

## Clases y funciones amigas: friend

1. Introducción a las funciones amigas (Funciones friend)
2. Funciones amigas
3. Clases amigas
4. Ejemplo: La clase fraccional. Métodos operadores amigos
5. Ejemplo: Las clases Coche y Camión.

1

## Clases y funciones amigas: friend

### Introducción: funciones amigas

Hemos visto que el acceso entre clases es imposible cuando definimos los miembros como `private`.

```
class Punto
{
  private :
    float x;
    float y;
  public :
    Punto ( );
    void visualizar ( );
};
```

Desde una clase no se puede acceder a los métodos o atributos privados de otra clase.

Desde una función normal tampoco se puede acceder a miembros privados de una clase.

2

## Clases y funciones amigas: friend

### Introducción: funciones amigas

Vamos a añadir un método público en la clase Punto llamado *distancia*, que calcula la distancia entre dos puntos.

```
class Punto
{
  private :
    float x;
    float y;
  public :
    Punto ( float a, float b );
    void visualizar ( );
    float distancia( Punto p );
} ;
```

```
float Punto:: distancia ( Punto p )
{ float d;
  d= sqrt( sqrt(p.x-x)+sqrt(p.y - y) );
  return d;
}
```

```
Punto c1(5, 2), c2(2,3);
dist = c1.distancia(c2);
```

Poco elegante

El método *distancia* tiene acceso a los atributos del objeto *c1* (receptor del mensaje), como a los atributos del objeto *c2* (objeto argumento).

3

## Clases y funciones amigas: friend

### Introducción: funciones amigas

El método *distancia* está operando sobre dos puntos, sería más vistoso poder calcular la distancia de la siguiente forma:

```
Punto c1(5, 2), c2(2,3);
dist = distancia ( c1, c2 );
```

Mucho mejor

Pero para ello, necesitamos que *distancia* sea una función de la forma:

```
float distancia ( Punto p1, Punto p2);
```

Función definida fuera de la clase Punto

Pero si sacamos la función *distancia* fuera de la clase *Punto*, ya no podemos acceder a los miembros privados de la clase *Punto*.

4

### Funciones amigas

Para resolver éste problema, es decir, para que una función externa a la clase pueda acceder a los atributos privados, C++ permite definir dicha función como **amiga** de la clase.

**En nuestro ejemplo, podemos definir la función distancia como amiga de la clase Punto.**

```
class Punto
{
private :
    float x;
    float y;
public :
    Punto ( float a, float b );
    void visualizar ( );
    friend float distancia( Punto p1, Punto p2);
} ;
```

Para ello utilizamos la palabra reservada **friend**

```
float distancia ( Punto p1, Punto p2)
{ float d;
  d= sqrt( sqrt(p2.x-p1.x)+sqrt(p2.y - p1.y) );
  return d;
}
```

### Funciones amigas

```
class Punto
{
private :
    float x;
    float y;
public :
    Punto ( float a, float b );
    void visualizar ( );
    friend float distancia( Punto p1, Punto p2);
} ;
```

```
float distancia ( Punto p1, Punto p2)
{ float d;
  d= sqrt( sqrt(p2.x-p1.x)+sqrt(p2.y - p1.y) );
  return d;
}
```

- La implementación de la función **distancia**, no hace uso del operador de ámbito (**::**) porque es una función amiga de **Punto**, pero no pertenece a la clase.
- La llamada no necesita hacerse a través de un objeto de la clase.
- Las funciones amigas no contienen el argumento implícito **this**.

## Clases y funciones amigas: friend

### Funciones amigas

- La utilidad de las funciones amigas es poder acceder a los datos privados de una o más clases.
- Una función declarada friend de una clase C, es una función no miembro de la clase, que puede acceder a los miembros privados de la clase.
- Una función amiga puede declararse en cualquier sección de la clase. No es miembro de la clase, por lo que no se ve afectada por los modificadores `private` y `public`.
- Una función puede ser amiga de una clase y miembro de otra.

### *Ocultación de la información*

Con la declaración de amigos, ¿existe una puerta trasera para burlar la ocultación de la información?

## Clases y funciones amigas: friend

### Funciones amigas

Habíamos visto que uno de los pilares de la programación orientada a objetos era la ocultación de la información. La declaración de funciones amigas permite dotar a la POO de una mayor flexibilidad.

¿existe una puerta trasera para burlar la ocultación de la información?

**Se preservan la seguridad y protección que proporcionan las clases.**

- Es la clase la que dice quiénes son sus amigos y pueden acceder a sus miembros privados.
- Ninguna función puede autodeclararse amiga y acceder a la privacidad de una clase sin que la propia clase tenga conocimiento de ello.

