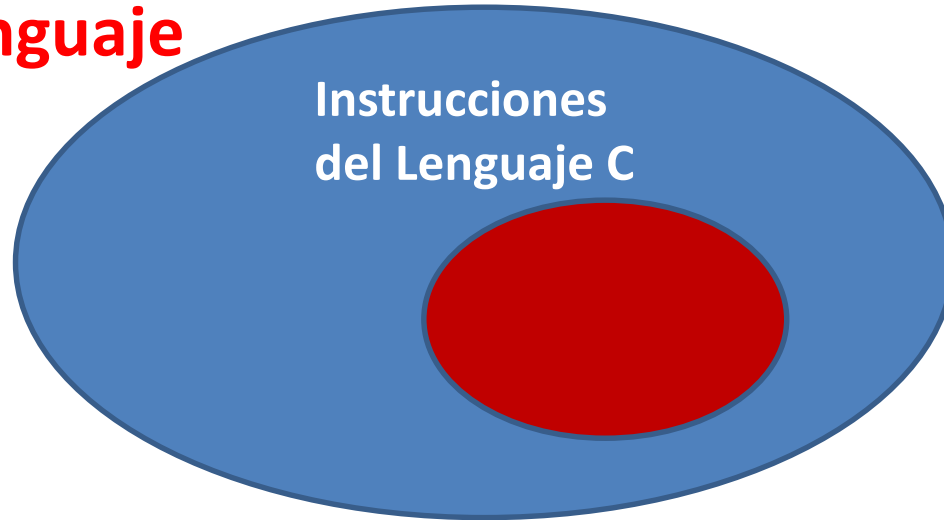


# DIFERENCIAS ENTRE C Y C++

El lenguaje C++ es un lenguaje de programación diferente del lenguaje C. Sin embargo, al igual que los lenguajes Java y C#, el lenguaje C++ ha tomado el núcleo de C para su implementación.

Esto quiere decir que, en términos referidos a la teoría de conjuntos, C es un subconjunto de C++.

# Instrucciones del Lenguaje C++



Esto quiere decir que todo lo que aprendimos sobre el lenguaje C lo podemos aplicar de la misma manera en un programa en lenguaje C++. Pero lo que hagamos en C++ no necesariamente lo podremos aplicar en C.

El lenguaje C fue diseñado por Dennis Ritchie a fines de los años 1960. En esa época no existían conceptos como: la programación orientada a objetos, la sobrecarga de funciones y operadores, la programación genérica mediante el uso de plantillas , etc.

A inicios de los años 1980, Bjarne Stroustrup diseñó el lenguaje de programación C++, enriqueciendo el núcleo de C, agregándole conceptos como la Programación Orientada a Objetos.

A continuación veremos algunas de estas diferencias:

# TIPO DE DATO bool Y OPERADORES LÓGICOS

**Tipo de dato bool:** El manejo de expresiones booleanas se realiza en el lenguaje C mediante valores enteros, esto es el valor 0 se considera como valor falso y un valor diferente de cero se considera verdadero. En el lenguaje C++ se sigue empleando la misma forma, sin embargo para hacer más legibles los programas se ha definido el tipo de “bool” y las constantes “false” (con un valor cero) y “true” con un valor diferente de cero, igual a uno).

## Operadores lógicos:

En el lenguaje C se emplean los operadores:

- **!** (not)
- **||** (or)
- **&&** (and)

En el lenguaje C++ se emplean, además, los operadores:

- **not**
- **or**
- **and**

INSTRUCCIÓN #include



En el lenguaje C estamos acostumbrados a escribir:

```
#include <stdio.h>
```

Ahora en el lenguaje C++, para diferenciar de los archivos obsoletos de C, los archivos de cabecera (header files) ya no tienen la extensión .h y, para diferenciarlas de sus propias bibliotecas de funciones, el lenguaje C++ a ha añadido al inicio del nombre de las bibliotecas del lenguaje C la letra “c”.

