Clase String

Repaso General
Unidades 4 y 5

(Capitulo 6 y 11 bibliografía)

» Ing. Ventre, Luis O.

CLASE STRING

- A lo largo del curso hemos usado por ejemplo el “objeto” cout, perteneciente a la clase “iostream”; sin conocer en detalle su estructura interna; la ventaja de los objetos es exactamente esta, poder utilizarlos sin mayor conocimiento de sus formalismos.

- Ahora usaremos otra clase proporcionada por la biblioteca estándar de C++ que es la clase **string**. Y crearemos objetos pertenecientes a esta clase antes de usarlos.

- Una clase, es un tipo de **dato no integrado en el compilador**, es necesario que el usuario lo construya usando código.

- Como todo tipo de dato un objeto de una clase, debe definir un conjunto de valores validos y un conjunto de operaciones.

CLASE STRING

- Los valores permitidos por la clase string se conoce como **literales de cadena**, que son cualquier secuencia de caracteres encerrada entre comillas.

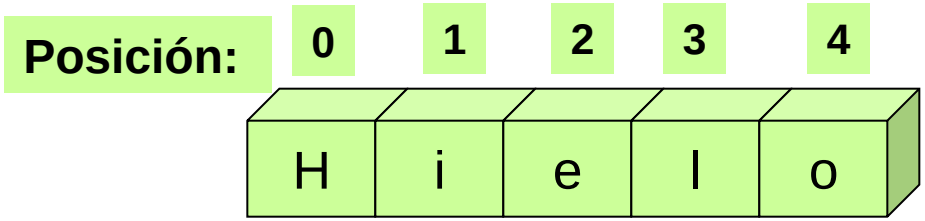
Ejemplos:
“Esta es una cadena”
“Hola mundo”
“xyz 123 *!·”

- Las comillas indican **inicio y final** y **no se almacenan con la cadena!!.**

- Por convención al **primer elemento** de una cadena se le asigna el **subíndice 0**.

CLASE STRING

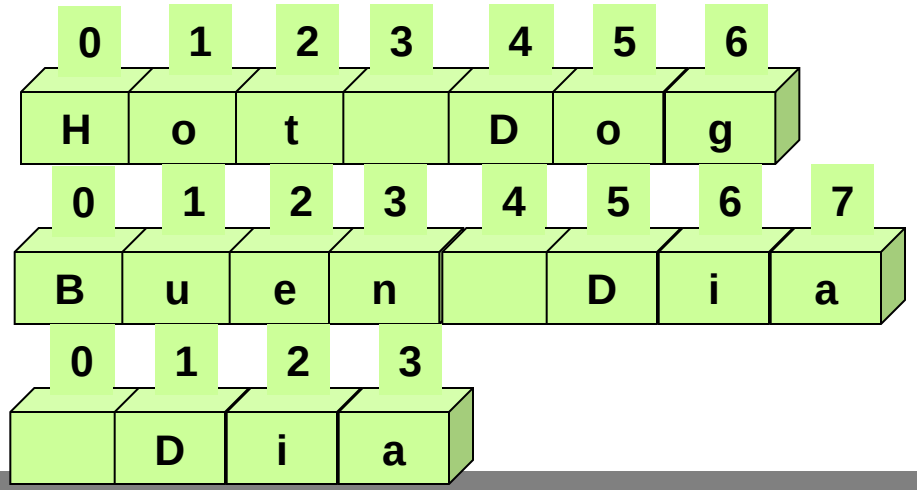
- Si se declara la cadena Hielo como un string la representación seria:



- FUNCIONES DE LA CLASE STRING:
- La clase string proporciona diversas funciones para declarar, crear, e inicializar una cadena.
- VER tabla 7.2 pagina 393 bibliografía.

CLASE STRING

- **Algunas de las funciones mas usadas son:**
- string nombreObjeto = “valor”
- string nombreObjeto (“valor”)
- string nombreObjeto (str1, m)
- **Ejemplos:**
- string str2=“Hot Dog”;
- string str3(“Buen Dia”);
- string str4(str3,4);



CLASE STRING – Entrada / Salida

- Además de inicializarse con los constructores mostrados anteriormente, **una cadena puede ingresarse por teclado e imprimirse en pantalla**.

