



- Unidad 2

INTRODUCCION A LA ESPECIFICACION DE PROGRAMAS IMPERATIVOS

(Capitulo 3 bibliografía)

Asignación, Formateo y entrada interactiva.



» Ventre, Luis O.



Operaciones de Asignación

Instrucciones de asignación:

Sintaxis:

variable = expresión

Expresión puede ser una **constante** en su versión mas simple.

En estas instrucciones, el valor de la constante a la **derecha del signo** se asigna a la variable a la **izquierda del signo**.

El signo de igual indica a la computadora, que **primero determine el valor de la expresión a la derecha**, y luego la asigne a la variable que esta a la izquierda.

Si la variable no fue inicializada cuando se declaro, debe dársele un valor mediante una *instrucción de asignación u operación de entrada*, **antes de ser utilizada** en cualquier expresión para obtener un resultado coherente.



Operaciones de Asignación

La expresión a la derecha del signo = puede ser una variable u cualquier otra expresión válida de C++.

Por lo tanto esta expresión puede ser cualquier combinación de constantes, variables y llamadas a funciones que puedan evaluarse para producir un resultado.

Ejemplos:

```
suma = 3 + 7;  
dif = 15 - 6;  
producto = .05 * 14.6;  
conteo = Contador + 1;  
pesoTOTAL = factor * peso;
```

Siempre debe asignársele un valor válido a las variables a la derecha del signo = para que el resultado tenga sentido...Cuidado!!!

Operaciones de Asignación

Otra consideración es que ya que el valor de una expresión es almacenado en la variable a la izquierda del signo = **SOLO UNA VARIABLE** puede escribirse en dicha posición.

Ej. Invalido:

~~cantidad + 1892 = 10;~~



Ej. Valido:

cantidad = 10;



El signo = se llama **operador de asignación**; y en vista que el operador de asignación tiene **menor precedencia que cualquier otro operador matemático**, el valor de cualquier expresión a la derecha del mismo será evaluado anteriormente a la asignación.

Operaciones de Asignación

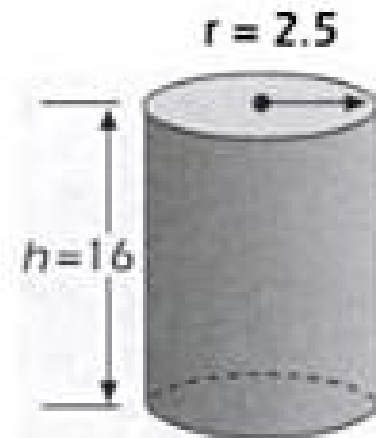
Instrucciones de asignación para calculo de volumen de un cilindro.

El volumen del cilindro esta dado por la formula:

$$\text{Volumen} = \pi * r^2 * h$$

Programa 3.1

```
// este programa calcula el volumen de un cilindro,  
// dados su radio y altura  
#include <iostream>  
using namespace std;  
  
int main()  
{  
    double radio, altura, volumen;  
  
    radio = 2.5;  
    altura = 16.0;  
    volumen = 3.1416 * radio * radio * altura;  
    cout << "El volumen del cilindro es " << volumen << endl;  
  
    return 0;  
}
```



Si no se han asignado valores a todas las vbles. se usan valores basura anteriores.
“No ve hacia delante”



Operaciones de Asignación

Coerción:

Según Wiki: Amenaza de utilizar violencia de cualquier tipo con el objetivo de condicionar el comportamiento de los individuos.