Esto es:

## LENGUAJE C:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

Propios de C++:

## LENGUAJE C++:

#include <cstdio>

#include <cstdlib>

#include <cstring>

#include <cmath>

#include <iostream>

#include <fstream>

#include <iomanip>

# ESPACIO DE NOMBRES

(“namespace”)

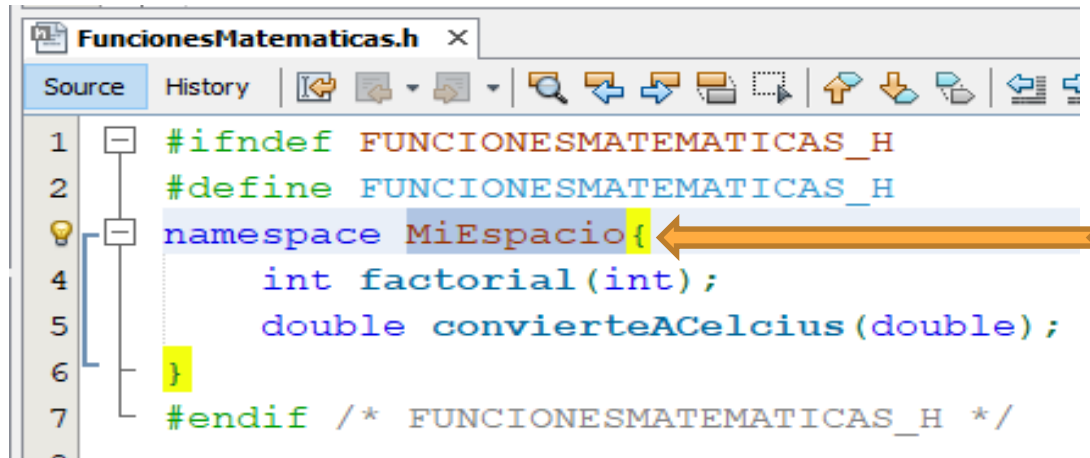
El lenguaje C++ se ha hecho tan popular que cada año se le incorporan más y más bibliotecas de funciones.

Eso ha traído algunos problemas, el más común se presenta con el **“choque entre elementos”**, que se produce cuando dos identificadores con el mismo nombre son incluidos en un proyecto desde dos bibliotecas de funciones distintas.

Para solucionar este problema se han creado lo que se denomina **“espacio de nombre”** o **“name sapce”**.

El espacio de nombre es precisamente nombre que se le ha dado a un conjunto de funciones.

El siguiente ejemplo muestra cómo se define un espacio de nombre en una biblioteca de funciones (\*):



The screenshot shows a code editor window titled 'FuncionesMatematicas.h'. The code defines a namespace 'MiEspacio' and contains two function declarations: 'int factorial(int);' and 'double convierteACelcius(double);'. A blue highlight is on the 'namespace MiEspacio{' line, and a yellow highlight is on the closing brace '}' on line 6. A large orange arrow points from the right towards the closing brace. The code is as follows:

```
1  #ifndef FUNCIONESMATEMATICAS_H
2  #define FUNCIONESMATEMATICAS_H
3  namespace MiEspacio{
4      int factorial(int);
5      double convierteACelcius(double);
6  }
7  #endif /* FUNCIONESMATEMATICAS_H */
```

Solo hay que colocar los encabezados de las funciones en un bloque como se muestra en la figura.

Los encabezados no se alteran aquí.

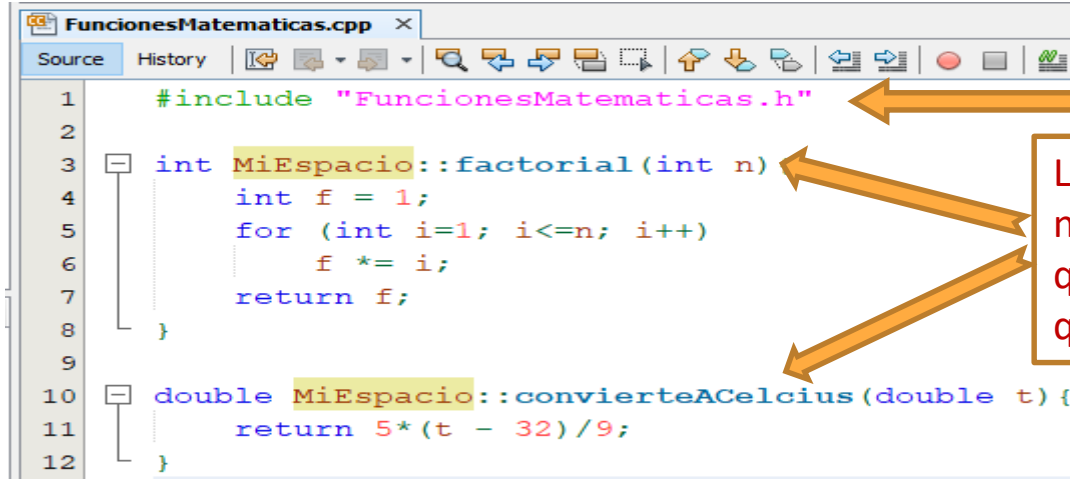
“MiEspacio” es el nombre que le hemos dado al “espacio de nombres” que hemos creado.

