

Clase 7. Sobrecarga de operadores

MODELO EDUCATIVO
TEC21

Sobrecarga de operadores





Sobrecarga
de
operadores

Sobrecarga de Op. – Conceptos Clave

Es el concepto de tomar operadores tales como $+$, $-$, $*$ entre otros y darles un significado para tipos de datos definidos por el usuario.

- Ej: se pueden definir clases y que en éstas se puedan presentar comportamientos usando operadores aritméticos.

No podemos cambiar de un operador

- Aridad
 - Precedencia - jerarquía
 - Asociatividad – izquierda a derecha o al contrario
-
- Solo se pueden sobrecargar operadores existentes

Operadores que no se pueden sobrecargar.

- :: (resolución de ámbito)
- . (selección de un miembro)
- .* (selección de un miembro referenciado por un puntero)
- ?: (operador condicional)

Antes de avanzar...funciones friend

¿Qué significa que una **función sea amiga** de una clase?

- Una función amiga no está dentro del ámbito de la clase pero tiene acceso a todos los miembros privados y protegidos de la misma.
- A pesar de que éstas funciones aparecen dentro del prototipo de la clase, NO son funciones miembro de ésta.
- Se define utilizando la palabra reservada **friend** en la declaración de la clase.

La sobrecarga de operadores se pueden implementar...

Como función miembro

NOTA: en este ejemplo para operador binario, en caso de ser unario no recibe parámetro

- Se declara:

```
<Tipo_retorno> operator op (<tipo_parámetro>);
```

```
Fraccion operator + (const Fraccion&);
```

- Se define/implementa:

```
<Tipo_retorno> NombreClase::operator op (<tipo_parámetro> var);
```

```
Fraccion Fraccion::operator + (const Fraccion& otra){...}
```


La sobrecarga de operadores se pueden implementar...

- **Como funciones amigas**

NOTA: en este ejemplo para operador binario, en caso de ser unario recibe un parámetro

- Se declara:

```
friend <Tipo_retorno> operator op (<tipo_par>, <tipo_par>);  
friend Fraccion operator-(const Fraccion&, const Fraccion&);
```

- Se define/implementa:

```
<Tipo_retorno> NombreClase::operator op (<tipo_par> var1, <tipo_par> var2){...}  
Fraccion operator -(const Fraccion& uno, const Fraccion& dos){...}
```

Hay casos especiales...

- Por ejemplo los operadores de flujo de entrada y de salida ($>>$, $<<$), siempre se deben implementar como funciones amigas.
- Los operadores de asignación ($=$), selección por índice ($[]$), llamada a funciones (" $()$ "), and selección de miembros ($->$) deben ser definidos como funciones miembro.

Sobrecarga de operadores de entrada y de salida

```
// Sobrecarga de operador de flujo de salida
friend ostream& operator<< (ostream &, const Fraccion&);

// Sobrecarga de operador de flujo de entrada
friend istream &operator>> (istream &, Fraccion&);
```

Declaración

```
// Sobrecarga de operador de flujo de salida
ostream &operator << (ostream & salida, const Fraccion& miFracc){
    salida << miFracc.num << "/" << miFracc.den;
    return salida;
}

// Sobrecarga de operador de flujo de entrada
istream &operator >> (istream & entrada, Fraccion& miFracc){
    entrada >> miFracc.num >> miFracc.den;
    return entrada;
}
```

Implementación

¿En qué otros casos necesitamos que se implementen como funciones amigas?

