FUNDAMENTOS DE PROGRAMACIÓN

Inicio Clase 02

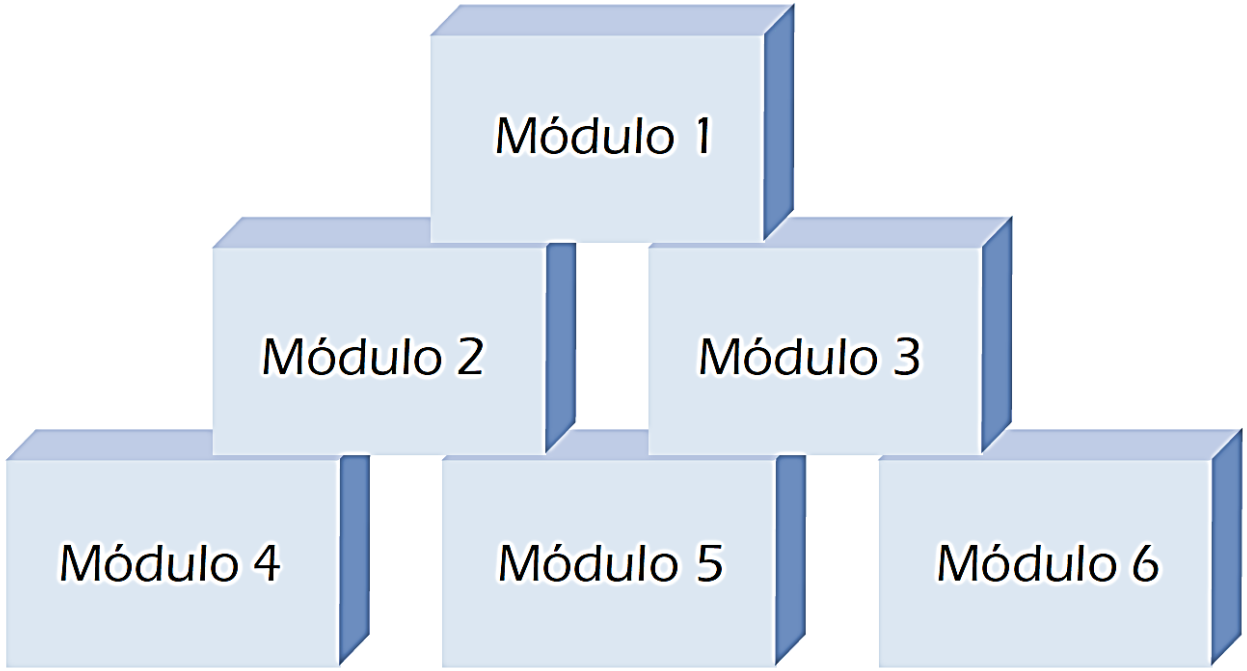
Profesor: Carlos Díaz

Clase 02: Introducción al C++

- Introducción a C++
- La función main()
- El objeto cout
- Tipos de datos enteros
- Tipos de datos de punto flotante
- Notación exponencial
- Operadores aritméticos
- Prioridad de operadores aritméticos
- Variables

