búsqueda y ordenamiento en arrays lo hemos visto usando un conjunto de clases inter relacionadas, que trabajan "en equipo". Primero definimos la clase Item, que vincula un código con un valor. Esta clase la incluimos en la clase ArrItems (Relación "tiene un"). Heredando de ArrItems (relación "es un") implementamos las dos clases de búsqueda y las tres de ordenamiento. Luego, sobre la marcha, sin pensarlo mucho, implementamos la clase CalcTime, para calcular los tiempos de ordenamiento. Un ***diagrama de todo esto*** *...*



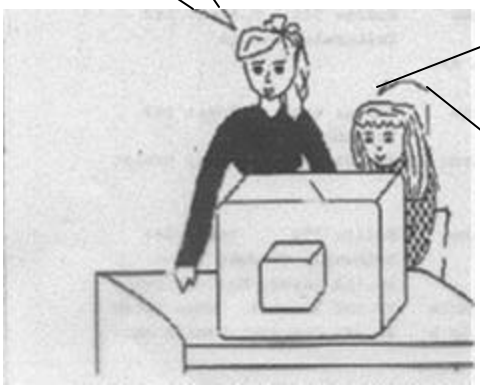
Las líneas representan las relaciones entre las clases:

De trazo: Tiene un; ArrItems tiene objetos Item

Continua: Es un: BusqSec, BusqBin, OrdBurb son objetos ArrItems

La última clase que se implementa, sobre la marcha, sin ánimo de integrarla a las otras, CalcTime, no implementa ninguna de estas relaciones. Su objeto solo tiene un par de atributos int propios, inicio y fin, no contiene ni hereda nada de ninguna de ellas. Únicamente su método calcular define y usa un objeto OrdBurb.

Hubiera quedado mejor si implementara alguna de estas relaciones ?



Si y no. Depende de lo que se quiera obtener. Para lo que nos pidieron, ***está bien como se hizo****. Pero si nos pidieran algo que requiriese una mayor integración, por ejemplo guardar en un array los tiempos de inicio y fin para ordenar arrays de distintos tamaños, con cada uno de los métodos que vimos, y después elaborar una gráfica comparando tiempos Vs cantidad de items en cada uno de los métodos,* ***sin duda****. Incluso podría afectar la forma como los relacionamos, por ejemplo para usar polimorfismo... Pero esto es tema de la U IV.*

ARRAYS MULTIDIMENSIONALES

Los arrays vistos anteriormente se conocen como arrays unidimensionales (una sola dimensión) y sus elementos se referencian mediante un único subíndice. Los arrays multidimensionales tienen más de una dimensión y, en consecuencia, de un índice. Son usuales arrays de dos dimensiones. Se les llama también como ***tablas*** o matrices. Es posible crear arrays de tantas dimensiones como se necesite, el límite puede ser nuestra capacidad de interpretación.

En memoria, un array bidimensional se almacena en un espacio lineal, fila tras fila. Si fuera tridimensional nos lo imaginamos constituido por planos, filas, columnas. Su almacenamiento es siempre lineal, en primer lugar tendríamos los elementos del primer plano, fila tras fila, luego el segundo plano y así sucesivamente.

La sintaxis de un array bi dimensional es:

<Tipo de dato> <Nombre Array> [<Numero de filas>] [<Numero de columnas>]

Algunos ejemplos de declaración de matrices.

```
char pantalla [25 ] [80];        // Para trabajr una pantalla en modo texto
int matCurs [10 ] [12];          // Cantidad de alumnos por materia/curso
```

Inicialización de arrays multidimensionales

Los arrays multidimensionales se pueden inicializar, al igual que los una dimensión, cuando se declaran. Esto lo hacemos asignándole a la matriz una lista de constantes separadas por comas y encerradas entre llaves, ejemplos.

```
int tablal[2] [3] = {51, 52, 53, 54, 55, 56}; // Tabla es una matriz de 2 filas, 3 columnas.

int tabla[2] [3] = {{51, 52, 53}, {54, 55, 56}};        // también puede hacerse así
```

