- Unidad 2

INTRODUCCION A LA ESPECIFICACION DE PROGRAMAS IMPERATIVOS

(Capítulos 2 y 3 bibliografía)

» Ventre, Luis O.

- Un buen programa requiere de planeamiento y diseño antes de su ejecución
- Programas **MODULARES**, estructura de segmentos interrelacionados para formar una **unidad completa**.
- Cada modulo realiza una tarea especifica
- Modulo = Clase o **función.**
- **Requerimiento base: CORRECTA IDENTIFICACION.**

- IDENTIFICADORES
- **Un identificador se puede formar con:**

  con letras

  minúsculas (de la ***a*** a la ***z***)

  mayúsculas (de la ***A*** a la ***Z***)

  con dígitos

  del ***0*** al ***9***

  con el carácter ***subrayado*** (_)

- **Un identificador NO se puede formar con:**

  con **espacios en BLANCO !!!**

  con otros caracteres como *** , ; . : - +**

- Además el primer carácter DEBE ser una **letra o el carácter subrayado _**
- Solo pueden **seguir a la letra inicial** los caracteres antes mencionados como posibles de formar un identificador
- El nombre de la función o identificador nunca puede ser una de las **PALABRAS CLAVES DEL LENGUAJE**
- El numero máximo de caracteres de un identificador lo impone el compilador (1024 mínimo según standard)
- Por ultimo el nombre de la función debe ser **MNEMONICO**.
- Recordar que el lenguaje C **es sensible al uso de mayúsculas** y minúsculas. Por lo tanto para el compilador será lo mismo:

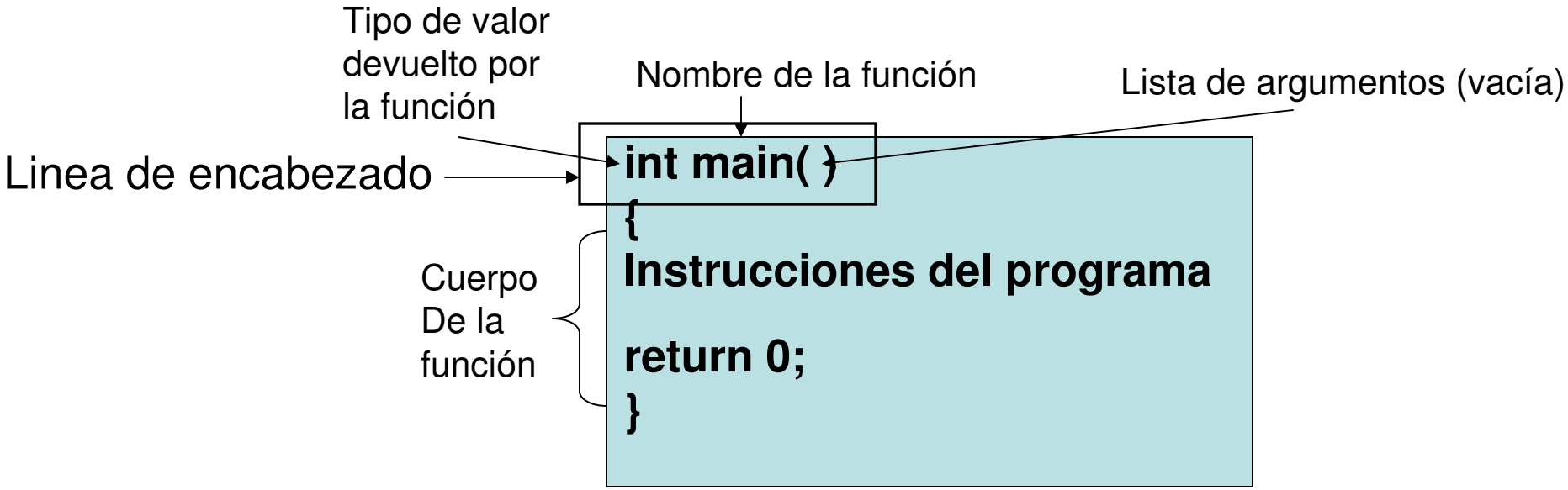
TOTAL, que Total, que TotaL??.....