(\*) La confección de proyectos empleando bibliotecas de funciones en archivos independientes de main.cpp se explicará en otro documento.

A partir de aquí las funciones ya no podrán nombrarse como estamos acostumbrados sino habrá que indicar el “espacio de nombres” al que pertenecen.

En la implementación:

La cláusula `#include` es necesaria, recuerde que cada módulo se compila independientemente de los otros, por lo que se requiere que el identificador “MiEspacio” debe ser definido antes de usarlo.

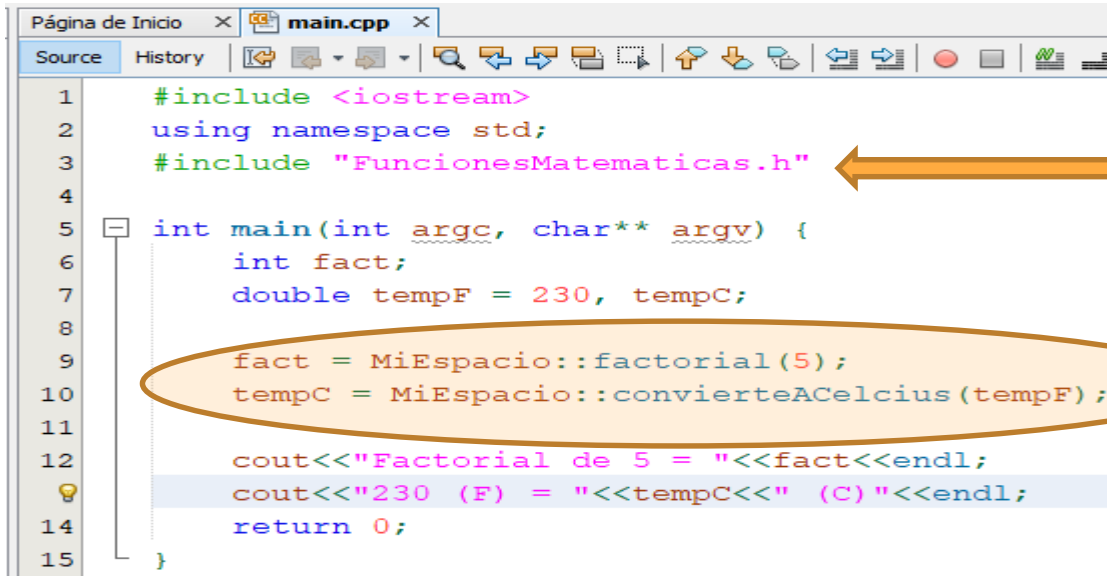


```
1  #include "FuncionesMatematicas.h"
2
3  int MiEspacio::factorial(int n)
4  {
5      int f = 1;
6      for (int i=1; i<=n; i++)
7          f *= i;
8      return f;
9  }
10 double MiEspacio::convierteACelcius(double t) {
11     return 5*(t - 32)/9;
12 }
```

Las funciones ya no pueden ser nombradas solo por su nombre, hay que indicar el espacio de nombre al que pertenece.

El espacio de nombres se separa del nombre de la función por el operador “::” que se denomina **“operador de ámbito”**.