En la programación, existen conversiones de datos, a lo largo de las instrucciones de asignación. Estos tipos de conversiones son llamados coerción, debido a que *el valor a la derecha del operador de asignación es **FORZADO** al tipo de datos de la variable*

Ejemplos de esto son la asignación de un número entero a una variable de número real, y viceversa con el agregado de la parte fraccionaria del número debido al truncamiento.

```
int temp;  
float real;
```

```
...
```

```
temp= 25.89;
```

```
real = 2;
```

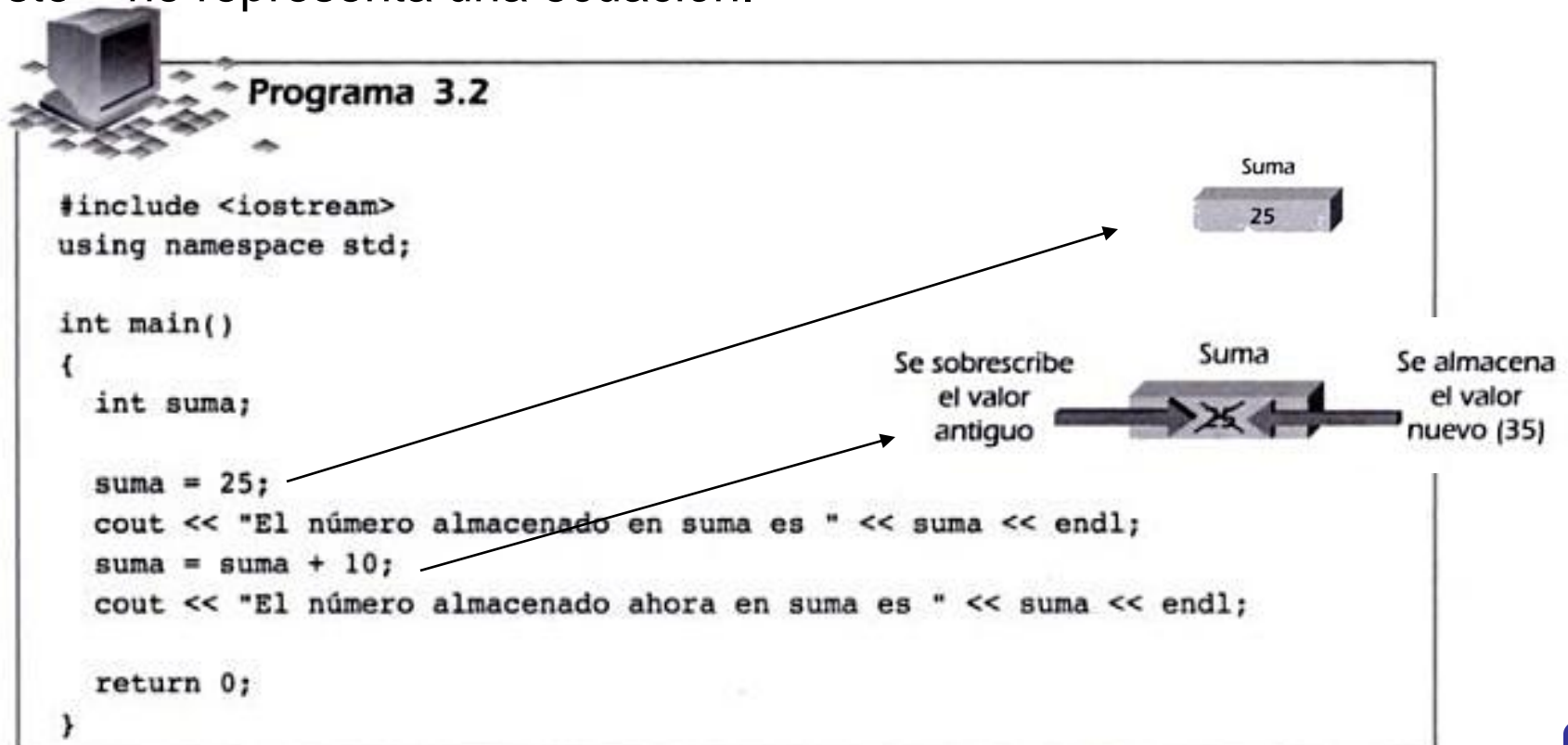
Que valores quedaran en temp?

Y en real?

Operaciones de Asignación

Variaciones de asignación:

En c++ esta permitido que la variable que esta a la izquierda del operador = este también en lado derecho de la expresión, debe entenderse que este = no representa una ecuación.