Palabras claves

- **else - do - while - for - switch - auto**
- **short - long - extern - static - default**
- **continue - break - register - sizeof - typedef**
- **char - int - float - double - if**
- **…**
- **Todas en detalle tabla 2.1 pagina 36 Bronson.**

- **La función MAIN()**

- Para facilitar la colocación y ejecución ordenada de los módulos, cada programa C++ debe tener **una y solo una función llamada main().**
- Esta se conoce como una función controladora, debido a que **indica secuencia.**



- La primer línea de la función se conoce como **línea de encabezado de la función**

- **Cada instrucción dentro del cuerpo de la función debe terminar en un ;**

- **El objeto COUT:**

Deriva de **Console out**, es un objeto de salida que deriva datos introducidos en él al dispositivo estándar de salida.

- Símbolo de inserción “<<” enviar a.

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

int main()
{
    cout<<"Hola MUNDO!";
    return 0;
}
```

- La primer línea del programa, es un **comando de pre-procesador** que utiliza palabra reservada include, este indica una acción a ejecutar antes de que el **compilador ejecute**.
- En este caso incluir el contenido del archivo invocado “iostream”.
- *Estos comandos de preprocesado no finalizan con ;*

- La instrucción **using namespace** especifica que los miembros de un **namespace** van a utilizarse frecuentemente en un programa. Esto permite al programador tener acceso a todos los miembros del **namespace** y escribir instrucciones mas concisas como:

```
cout<<"hola";
```

en vez de:

```
std::cout<<“Hola”;
```

- **Secuencia de escape: \n**
- Envían información para comenzar una línea nueva.

- Se puede utilizar secuencias de escape en cualquier parte del mensaje enviado al objeto cout.
- Cual es la salida del siguiente programa?

```
Computadoras por todo lados
 tan lejos como

   pueda llegar C
```

```
#include <iostream>

using namespace std;

int main()
{
    cout<<"Computadoras por todo lados \n tan lejos como\n\n   pueda llegar C";

    getchar();
    return 0;

}
```

Estilo de programación

- Los programas en C++ comienzan su ejecución en la función main, y como todo programa solo puede tener un punto de inicio, solo puede existir una **UNICA función main.**

- Es importante destacar que puede colocarse mas de una instrucción por línea; **C++ ignora todo espacio en blanco** (excepto cadenas, identificadores, comillas y palabras clave).

```
int main
(
){
cout<<“Hola Mundo”;
return 0;
}
```

Estilo de programación pobre poco entendible

```
int main ( )
{

    cout<<“Hola Mundo”;

    return 0;
}
```

Estilo programación correcto

Estilo de programación

- **Comentarios:**
- Son observaciones explicativas que se hacen dentro de un programa.
- C++ acepta dos tipos de comentario: ***de línea y de bloque***
- El compilador **ignora** todos los comentarios.

Un comentario de línea comienza con las // y continua hasta el final de la línea.

Un comentario de línea puede escribirse al inicio de la misma o al final de la instrucción de esa línea.

```
//este programa despliega un mensaje
#include <iostream>
using namespace std;