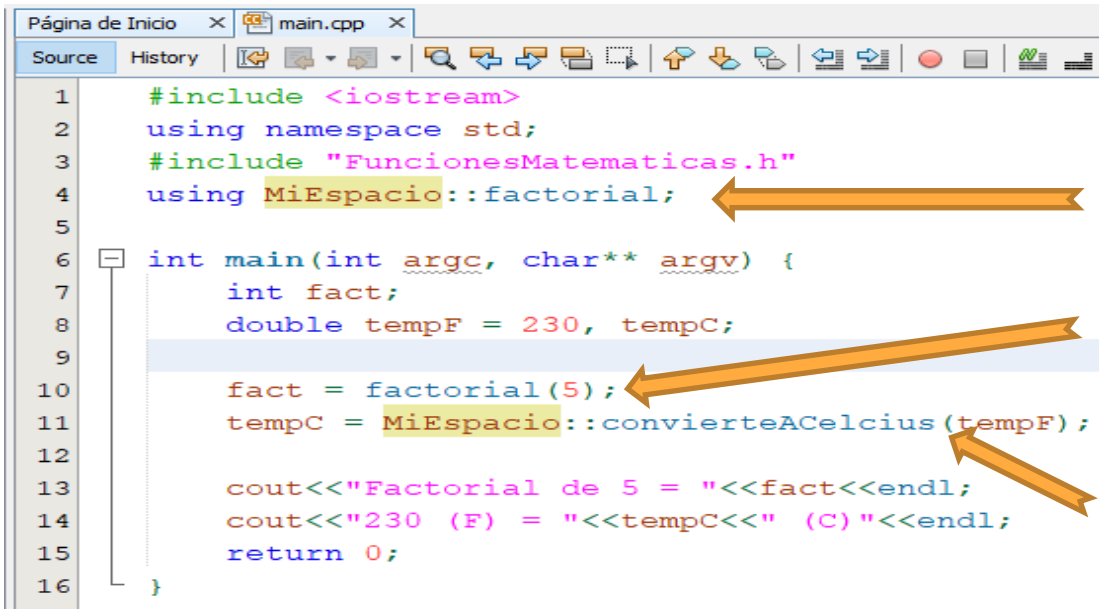
En la función main u otro módulo:



```
1  #include <iostream>
2  using namespace std;
3  #include "FuncionesMatematicas.h"
4
5  int main(int argc, char** argv) {
6      int fact;
7      double tempF = 230, tempC;
8
9      fact = MiEspacio::factorial(5);
10     tempC = MiEspacio::convierteACelcius(tempF);
11
12     cout<<"Factorial de 5 = "<<fact<<endl;
13     cout<<"230 (F) = "<<tempC<<" (C) "<<endl;
14     return 0;
15 }
```

# MODOS DE SIMPLIFICAR EL USO DE FUNCIONES DEFINIDAS EN UN ESPACIO DE NOMBRES

Cuando se requiere usar una función de un espacio de nombres varias veces en un módulo:



```
1  #include <iostream>
2  using namespace std;
3  #include "FuncionesMatematicas.h"
4  using MiEspacio::factorial;
5
6  int main(int argc, char** argv) {
7      int fact;
8      double tempF = 230, tempC;
9
10     fact = factorial(5);
11     tempC = MiEspacio::convierteACelcius(tempF);
12
13     cout<<"Factorial de 5 = "<<fact<<endl;
14     cout<<"230 (F) = "<<tempC<<" (C)"<<endl;
15     return 0;
16 }
```

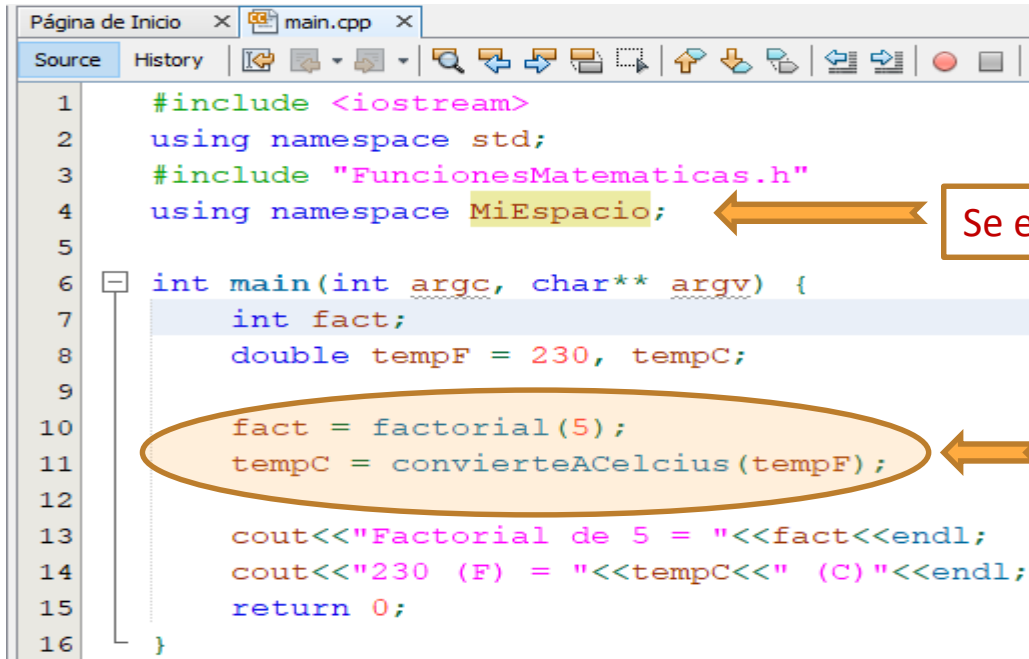
Se emplea la cláusula “using”.