- Puede utilizarse **cout, cin o getline**. Cin tiene la desventaja que deja de ingresar datos a la cadena cuando se ingresa **un espacio en blanco.**

- El método getline, permite determinar **cual será el elemento de fin de cadena**, o por omisión es el carácter de escape de línea nueva **\n** que se ingresa mediante la tecla **enter.**

- Forma mas general del método getline:

```
getline ( cin, strObj, carácter-de-terminación)
```

CLASE STRING – Entrada / Salida

- Ejemplos: En el primer caso, introduciré vía terminal o teclado los literales de la cadena str1 hasta que se ingrese “!” no incluido en cadena.
- En el segundo ejemplo, ingresare hasta que se presione enter, y en el tercero??

```
getline ( cin, str1, !)
```

```
getline ( cin, str1, \n)
```

```
getline ( cin, str1)
```

- Los objetos string, tienen ventajas como la asignación dinámica en memoria, y los métodos disponibles en la tabla 7.4 página 399 donde se puede determinar el largo, concatenar etc…

REPASO

- APLICACIONES - ARREGLOS Pág. 627 Bibliogr.

- ***Desarrollar un programa que acepte una lista de un máximo de 100 voltajes como entrada, determine tanto el promedio como la desviación estándar de los voltajes introducidos y luego despliegue los resultados.***

- Del análisis del enunciado se observa que se requieren dos salidas, el valor promedio de los voltajes ingresados y la desviación estándar. Como pueden ingresarse hasta 100 voltajes, la primer entrada será cuantos voltajes deseara ingresar.

REPASO

- Para el calculo del promedio de los valores ingresados, es simple deben sumarse el total de ingresos y dividirse por la cantidad.
- Para el calculo de la desviación estándar se debe realizar la siguiente formula.

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2}$$

- Los pasos para el calculo de la desviación estándar serán:
- ***Calcular el promedio.***
- ***Restar el promedio a cada voltaje individual; esto generara un nuevo grupo de datos llamado desviaciones.***
- ***Elevar al cuadrado cada una de las desviaciones.***
- ***Sumar las desviaciones cuadradas y dividir por la cantidad de desviaciones.***
- ***La raíz cuadrada del numero encontrado en el ultimo paso es la desviación estándar.***

REPASO

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

double calculades(double[], int, double);

int main()
{
  const int NUMELS = 100;

  int i, numvoltios;
  double voltios[NUMELS];
  double promedio, desvest;
  double sumavoltios = 0.0;

  cout << "Ingrese el numero de voltajes a ser analizados: ";
  cin  >> numvoltios;

  // lee los voltajes ingresados y los totaliza
  for (i = 0; i < numvoltios; i++)
  {
    cout << "Ingrese el voltaje " << i+1 << ": ";
    cin  >> voltios[i];
    sumavoltios = sumavoltios + voltios[i];
  }
```

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2},$$

REPASO

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2},$$

```
  }
  // calcula y despliega promedio
  promedio = sumavoltios / numvoltios;
  cout << "\nEl promedio de los voltajes es "
       << setw(11) << setiosflags(ios::showpoint)
       << setprecision(8) << promedio << endl;

  desvest=calculades(voltios, numvoltios, promedio);

  cout << "La desviacion estandard de los voltajes es "
       << desvest << endl;

system("PAUSE");
return 0;
}
```

Pasajes por valor, solo copia su contenido

Pasaje de arreglo como argumento

REPASO

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2},$$

La función que calcula la desviación es:

```
double calculades( double volts[], int numvolts, double promedi)
{
    int i;
    double sumadesv=0.0;
    double desvest=0.0;

  for (i = 0; i < numvolts; i++)//sumatoria de desviaciac. cuad.
  sumadesv = sumadesv + pow((volts[i] - promedi),2);

  desvest = sqrt(sumadesv/numvolts); //divido y saco raiz cuad.

  return desvest;

}
```

REPASO

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \overline{x})^2},$$

La salida producida por la ejecución es:

```
Ingrese el numero de voltajes a ser analizados: 10
Ingrese el voltaje 1: 98
Ingrese el voltaje 2: 82
Ingrese el voltaje 3: 67
Ingrese el voltaje 4: 54
Ingrese el voltaje 5: 78
Ingrese el voltaje 6: 83
Ingrese el voltaje 7: 95
Ingrese el voltaje 8: 76
Ingrese el voltaje 9: 68
Ingrese el voltaje 10: 63

El promedio de los voltajes es   76.400000
La desviacion estandard de los voltajes es 13.154467
Presione una tecla para continuar . . .
```

Recomendación: Ver segundo caso de aplicación, lista ordenada página 630 Bibliografía.

- Unidad 4 – Unidad 5

REPASO

» Ventre, Luis O.

Declaración de Funciones y Parámetros

- ANTES QUE UNA FUNCION PUEDA SER LLAMADA DEBE SER DECLARADA, LA DECLARACION INICIAL DE LA MISMA SE CONOCE COMO **PROTOTIPO DE LA FUNCION**

- **En el prototipo de una función se puede reconocer, su NOMBRE, sus datos validos como ARGUMENTOS Y EL ORDEN, y su valor DEVUELTO.**

- **Por ejemplo el siguiente prototipo:**

```
void encontrarMax(int, int);
```

Declara una función llamada encontrarMax, la cual recibe dos valores enteros como parámetro y no devuelve ningún valor(void).

Declaración de Funciones y Parámetros

- La forma general de escritura de un prototipo de función es:

```
tipo-de-datos-a-devolver  nombre-de-función  (lista de tipos de datos argumento)
```

- **Ejemplos de prototipos de funciones:**

```
int fmax(int, int);

double intercambio(int, char, char, double);
```

- **Los prototipos de funciones permiten la verificación de errores en los tipos de datos por el compilador. Y asegura la conversión de todos los argumentos enviados al tipo de datos declarado.**

Devolver un solo valor

Para que la función retorne el valor solo es necesario colocar la instrucción:

```
return expresion;
```

Debe cuidarse para evitar errores indeseados que el **tipo de dato devuelto por la función** y el **tipo utilizado en la instrucción de return coincidan!.**

Desde el punto de vista del receptor, la función que llama debe:

* Ser alertada del tipo de valor a esperar (**prototipo de función**)
* Usar de manera apropiada el valor.

La variable utilizada para almacenar el valor devuelto debe ser del mismo tipo de dato.

```
max = encontrarMax(num1, num2);
```

```
cout<<encontrarMax(num1, num2);
```

PASAJE ARGUMENTOS

Al llamar a una funcion sumar, y colocar el nombre de la variable total como argumento:

```
int total=10;
sumar(total);
```

por valor

```
void sumar(int a)
```

Si el prototipo de la funcion, en su argumento tiene una declaracion de int a, SOLO se copia el valor de la variable total en la variable a.

Y ésta última NO PUEDE modificar el valor de la variable TOTAL.

2 variables diferentes: **a y total**

por referencia

```
void sumar(int& a)
```

Si el prototipo de la funcion, en su argumento tiene una declaracion de parametro de REFERENCIA con & a, pasa a ser una referencia de total.

Y al MODIFICAR esta última en la función se modifica TOTAL.

1 misma variable, dos nombres dif.

PASAJE ARGUMENTOS

Al llamar a una funcion sumar, si deseo pasarle uno o mas valores de un arreglo por valor, debo colocar cada subindice:

```
int total[10];
sumar(total[0],total[1]);
```

↓

```
void sumar(int a, int b)
```

El prototipo de funcion tiene como argumento dos variables del mismo tipo de los elementos del arreglo.

Se copian los valores de los elementos del arreglo a las variables nuevas a y b

```
int total[10];
sumar(total);
```

↓

```
void sumar(int a[ ])
```

El prototipo de la funcion alerta que el argumento es un arreglo. De esta forma se tiene acceso a todo el arreglo y se modifica desde la funcion los valores del mismo

1 mismo arreglo, dos nombres dif.

Devolver múltiples valores

Existen ocasiones en donde es necesario darle a la función llamada acceso directo a las variables de la función que llama.

Para lograr esto se necesita enviarle a la función llamada, **la dirección de la variable.**

Una vez que la función llamada conoce la dirección de la variable "conoce donde vive" y puede tener acceso y cambiar su valor de manera directa.

Este tipo de transmisión se llama **"pasaje o transmisión por referencia".**

C++ proporciona dos tipos de parámetros de dirección, referencia y apuntadores. En esta materia veremos referencia.

Transmisión de parámetros de referencia

La invocación a una función con pasaje por referencia desde el emisor es idéntica. **Solo debe declararse los tipos de parámetros como referencia**; esto se hace con la siguiente sintaxis:

```
tipo-de-datos& nombre-referencia
```

Ej.:

```
double& num1;
```

Indica que num1 es un parámetro de referencia que se utilizara para almacenar la dirección de un double. O leer al revés por ejemplo num1 es la dirección de una variable de precisión doble.

Arreglos Unidimensionales

- Lista de valores relacionadas con **“el mismo tipo de datos”** que se almacena bajo un **“nombre de grupo único”**.

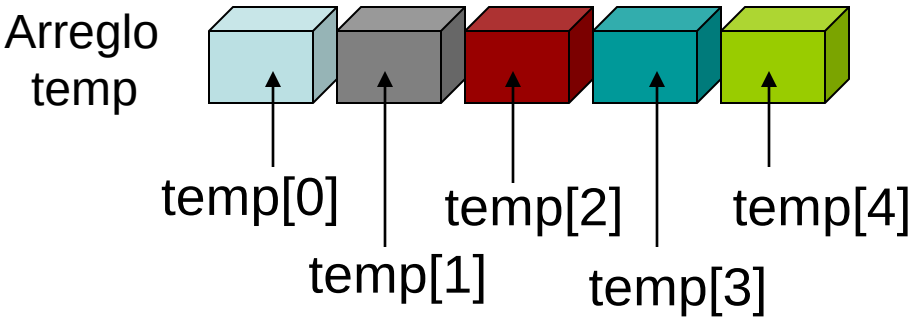
- En la declaración de un arreglo de dimensión única es necesario indicar:
  - ***El tipo de datos del conjunto***
  - ***El nombre del arreglo o del grupo.***
  - ***La cantidad de elementos del grupo entre corchetes [ ]***

Sintaxis de declaración:

```
tipo-de-datos nombreArreglo [numero-elementos]
```

Arreglos Unidimensionales

- Continuando con el ejemplo anterior los subíndices de los componentes serán:



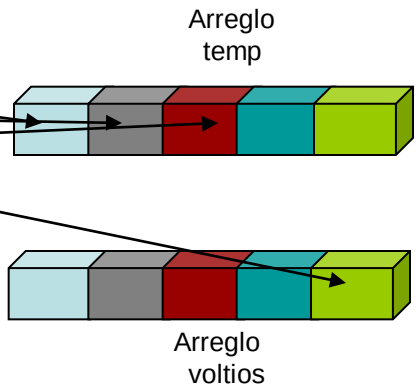
- ***Una vez declarado el array, cada componente es una variable indexada ya que debe darse su nombre y su subíndice para hacer referencia a ese elemento.***

Entrada y Salida de valores del arreglo

- ***A los objetos del arreglo se les puede asignar valores de manera interactiva usando cin!.***
- ***Ej:***

```
cin>> temp[0];

cin>> temp[1] >> temp[2] >> voltios [4];
```



- *De manera análoga, puede utilizarse un ciclo for para la introducción interactiva de todos los datos del arreglo:*

```
for ( i=0; i<numels; i++)
{
   cout<<“Introduzca el elemento “<<i;
   cin>>temp[i];
}
```

Inicialización de arreglos

- ***Al igual que las variables vistas, los arreglos pueden inicializarse cuando son declarados, la DIFERENCIA entre ambas declaraciones radica en que los valores del arreglo deben ir entre llaves { }***