int main()
{
   cout<<"Hola MUNDO!"; //esto produce el despliegue
   //este comentario produce
     un error de c++
   return 0;
}
```

Estilo de programación

- Cuando el comentario contiene mas de 2 líneas, es conveniente utilizar comentarios de bloque.
- **Estos comienzan con /* y finalizan en */.**
- La utilización escasa de comentarios es señal de mala programación, y se hace presente cuando se desea mantener o ser leído por otro programador.
- La utilización en extremo de comentarios es señal de mala programación, debido a que la lógica del programa no corresponde al un pensamiento sencillo.

EJEMPLO

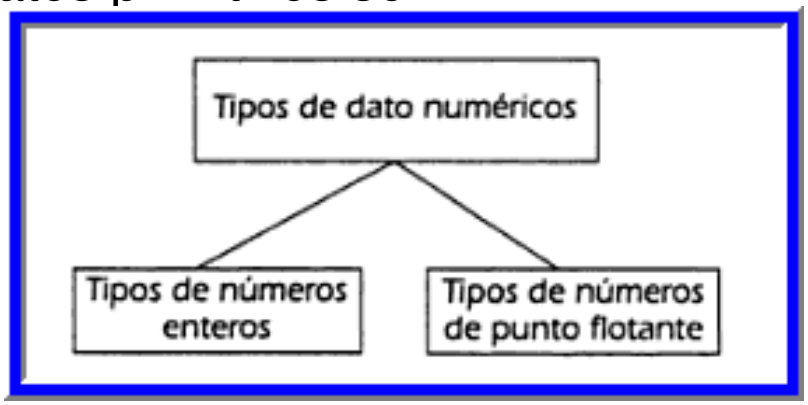
```
/* este es un comentario
En bloque que ocupa
Un total de 4 lineas de texto
Y finaliza aqui */
```

Tipos de datos

- El objetivo de todo programa es procesar datos, es fundamental para esto la clasificación de estos datos en tipos específicos.
- Los **tipos de datos permitidos y las operaciones apropiadas** definidas para cada tipo, se conocen como **tipo de dato**. Por lo tanto un tipo de datos es un rango de valores y un conjunto de operaciones que pueden aplicarse a esos valores.

- Tipo de *dato de clase y tipos de datos integrados (o primitivos).*

- Los tipos de datos primitivos son:



Tipos de datos

- **Tipos de datos enteros:**
- C++ proporciona 9 tipos de datos enteros, la diferencia esencial entre ellos es la cantidad de almacenamiento usado por cada tipo. Esto afecta el rango de valores que pueden tomar

Tipos de datos enteros:

* **bool** * **char** * short int * **int**
* long int * unsigned char * unsigned short int
* unsigned int * unsigned long int

- Los 3 tipos mas usados son *int, char y bool.*

- **El tipo de dato INT**

➢ Conjunto de los números enteros positivos y negativos
➢ En la práctica están limitados por el almacenamiento -> max 4 bytes
➢ Operaciones básicas: aritméticas, relacionales

Tipos de datos

- Enteros validos: 0 5 -10 +25 1000 ….
- Enteros inválidos: $255.62 3. 1482.32 +6.0

- **El tipo de dato CHAR**
- Almacena **caracteres individuales** y mediante el código ASCII se almacena internamente como enteros.
- Incluyen las letras del alfabeto mayúsculas y minúsculas
- Incluye los diez dígitos 0 al 9.
- Incluye símbolos especiales como + $. , !
- **Un valor de carácter es cualquiera antes mencionado encerrado entre comillas simples.**

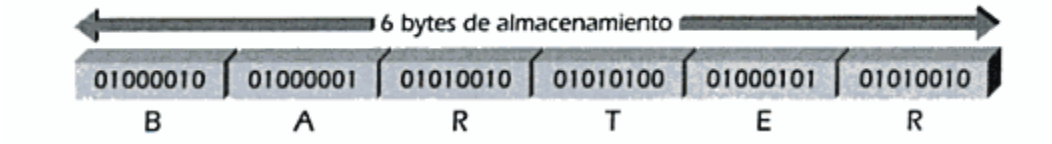


Tabla 2.3 Los códigos ASCII para las letras mayúsculas

Letra	Código ASCII	Letra	Código ASCII
A	01000001	N	01001110
B	01000010	O	01001111
C	01000011	P	01010000
D	01000100	Q	01010001
E	01000101	R	01010010
F	01000110	S	01010011

Tipos de datos

- El carácter de escape: "\"
- Indica al compilador que **"escape"** de la forma normal de interpretar algunos caracteres.
- La combinación de este carácter de escape junto a estos caracteres se denomina **secuencia de escape**

Tabla 2.4 Secuencias de escape

Secuencia de escape	Carácter representado	Significado	Código ASCII
\n	Línea nueva	Se mueve a una línea nueva	00001010
\t	Tabulador horizontal	Se mueve a la siguiente posición del tabulador horizontal	00001001
\v	Tabulador vertical	Se mueve a la siguiente posición del tabulador vertical	00001011
\b	Retroceso	Retrocede un espacio	00001000
\r	Retorno de carro	Mueve el cursor al inicio de la línea actual; se escribe para sobrescribir	00001101
\f	Alimentación de forma	Expulsa una hoja para iniciar otra	00001100
\a	Alerta	Emite una alerta (por lo general un sonido de campana)	00000111

Tipos de datos

- **El tipo de dato BOOL:**
- Se utiliza para representar datos booleanos (**lógicos**), se restringe la cantidad de valores posibles a 2: verdadero o falso.
- Este tipo de datos es fundamental cuando un programa debe **evaluar una condición especifica de un estado a verdadero o falso**.

- Determinación del tamaño de almacenamiento:
- Operador denominado **sizeof**()
- Proporciona el numero de bytes utilizado por cualquier nombre de tipo de datos incluidos dentro del paréntesis.