La función ya no requiere el “espacio de nombres”.

Las otras funciones sí.



Cuando se requiere usar muchas funciones de un mismo espacio de nombres en un módulo:



```
1  #include <iostream>
2  using namespace std;
3  #include "FuncionesMatematicas.h"
4  using namespace MiEspacio;
5
6  int main(int argc, char** argv) {
7      int fact;
8      double tempF = 230, tempC;
9
10     fact = factorial(5);
11     tempC = convierteACelcius(tempF);
12
13     cout<<"Factorial de 5 = "<<fact<<endl;
14     cout<<"230 (F) = "<<tempC<<" (C)"<<endl;
15     return 0;
16 }
```

Se emplea la cláusula **“using namespace”**.

Ninguna de las funciones requerirán del “espacio de nombres”.

## BIBLIOTECAS ESTÁNDAR DE C++

En el caso de las bibliotecas estándar de C++ como **“iostream”**, **“fstream”**, **“iomanip”**, **etc.** , todos los elementos definidos allí han sido declarados en el espacio de nombres **“std”**, por lo que para usarlos requerirá emplear alguna de las formas explicadas atrás.

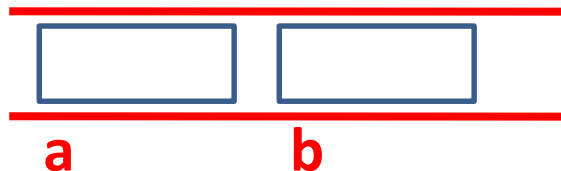
En cuanto a las bibliotecas correspondientes al lenguaje C como **“cstdio”**, **“cstdlib”**, **“cstring”**, **“cmath”**, **etc.**, sus elementos siguen igual, esto es no se han definido dentro de un espacio de nombres.

# REFERENCIAS

Una referencia es una variable que ha sido definida de una manera espacial.

## VARIABLES SIMPLES

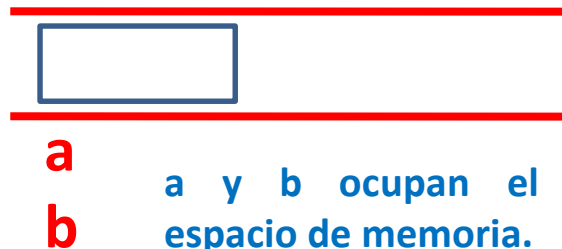
```
int a = 3;  
int b = a;
```



a y b ocupan espacios de memoria diferentes.

## REFERENCIAS

```
int a = 3;  
int &b = a; ← b es una referencia
```



a y b ocupan el mismo espacio de memoria.  
Se dice que **b** es el **alias** de **a**.

# PARÁMETROS POR REFERENCIA

El lenguaje C++, a diferencia del lenguaje C, si define parámetros por referencia.

La forma cómo se especifica este tipo de parámetro es muy simple, sin embargo la manera como se definen puede causar confusión y hasta dolores de cabeza a aquellos acostumbrados a trabajar con la forma cómo se manejan los parámetros desde el lenguaje C.

Sin embargo es muy importante dominar su uso ya que en programas complejos, la forma de C los volvería más complejos aun.

# INVOCACIÓN A UNA FUNCIÓN CON PARÁMETROS POR REFERENCIA

## LENGUAJE C:

```
int a, b = 5;  
a = f(&b);
```

Se simula, enviando la dirección de memoria de la variable.



## LENGUAJE C++:

```
int a, b = 5;  
a = f(b);
```

Solo se coloca la variable.

**Error común en C++:** Si el parámetro se define por referencia, no puede mandar un valor constante o una expresión:

`a = f(3);` ❌

`a = f(3*b);` ❌

# IMPLEMENTACIÓN A UNA FUNCIÓN CON PARÁMETROS POR REFERENCIA

## LENGUAJE C:

```
int f(int *b){  
    ...  
    *b += 5;  
    ...  
}
```

El dato es recibido por un puntero, y como tal se debe emplear.



## LENGUAJE C++:

```
int f(int &b){  
    ...  
    b += 5;  
    ...  
}
```

Se coloca un & delante del parámetro, luego solo se le usa como una variable común.