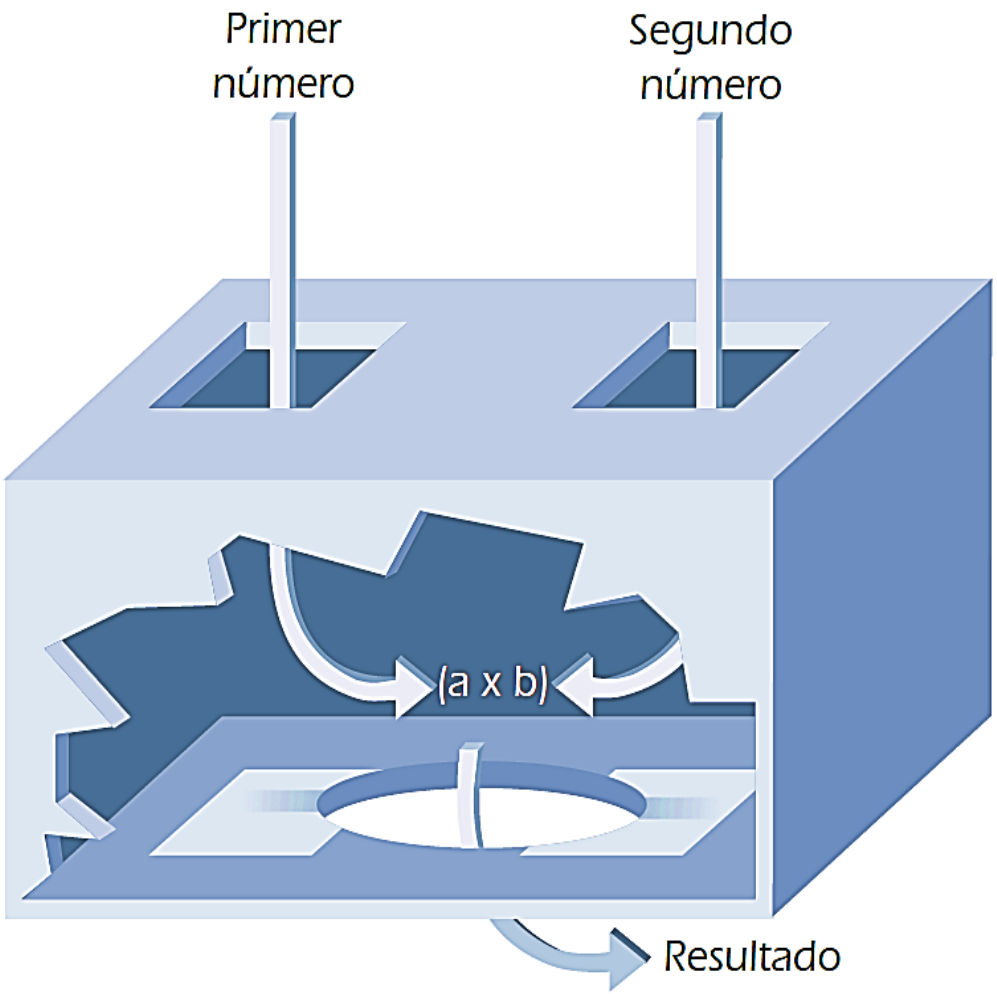
Introducción a C++

- Un programa C++ se construye combinado tantos **módulos** como sea necesario para resolver el problema.
- Cada módulo puede programarse por separado y luego integrarse conforme se completan.
- En C++ los módulos pueden ser clases o funciones.