Operaciones de Asignación

Las expresiones que usan la misma variable de ambos lados pueden escribirse usando los siguientes **atajos de operadores de asignación**.

`+= -= *= /= %=`

De esta forma la expresión:

**Precio `*=` tasa
es equivalente a
Precio = Precio * tasa**

Ver siguiente ejemplo!...OJO....

```
6      int a,b,c;  
7  
8      a=5;  
9      b=2;  
10  
11     a*=b+6;  
12  
13     cout<<a;  
14     |
```





Operaciones de Asignación

Acumulación:

Las expresiones antes mencionadas, se utilizan cuando se requieren **acumular subtotales**. En el siguiente programa se observa la acumulación de resultados mediante instrucciones de acumulación:

variable = variable + valornuevo

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

Int main()
{
    int suma;
    suma =0;
    cout<<"El valor de suma se inicializo en "<<suma<<endl;
    suma = suma +96;
    cout<<" suma ahora es"<<suma<<endl;
    suma = suma +70;
    cout<<" suma ahora es"<<suma<<endl;
    ...
    return 0;
}
```

El valor de suma se inicializo en 0
suma ahora es 96
suma ahora es 166
...



Operaciones de Asignación

Conteo:

Una instrucción muy similar a la de acumulación es la **de conteo** y tiene la siguiente forma:

```
variable = variable + numeroFijo;
```

Ejemplos de esto seria:

```
cuenta = cuenta + 1;
```

```
i = i + 1;
```

```
n = n + 2;
```

Para los casos especiales en donde las variables se incrementan en 1, C++ brinda operadores unitarios.

Operador de incremento “++”



Operaciones de Asignación

En el caso de la expresión $\text{variable} = \text{variable} + 1$; puede ser reemplazada por la expresión

$\text{variable}++$ ó $++\text{variable}$

La diferencia entre ellos: si el operador de incremento $++$ aparece antes de la variable se llama **operador de prefijo para incremento** y cuando aparece después se llama **operador de postfijo para incremento**. Esta diferencia es importante SOLO cuando la variable que es incrementada se usa en una expresión de asignación.

Se debería utilizar		
..... prefijo	postfijo
$n = n + 1;$ $k = n;$	$k = ++n;$	$k = n;$ $n = n + 1;$
	←	→



Operaciones de Asignación

Además del operador de incremento, C++ proporciona un operador de decremento –

Como puede esperarse el mismo puede utilizarse de igual manera que el operador de incremento.

variable-- ó --variable

Su definición es operador de prefijo para decremento y operador de postfijo para decremento. Y su funcionamiento es análogo al operador antes visto.

Se debería utilizar			
	prefijo	postfijo	
..... n = n - 1; k = n;	$k = --n;$	$k = n--;$ k = n; n = n - 1;
	←	→	



Formato para salida

Además de desplegar resultados correctos es importante que un programa despliegue sus resultados en forma atractiva y clara.

El formato de los números desplegados por cout puede controlarse con los manipuladores de ancho de campo.

Manipuladores
mas usados
Pág 123.
Tabla 3.1

Manipulador	Acción
<code>setw(<i>n</i>)</code>	Establece el ancho de campo en <i>n</i> .
<code>setprecision(<i>n</i>)</code>	Establece la precisión del punto flotante en <i>n</i> lugares. Si se designa el manipulador <code>fixed</code> , <i>n</i> especifica el número total de dígitos desplegados después del punto decimal; de otra manera, <i>n</i> especifica el número total de dígitos significativos desplegados (números enteros más dígitos fraccionarios).
<code>setfill('x')</code>	Establece el carácter de relleno a la izquierda por omisión en <i>x</i> . (El carácter de relleno principal por omisión es un espacio, el cual es la salida para rellenar el frente de un campo de salida siempre que el ancho del campo es mayor que el valor que se está desplegando.)
<code>setiosflags(<i>flags</i>)</code>	Establece el formato de los indicadores (véase la tabla 3.3 para las configuraciones de los indicadores).
<code>scientific</code>	Establece la salida para desplegar números reales en notación científica.
<code>showbase</code>	Despliega la base usada para los números. Se despliega un 0 a la izquierda para los números octales y un 0x a la izquierda para números hexadecimales.
<code>showpoint</code>	Siempre despliega seis dígitos en total (combinación de partes enteras y fraccionarias). Rellena con ceros a la derecha si es necesario. Para valores enteros mayores, revierte a notación científica.
<code>showpos</code>	Despliega todos los números positivos con un signo de + a la izquierda.



Formato para salida

Ejemplo: el manipulador de ancho de campo **setw()** se usa para establecer el ancho de campo desplegado.

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

Int main()
{
cout <<setw(3) << 6 <<endl;
    <<setw(3) << 18 <<endl;
    <<setw(3) << 124 << endl;
    << “---\n”
    <<(6+18+124) << endl;

return 0;
}
```

