



Introducción a c++ y la programación orientada a objetos

| Temperaturas | Códigos | Voltajes |
|--------------|---------|----------|
| 95.75 | Z | 98 |
| 83.0 | C | 87 |
| 97.625 | K | 92 |
| 72.5 | L | 79 |
| 86.25 | | 85 |
| | | 72 |

Arreglos y Vectores

El nombre del arreglo es c

| | | | |
|--|---------|------|-------|
| Número de posición del elemento dentro del arreglo c | c[0] | -45 | |
| | c[1] | 6 | |
| | c[2] | 0 | |
| | c[3] | 72 | |
| Nombre de un elemento individual del arreglo | c[4] | 1543 | Valor |
| | c[5] | -89 | |
| | c[6] | 0 | |
| | c[7] | 62 | |
| | c[8] | -3 | |
| | c[9] | 1 | |
| | c[10] | 6453 | |
| | c[11] | 78 | |

```
int main()
{
    int C[10];
    for (int j =0; j<10;j++){
        C[j]=0;
    }
    cout << "elemento " << " valor" << endl;

    for (int j =0; j<10;j++){
        cout << j << C[j] << endl;
    }
    return 0;
}
```

int A[5]: // A es un arreglo de 5 enteros

A[j] j es el número de la posición del elemento dentro del arreglo

Arreglos y Arreglos stl

```
int main()
{
    int C[10];
    for (int j =0; j<10;j++){
        C[j]=0;
    }
    cout << "elemento " << " valor" << endl;

    for (int j =0; j<10;j++){
        cout << j << C[j] << endl;
    }
    return 0;
}
```

#include <array>

```
int main() {
    array<int, 10> n; //n es un arreglo de 10 enteros

    // initialize elements of array n to 0
    for (size_t i{0}; i < n.size(); ++i) {
        n[i] = 0; //establece el elemento en la ubicación i a 0
    }

    cout << "Elemento" << setw(10) << "Valor" << endl;
    // imprime el valor de cada elemento del arreglo
    for (size_t j{0}; j < n.size(); ++j) {
        cout << setw(7) << j << setw(10) << n[j] << endl;
    }
    return 0;
}
```

Inicialización de un arreglo en una declaración

```
int temp[5] = {98, 87, 92, 79, 85};  
char codigos[6] = {'m', 'u', 'e', 's', 't', 'r', 'a'};  
double pendientes[7] = {11.96, 6.43, 2.58, .86, 5.89, 7.56, 8.22};
```

```
int galones[20] = {19, 16, 14, 19, 20, 18,  
                  12, 10, 22, 15, 18, 17,  
                  16, 14, 23, 19, 15, 18,  
                  21, 5};
```

```
int A[]={1,2,3,4,5};
```

```
int B[5]={1,2,3,4,5};
```

```
int C[7]={1,2,3,4,5,6}; //???
```

```
int main()  
{  
    int n[ 10 ] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };  
  
    cout << "Elemento" << setw( 13 ) << "Valor" << endl;  
  
    for ( int i = 0; i < 10; i++ ){  
        cout << setw( 7 ) << i << setw( 13 ) << n[ i ] << endl;  
    }  
    return 0;  
}
```

Tamaño arreglo con una variable constante y establecimiento de los elementos de un arreglo con cálculos

```
#include<iostream>

using namespace std;

int main()
{
    // la variable constante se puede usar para especificar el tamaño de los arreglos
    const int tamañoArreglo = 10; // debe inicializarse en la declaración

    int s[ tamañoArreglo ]; // el arreglo s tiene 10 elementos

    for ( int i = 0; i < tamañoArreglo; i++ ){
        s[ i ] = 2 + 2 * i; // establece los valores
    }

    cout << "Elemento" << setw( 13 ) << "Valor" << endl;

    for ( int j = 0; j < tamañoArreglo; j++ )
        cout << setw( 7 ) << j << setw( 13 ) << s[ j ] << endl;

    return 0;
}
```

Esto compila?

Si no se asigna un valor a una variable constante cuando se declara es un error de compilación.

E. Código
example1_barras

E. Código
example2_contadores (datos)

E. Código
example3_encuesta

"C++ no cuenta con comprobación de límites para evitar que la computadora haga referencia a un elemento que no existe."

Uso de arreglos tipo carácter para almacenar y manipular cadenas

```
char cadena1[] = "first";  
char cadena1[] = { 'f', 'i', 'r', 's', 't', '\0' };
```

caracter nulo '\0'

Todas las cadenas representadas mediante arreglos de caracteres terminan con este carácter.

Sin él, este arreglo representaría tan sólo un arreglo de caracteres, no una cadena

```
char cadena2[ 20 ];  
cin >> cadena2;
```

Es responsabilidad del programador asegurar que el arreglo en el que se coloque la cadena sea capaz de contener cualquier cadena que el usuario escriba en el teclado.

```
int main(){  
    char cadena1[ 20 ];  
    char cadena2[] = "literal de cadena";  
  
    // lee la cadena del usuario y la coloca en el arreglo cadena1  
    cout << "Escriba la cadena \"hola todos\": ";  
    cin >> cadena1;  
  
    cout << "cadena1 es: " << cadena1 << "\ncadena2 es: " << cadena2;  
    cout << "\ncadena1 con espacios entre caracteres es:\n";  
  
    // imprime caracteres hasta llegar al caracter nulo  
    for ( int i = 0; cadena1[ i ] != '\0'; i++){  
        cout << cadena1[ i ] << ' ';  
    }  
  
    cin >> cadena1; // lee "todos"  
    cout << "\ncadena1 es: " << cadena1 << endl; // NOTE: cuidado con  
    la linea fantasma. aca debe usar getline(cin,string)  
  
    return 0;  
}
```

Arreglos locales estáticos

```
void inicArregloStatic( void )    E.Codigo
{                                example5_arreglosstatic
// inicializa con 0 la primera vez que se llama a la función
static int arreglo1[ 3 ];

cout << "\nValores al entrar en inicArregloStatic:\n";

for ( int i = 0; i < 3; i++ ){
    cout << "arreglo1[" << i << "] = " << arreglo1[ i ] << "
";
}

cout << "\nValores al salir de inicArregloStatic:\n";

// modifica e imprime el contenido de arreglo1
for ( int j = 0; j < 3; j++ )
    cout << "arreglo1[" << j << "] = " << ( arreglo1[ j ] +=
5 ) << " ";
}
```

Podemos aplicar `static` a la declaración de un arreglo local, de manera que el arreglo no se cree e inicialice cada vez que el programa llame a la función, y no se destruya cada vez que termine la función en el programa. Esto puede mejorar el rendimiento, en especial cuando se utilizan arreglos extensos.