Acceso a los elementos de arrays bidimensionales

Se puede acceder a los elementos de arrays bidimensionales de igual forma que a los elementos de un array unidimensional. La diferencia reside en que en los elementos bidimensionales deben especificarse los índices de la fila y columna.
El formato general para asignación directa de valores a los elementos:

```
inserción de elementos // Asignación de valores
<nombre matriz >[índice fila] [índice columna] =valor elemento;
```

extracción de elementos // Obtención de valores
<variable> = <nombre array> [índice fila] [índice columna];

Algunos ejemplos de asignación de valor:

```
tabla[2] [3] = 4.5;
matCurs[2] [4] = 50;  //  Asignamos 50 alumnos a la materia 2 del curso 4
matCurs[2] [4] += 10;//  y le agregamos 10 mas ...
```

y de extracción de valores:

```
ventas = tabla[1] [1];
dia = Semana[3] [6];
```

Lectura y escritura de elementos de arrays bidimensionales

Las sentencias *cín y cout* se utilizan para asignar / extraer elementos de arrays. Por ejemplo,

```
cín >> tabla[2] [3];     //       El valor digitado queda almacenado en tabla[2] [3];
cout  << tabla[2] [3]; //       Lo mostramos
```

Acceso a elementos mediante bucles

Se puede recorrer elementos de arrays bidimensionales mediante bucles anidados.
Si recorremos la matriz procesando una fila tras otra, la sintaxis es:

```
for  (int fila = 0; fila < nunFilas; ++fila)        // Ciclo externo para recorrer filas,
     for (int col = 0; col < numCol; ++col)         // Ciclo interno para recorrer elementos columna
        Procesar elemento[fila] [col];              // Algo hacemos con el elemento en cuestión
```

Si recorremos la matriz procesando una columna tras otra, la sintaxis es:

```
for (int col = 0; col < numCol; ++col)          // Ciclo externo para recorrer columna
    for  (int fila = 0; fila < nunFilas; ++fila)    // Ciclo externo para recorrer elementos fila,
        Procesar elemento[fila] [col];              // Algo hacemos con el elemento en cuestión
```

Donde *Procesar* elemento[fila] [col] puede ser:

```
        cin >>  elemento[fila] [col];      // Si estamos leyendo
        elemento[fila] [col] = 0;          // Si estamos inicializando a cero
        cout << "fila: "<<fila<<", col: "<<col <<", valor: "<<elemento[fila] [col] // Exhibiendo
```

Si la matriz es cuadrada y lo que necesitamos es recorrer la diagonal principal.

```
for (int ind = 0; ind < numFil; ++ind)          // Necesitamos un solo ciclo
        Procesar elemento[ind] [ind];           // Algo hacemos con el elemento en cuestión
```

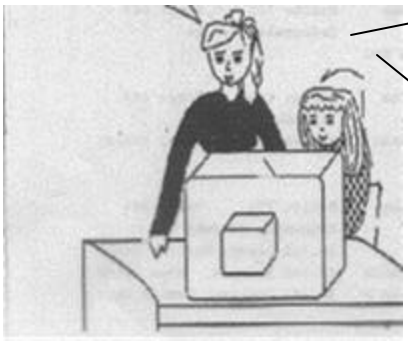
Arrays de más de dos dimensiones

C++ no limita la cantidad de dimensiones de un array o matriz. El límite ocurre por la falta de necesidad de representar organizaciones de datos de mas de 3 o 4 dimensiones. Cada dimensión es un acceso. Así como un array bidimensional puede considerarse como un conjunto de arrays lineales, uno tridimensional es un conjunto de arrays bidimensionales o planos. Los elementos de este array serán accedidos por una subindicación, de izquierda a derecha, planos, filas, columnas.

Un ejemplo bastante típico de un array de tres dimensiones es un libro: supongamos que tiene 500 páginas, cada pagina 45 líneas, cada línea 80 caracteres. No hay figuras. Este array se declara:

char libro[500] [45] [80]; // Ocupará 500*45*80= 1.800.000 bytes.

Necesitamos un ejemplo, un caso concreto, de la vida real, donde haya que recorrer filas, columnas, hacer cálculos, ...