Los enteros se justifican
A la derecha del ancho de campo



La salida de este programa es:

```
    6
   18
  124
   ---
 148
```



Formato para salida

El manipulador de ancho de campo `setw()` se **debe incluir en cada ocurrencia** de un numero insertado en el flujo de datos enviado a `cout`. O sea el mismo solo se aplica a la siguiente inserción de datos inmediata.

Cuando usamos manipuladores que requieren un argumento se debe incluir el archivo de encabezado **IOMANIP** como **parte de programa**.

```
#include <iomanip>
```

*A su vez, dar formato completo a números de punto flotante requiere el uso de **tres manipuladores de ancho de campo**. El primer manipulador establece el ancho del campo, el segundo fuerza el despliegue de un punto decimal y el tercero determina cuantos dígitos significativos se desplegaran a la derecha del punto.!!*



Formato para salida

Ejemplo.

```
cout<<"|"<<setw(10)<<fixed<<setprecision(3)<<25.67<<"|";
```

El manipulador setw(10) indica a cout que despliegue el numero en un campo total de 10.

El manipulador fixed fuerza de manera explicita el despliegue de un punto decimal y designa que el manipulador setprecision se usa para designar el numero de dígitos a desplegar luego del punto decimal. Sin la designacion de fixed, setprecision determina el numero de dígitos total desplegado, el cual incluye partes enteras y fraccionarias.

```
| 25.670|
```

Recomendación: VER EJEMPLOS TABLA 3.2 pagina 126 Bibliografia



Formato para salida

Existe un manipulador que utiliza señales como banderas para indicar el estado de una condición específica. Manipulador `setiosflag()` donde `ios` se deriva de las iniciales de `input output stream` (flujo de entrada salida). Como se ha visto, los numero enviado a `cout`, tienen justificación derecha, para modificar la justificación se usa `setiosflags(ios::left)`.

Indicador	Significado
<code>ios::fixed</code>	Siempre muestra el punto decimal con seis dígitos después del punto decimal. Rellena con ceros a la derecha si es necesario. Este indicador tiene precedencia si se establece con el indicador <code>ios::showpoint</code> .
<code>ios::scientific</code>	Usa despliegue exponencial en la salida.
<code>ios::showpoint</code>	Siempre despliega un punto decimal y seis dígitos significativos en total (combinación de partes enteras y fraccionarias). Rellena con ceros a la derecha después del punto decimal si es necesario. Para valores enteros más grandes, revierte a notación científica a menos que esté establecido el indicador <code>ios::fixed</code> .
<code>ios::showpos</code>	Despliega un signo + a la izquierda cuando el número es positivo.
<code>ios::left</code>	Justifica a la izquierda la salida.
<code>ios::right</code>	Justifica a la derecha la salida.



Formato para salida

Debido a que el manipulador antes visto requiere de argumentos o parámetros para su determinación, se puede llamar ***manipulador parametrizado***.