## Clases y funciones amigas: friend

### Clases amigas

Si queremos que todos los métodos de una clase sean amigos de otra, entonces declaramos toda la clase como amiga.

```
friend class NombreClase;
```

Para ello utilizamos la siguiente declaración

```
class Clase2;
```

Prototipo para que la Clase1 reconozca la existencia de la Clase2

```
class Clase1;
```

```
{  
private :  
float x, y;  
public :  
void visualizar ( );  
friend class Clase2;  
};
```

Desde la clase2 se puede acceder a los miembros privados de la Clase1

Clase2 es amiga de Clase1

9

## Clases y funciones amigas: friend

### Ejemplos: La clase Fraccional

```
class Fraccional  
{  
private:  
int numerador ;  
int denominador ;  
void simplificar( ) ;  
public:  
Fraccional(int n=0, int d=1);  
Fraccional operator + ( Fraccional f );  
Fraccional operator - (Fraccional f);  
Fraccional operator * (Fraccional f);  
Fraccional operator / (Fraccional f);  
int ac_numerador( );  
int ac_denominador( );  
void mu_numerador (int n);  
void mu_denominador ( int d);  
};
```

```
...  
f3 = f1 + f2 ;  
...
```

¿Cómo hay que modificar la clase para poder sumar un entero **int** a un **Fraccional** ?

```
Fraccional operator + ( int n );
```

10

## Clases y funciones amigas: friend

### Ejemplos: La clase Fraccional

```
class Fraccional
{
  private:
    int numerador ;
    int denominador ;
    void simplificar ( ) ;
  public:
    Fraccional(int n=0, int d=1);
    Fraccional operator + ( Fraccional f );
    Fraccional operator + ( int n );
    Fraccional operator - (Fraccional f);
    Fraccional operator * (Fraccional f);
    Fraccional operator / (Fraccional f);
    ....
    ....
};
```

```
Fraccional Fraccional :: operator + (int n )
{
  Fraccional aux;
  aux.numerador = numerador + denominador * n;
  aux.denominador = denominador;
  aux.simplifica();
  return aux;
}
```

```
...
Fraccional f1, f2, f3;
f3 = f1 + f2 ;
f3 = f2 + 5;
...
f3 = 5 + f2;
```

**Error !!!**

11

## Clases y funciones amigas: friend

### Ejemplos: La clase Fraccional

Una forma de solucionar el error anterior es definir como amiga la función que suma un fraccional a un entero:

```
class Fraccional
{
  private:
    .....
  public:
    Fraccional(int n=0, int d=1);
    Fraccional operator + ( Fraccional f );
    Fraccional operator + ( int n );
    friend Fraccional operator + ( int n , Fraccional f);
    Fraccional operator - (Fraccional f);
    ....
    ....
};
```

```
...
Fraccional f1, f2, f3;
f3 = f1 + f2 ;
f3 = f2 + 5;
...
f3 = 5 + f2;
```

Todo funciona correctamente

12

## Clases y funciones amigas: friend

### Ejemplos: La clase Fraccional

```
class Fraccional
{
  private:
    .....
  public:
    Fraccional(int n=0, int d=1);
    Fraccional operator + ( Fraccional f );
    Fraccional operator + ( int n );
    friend Fraccional operator + ( int n , Fraccional f);
    Fraccional operator - (Fraccional f);
    ....
};
```

```
Fraccional operator + (int n , Fraccional f)
{
  Fraccional aux;
  aux.numerador = f.numerador + f.denominador * n;
  aux.denominador = f.denominador;
  aux.simplifica();
  return aux;
}
```

No hace uso del  
operador de  
ámbito (::)

13

## Clases y funciones amigas: friend

### Ejemplo: Las clases Coche y Camión

```
class Camion; // referencia anticipada

class Coche
{ private:
  int plazas;
  int velocidad;
public:
  coche(int p, int v)
  {
    plazas = p;
    velocidad = v;
  }
  void mas_veloz( Camion t );
};
```

```
...
Coche c;
Camion g;
c.mas_veloz(g);
...
```

```
class Camion
{ private:
  int peso;
  int velocidad;
public:
  camion(int p, int v)
  {
    peso = p;
    velocidad = v;
  }
};
```

14

## Clases y funciones amigas: friend

### Ejemplo: Las clases Coche y Camión

```
class Camion; // referencia anticipada
```

```
class Coche  
{ private:  
  int plazas;  
  int velocidad;  
public:  
  coche(int p, int v)  
  {  
    plazas = p;  
    velocidad = v;  
  }  
  void mas_veloz( Camion t );  
};
```

El método `mas_veloz` accede al atributo `velocidad` de la clase `Camión`.

**NO PERMITIDO**

```
void Coche::mas_veloz ( Camion t )  
{  
  int comp;  
  comp = velocidad - t.velocidad;  
  if (comp < 0)  
    cout << " El camión es más rápido " ;  
  else if (comp == 0)  
    cout << " Son igual de rápidos " ;  
  else  
    cout << " El coche es más rápido " ;  
}
```

15

## Clases y funciones amigas: friend

### Ejemplo: Las clases Coche y Camión

Para resolver ese problema, basta con que la clase `Camión` dé permiso a la función `mas_veloz` de la clase `Coche` para acceder a sus atributos. Esto es, declarar a `mas_veloz` como amiga de `Camion`.

```
class Camion; // referencia anticipada
```

```
class Coche  
{ private:  
  int plazas;  
  int velocidad;  
public:  
  coche(int p, int v)  
  {  
    plazas = p;  
    velocidad = v;  
  }  
  void mas_veloz( Camion t );  
};
```

```
class Camion  
{ private:  
  int peso;  
  int velocidad;  
public:  
  camion(int p, int v)  
  {  
    peso = p;  
    velocidad = v;  
  }  
  friend void Coche::mas_veloz(Camion t);  
};
```

Operador de resolución de ámbito

16