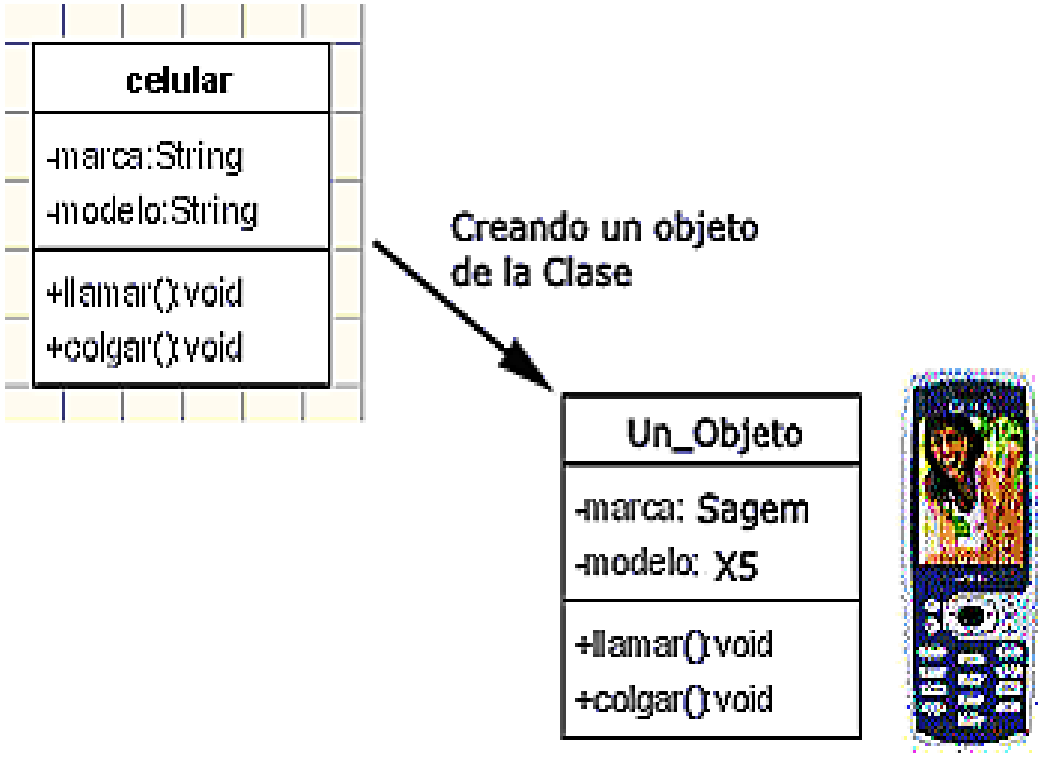
Introducción a C++

- Una **función** puede considerarse como una pequeña máquina que transforma los datos que recibe en un producto terminado.
- En el ejemplo la función acepta dos números como entrada y los multiplica para producir una salida.
- Una función encapsula un conjunto de operaciones.