Paso de arreglos a funciones

C++ pasa los arreglos a las funciones por referencia.

las funciones llamadas pueden modificar los valores de los elementos en los arreglos originales.

El paso de arreglos por referencia tiene sentido por cuestiones de rendimiento. Si los arreglos se pasaran por valor, se pasaría una copia de cada elemento. Para los arreglos extensos que se pasan con frecuencia, esto requeriría mucho tiempo y una cantidad considerable de almacenamiento para las copias de los elementos del arreglo.

Observe la extraña apariencia del prototipo
void FunA(int [], int); //Arreglo y tamaño

C++ ignoran los nombres de las variables en los prototipos

void FunArreglo(int NameA[], int VariableSizeA);

E. Codigo example6_modificarA

instrucción for basada en rango

```
int main() {  
    array<int, 5> items{1, 2, 3, 4, 5};  
  
    cout << "items before modification: ";  
    for (int item : items) {  
        cout << item << " ";  
    }  
  
    for (int& itemRef : items) {  
        itemRef *= 2;  
    }  
  
    cout << "\nitems after modification: ";  
    for (int item : items) {  
        cout << item << " ";  
    }  
  
    cout << endl;  
}
```



```

#include <iostream>
#include <iomanip>

using namespace std;

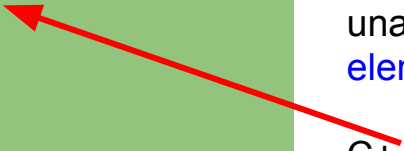
void tratarDeModificarArreglo( const int [] );

int main()
{
    int a[] = { 10, 20, 30 };

    tratarDeModificarArreglo( a );
    cout << a[ 0 ] << ' ' << a[ 1 ] << ' ' << a[ 2 ] << '\n';
    return 0;
}

void tratarDeModificarArreglo( const int b[] )
{
    b[ 0 ] /= 2; // error de compilación
    b[ 1 ] /= 2;
    b[ 2 ] /= 2;
}

```



Principio de menor privilegio.

Las funciones no deben recibir la capacidad de modificar un arreglo, a menos que sea absolutamente necesario

se puede encontrar con situaciones en las que una función **no tenga permitido modificar los elementos de un arreglo**.

C++ cuenta con el calificador de tipos **const**.

Cuando una función especifica un parámetro tipo arreglo al que se antepone el calificador **const**, los elementos del arreglo se hacen constantes en el cuerpo de la función

E. Código [librocalicar1 \(clase6\)](#)

E. Código [example7_busqueda_lineal](#)

E. Código [example8_busqueda_insercion](#)

E. Código [example8_sort_new](#)

Arreglos multidimensionales

| | Columna 0 | Columna 1 | Columna 2 | Columna 3 |
|--------|-------------|-------------|-------------|-------------|
| Fila 0 | a[0][0] | a[0][1] | a[0][2] | a[0][3] |
| Fila 1 | a[1][0] | a[1][1] | a[1][2] | a[1][3] |
| Fila 2 | a[2][0] | a[2][1] | a[2][2] | a[2][3] |

Diagram illustrating the structure of a 2D array. The array is represented as a table with rows (Fila 0, Fila 1, Fila 2) and columns (Columna 0, Columna 1, Columna 2, Columna 3). Each element is accessed using the format `a[row][column]`. Arrows point from the labels "Subíndice de columna", "Subíndice de fila", and "Nombre del arreglo" to the corresponding parts of the array notation in the table.

```
int b[ 2 ][ 2 ] = { { 1, 2 }, { 3, 4 } };
```

`b[0][0] = 1, b[0][1] = 2, b[1][0] = 3, b[1][1] = 4,`

```
int b[ 2 ][ 2 ] = { { 1 }, { 3, 4 } };
```

`b[0][0] = 1, b[0][1] = 0, b[1][0] = 3 y b[1][1] = 4`

Tarea, no calificable.

¿Como se haria esto con el template "array"?

```
void imprimirArreglo( const int a[ 3 ] );
```

```
int main()
{
```

```
int arreglo1[ 2 ][ 3 ] = { { 1, 2, 3 }, { 4, 5, 6 } };
```

```
int arreglo2[ 2 ][ 3 ] = { 1, 2, 3, 4, 5 };
```

```
int arreglo3[ 2 ][ 3 ] = { { 1, 2 }, { 4 } };
```

```
cout << "Los valores en arreglo1 por fila:" << endl;
```

```
imprimirArreglo( arreglo1 );
```

```
cout << "\nLos valores en arreglo2 por fila:" << endl;
```

```
imprimirArreglo( arreglo2 );
```

```
return 0;
```

```
}
```

```
void imprimirArreglo( const int a[ 3 ] )
```

```
{
```

```
// itera a través de las filas del arreglo
```

```
for ( int i = 0; i < 2; i++ ){
```

```
// itera a través de las columnas de la fila actual
```

```
for ( int j = 0; j < 3; j++ )
```