```
int temp[5] = { 98, 87, 92, 79, 85};

char codigos[6] = { 'm', 'u', 'e', 's', 't', 'r', 'a'};
```

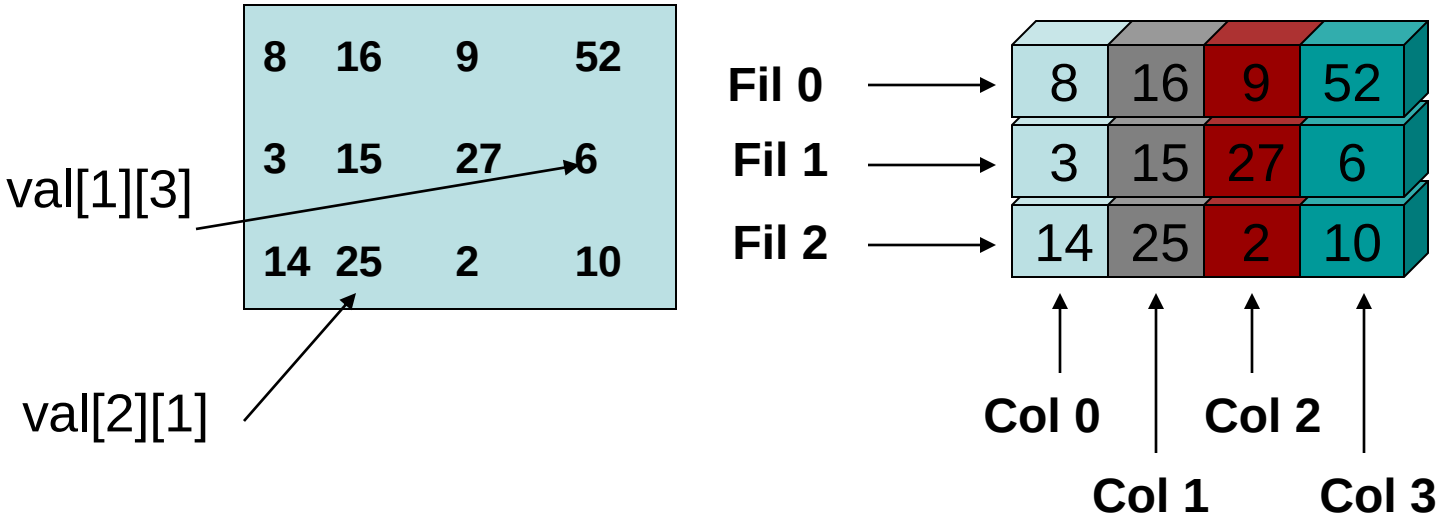
- ***La inicialización de valores puede extenderse a múltiples líneas:***

```
int voltios [9] = { 98, 87, 92,
                    79, 85, 66,
                    94, 55, 67};
```

- ***Si el numero de valores es menor serán inicializados a "0".***

Arreglos BIDIMENSIONALES

- **Un arreglo bidimensional, a veces llamado tabla, es un arreglo de elemento que posee filas y columnas.** Por ejemplo un arreglo bidimensional de números enteros se observa a continuación**:**



- **Para reservar los lugares de almacenamiento en su declaración deben incluirse el numero de filas y el numero de columnas**

```
int val [3] [4];
```

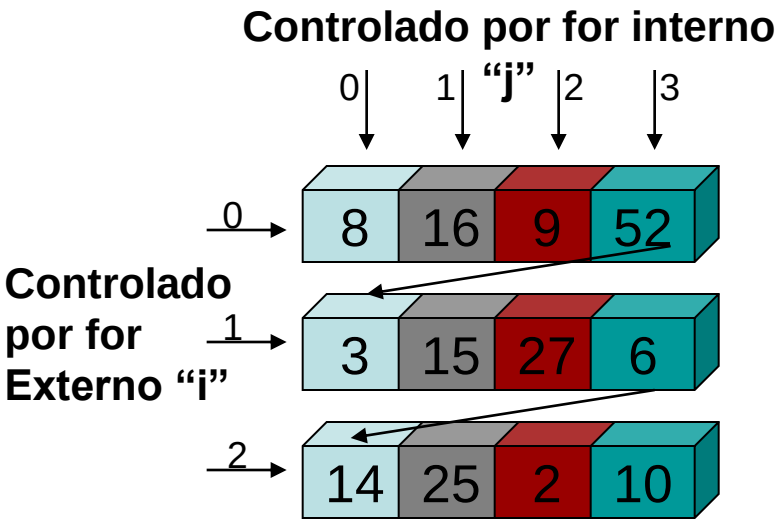
Entrada y salida de valores Arreglos BIDIMENSIONALES

- Un **ciclo for** exterior recorrerá **las filas** o renglones.

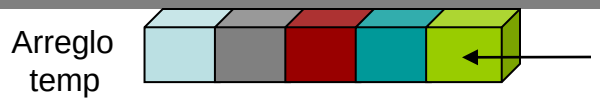
- Un **ciclo for** interior recorrerá **las columnas**.