```
int main()
{
  cout << "\nTipo de datos  Bytes"
       << "\n---------  -----"
       << "\nint           " << sizeof(int)
       << "\nchar          " << sizeof(char)
       << "\nbool          " << sizeof(bool)
       << '\n';

  return 0;
```

```
Tipos de datos   Bytes
--------------   -----
inf                4
char               1
bool               1
```

Tipos de datos

- **Tipos de datos con signo y sin signo**
- El lenguaje proporciona un calificador de tipo de dato que permite indicar si los datos a utilizar **tendrán signo o no**. Un tipo de dato con signo puede almacenar tanto el 0 como valores positivos y además negativos. Un tipo de dato sin signo solo podrá almacenar 0 y valores positivos.
- De esta forma puede **customizarse** el uso del rango de valores de un tipo de dato.

Tipo de Datos	Tamaño del Almacenamiento	Rango de Valores
char	1 byte	256 caracteres
bool	1 byte	Verdadero o Falso
short int	2 bytes	"-32768 a 32767"
unsigned short int	2 bytes	0 a 65535
int	4 bytes	"-2147483648 a 2147483647"
unsigned int	4 bytes	0 a 4294967295
long int	4 bytes	"-2147483648 a 2147483647"
unsigned long int	4 bytes	0 a 4294967295

Tipos de datos

- **Tipos de datos de punto flotante:**
- Son llamados números reales, pueden ser el 0, cualquier numero positivo o negativo que contenga un **punto decimal**.
- Igual que en los enteros no se permiten signos especiales como el $, etc.
- C++ acepta tres tipos de datos de punto flotante, cuya diferencia otra vez es el espacio de almacenamiento utilizado para cada uno.

- **Float Double Long Double.**

- La mayoría de los compiladores utilizan el doble de espacio para almacenar un double vs un float. Por esto usualmente se conoce a un valor FLOAT como de precisión “simple” y un DOUBLE como un dato de **precisión “doble”.**
- Almacenamiento de un float 4 bytes
- Almacenamiento de un double 8 bytes.

Tipos de datos

- **Notación Exponencial:**
- Los números en punto flotante pueden escribirse en notación exponencial. Esta notación es semejante a la notación científica y se utiliza para expresar en forma compacta valores grandes y pequeños.

	Decimal	Exponencial	Científica
•	1625	1.625 e3	1.625×10^{3}
•	63421	6.3421 e4	6.3421×10^{4}
•	0.000625	6.25 e-4	6.25×10^{-4}
•	0.00731	7.31 e-3	7.31×10^{-3}

Operaciones Aritméticas

- Anteriormente vimos los valores posibles para los tipos de datos integrados de c++. Ahora veremos las operaciones aritméticas posibles para estos.
- Los ***enteros y los reales*** pueden:

  Sumarse Restarse Multiplicarse Dividirse

- Aunque por lo general es mejor no mezclar la utilización de enteros con reales.

- Los operadores aritméticos son:

  Adición + Sustracción - Multiplicación * División /

  División de modulo %

Estos operadores son conocidos como **operadores binarios**, debido a la cantidad de operandos necesarios para la operación.

Operaciones Aritméticas

- Además de los operadores binarios, existen los **operadores unitarios**, los cuales solo requieren de un operando.
- Expresión aritmética binaria simple:

valorliteral operador valorliteral

Se puede utilizar cout, para desplegar en pantalla el resultado de cualquier expresión aritmética.

Además a cout pueden enviarse dos o mas piezas de datos, por ejemplo una cadena y una expresión aritmética, siempre separando cada pieza por el propio símbolo de inserción <<.

Ej.:

```
cout<<“La suma de 12.2 y 15.754 es igual a: “<< (12.2 + 15.754);
```

Produce…

```
La suma de 12.2 y 15.754 es igual a: 27.954
```

Operaciones Aritméticas

- cout, puede extenderse a mas de una línea, pero solo debe llevar un ; al final de la instrucción y si hay una cadena de carácter esta no puede extenderse de una línea.

```
cout << "La suma de 12.2 y 15.754 es "
     <<  (12.2 + 15.754);
```

- Ej.

Programa 2.6

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

int main()
{
  cout << "15.0 más 2.0 es igual a "           << (15.0 + 2.0) << endl
       << "15.0 menos 2.0 es igual a "         << (15.0 - 2.0) << endl
       << "15.0 por 2.0 es igual a "           << (15.0 * 2.0) << endl
       << "15.0 dividido entre 2.0 es igual a " << (15.0 / 2.0) << endl;

  return 0;
}
```

```
15.0 más 2.0 es igual a 17
15.0 menos 2.0 es igual a 13
15.0 por 2.0 es igual a 30
15.0 dividido entre 2.0 es igual a 7.5
```

Operaciones Aritméticas

- Expresión: **combinación de operadores y operandos** que pueden ser evaluados para producir un valor. Existen 3 tipos de expresiones:

  Enteras *De punto flotante* *En modo mixto*

- División de enteros: **parte fraccionaria truncada**. 15/2 = ?

- Si fuese necesario la parte fraccionaria c++ **proporciona el operador modulo, u operador de residuo "%".** Este captura el residuo cuando un numero entero es dividido entre otro numero entero.

- Ej. 9 % 4 es 1 (el residuo cuando 9 se divide entre 4 es 1)
  17 % 3 es 2 (el residuo cuando 17 se divide entre 3 es 2)

  Que resultado generara la expresión 3 % 5 ?...

Operaciones Aritméticas

- **Operador Unitario:**

Este tipo de operador opera sobre un solo operando; uno de estos operadores es el de *negación*, el mismo utiliza el mismo signo que la resta pero precediendo al operando invierte su signo.

Ver tabla de resumen de operadores aritméticos Tabla 2.7 – Pág.66.

- **Precedencia del operador y asociatividad:**

Existen ciertas reglas para escribir expresiones que contengan mas de un operador aritmético.

1-Nunca deben colocarse **2 operadores binarios uno seguido del otro**

2-**Los paréntesis priorizan** el cálculo de la expresión. Esto permite alterar el orden.

3-Paréntesis dentro de paréntesis se evalúan de adentro para afuera.

4-Nunca usar paréntesis para indicar multiplicación se usa *.