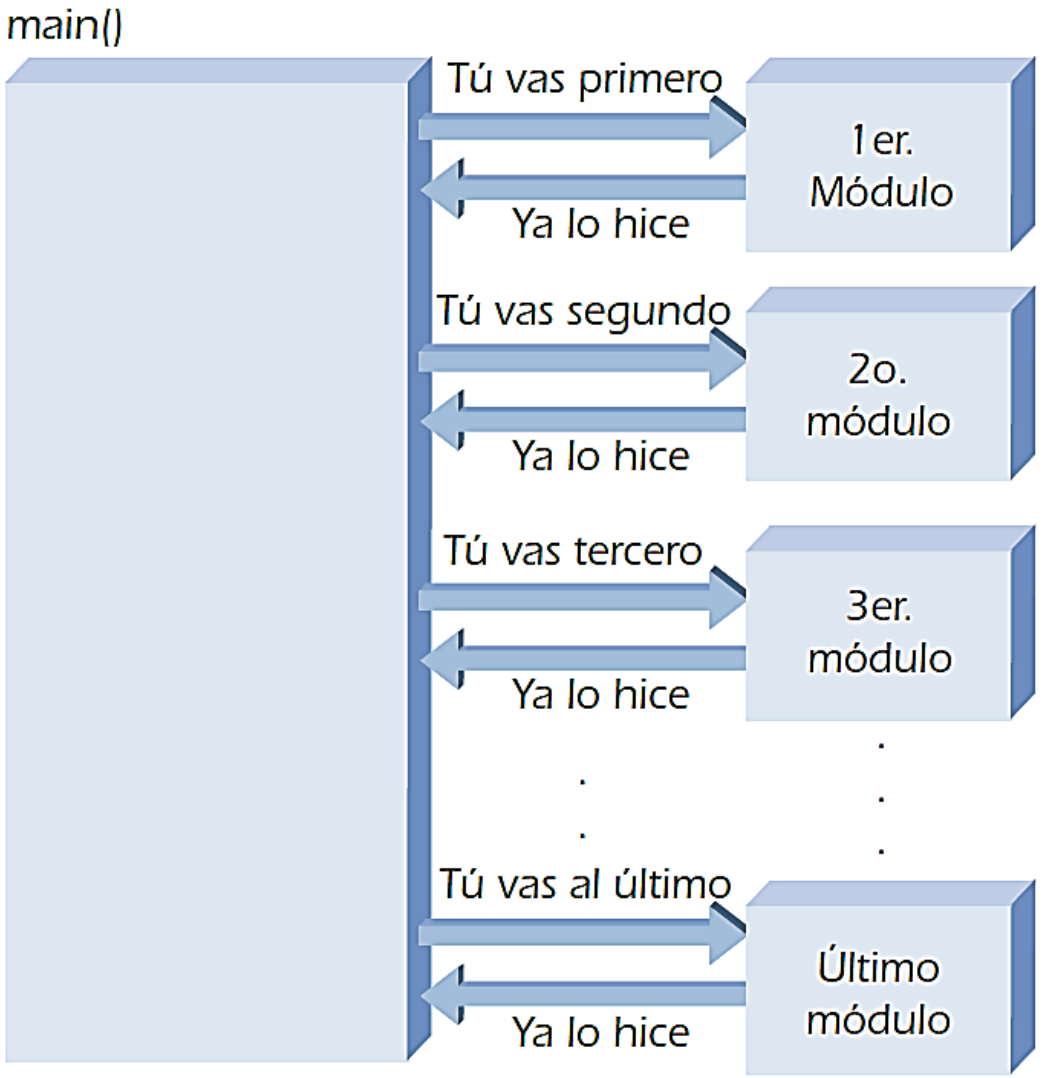
Introducción a C++

- Una **clase** es una unidad más complicada que una función, debido a que contiene datos como funciones apropiadas para manipular los datos.
- Una clase encapsula tanto datos como uno o más conjuntos de operaciones.
- Puede considerarse como una pequeña fábrica que puede crear objetos de esa clase.



La función main()

- Para facilitar la colocación y ejecución ordenada de los módulos, C++ debe tener una y sólo una función **main()**.
- La función main() se conoce como función controladora, porque indica a los otros módulos la secuencia en que deben ejecutarse.



El objeto cout

- Su nombre deriva de Console OUTput, es un objeto de salida que envía datos al dispositivo estándar de salida.

```
#include <iostream>
using namespace std;
int main()
{
	cout<<"Hola mundo!";
	return 0;
}
```

Explicación

- **#include <iostream>**: causa que el archivo iostream sea insertado donde aparece el comando #include.
- El iostream es una parte de la biblioteca estándar que contiene, entre otro código, dos clases llamadas istream y ostream. Que permiten la entrada (cin) y salida (cout) de datos respectivamente.
- **using namespace std**: Le dice al compilador dónde encontrar los archivos en ausencia de cualquier designación explicita adicional.
- Como el archivo iostream esta contenido dentro del espacio de nombres std, el compilador usara automáticamente los objetos cin y cout.
- El uso de espacios de nombres permite crear clases y objetos propios con los mismos nombres provistos por la librería estándar.