```
for ( i=0; i<FILAS; i++) // ciclo externo
{
    for( j=0; j<COLUMNAS; j++) //ciclo interno
    {
      cout<<“Ingrese el elemento temp“<<i<<j;
      cin>>temp[i][j];
}
```

```
for ( i=0; i<FILAS; i++) // recorre filas
{   cout<<endl;
    for( j=0; j<COLUMNAS; j++) // recorre colum.
    {
      cout<<“El elemento temp“<<i<<j<<“es: “;
      cout>>temp[i][j];
}
```



Arreglos Como ARGUMENTOS



- Ante este problema es posible poner a disposición de la función todo el arreglo. Esto se logra con la siguiente llamada:

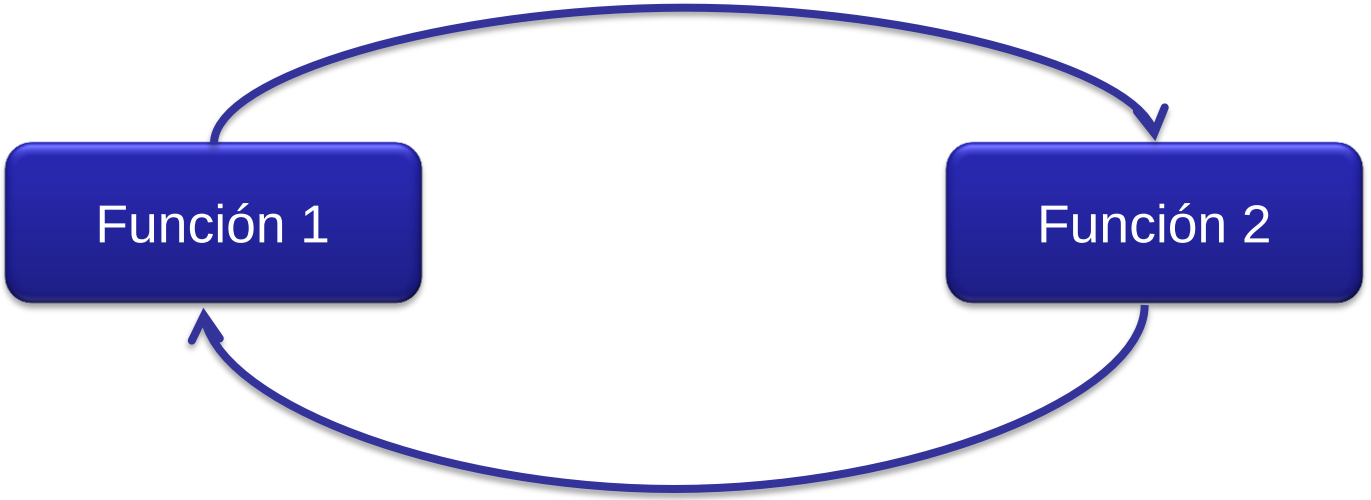
```
hallarmin(voltios);
```

- Esta llamada, envía a la función la dirección del primer componente del arreglo, y de esta manera en la función se accede directamente a todo el arreglo de manera análoga a como sucedía con el “pasaje por referencia”.

- En el caso del llamado anterior la función llamada debe ser alertada que el argumento es un arreglo por lo que su encabezado será:

```
int hallarmin (int voltios[5])
{….
```

- Hemos visto que DESDE una función, podemos llamar a OTRA función.



- Que pasa, si esta segunda FUNCION, llama a la PRIMERA que la invoco a ella?

- Y peor aun, que pasaría si una función se llama a si misma?

- Ambos casos son posibles!.

- Una función que se llama a si mismo es **RECURSIVIDAD DIRECTA**.
- El anterior, que incluye una función numero 2, es llamado **RECURSIVIDAD INDIRECTA**.