Operaciones Aritméticas

- **Niveles de precedencia y asociatividad:**
- Existen 3 niveles de precedencia, los cuales determinan el orden de resolución de la expresión aritmética.
- **P1:** Primero todas las *negaciones.*
- **P2:** *Multiplicación, división y modulo.* Cuando hay mas de un operador de estos se resuelve de izquierda a derecha.
- Cual es el resultado de: 40/8%3*5 = ?
- **P3:** *Adición y sustracción* se calculan al ultimo. Al igual que con los p2 si hay mas de una adición o sustracción se resuelven de izquierda a derecha.
- **Que resultado dará 6.0*6/4 = ?**
- **….9!**
- **Y 40 / 8 % 3 * 5 = ?**
- **….10!**
- **Y 6 + 10 / 5 – 2?**
- **….6!**

Operador	Asociatividad
unitario –	derecha a izquierda
* / %	izquierda a derecha
+ –	izquierda a derecha

Variables e instrucciones de declaración

- Recordemos que antiguamente se hacia referencia a la ubicación donde se almacenaban los datos. Esto era engorroso.
- Ej. para sumar 2 números:

Coloque un 45 en la ubicación 1652
Coloque un 12 en la ubicación 2548

Para sumar los dos números que se acaban de almacenar y guardar el resultado en otra ubicación de memoria, por ejemplo en la ubicación 3000, se necesita una instrucción comparable a

Sume el contenido de la ubicación 1652
al contenido de la ubicación 2548
y almacene el resultado en la ubicación 3000

Variables e instrucciones de declaración

- Para simplificar esto en c++ al lugar de dirección de memoria se le dice **"variable". Por lo tanto una variable es un nombre simbólico utilizado por el programador para designar un espacio de almacenamiento de la memoria.**
- Se utiliza el termino variable porque el contenido de esa posición de memoria puede **cambiar o variar**.
- Para dar un nombre a la variable se debe cumplir con los **identificadores** antes vistos.
- Ahora el ejemplo quedaría:


```
num1 = 45;
num2 = 12;
total = num1 + num2;
```

Variables e instrucciones de declaración

- Cada una de estas tres instrucciones se denomina ***instrucción de asignación.*** Porque le indica a la computadora que **asigne o almacene un valor en la variable.**

```
num1 = 45;
num2 = 12;
total = num1 + num2;
```

- Toda instrucción de asignación tiene un signo =, y a la **izquierda un nombre de variable**, a la cual se le asigna el valor ubicado a la **derecha del signo =.**

- Ya que la utilización de variables libera al programador de saber donde y como esta almacenado el valor del dato, es necesario indicar el **tipo de datos que se van a almacenar** en la variable para que el compilador **asigne el espacio correspondiente**.

Variables e instrucciones de declaración

- **Instrucciones de declaración:**
- **Darle un nombre a una variable** y esp
  pueden almacenarse en ella se logra co
  declaración; que tiene la forma:

Tipo-de-datos nombreDeVariable;
int suma; ó float primernumero; ó double s

Las instrucciones de declaración pueden c
programa, pero es una buena practica de p
inicio y separadas por una línea en blanco

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

int main()
{
   int primero;
   double segundo;
   char a;

   otras instrucciones;

    return 0;
}
```

Instrucc. declarac.

Además, toda variable DEBE SER DECLARADA antes de ser USADA.

Variables e instrucciones de declaración

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

int main()
{
  double calif1; // declara calif1 como una variable de precisión doble
  double calif2; // declara calif2 como una variable de precisión doble
  double total;   // declara total como una variable de precisión doble
  double promedio; // declara promedio como una variable de precisión doble

  calif1 = 85.5;
  calif2 = 97.0;
  total = calif1 + calif2;
  promedio = total/2.0;  // divide el total entre 2.0
  cout << "El promedio de las calificaciones es " << promedio << endl;

  return 0;
}
```

```
El promedio de las calificaciones es 91.25
```

Variables e instrucciones de declaración

- **Declaraciones Múltiples:**
- Las variables del mismo tipo pueden declararse en una sola instrucción, utilizando solo una vez el tipo de dato y luego la lista de nombres de variables separadas por una “,”.