Además de este puede utilizarse el manipulador showbase seguido del carácter de enviar a << y la base numérica deseada para realizar conversiones, de números decimales a números octales o hexadecimales. Ver prog. 3.7 pág 130.

El compilador considera el ingreso de un numero en base octal si se le antecede un 0, y considera un numero ingresado en base hexadecimal si se le antecede 0x; de esta forma el compilador por defecto lo pasa a decimal.

Formato para salida

Ejemplo del caso:



Programa 3.8

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

int main()
{
    cout << "El valor decimal de 025 es " << 025 << endl;
        << "El valor decimal de 0x37 es " << 0x37 << endl;

    return 0;
}
```

La salida producida será:

```
El valor decimal de 025 es 21
El valor decimal de 0x37 es 55
```

Recomendación ver figura 3.8 página 132 Bibliografía.



Biblioteca de funciones Matemáticas

Ya hemos visto los operadores aritméticos de C++, pero no existen operadores para realizar una potencia de un número, o una raíz cuadrada o encontrar valores trigonométricos.

Para facilitar estos cálculos **C++ proporciona funciones preprogramadas** que pueden incluirse. Antes de usar dicha función debemos saber:

- Nombre de la función matemática deseada
- Que hace?
- Tipo de datos requerido por la función (parámetros)
- El tipo de dato entregado por la misma (resultado)
- Como incluir la biblioteca.

La biblioteca que contiene estas funciones matemáticas se llama **cmath**!!

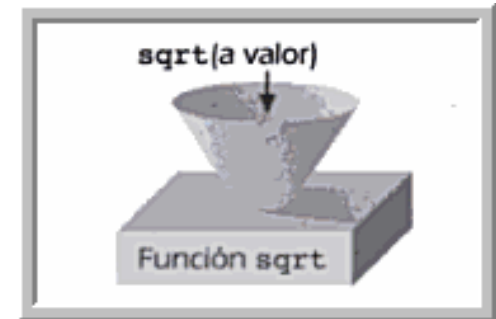
Biblioteca de funciones Matemáticas

Ej.: la función `sqrt(numero)`.

Su nombre es “sqrt”; esta función calcula la raíz cuadrada del numero pasado a la misma. los paréntesis proporcionan un embudo para enviar esos datos a la función. **Estos son sus argumentos, y constituyen los datos de entrada de la función.**

El argumento para la función sqrt debe ser un valor real.

Sobrecarga de funciones, hay 3 sqrt reales.



El resultado devuelto es un numero de precisión doble.

La biblioteca debe ser incluida:

```
#include <cmath>
```



Biblioteca de funciones Matemáticas

Algunas funciones mas de la biblioteca cmath son:

abs(a)	valor absoluto de a
pow(a1,a2)	a1 elevado a la potencia a2
sqrt(a)	raiz cuadrada de a
sin(a)	seno de a (Toda FUNCION trigonométrica argumento en radianes ojo!!!)
...etc	

Los argumentos de las funciones también pueden ser expresiones.

Ej: `sqrt (4.0 + 5.3 * 4.0)` `abs (omega – phi) ...`

$$= 4 * \sqrt{4.5 * 10.0 - 9.0} - 2.0$$

$$= 4 * \sqrt{36} - 2.0$$

$$= 4 * 6 - 2.0$$

$$= 24 - 2.0 \longrightarrow = 22.0$$



Biblioteca de funciones Matemáticas

Ejemplo: El programa 3.9 determina el tiempo que tarda una pelota en golpear el suelo después de caer desde 800 pies. La formula matemática:

$$\text{tiempo} = \sqrt{2 * \text{distancia} / g}$$



Programa 3.9