Y si... Tengo el ejemplo perfecto. Las temperaturas de la semana pasada, hora por hora, todos los días. El comportamiento de una class Temperat incluye la capacidad de modificar/ informar una dada temperatura, además los promedios, por día o a una hora determinada, cual es el día mas frío, que día hizo mas frío (Cosas distintas). O también podemos definir una clase básica de comportamiento mínimo y de allí heredar y definir una clase mas sofisticada. Esto me gusta mas ...

```
#include <stdlib.h>
#include <iostream.h>
class Temperat {      // Una semana de temperaturas, cada dia, cada hora.
protected:            // parte privada
        int     temp [7] [24]; // Un objeto matriz de temperaturas
        virtual int     getTemp(int,int);
        virtual void    setTemp(int, int, int);
public:
        Temperat();             //      Constructor sin argumentos
        virtual void    toString(int,int); //     Exhibimos
        virtual void implementar();// Implementando el demo
        //  Otros métodos, si fuese necesario ...
};
Temperat::Temperat(){
        for(int dia = 0;dia < 7;++dia)
                for (int  hora = 0;hora < 24; ++hora)
                        temp[dia][hora] = (float)random(50);
}
int Temperat::getTemp(int dia, int hora){
        return temp[dia][hora];
}
void Temperat::setTemp(int dia, int hora, int temper){
        temp[dia][hora] = temper;
}

void    Temperat::toString(int dia, int hora){ //      Exhibimos
         cout << "La temperatura del día "<<dia<<", a la hora "<<hora<<", es: ";
         cout << temp[dia][hora]<<endl;
}
void Temperat::implementar(){     // Implementando el demo
int dia, hora, temp;
         cout << "Demo uso class Temperat (Fin hora: 999)\n";
         cout << "De que día desea conocer temperaturas ? ";
         cin  >> dia;
         cout << "A que hora ? ";
         cin >> hora;
         while(hora!=999){
                toString(dia,hora);
                cout << "A que valor corrijo ? ";
                cin >> temp;
                setTemp(dia,hora,temp);
                cout << "Otra hora ? ";
                cin >> hora;
```

```
        }
        cout << "Muchas gracias "<< endl;
}
# include <G:\Borland\BC45\Ejemplos\AED2004\Unidad3\Temperat\Temperat.cpp>
void main(){
        Temperat temp;
        temp.implementar();
```

```
(Inactive G:\BORLAND\BC45\EJEMPLOS\AED2004\UNIDAD3\TEMPERAT\MAINTEMP...
Demo uso class Temperat (Fin hora: 999)
De que día desea conocer temperaturas ? 3
A que hora ? 11
La temperatura del día 3, a la hora 11, es: 44
A que valor corrijo ? 8
Otra hora ? 11
La temperatura del día 3, a la hora 11, es: 8
A que valor corrijo ? 0
Otra hora ? 12
La temperatura del día 3, a la hora 12, es: 45
A que valor corrijo ? 45
Otra hora ? 999
Muchas gracias
};
```

Profe, el día 3 es martes o miércoles ? Comenzamos con el lunes o el domingo? Y el primer día es el 0 o el 1 ? ...