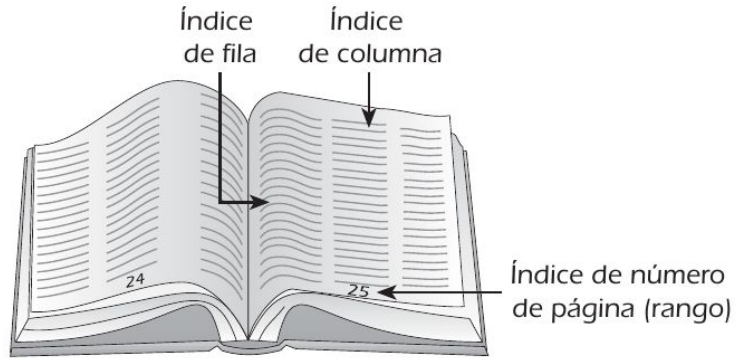
```
    cout << a[ i ][ j ] << ' ';
```

```
    cout << endl; // empieza nueva línea de salida
```

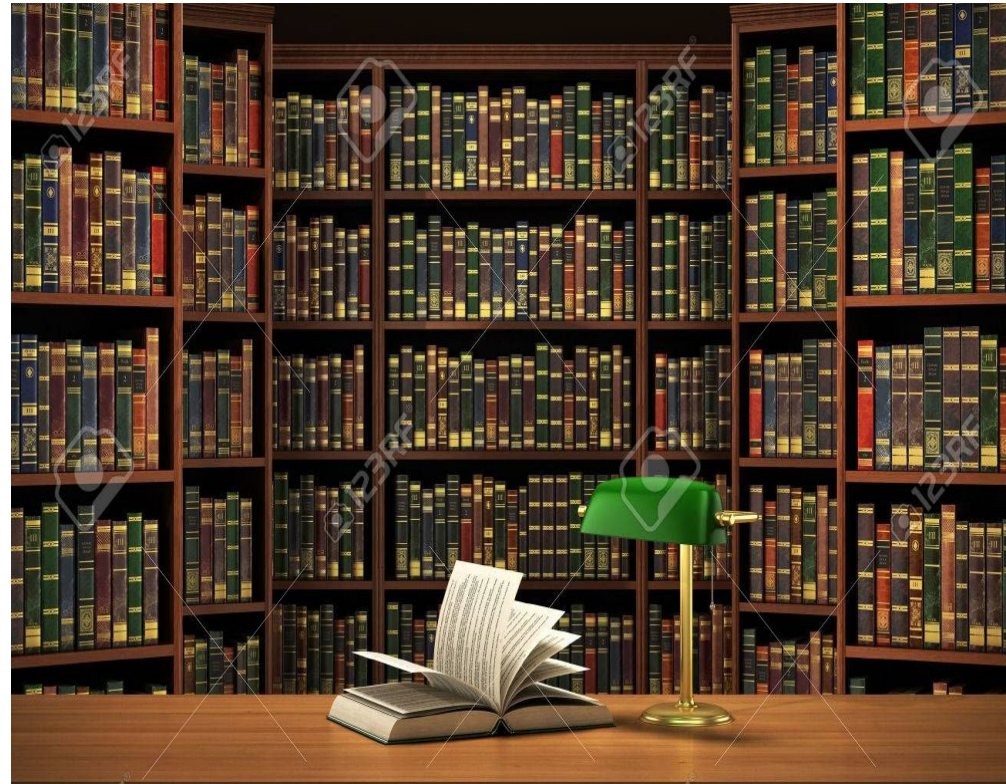
```
}
```

```
}
```

Arreglos multidimensionales



© Can Stock Photo - csp33529995



E. Codigo librocalicar2 (clase6)

La clase vector de STL (standard template library)

Una de las dificultades del lenguaje C es la implementación de contenedores (vectores, listas enlazadas, conjuntos ordenados) genéricos, de fácil uso y eficaces. Para que estos sean genéricos por lo general estamos obligados a recurrir a punteros genéricos (void *) y a operadores de cast. Es más, cuando estos contenedores están superpuestos unos a otros (por ejemplo un conjunto de vectores) el código se hace difícil de utilizar.

Para responder a esta necesidad, la STL (standard template library) **implementa un gran número de clases template describiendo contenedores genéricos para el lenguaje C++**.

std::pair<T1,T2>
std::list<T,...>
std::vector<T,...>
std::set<T,...>
std::map<K,T,...>

```
#include <iostream>
#include <string>
#include <list>
```

```
int main(){
```

```
    std::list<int> ma_lista;
    ma_lista.push_back(4);
    ma_lista.push_back(5);
    ma_lista.push_back(4);
    ma_lista.push_back(1);
```

```
    std::list<int>::const_iterator lit (mi_lista.begin()),
    lend(mi_lista.end());
```

```
    for(;lit!=lend;++lit) {
        std::cout << *lit << ' ';
    }
```

```
    std::cout << std::endl;
    return 0;
```

```
}
```

La clase vector de STL (standard template library)

| Funciones (métodos de clase) y operaciones | Descripción |
|---|--|
| <code>vector<TipoDatos> nombre</code> | Crea un vector vacío con tamaño inicial dependiente del compilador |
| <code>vector<TipoDatos> nombre(fuente)</code> | Crea una copia del vector fuente |
| <code>vector<TipoDatos> nombre(n)</code> | Crea un vector de tamaño <i>n</i> |
| <code>vector<TipoDatos> nombre(n, elem)</code> | Crea un vector de tamaño <i>n</i> con cada elemento inicializado como <i>elem</i> |
| <code>vector<TipoDatos> nombre(src.beg, src.end)</code> | Crea un vector inicializado con elementos de un contenedor fuente que comienza en <i>src.beg</i> y termina en <i>src.end</i> |
| <code>~vector<TipoDatos>()</code> | Destruye el vector y todos los elementos que contiene |
| <code>nombre[índice]</code> | Devuelve el elemento en el índice designado, sin comprobación de límites |
| <code>nombre.at(índice)</code> | Devuelve el elemento en el argumento del índice especificado, sin comprobación de límites en el valor del índice |
| <code>nombre.front()</code> | Devuelve el primer elemento en el vector |
| <code>nombre.back()</code> | Devuelve el último elemento en el vector |
| <code>dest = src</code> | Asigna todos los elementos del vector <i>src</i> al vector <i>dest</i> |