```
#include <iostream> // esta línea puede colocarse en segundo lugar en vez
                    // de en primero
#include <cmath>     // esta línea puede colocarse en primer lugar en vez
                    // de en segundo

using namespace std;

int main()
{
    int altura;
    double tiempo;

    altura = 800;
    tiempo = sqrt(2 * altura / 32.2);
    cout << "Tardará " << tiempo << " segundos en caer "
         << altura << " pies.\n";

    return 0;
}
```

Tardará 7.04907 segundos en caer 800 pies.



Biblioteca de funciones Matemáticas

Moldes:

En c++, además de las conversiones de datos implícitas, se proporciona conversiones explícitas definidas por el usuario. El operador usado para forzar la conversión es el **operador de molde o cast**.

Existen operadores de molde en tiempo de compilación y en tiempo de ejecución.

En tiempo de compilación es un operador unitario con sintaxis:

```
tipoDeDatos(expresion) -----> int( a * b )
```

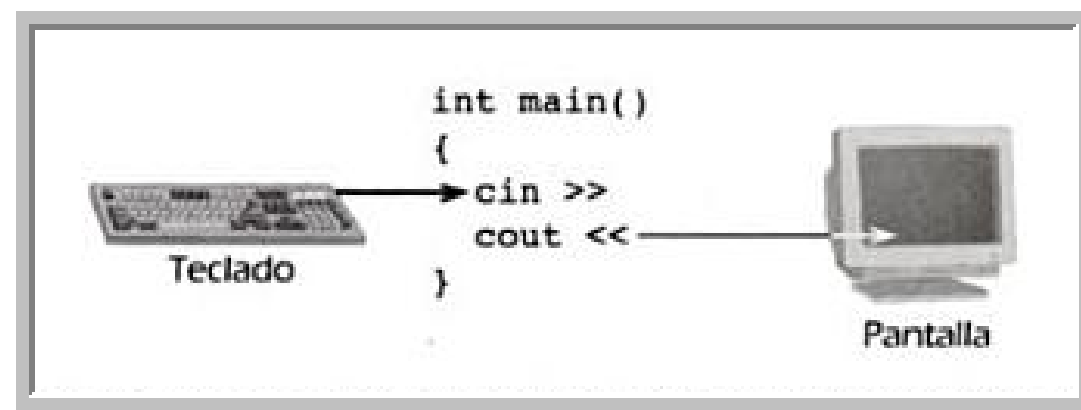
El ultimo standard mas reciente de c++, incorpora la conversión en tiempo de ejecución, la cual es **verificada** y se aplica en tiempo de ejecución si produce un valor valido.

```
staticCast<tipo-de-datos> (expresión) ej. staticCast<int>(a * b)
```


Ingreso de datos al programa **CIN**

Los datos en programas que solo se **van a ejecutar una** vez puede incluirse en forma directa y **fija** en el programa.

Pero cuando el objetivo es realizar un programa que cumpla con diferentes propósitos y pueda modificarse sin tener que ser reescrito. Así como c++ proporciona el objeto cout para la salida de datos en pantalla, también proporciona el **objeto CIN; que permite al usuario introducir un valor en la terminal**. El valor ingresado se almacena de manera directa en una variable.





Ingreso de datos al programa CIN

Cuando se encuentra una instrucción `cin >> num1;` la computadora **detiene la ejecución del programa y acepta datos del teclado.**

La pausa no finaliza hasta que el usuario no presiona la tecla return, luego de ingresar los datos.

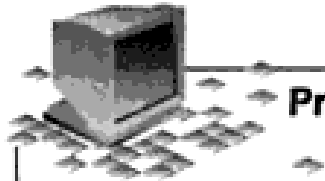
El objeto `cin` almacena el elemento ingresado por el usuario en la variable mostrada después del ***operador de extracción >> (“obtener de”)***.

El programa continua luego de ingreso de datos.

El objeto `cin`, puede utilizarse de igual manera que el objeto `cout`, permitiendo **el ingreso de mas de un dato en el mismo comando.** Por ejemplo: `cin >> num1 >> num2;` produce dos valores que se leen de la terminal y se asignan a las variables `num1` y `num2`.

Ojo entre variable y variable ingresadas debe existir un **espacio o return**

Ingreso de datos al programa CIN



Programa 3.12

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

int main()
{
    double num1, num2, producto;

    cout << "Por favor introduzca un número: ";
    cin >> num1;
    cout << "Por favor introduzca otro número: ";
    cin >> num2;
    producto = num1 * num2;
    cout << num1 << " por " << num2 << " es " << producto << endl;

    return 0;
}
```

indicador de comandos

pausa ejecución hasta
ingreso datos y tecla
return


Se almacena en esta vble

```
Por favor introduzca un número: 30
Por favor introduzca otro número: 0.05
30 por 0.05 es 1.5
```