Hum ... Es verdad, no está muy claro que digamos ... Nadie se refiere a los días de la semana como 1, 2, 3 ... ya se que haremos ... No pensaba trabajar tanto, pero bueno ...
*Lo que necesitamos que la clase Tempmas, que hereda de Temperat, **tenga un** objeto DiasSem (Dias de la semana). Y que tiene que hacer este objeto en TempMas ? Nada mas que convertir la hilera "lunes" o "martes" en 0,1, que es como la matriz temp lo necesita. En términos informáticos digamos que debe hacer de interface entre el lenguaje humano y la máquina.*

```
#include <string.h>
#include <iostream.h>
#include <stdlib.h>
char *dia[] = {"lunes","martes","miercoles","jueves","viernes","sábado","domingo"};
        // Un array de punteros a caracteres ya inicializado
class DiasSem {       // Los días de la semana.
protected:
public:
        int     getNumero(char *);    // Dado el nombre, retorna el númeron
        char* getNombre(int);         // Dado el número, retorna el nombre
        void implementar();
//  Otros métodos, si fuese necesario ...
};

int DiasSem::getNumero(char * dias){
        int res  = 8; // No existe día 8 ...
        for(int ind = 0;ind < 7;++ind)
                if(strcmp(dia[ind],dias)==0){
                        res=ind; break;
```

```
                }
        return res;
}

char * DiasSem::getNombre(int num){
        return (num < 8 ? dia[num]:"Noexiste");
}

void DiasSem::implementar(){      // Implementando el demo
        char dia[12];
        int existe = 0;
        cout << "Digite nombre día (Salir: xxx)" << endl;
        do {
                cin.getline(dia,10);
                existe = getNumero(dia);
                if(existe < 8){    // Es válido
                        cout << "El dia existe, su posición es: "<<existe<<endl;
                        cout << "Otro nombre ";
                }
                else
                        cout << "El dia informado no existe \n";
        } while(existe < 8);     // Nombres de días válidos
        cout << "Demo terminado !!!"<<endl;
}
#include <G:\Borland\BC45\Ejemplos\AED2004\Unidad3\DiasSem\DiasSem.cpp>
void main(){
        DiasSem dia;
        dia.implementar();
};
```

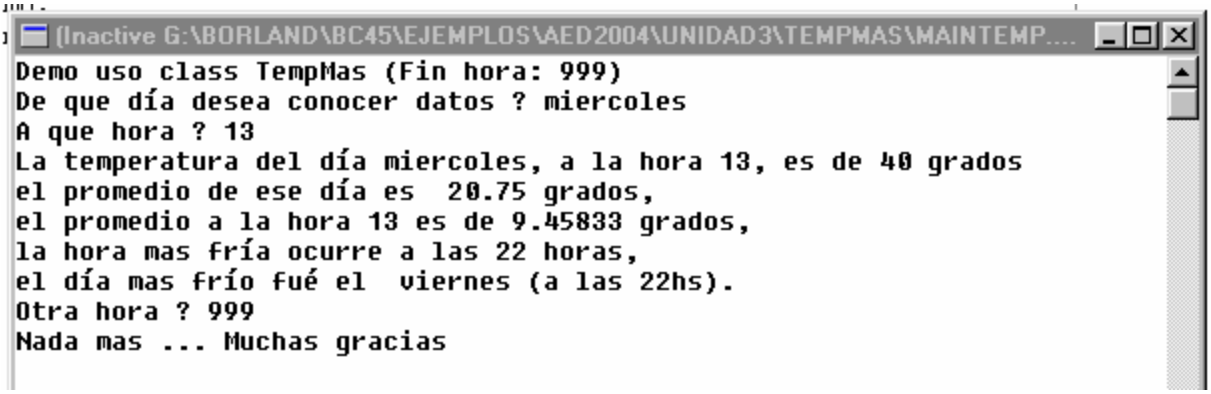
```
(Inactive G:\BORLAND\BC45\EJEMPLOS\AED2004\UNIDAD3\DIASSEM\MAINDIAS.EXE)
Digite nombre día (Salir: xxx)
viernes
El dia existe, su posición es: 4
Otro nombre jueves
El dia existe, su posición es: 3
Otro nombre sábado
El dia existe, su posición es: 5
Otro nombre sabado
El dia informado no existe
Demo terminado !!!
```