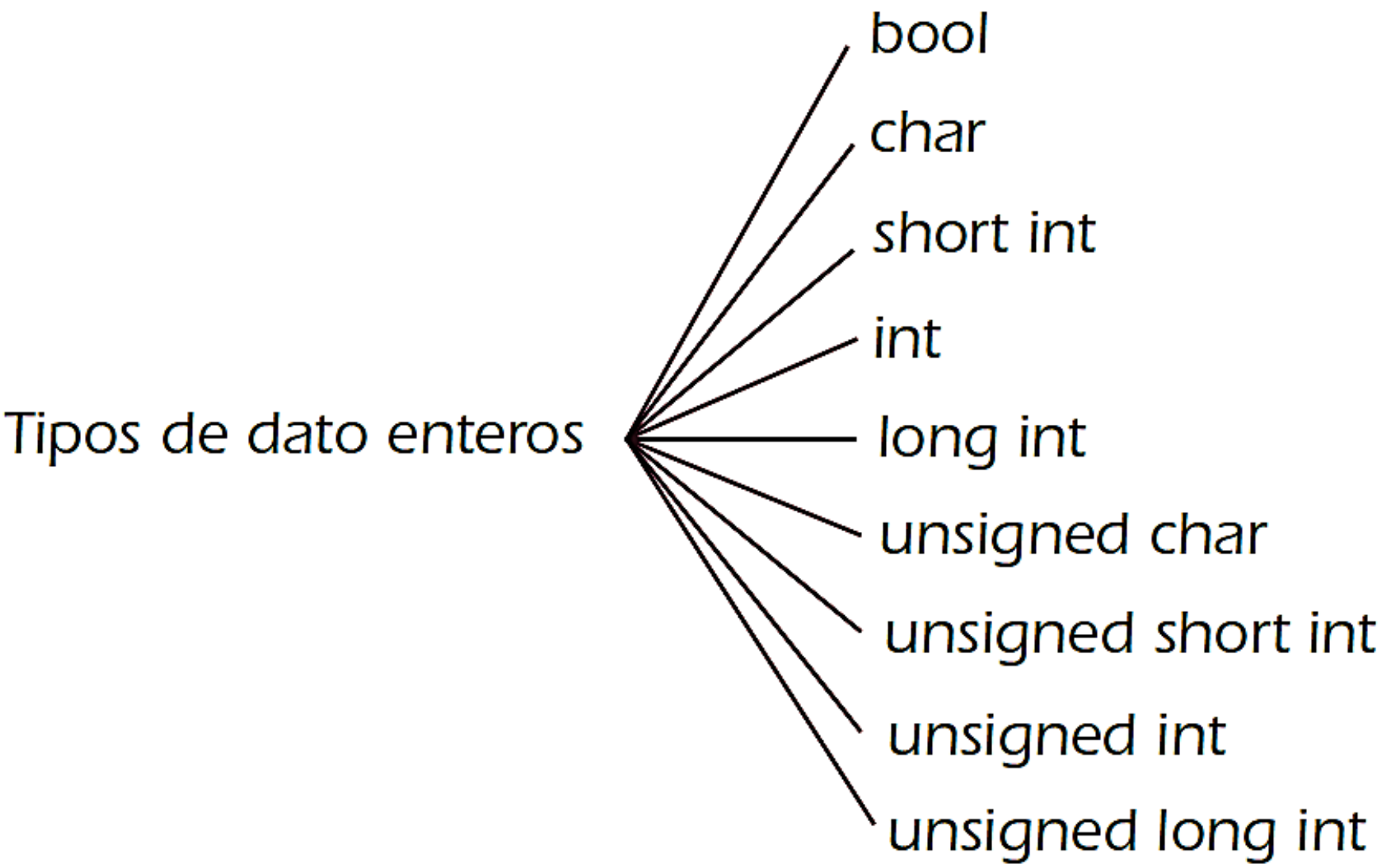
Ejemplo con namespace

```
#include <iostream>
using namespace std;
namespace miEspacio
{
   int miValor;
}
int main()
{
   int miValor = 3;
   miEspacio::miValor = 4;
   cout << miValor << endl; // imprime '3'
   cout << miEspacio::miValor << endl; // imprime '4'
   return 0;
}
```

Otro ejemplo con namespace

```
#include <iostream>
using namespace std;
namespace miEspacio
{
    int cout;
}
int main()
{
   miEspacio::cout = 5;
   cout << miEspacio::cout << endl; // imprime '5'
   return 0;
}
```

Tipos de datos enteros



Observaciones

- El tipo de datos **bool** se utiliza para representar datos booleanos (lógicos). Por ello esta restringido a solo dos tipos de valores: verdadero (true) o falso (false).
- El tipo de dato **char** se almacena usando códigos ASCII (**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange).

```
#include <iostream>
using namespace std;
int main()
{
	for (int i=0;i<=255;i++)
		cout<<"\t"<<i<<"\t"<<(char) i<<endl;
	return 0;
}
```