**EL COMPILADOR OCULTA AL DESARROLLADOR EL PUNTERO**



# PUNTEROS

# PUNTERO NULO

## LENGUAJE C:

**NULL:** define un valor 0 (cero) o una dirección nula. Requiere incluir la biblioteca `stdlib.h` donde se define.

Es válido:

```
int a, *pt;  
a = NULL;  (*)  
p = NULL;
```

(\*) Solo produce una advertencia (warning).



## LENGUAJE C++:

**nullptr:** define una dirección nula. Pertenece al núcleo de C++ por lo que no requiere definición.

Es válido:

```
int a, *pt;  
p = nullptr;
```

No es válido:

```
a = nullptr;
```



# ASIGNACIÓN DINÁMICA

## LENGUAJE C:

**malloc:** es una función. Requiere una operación “cast” según el tipo de puntero. Es necesario calcular el espacio que se desea asignar. Requiere incluir la biblioteca `stdio.h` donde se define.



```
int *pt;  
pt = (int*)malloc(sizeof(int));  
pt = (int*)malloc(sizeof(int)*10);
```



## LENGUAJE C++:

**new:** es un operador. Pertenece al núcleo de C++ por lo que no requiere definición. **NO REQUIERE** operaciones “cast” ni cálculos.

```
int a, *pt;  
pt = new int;  
pt = new int[10];
```

OTRAS DIFERENCIAS  
SIGNIFICATIVAS DEL  
LENGUAJE C++ QUE SE  
VERÁN MÁS ADELANTE O  
EN LP1.

**Sobrecarga de funciones:** Se puede definir dos o más funciones con el mismo nombre y que hagan cosas diferentes.

**Sobrecarga de operadores:** Se le puede dar a un operador una función diferente a la que usualmente tiene. Por ejemplo al “+” se puede usar para otra cosa que no sea sumar.

**Entrada y salida de datos:** El lenguaje C usa funciones como “scanf” y “printf”, el Lenguaje C++ emplea objetos y sobrecarga de operadores como **cin** y **cout**, **>>** y **<<**.

**Plantillas:** Es una herramienta del lenguaje C++ que permite definir una sola función o clase y que de manera automática el compilador cree varias funciones o clase en donde cada una de ellas se adapte a tipo de dato diferente (por diferentes que sean), según se necesite.

Por ejemplo, en el lenguaje C (math.h) se definen dos funciones de valor absoluto: **abs** y **fabs**, una para enteros (**int**) y la otra para valores de punto flotante (**double**), en C++ se definiría una sola plantilla que pueda llamarse simplemente **abs**.

(continúa en la siguiente lámina)

(continuación de la anterior lámina)

Luego el compilador creará una función por cada tipo que uno necesite (**int**, **long**, **float**, **double**, **etc.**), y para poder utilizarlas solo debemos usar el identificador **abs** con el tipo que queramos sin cambio alguno.

Ejemplo:

```
int a=-3, b;  
float p=-3.5, w;  
double f=-4653.83, g;  
  
b = abs(a);  
w = abs(p);  
g = abs(f);
```

**Clases y Objetos:** Si bien aún se puede continuar usando estructuras (**struct**), en lenguaje C++ las estructuras se consideran elementos obsoletos.

Las clases son una evolución de las estructuras y son los elementos básicos de la Programación orientada a objetos (**PPO**).

En términos de la programación tradicional, una **clase** corresponde a un “tipo dato” mientras que un **objeto** corresponde a una “variable”.



**La Clase string:** En el lenguaje C, las cadenas de caracteres se definen como arreglos de caracteres y se manipulan con la biblioteca de funciones **"string.h"**. En C++ las cadenas se manejan de la misma manera, claro está empleando la biblioteca **"cstring"**, sin embargo también ha definido una biblioteca denominada **"string"** (**cstring** ≠ **string**). En esta biblioteca se define la **"clase string"**(<sup>\*</sup>), con la que se pueden definir objetos de esta clase.

Esta clase se puede considerar como una **caja negra** que permite encapsular la manipulación de cadenas de caracteres. La finalidad de este curso no es la de emplear elementos sin saber cómo funcionan, por lo que **en el curso no estará permitido su uso de esta biblioteca**. Al final del semestre usted estará suficientemente capacitado como para poder implementar una similar.

(<sup>\*</sup>) Recuerde que una clase es un tipo de datos y un objeto es una variable.