*Bueno, ahora estamos en condiciones de hacer lo que nos piden. La class **DiaSem** puede convertir nombre del día en su número de orden y viceversa. Entonces, en la **class TempMas**, que hereda públicamente de **class Temperat,** incluimos un objeto **DiaSem** y usamos su comportamiento getNumero() y GetNombre() conforme precisamos en sus funciones miembro implementar() y listo ...*

```
#include <stdlib.h>
#include <iostream.h>
#include <G:\Borland\BC45\Ejemplos\AED2004\Unidad3\Temperat\Temperat.cpp>
#include <G:\Borland\BC45\Ejemplos\AED2004\Unidad3\DiasSem\DiasSem.cpp>
class TempMas:public Temperat{   // Tratamiento de temperaturas potenciado.
protected:              // parte privada
        DiasSem dias;                   // Necesitamos tener un objeto de DiasSem
        virtual float   promDia(int);           //  Temperatura promedio del día
        virtual float   promHora(int);          //  y de la hora, en distintos días.
```

```
        virtual int     horaFria(int);          //  La hora mas fría de un dia determinado
        virtual char*   diaFrio(int);           //  y el día mas frío a una hora dada.

public:
        TempMas();                              //      Constructor sin argumentos
        void    toString(int, int);             //      Redefinimos esta función
        void implementar();                     //  y también esta ...
        //  Otros métodos, si fuese necesario ...
};

TempMas::TempMas():Temperat(){;}

float TempMas::promDia(int dia){
        int total=0.0;  //      p/totalizar temperaturas
        if (dia == 8)return 999.99;     //      Si nombre día mal informado => 999.99;
        for(int hor=0;hor<24;++hor)total+=getTemp(dia,hor);
        return (float)total/24;
}

float TempMas::promHora(int hor){
        int total=0.0;  //      p/totalizar temperaturas
        for(int dia=0;dia<7;++dia)total+=getTemp(dia,hor);
        return (float)total/24;
}

int     TempMas::horaFria(int dia){ //  La hora mas fría de un dia determinado
        int tFria = getTemp(dia,0);     // Tomamos el primer casillero
        int hFria = 0;
        for(int h=1;h<24;++h)
                if(tFria >= getTemp(dia,h)){ // Buscamos la hora a la que se da la
                        tFria = getTemp(dia,h);  // temperatura mas fría. Si varias, la
                        hFria = h;              // última
                }
        return hFria;
}

char* TempMas::diaFrio(int hs){   //  y el día mas frío a una hora dada
        int tFria = getTemp(0, hs);     // Tomamos el primer casillero
        int dFrio = 0;
        for(int d=1;d<7;++d)
                if(tFria >= getTemp(d,hs)){ // Buscamos el día que a la hora consulta-
                        tFria  =  getTemp(d,hs); // da tiene la menor temperatura. Si hay
                        dFrio = d;              // varios, el último.
                }
        return dias.getNombre(dFrio);  // El comportamiento getNombre (Objeto dias,
}               // class DiaSem) convierte el int dFrio a un nombre

void    TempMas::toString(int numDia, int hora){ //          Exhibimos
        char * nomDia = dias.getNombre(numDia);
         cout << "La temperatura del día "<<nomDia<<", a la hora "<<hora<<", es de ";
         cout << temp[numDia][hora]<<" grados"<<endl;
}

void TempMas::implementar(){      // Implementando el demo
        int hora,dia;
        char nomDia[12];
         cout << "Demo uso class TempMas (Fin hora: 999)\n";
         cout << "De que día desea conocer datos ?";
         cin.getline(nomDia,10);
         dia=dias.getNumero(nomDia); // Comportamiento del objeto clase DiaSem
         if(dia == 8)    // No hay tal día
                cout << "No existe " << nomDia << endl;
         else {
```

```
                cout << "A que hora ? ";
                cin >> hora;
                while(hora!=999){
                        toString(dia,hora);
                        cout << "el promedio de ese día es  "<<promDia(dia);
                        cout <<" grados,"<<endl;
                        cout << "el promedio a la hora "<<hora<<" es de ";
                        cout <<promHora(hora)<<" grados,"<<endl;
                        cout << "la hora mas fría ocurre a las ";
                        cout <<horaFria(dia)<<" horas,"<<endl;
                        cout << "el día mas frío fué el  ";
                        cout <<diaFrio(hora)<<" (a las "<< horaFria(dia)<< "hs).\n";
                        cout << "Otra hora ? ";
                        cin >> hora;
                }       // while
        }   // else
        cout << "Nada mas ... Muchas gracias "<< endl;
}

# include <G:\Borland\BC45\Ejemplos\AED2004\Unidad3\TempMas\TempMas.cpp>
void main(){
        TempMas demo;
        demo.implementar();
```




```
};
```

Diagrama de clases