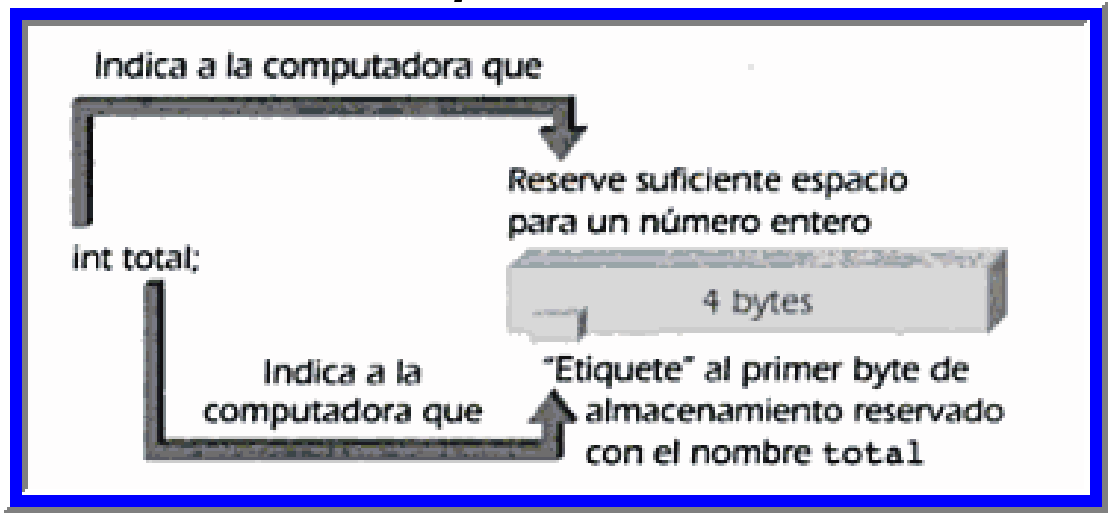
Tipo-de-Datos listaDeVariables
double calif1, calif2, total, promedio;

En las **instrucciones de declaración** también puede asignársele un valor inicial a la variable, esto se denomina ***inicializar*** la variable.

```
double calif1 = 87.0;
double calif2 = 93.5;
double total;
```

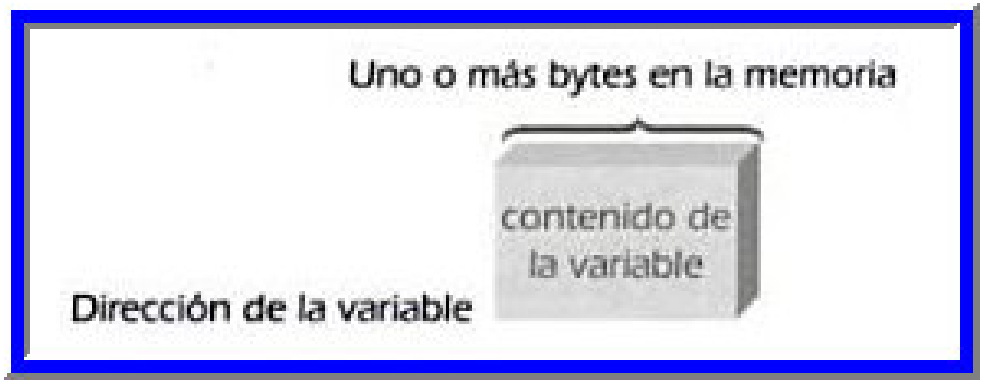
Variables e instrucciones de declaración

- **Asignación de memoria:**
- Las instrucciones de declaración cumplen funciones en el software para facilitar la programación y la prevención de errores de tipeo. Pero además en el hardware fuerzan al compilador a **reservar el espacio de memoria correspondiente** al tipo de dato solicitado.
- Las instrucciones de declaración utilizadas con este propósito son llamadas ***instrucciones de definición.***
- ***Conocer el tamaño de memoria a reservar le permite al compilador almacenar o recuperar el numero correcto de bytes***



Variables e instrucciones de declaración

- **Dirección de una variable:**
- Cada variable tiene 3 elementos importantes: su **tipo de datos**, el **valor almacenado** en la variable y la **dirección de la variable**.