E. Código example9_vector

E. Código example10_fibo

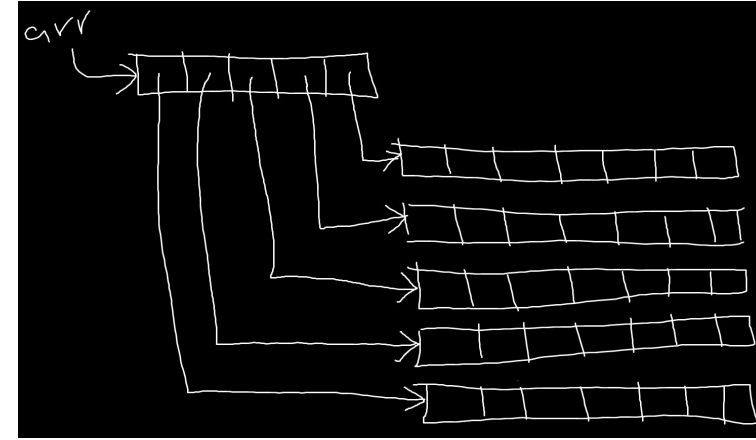
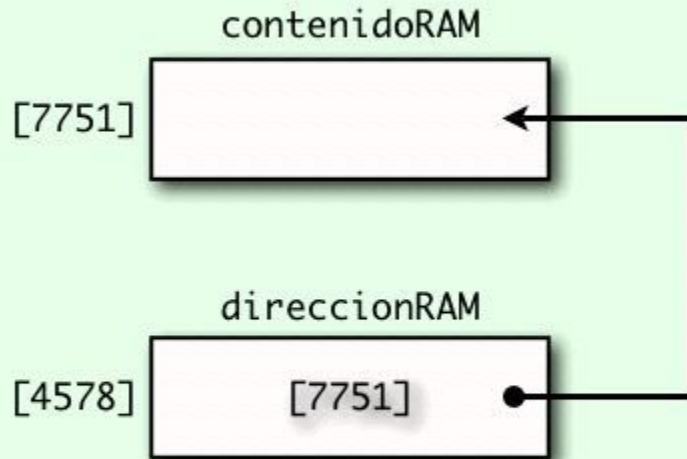
Podemos poner un poco de luz para ver la grandeza de Khazad-dûm



- ¿Qué es este nuevo horror Gandalf?
- Apuntadores, un demonio del mundo antiguo.
¿es éste un poder que alguno de ustedes puede enfrentar?



Apuntadores, los arreglos y las cadenas estilo C

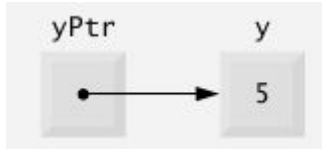


Los apuntadores nos dan acceso al funcionamiento interno de la computadora y la estructura de almacenamiento básico

Declaraciones e inicialización de variables apuntdores

```
Int y = 5; // Decalara la variable
int *yPtr; //Decalara la variable apuntdor

yPtr = &y; // asigna la direcion de y a yPtr
```



variables que se usan para almacenar direcciones de memoria

```
double *B_mass, *B_px, *B_py, *B_pz;

int*    J_mass, J_px, Jpy, Jpz; ???
```

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

```
int main()
{
    int a;
    int *aPtr; // aPtr es un int * -- apuntdor a un entero

    a = 7;
    aPtr = &a; // aca asignamos la direccíon de "a" a "aPtr"

    cout << "La direccíon de a es " << &a
          << "\nEl valor de aPtr es " << aPtr;
    cout << "\n\nEl valor de a es " << a
          << "\nEl valor de *aPtr es " << *aPtr;
    cout << "\n\nDemostracíon de que * y & son inversos "
          << "uno del otro.\n&*aPtr = " << &*aPtr
          << "\n*&aPtr = " << *&aPtr << endl;

    return 0;
}
```


Declaraciones e inicialización de variables apunadores

Inicialice todos los punteros para evitar que apunten a áreas de memoria desconocidas o no inicializadas.

Pointers should be initialized to **nullptr** (added in C++11) or to a memory address either when they're declared or in an assignment.

En versiones anteriores de C++, el valor especificado para un **puntero nulo era 0 o NULL**.

```
int y{5}; // declara la variable y
int* yPtr{nullptr}; // declara la variable puntero yPtr
yPtr = &y; // asigna la dirección de y a yPtr
```

```
*yPtr = 9;
cin >> *yPtr;
```





Detector Model ?

- Tracker Barrels ☐
- Tracker Endcaps ☐
- ECAL Barrel ☒
- ECAL Endcaps ☐
- ECAL Preshower ☐
- HCAL Barrel ☐
- HCAL Endcaps ☐
- HCAL Outer ☒
- HCAL Forward ☐
- Drift Tubes (muon) ☐
- Cathode Strip Chambers (muon) ☐
- Resistive Plate Chambers (muon) ☐

Tracking ?