Rango de almacenamiento de datos entero

Nombre del tipo de datos	Tamaño del **almacenamiento** (en bytes)	Rango de valores
`char`	1	256 caracteres
`bool`	1	verdadero (lo cual es considerado como cualquier valor positivo) y falso (lo cual es un cero)
`short int`	2	–32,768 a +32,767
`unsigned short int`	2	0 a 65,535
`int`	4	–2,147,483,648 a +2,147,483,647
`unsigned int`	4	0 a 4,294,967,295
`long int`	4	–2,147,483,648 a +2,147,483,647
`unsigned long int`	4	0 a 4,294,967,295

Tamaño de almacenamiento de datos entero

```
#include <iostream>
using namespace std;
int main()
{
	cout<<"Tama\xa4o de bool es de:\t\t\t"<<sizeof(bool)<<" bytes.\n";
	cout<<"Tama\xa4o de char es de:\t\t\t"<<sizeof(char)<<" bytes.\n";
	cout<<"Tama\xa4o de short int es de:\t\t"<<sizeof(short int)<<" bytes.\n";
	cout<<"Tama\xa4o de int es de:\t\t\t"<<sizeof(int)<<" bytes.\n";
	cout<<"Tama\xa4o de long int es de:\t\t"<<sizeof(long int)<<" bytes.\n";
	cout<<"Tama\xa4o de unsigned char es de:\t\t"<<sizeof(unsigned char)<<" bytes.\n";
	cout<<"Tama\xa4o de unsigned short int es de:\t"<<sizeof(unsigned short int)<<" bytes.\n";
	cout<<"Tama\xa4o de unsigned int es de:\t\t"<<sizeof(unsigned int)<<" bytes.\n";
	cout<<"Tama\xa4o de unsigned long int es de:\t"<<sizeof(unsigned long int)<<" bytes.\n";
	return 0;
}
```

Observaciones

- Los tipos de datos sin signo (**unsigned**) solo permite utilizar valores no negativos, es decir, cero y positivos.
- Al no utilizar un bit como signo los tipos unsigned proporcionan el doble de rango que su contraparte con signo.
- Ejemplo: Un short int tiene 2 bytes, o sea 16 bits, pero emplea un bit para el signo, así que su rango esta comprendido desde:

  -2^{15} hasta $2^{15}-1$. Es decir -32768 hasta 32767.

- Ejemplo: Un unsigned short int tiene 2 bytes, o sea 16 bits, pero como no tiene signo, su rango esta comprendido desde:

  0 hasta $2^{16}-1$. Es decir 0 hasta 65535.

Ejemplo

```
#include <iostream>
using namespace std;
int main()
{
        short int x=32767;
        short int y=-32768;
        cout<<"El valor de x es: "<<x<<endl;
        x=x+1;
        cout<<"El valor de x+1 es: "<<x<<endl; //imprime -32768
        cout<<"El valor de y es: "<<y<<endl;
        y=y-1;
        cout<<"El valor de y-1 es: "<<y<<endl; //imprime 32767
        return 0;
}
```

Otro ejemplo

```
#include <iostream>
using namespace std;
int main()
{
        unsigned short int x=65535;
        unsigned short int y=0;
        cout<<"El valor de x es: "<<x<<endl;
        x=x+1;
        cout<<"El valor de x+1 es: "<<x<<endl; //imprime 0
        cout<<"El valor de y es: "<<y<<endl;
        y=y-1;
        cout<<"El valor de y-1 es: "<<y<<endl; //imprime 65535
        return 0;
}
```

Tipos de datos de punto flotante

- Un número de **punto flotante**, al cual se llama **número real**, puede ser cualquier número cero, positivo o negativo que contenga un punto decimal.