Ingreso de datos al programa CIN

Se observa un ejemplo con el ingreso de 3 valores en una sola instrucción.



Programa 3.13

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

int main()
{
    int num1, num2, num3;
    double promedio;

    cout << "Introduzca tres números enteros: ";
    cin >> num1 >> num2 >> num3;
    promedio = (num1 + num2 + num3) / 3.0;
    cout << "El promedio de los números es " << promedio << endl;

    return 0;
}
```

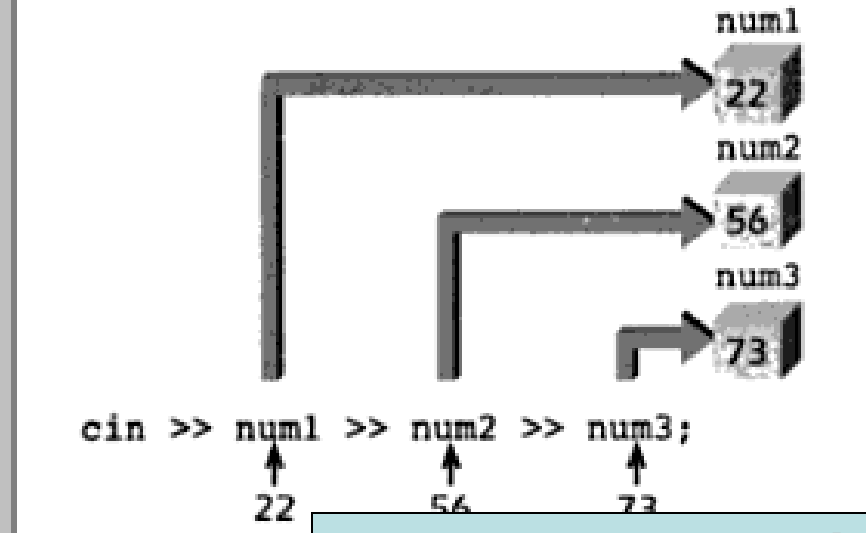


Diagrama de flujo que muestra la ejecución de la instrucción `cin >> num1 >> num2 >> num3;` con los valores 22, 56 y 73 ingresados. Las flechas indican el flujo de datos desde los valores ingresados hacia las variables `num1`, `num2` y `num3`.

**Cuidado: Que pasaría si se quitaran los paréntesis?.....
....Y si dividido en 3?**

Introduzca tres números enteros: 22 56 73
El promedio de los números es 50.3333



Ingreso de datos al programa CIN

La operación de cin, se **encarga de forzar los valores ingresados a los tipos de datos de las variables asociadas**. Por lo tanto es importante tener cuidado en esto.

Ej.:

Suponga que se introducen en respuesta a **cin>>num1>>num2>>num3;** donde num1 y num3 son variables de precisión doble y num2 es una variable entera.

56	22.879	33.923
----	--------	--------

El valor 56, será convertido en 56.0, luego se esperará el ingreso de un entero, así que se tomará 22.879 como 22 para num2, y luego para num3 se buscará un real encontrando .879. La última entrada es extra y es ignorada.



Constantes simbólicas

Existen constantes usadas dentro de un programa que tienen un significado reconocido fuera del contexto del programa. Por ejemplo el número 3,1416, el cual es π ; o 32,2 pies/sec² que es la constante gravitacional.

Estos números son llamados números mágicos, y cuando aparecen muchas veces en un programa se vuelven una **fente potencial de error**.

Para no tener números mágicos diseminados por todo el programa, c++ permite usar un nombre simbólico en lugar de su valor. Y si alguna vez se necesita cambiar su valor solo debe cambiarse una vez.

Equiparar números con nombres simbólicos se logra con el **calificador CONST**. Esto indica que el identificador que lo continua solo puede ser leído después de inicializarse.



Constantes simbólicas

Ej.:

```
const double PI = 3.1516;  
const double DENSIDAD = 0.238;  
const int MAXNUM = 100;
```

Una vez que se crea un identificador const, y se inicializa su valor **no puede cambiarse**.

Una vez declarado puede usarse en lugar de los números que representa

```
circun = 2 * PI * radio;  
peso = DENSIDAD * volumen;
```

Debido a que una declaración const equipara un valor constante con un identificador, estos identificadores se conocen como **constantes simbólicas o constantes nombradas**.



```
// Este programa determina el peso de un cilindro de acero
// al multiplicar el volumen del cilindro por su densidad
// El volumen del cilindro está dado por la fórmula  $PI * pow(radio,2) * altura$ .
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;
```

Constantes simbólicas

```
int main()
{
    const double PI = 3.1416;
    const double DENSIDAD = 0.284;
    double radio, altura, peso;

    cout << "Introduzca el radio del cilindro (en pulgadas): ";
    cin >> radio;
    cout << "Introduzca la altura del cilindro (en pulgadas): ";
    cin >> altura;
    weight = DENSIDAD * PI * pow(radio,2) * altura;
    cout << setiosflags(ios::fixed)
        << setiosflags(ios::showpoint)
        << setprecision(4)
        << "El cilindro pesa " << peso;

    return 0;
}
```

Uso de una función matemática
potencia

```
Introduzca el radio del cilindro (en pulgadas): 3
Introduzca la altura del cilindro (en pulgadas): 12
El cilindro pesa 96.3592 libras
```




Constantes simbólicas

Al utilizar estas dos constantes simbólicas, nos permite; con PI, utilizar una etiqueta mas conocida por la mayoría de las personas. Y con DENSIDAD, la posibilidad de cambiar de material alterando solo este valor. **Y así poder utilizar este mismo programa para otros cálculos.**

Por supuesto que si deben hacerse muchas variaciones de la DENSIDAD; es mejor utilizar una variable en su lugar, y aquí podemos marcar una diferencia entre constante simbólica y variable.

Una variable puede cambiar su valor en cualquier parte del programa, mientras que el objetivo de una constante nombrada es **no modificar su valor una vez definido.**

Una vez definida densidad una instrucción como:

DENSIDAD=0.156; carece de sentido y producira un mensaje de error ya que equivaldría a colocar $0.284=0.156!$?



Constantes simbólicas

Pueden usarse las constantes simbólicas para representar expresiones constantes, y pueden utilizarse las constantes simbólicas ya definidas para definir una nueva; por ejemplo:

```
const double PI=3.1416;  
const double GRAD_A_RAD= PI/180.0;
```

Posteriormente puedo utilizarlo en una expresión matemática, recordando que las funciones trigonométricas usan de argumento ángulos en radianes.

```
altura = distancia * sin(angulo * GRAD_A_RAD)
```



Errores Comunes

Como se menciono en el capitulo anterior, de acuerdo al material de este capitulo es importante conocer los siguientes posibles errores comunes:

- ***Olvidar asignar o inicializar valores para todas las variables antes que estas se usen en una expresión.***
- ***Utilizar una función matemática sin incluir la declaración de pre-procesador `#include<cmath>`***
- ***Utilizar una funcion de biblioteca matemática sin pasar el numero correcto de argumentos, ej. `Pow`***
- ***Utilizar los operadores unitarios de incremento o decremento a expresiones ej. `(count + n)++`; esto es incorrecto solo puede aplicarse a variables individuales***
- ***Olvidar separar las variables a introducir en cin con `>>`***