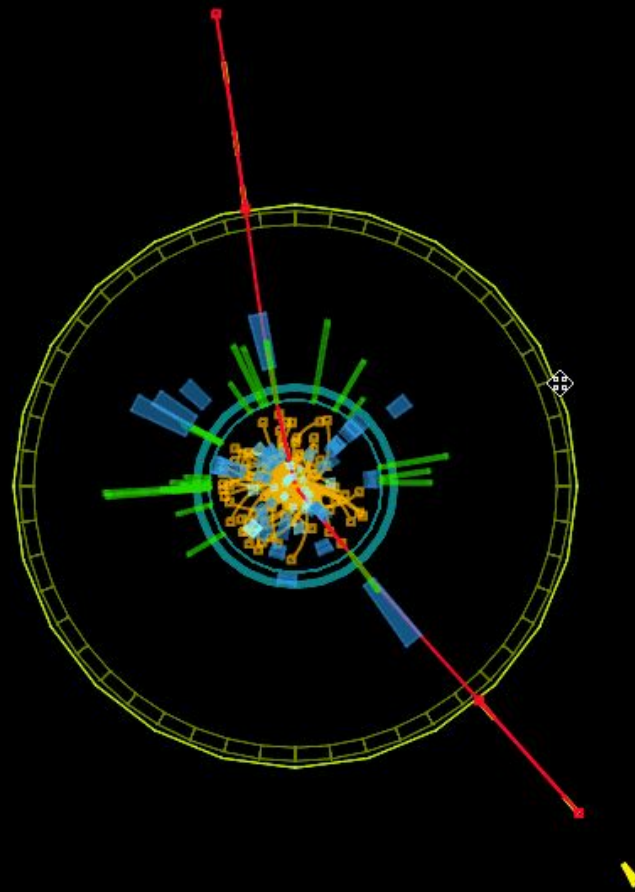
- Tracks (reco.) ☒
- Clusters (Si Pixels) ☐
- Clusters (Si Strips) ☐
- Rec. Hits (Tracking) ☐

ECAL ?

- Barrel Rec. Hits ☒ ▶
- Endcap Rec. Hits ☐ ▶
- Preshower Rec. Hits ☐ ▶

HCAL ?

- Barrel Rec. Hits ☒ ▶
- Endcap Rec. Hits ☒ ▶
- Forward Rec. Hits ☒ ▶
- Outer Rec. Hits ☐ ▶



Paso de argumentos a funciones por referencia mediante apuntadores

Paso por referencia

Type Fun (type &, type &) ;

mediante apuntadores

Type Fun (type *, type *) ;

En general, para funciones "sencillas", gana la conveniencia de la notación y se usan referencias. Sin embargo, **al transmitir arreglos a funciones el compilador transmite de manera automática una dirección. Esto dicta que se usarán variables apuntadoras para almacenar la dirección.**

E. Codigo example2_ArgPorReren.cpp (clase7)

Uso de const con apuntadores

-Si un valor no cambia (o no debe cambiar) en el cuerpo de una función que lo recibe, el parámetro se debe declarar const para asegurar que no se modifique por accidente.

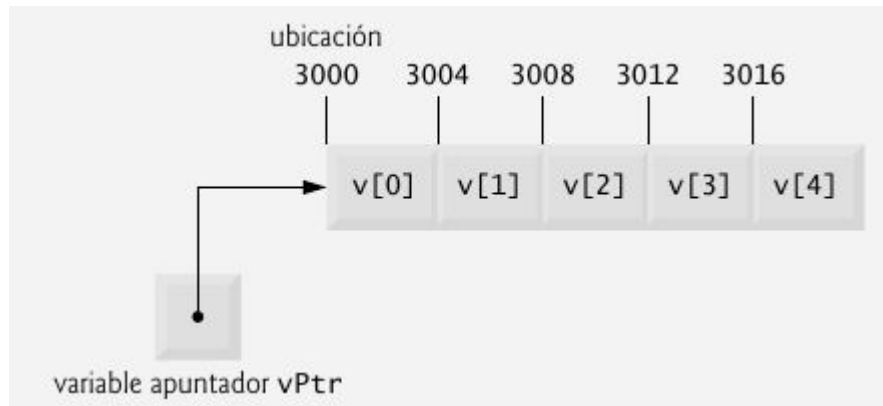
-Antes de usar una función, compruebe su prototipo para determinar los parámetros que puede modificar.

- Cuando el compilador encuentra el parámetro de una función para un arreglo unidimensional de la **forma int b[] , convierte el parámetro a la notación de apuntador int *b.** Ambas formas de declarar un parámetro de función como arreglo unidimensional son intercambiables.

E. Codigo example3_Const.cpp (clase7)

E. example4_OrdeSelec.cpp (clase7)

E. example5_sizeof.cpp (clase7)



```
int *vPtr = v;  
int *vPtr = &v[ 0 ];
```

```
vPtr +=2 ;    (3000 + 2*4 = 3008 )
```

En el arreglo v, vPtr apuntaría ahora a v[2]

new example:

```
vPtr +=4 ;
```

En el arreglo v, vPtr apuntaría ahora a v[4]

```
vPtr -=3 ;   ???
```

Aritmética de apuntadores

Se pueden realizar varias operaciones aritméticas con los apuntadores. Un apuntador se puede incrementar (++) o decrementar (--), se puede sumar un entero a un apuntador (+ o +=), se puede restar un entero de un apuntador (- o -=), o se puede restar un apuntador de otro apuntador del mismo tipo.

NOTA: La mayoría de las computadoras de la actualidad tienen enteros de dos o de cuatro bytes. Algunos de los equipos más recientes utilizan enteros de ocho bytes. Debido a que los resultados de la aritmética de apuntadores dependen del tamaño de los objetos a los que apunta un apuntador, **la aritmética de apuntadores es dependiente del equipo.**

Relación entre apuntadores y arreglos

Los arreglos y los apuntadores están estrechamente relacionados en C++ y se pueden utilizar de manera casi intercambiable. **El nombre de un arreglo se puede considerar como un apuntador constante.** Los apuntadores se pueden utilizar para realizar cualquier operación en la que se involucren los subíndices de arreglos.

```
int b[ 5 ];  
int *bPtr;
```

E. Código example6_notacionApun.cpp (clase7)

```
bPtr = b; // asigna la dirección del arreglo b a bPtr  
bPtr = &b[ 0 ]; // también asigna la dirección del arreglo b a bPtr
```

El elemento `b[3]` del arreglo se puede referenciar de manera alternativa con la siguiente expresión de apuntador:

`*(bPtr + 3)` y en general

`b[i] = *(bPtr + i)`

E. Códigos
example7_copystringArreglos.cpp
example9_ApuntFunc.cpp
example10_ArregApuntFun.cpp
(clase7)



NOTA1: El nombre del arreglo (**que es const de manera implícita**) se puede tratar como apuntador y se puede utilizar en la aritmética de apuntadores. Por ejemplo, la expresión `*(b + 3)`

Sin embargo

`b += 3`

produce un error de compilación, trata de modificar una constante.