Tipo	Almacenamiento	Rango absoluto de valores (+ y –)
`float`	`4 bytes`	`1.40129846432481707e-45` a `3.40282346638528860e+38`
`double` y `long double`	`8 bytes`	`4.94065645841246544e-324` a `1.79769313486231570e+308`

Tamaño de almacenamiento de datos de punto flotante

```
#include <iostream>
using namespace std;
int main()
{
	cout<<"Tama\xa4o de float es de:\t\t\t"<<sizeof(float)<<" bytes.\n";
	cout<<"Tama\xa4o de double es de:\t\t\t"<<sizeof(double)<<" bytes.\n";
	cout<<"Tama\xa4o de long double es de:\t\t"<<sizeof(long double)<<" bytes.\n";
	return 0;
}
```

Notación exponencial

Notación decimal	Notación exponencial	Notación científica
1625.	1.625e3	1.625×10^3
63421.	6.3421e4	6.3421×10^4
.00731	7.31e-3	7.31 x 10-3
.000625	6.25e-4	6.25 x 10-4

```
#include <iostream>
using namespace std;
int main()
{
        float num1, num2;
        num1=-223e-3;
        num2=3e2;
        cout<<num1<<" + "<<num2<<" = ";
        cout<<num1+num2<<endl;
        return 0;
}
```

Operadores aritméticos

- Los números enteros y reales pueden sumarse, restarse, multiplicarse y dividirse.

Operación	Operador
Adición	+
Sustracción	–
Multiplicación	*
División	/
División de módulo	%

- En general no conviene mezclar números enteros y reales, pues pueden obtenerse resultados impredecibles. Por ejemplo 'A'+1 produce 'B' .
- Si ambos operando son enteros, el resultado es entero.
- Si un operando es real, el resultado es real.

Ejemplo

```
#include <iostream>
using namespace std;
int main()
{
	char letra='A';//codigo ASCII 65
	int numero=1;
	letra=letra+1;
	numero=numero+'A';
	cout<<"La letra es: "<<letra<<endl; //imprime B
	cout<<"El n\xA3mero es: "<<numero<<endl; //imprime 66
	return 0;
}
```

Otro ejemplo

```
#include <iostream>
using namespace std;
int main()
{
	int dividendo=27;
	int divisor=4;
	cout<<"El cociente es: "<<dividendo/divisor<<endl; //imprime 6
	cout<<"El residuo es: "<<dividendo%divisor<<endl; //imprime 3
	cout<<"El resultado con decimales es:\n";
	cout<<(float)dividendo/divisor<<endl; //imprime 6.75
	return 0;
}
```