- Normalmente el programador esta interesado en el **contenido de la variable** y no en el lugar donde se encuentra físicamente en la memoria.

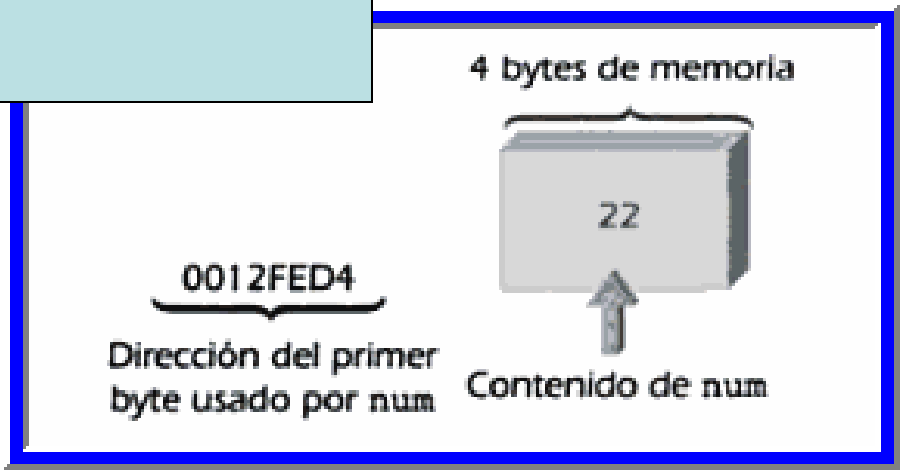
- Para determinar la dirección de una variable en memoria, se usa el operación de dirección **"&"** ("la dirección de").

Variables e instrucciones de declaración

- **Supongamos un programa que incluye:**

```
…..
int num;

num=22;

cout<<“El valor almacenado en num es ”<<num<<endl;

cout<<“La dirección donde esta num es =“<<&num<<endl;

….
```

- La salida que producirá será:



El valor almacenado en num es 22
La dirección donde esta num es =0012FED4

Errores comunes de programación

- **Parte de aprender un determinado lenguaje, es cometer los errores elementales que se encuentran normalmente al comenzar a programar. Pueden ser frustrantes.**

- Algunos de los errores mas comunes cuando se empieza a programar en C++ son:

- 1-Omitir los paréntesis después del encabezado de función main

- 2-Omitir o escribir de manera incorrecta la llave de apertura { que indica el inicio de un cuerpo de función.

- 3-Omitir o escribir de manera incorrecta la llave de cierre } que indica el final de un cuerpo de función.

Errores comunes de programación

- 4- Escribir mal el nombre de un objeto o función, por ej. escribir **cot** en vez de **cout**.
- 5-Olvidar enviar una cadena enviada a cout con un símbolo de comillas.
- 6-Olvidar separar flujos de datos individuales pasados a cout con un símbolo de inserción (“enviar a”) <<.
- 7-Omitir el punto y coma al final de cada instrucción de c++
- 8-Agregar un punto y coma al final del comando de pre-procesador #include.
- 9-Escribir incorrectamente la O en vez de cero. Y lo mismo para la letra I y el 1.

Errores comunes de programación

- 10-Olvidar declarar todas las variables utilizadas en el programa.
- 11-Almacenar un tipo de datos inapropiado en una variable declarada. Conversión al tipo de datos destino.
- 12-Usar una variable en una expresión antes de asignarle ningún valor a esa variable!. **OJO antes de inicializar una variable que tiene?.**
- 13-Dividir valores enteros en forma incorrecta en expresiones grandes: Ej.: **3.425 + 2/3 +7.9** dará el mismo resultado que **3.425 + 7.9**
- 14-Mezclar tipos de datos en la misma expresión sin entender el efecto que produce.