NOTA2: Los apuntadores pueden usar subíndices de la misma forma que los arreglos. Por ejemplo, la expresión `bPtr[1]`

Procesamiento de cadenas basadas en apuntador

| Prototipo de función | Descripción de función |
|--|---|
| <code>char *strcpy(char *s1, const char *s2);</code> | Copia la cadena s2 en el arreglo de caracteres s1. Se devuelve el valor de s1. |
| <code>char *strncpy(char *s1, const char *s2, size_t n);</code> | Copia como máximo n caracteres de la cadena s2 y los coloca en el arreglo de caracteres s1. Se devuelve el valor de s1. |
| <code>char *strcat(char *s1, const char *s2);</code> | Adjunta la cadena s2 a s1. El primer carácter de s2 sobrescribe el carácter nulo de terminación de s1. Se devuelve el valor de s1. |
| <code>char *strncat(char *s1, const char *s2, size_t n);</code> | Adjunta como máximo n caracteres de la cadena s2 a la cadena s1. El primer carácter de s2 sobrescribe el carácter nulo de terminación de s1. Se devuelve el valor de s1. |
| <code>int strcmp(const char *s1, const char *s2);</code> | Compara la cadena s1 con la cadena s2. La función devuelve un valor de cero, menor que cero o mayor que cero si s1 es igual a, menor que, o mayor que s2, respectivamente. |
| <code>int *strncmp(const char *s1, const char *s2, size_t n);</code> | Compara hasta n caracteres de la cadena s1 con la cadena s2. La función devuelve cero, menor que cero o mayor que cero, si la porción del carácter n de s1 es igual a, menor que, o mayor que la correspondiente porción del carácter n de s2, respectivamente. |
| <code>size_t strlen(const char *s);</code> | Determina la longitud de la cadena s. Se devuelve el número de caracteres antes del carácter nulo de terminación. |

```
char color[] = "azul";  
const char *colorPtr = "azul";
```

```
char color[] = { 'a', 'z', 'u', 'l', '\0' };
```

Si no se asigna suficiente espacio en un arreglo de caracteres para almacenar el carácter nulo que termina una cadena, se produce un error.

Default

```
char enunciado[ 80 ];  
cin.getline( enunciado, 80, '\n' );
```

E. Codigos
example11_strcopy.cpp
example12_strcat.cpp
example13_strcmp.cpp
example14_strtok.cpp
example15_strlen.cpp
(clase7)

Tarea: Segui2_1

Generalidades acerca del manejo de excepciones

Realizar una tarea

Si la tarea anterior no se ejecutó correctamente

Realizar el procesamiento de los errores

Realizar la siguiente tarea

Si la tarea anterior no se ejecutó correctamente

Realizar el procesamiento de los errores

...

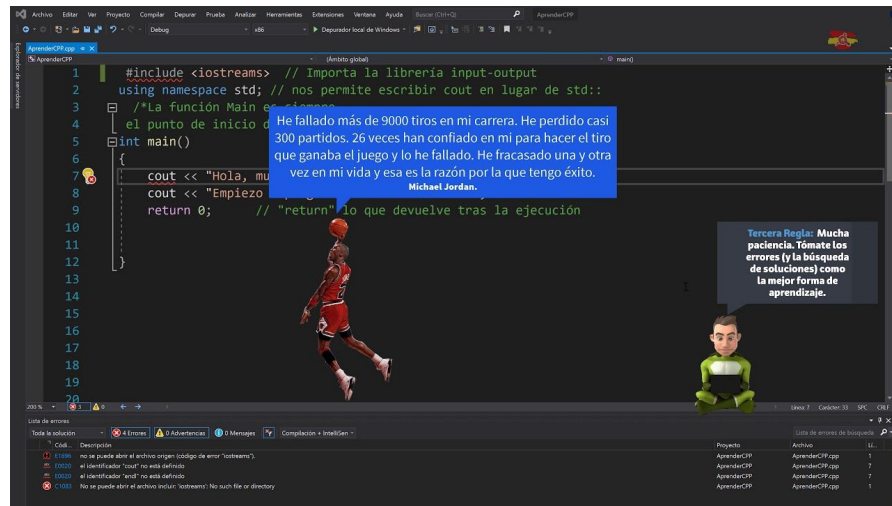
...

...

...

Aunque esta forma de manejo de excepciones funciona

Si los problemas potenciales ocurren con poca frecuencia, **al entremezclar la lógica del programa y la lógica del manejo de errores se puede degradar el rendimiento del programa**, ya que éste debe realizar pruebas (tal vez con frecuencia) para determinar si la tarea se ejecutó en forma correcta, y si se puede llevar a cabo la siguiente tarea.



El manejo de excepciones proporciona un mecanismo estándar para procesar los errores. **Esto es especialmente importante cuando se trabaja en un proyecto con un equipo extenso de programadores.**


```

int main()
{
    int x = 1;
    // instrucciones preliminares.....

    cout << "antes del try \n";
    try {
        cout << "dentro del try \n";
        if (x < 0){
            throw x; // el tipo de excepcion sera un entero
            cout << "despues del throw (NO se debe
ejecutar) \n";
        }
    }
    catch (int x ) {
        cout << "Exception Caught \n";
    }

    cout << "despues del catch (esto seguira
ejecutandose) \n";

    return 0;
}

```

E. Codigo example_throw.cpp (clase8)
E. Codigo example2.cpp (clase8)

bloque try: Encierra instrucciones que podrian ocasionar excepciones e instrucciones que se debria omitir si ocurren excepciones