Prioridad de operadores aritméticos

Operador
Negación –
* / %
+ –

8 + 5 * 7 % 2 * 4 =

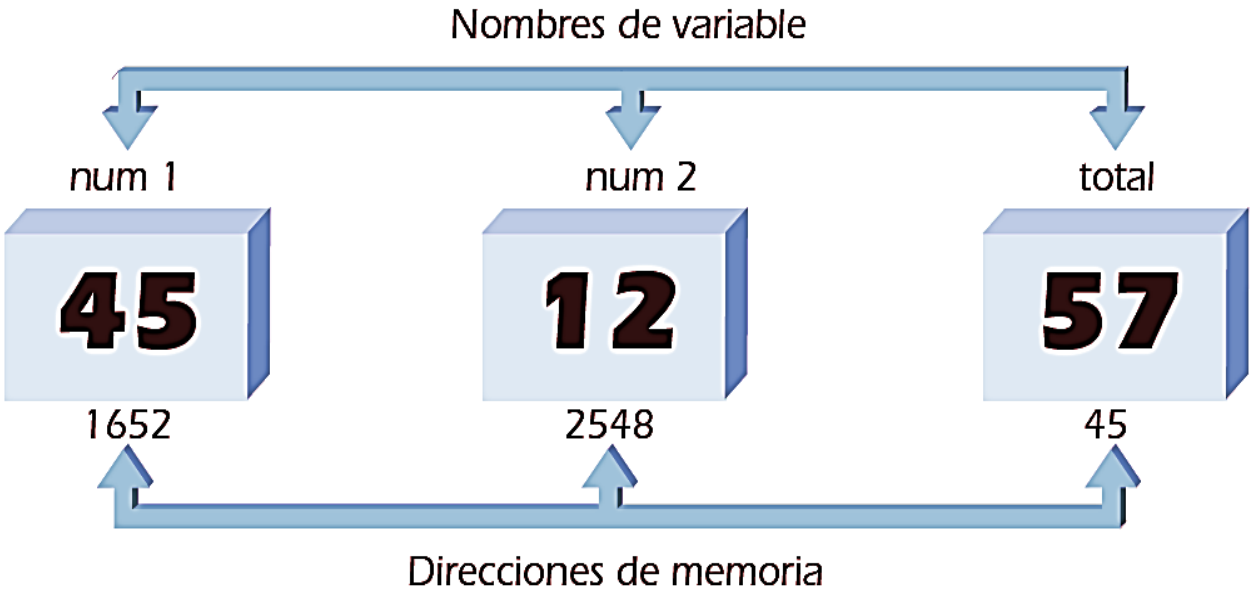
8 + 35 % 2 * 4 =

8 + 1 * 4 =

8 + 4 = 12

Variables

- Cada valor entero o real se almacena en la memoria de la computadora y se recuperan de ella.
- Una variable es tan solo un nombre dado por el programador para referirse a ubicaciones de almacenamiento de la computadora.
- Se usa el termino variable porque el valor almacenado por la variable puede cambiar.



Dirección de una variable

```
#include <iostream>
using namespace std;
int main()
{
	int num;
	num=22;
	cout<<"El valor almacenado en num es "<<num<<endl;
	cout<<"La direcci\xA2n de num = "<<&num<<endl;
	return 0;
}
```

Ejercicio 1

El conjunto de ecuaciones lineales

$$a_{11}X_1 + a_{12}X_2 = c_1$$

$$a_{21}X_1 + a_{22}X_2 = c_2$$

puede resolverse usando la regla de Cramer:

$$X_1 = \frac{c_1 a_{22} - c_2 a_{12}}{a_{11} a_{22} - a_{12} a_{21}}$$

$$X_2 = \frac{c_2 a_{11} - c_1 a_{21}}{a_{11} a_{22} - a_{12} a_{21}}$$

Usando estas ecuaciones, escriba, compile y ejecute un programa en C++ para encontrar los valores X_1 y X_2 que satisfagan las siguientes ecuaciones:

$$3X_1 + 4X_2 = 40$$

$$5X_1 + 2X_2 = 34$$

Ejercicio 2

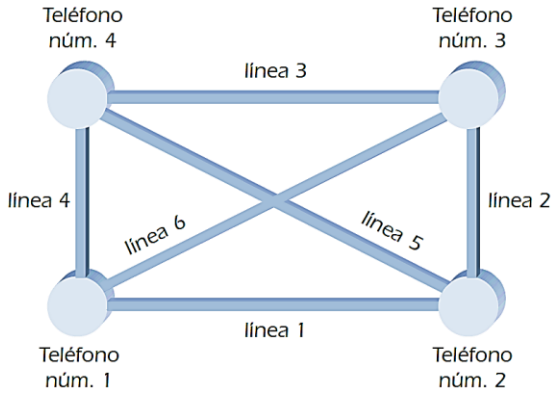
Una red telefónica conectada en forma directa es aquella en la que todos los teléfonos en la red están conectados en forma directa y no requieren una estación de conmutación central para establecer llamadas entre dos de ellos. Por ejemplo, las instituciones financieras en Wall Street usan una red así para mantener líneas telefónicas abiertas en forma directa y continua entre las empresas.

El número de líneas directas necesarias para mantener una red conectada en forma directa para n teléfonos está dado por la fórmula:

$$líneas = n(n - 1)/2$$

Por ejemplo, conectar en forma directa cuatro teléfonos requiere 6 líneas individuales (véase la figura).

Agregar un quinto teléfono a la red ilustrada en la figura requeriría 4 líneas adicionales para un total de 10 líneas.



Usando la fórmula dada, escriba un programa en C++ que determine el número de líneas directas requeridas para 100 teléfonos, y las líneas adicionales requeridas si se fueran a agregar 10 teléfonos nuevos a la red.

FUNDAMENTOS DE PROGRAMACIÓN

Fin Clase 02

Profesor: Carlos Díaz