Palabra clave throw: representa el tipo de excepcion que se va a lanzar

catch: sirven como manejadores de excepciones para cualesquiera excepciones lanzadas por las instrucciones en el bloque try

```

cout << endl;
cout << "aca una excepcion general. \n";

try {
    throw 10;
}
catch (char excp) {
    cout << "Caught " << excp;
}
catch (...) {
    cout << "por default \n";
}

```

La clase string: flujos de cadena

```
string texto( "Hola" );  
string nombre( 8, 'x' ); // cadena de 8 caracteres 'x'  
string mes = "Marzo"; // igual que: string mes( "Marzo" );
```

string no proporciona conversiones de int o char a string en una definición string.

```
string error1 = 'c';  
string error2( 'u' );  
string error3 = 22;  
string error4( 8 );
```

```
string objetoString;  
cin >> objetoString;
```

La función getline también se sobrecarga para objetos string:

```
getline( cin, cadena1 );
```

String in C++

C style String

C++ style String

```
Char e[] = "geeks"  
Char e1[] = {'g', 'f', 'g', '10'};  
Char * C = "geeksforgeeks";
```

```
String str = ("gfg");  
String str = "" g;  
String str ; str = "gfg";
```



E.Codigo example3_concatenacion.cpp (clase8)
E.Codigo example4_comparar.cpp (clase8)
E.Codigo example5_subcadenas.cpp (clase8)
E.Codigo example6_caracteristicas.cpp (clase8)

Constructores de clase string

| Constructor | Descripción | Ejemplos |
|--|--|--|
| <code>string nombreObjeto = valor</code> | Crea e inicializa un objeto de cadena a un valor que puede ser un literal de cadena, un objeto de cadena declarado con anterioridad o una expresión que contiene literales de cadena y objetos de cadena | <code>string str1 = "Buenos dias";</code> <code>string str2 = str1;</code> <code>string str3 = str1 + str2;</code> |
| <code>string nombreObjeto (valorCadena)</code> | Produce la misma inicialización que el anterior | <code>string str1 ("Hot");</code> <code>string str1 (str1 + " Dog");</code> |
| <code>string nombreObjeto (str, n)</code> | Crea e inicializa un objeto de cadena con una subcadena del objeto de cadena <code>str</code> , iniciando en la posición índice <code>n</code> de <code>str</code> | <code>string str1(str2, 5)</code> Si <code>str2</code> contiene la cadena Buenos dias, entonces <code>str1</code> se convierte en la cadena <code>dias</code> |
| <code>string nombreObjeto (str, n, p)</code> | Crea e inicializa un objeto de cadena con una subcadena del objeto de cadena <code>str</code> , iniciando en la posición índice <code>n</code> de <code>str</code> y contiene <code>p</code> caracteres | <code>string str1(str2, 5,2)</code> Si <code>str2</code> contiene la cadena Buenos dias, entonces <code>str1</code> se vuelve la cadena <code>di</code> |
| <code>string nombreObjeto (n, char)</code> | Crea e inicializa un objeto de cadena con <code>n</code> copias de <code>char</code> | <code>string str1(5, '*')</code> Esto hace a <code>str1 = "*****"</code> |
| <code>string nombreObjeto;</code> | Crea e inicializa un objeto de cadena para representar una secuencia de caracteres vacía (igual a la cadena <code>nombreObjeto = ""</code> ; el largo de la cadena es 0) | <code>string mensaje;</code> |

Los métodos de procesamiento de la clase string

| Método/Operación | Descripción | Ejemplo |
|----------------------------------|---|---------------------------------------|
| <code>int length()</code> | Devuelve la longitud de la cadena implícita | <code>string.length()</code> |
| <code>int size()</code> | Igual que la anterior | <code>string.size()</code> |
| <code>at(int index)</code> | Devuelve el carácter en el índice especificado y lanza una excepción si el índice es inexistente | <code>string.at(4)</code> |
| <code>int compare(string)</code> | Compara dos cadenas; devuelve un valor negativo si la cadena implicada es menor que <code>str</code> , cero si son iguales y un valor positivo si la cadena implicada es mayor que <code>str</code> | <code>string1.compare(string2)</code> |
| <code>c_str()</code> | Devuelve la cadena como una cadena C terminada en null | <code>string1.c_str()</code> |

| | | |
|----------------------------|---|---------------------------------|
| <code>bool empty</code> | Devuelve verdadero si la cadena implicada está vacía; de lo contrario, devuelve falso | <code>string1.empty()</code> |
| <code>erase(ind,n);</code> | Elimina <code>n</code> caracteres de la cadena implicada, empezando en el índice <code>ind</code> | <code>string1.erase(2,3)</code> |
| <code>erase(ind)</code> | Elimina todos los caracteres de la cadena implicada, empezando desde el índice <code>ind</code> hasta el final de la cadena. La longitud de la cadena restante se convierte en <code>ind</code> | <code>string1.erase(4)</code> |

Tarea (NO calificable). Leer las secciones 9.5 y 9.6 de texto.

Procesamiento de archivos



```
#include<iostream>
#include<fstream>
#include<cstdlib> // necesaria para exit()
#include<string>
```

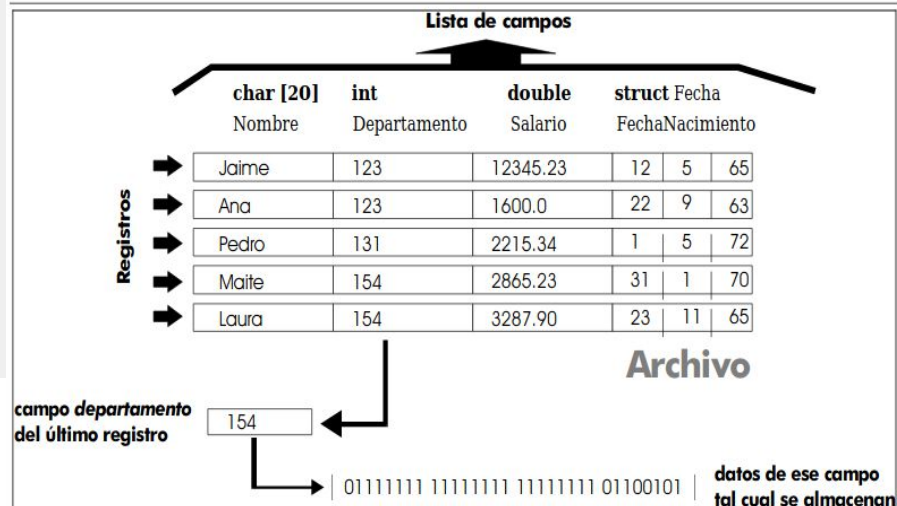
```
int main()
{
    cout << "xxxxxxx"<< endl;
    return 0;
}
```

```
-fstream archivoClientesEntrada( "clientes.txt",
ios::in );
```

```
-ifstream archivoClientesEntrada( "clientes.txt" );
```

```
string name_file = "clientes.txt"; // txt, tex, dat, etc
ifstream archivo_entr;
archivo_entr.open(name_file.c_str());
```

Prior to C++11, the filename was specified as a pointer-based string—as of C++11, it also can be specified as a string object.



Métodos de estado del archivo

| Indicador | Descripción |
|----------------|---|
| ios::in | Abre un archivo de texto en modo de entrada |
| ios::out | Abre un archivo de texto en modo de salida |
| ios::app | Abre un archivo de texto en modo anexar |
| ios::ate | Va al final del archivo abierto |
| ios::binary | Abre un archivo binario en modo de entrada (es archivo de texto el valor por omisión) |
| ios::trunc | Elimina el contenido del archivo si existe |
| ios::nocreate | Si el archivo no existe, la apertura falla |
| ios::noreplace | Si el archivo existe, falla la apertura para salida |

| Prototipo | Descripción |
|-----------|--|
| fail() | Devuelve un valor booleano verdadero si el archivo no se ha abierto con éxito; de lo contrario, devuelve un valor booleano falso . |
| eof() | Devuelve un valor booleano verdadero si se ha intentado leer más allá del final del archivo; de lo contrario, devuelve un valor booleano falso . El valor se vuelve verdadero sólo cuando se lee el primer carácter después del último carácter de archivo válido. |
| good() | Devuelve un valor booleano verdadero mientras el archivo está disponible para uso del programa. Devuelve un valor booleano falso si se ha intentado una lectura después del final del archivo. El valor se vuelve falso sólo cuando se lee el primer carácter después del último carácter de archivo válido. |
| bad() | Devuelve un valor booleano verdadero si se ha intentado una lectura después del final del archivo; de lo contrario, devuelve un valor falso . El valor se vuelve verdadero sólo cuando se lee el primer carácter después del último carácter de archivo válido. |

Funciones de marcadores de posición del archivo

| Nombre | Descripción |
|----------------------------------|--|
| <code>seekg(offset, mode)</code> | Para archivos de entrada, se mueve a la posición de desplazamiento indicada por el modo. |
| <code>seekp(offset, mode)</code> | Para archivos de salida, se mueve a la posición de desplazamiento indicada por el modo. |
| <code>tellg(void)</code> | Para archivos de entrada, devuelve el valor actual del marcador de posición del archivo. |
| <code>tellp(void)</code> | Para archivos de salida, devuelve el valor actual del marcador de posición del archivo. |

“Un objeto de flujo de archivo puede utilizarse como un argumento de función. El único requisito es que el parámetro formal de la función sea una referencia”

void inOut(ofstream&);

Tarea (NO calificable). Leer la seccion 8.4, y 8.5 de texto.

Ejemplo: la clase Tiempo

```
// evita múltiples inclusiones del archivo de encabezado
```

```
#ifndef TIEMPO_H
```

```
#define TIEMPO_H    E.Codigo time1 (clase9)  
                  E.Codigo example2_alcance  
...               E.Codigo FunUtilitarias  
#endif            E.Codigo time2
```

```
void Tiempo::imprimirEstandar() ← Sin parametros
```

```
{  
cout << ( ( hora == 0 || hora == 12 ) ? 12 : hora % 12 ) << ":"  
<< setfill( '0' ) << setw( 2 ) << minuto << ":" << setw( 2 )  
<< segundo << ( hora < 12 ? " AM" : " PM" );  
}
```

“setfill” es una opción “pegajosa”

Por lo general, el uso de una metodología de programación orientada a objetos puede simplificar las llamadas a las funciones, al reducir el número de parámetros que se deben pasar. Este beneficio de la programación orientada a objetos se deriva del hecho de que al encapsular los miembros de datos y las funciones miembro dentro de un objeto, se proporciona a las funciones miembro el derecho de acceder a los miembros de datos.



```
Tiempo llovertime; // objeto de tipo Tiempo  
Tiempo arregloDeTiempos[ 5 ], // arreglo de 5 objetos Tiempo  
Tiempo &LLegolaLLuvia = llovertime; // referencia a un objeto Tiempo  
Tiempo *tiempoPtr = &LLegolaLLuvia; // apuntador a un objeto
```


Destructores

- Es un error de sintaxis tratar de pasar argumentos a un destructor, especificar un tipo de valor de retorno para un destructor (ni siquiera se puede especificar void), devolver valores de un destructor o sobrecargarlo.
- El constructor para un objeto local automático se llama cuando la ejecución llega al punto en el que se define ese objeto; el correspondiente destructor se llama cuando la ejecución sale del alcance del objeto (es decir, el bloque en el que se define el objeto ha terminado de ejecutarse).
- Los objetos globales y static se destruyen en el orden inverso de su creación.

```
class CrearYDestruir
{
    public:
        CrearYDestruir( int, string ); // constructor
        ~CrearYDestruir(); // destructor

    private:
        int idObjeto; // número de ID para el objeto
        string mensaje; // mensaje que describe al objeto
};
```

E, Código Destruidores