- Ejemplo:
- Tenemos una clase que representan coordenadas rectangulares **CoordRect** y otra que representa coordenadas polares **CoordPol**
 - Cuando en la clase CoordRect quisiéramos sobrecribir el operador + para que el primer operando fuera un objeto de la clase CoordPol y el segundo operando el de la clase CoordRect:

CoordRect cr1;

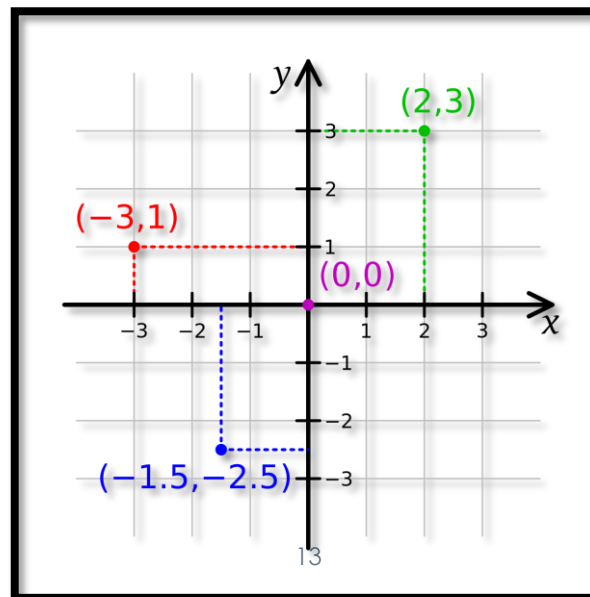
CoordPol cp1;

Y queremos que funcione esto: cp1 + cr2

Sobrecarga - Escenario

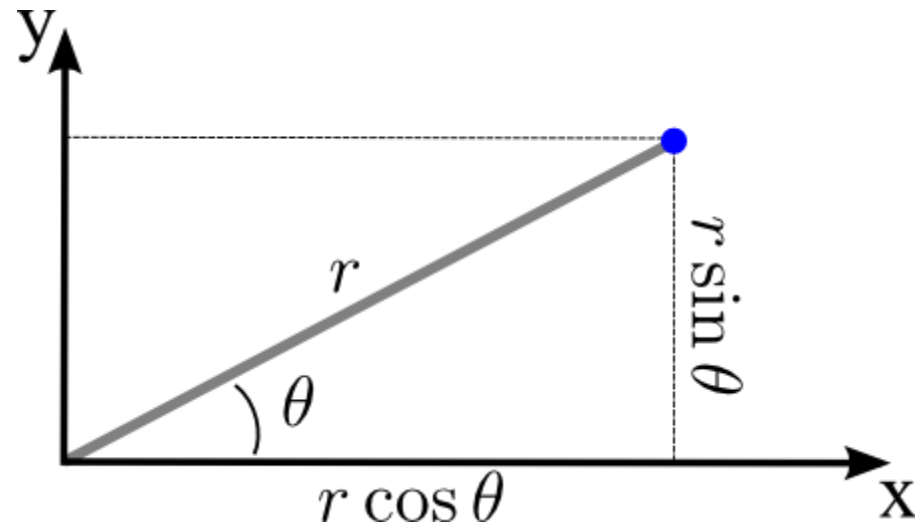
Asumamos que se tiene dos clases que representan coordenadas polares y rectangulares.

Las coordenadas rectangulares manejan puntos en un plano cartesiano en el formato (x, y) .



Sobrecarga - Escenario

Las coordenadas polares es como se manejan puntos en un plano cartesiano usando ángulos y una dimensión, ej: (4 , 30°).



Sobrecarga - Escenario

- La suma y resta de coordenadas es mucho más sencillo de hacer en la forma rectangular.
- La división y multiplicación es mucho más sencilla de hacer en el formato polar.

Sobrecarga

- De no permitirse la sobrecarga de operadores, la suma de dos coordenadas (en formato rectangular) se podría hacer usando una de estas dos opciones.

```
Rect r1(1,2);  
Rect r2(3,2);  
  
int x = r1.x + r2.x;  
int y = r1.y + r2.y;  
  
//o  
  
r1.sumar (r2);
```

Mala práctica de desarrollo de software.

Válido pero se puede mejorar.

Sobrecarga

- Es mucho más claro escribir algo así

```
Rect r1(1,2);  
Rect r2(3,2);  
Rect r3;  
  
r3 = r1 + r2
```

Es claro que hay una suma entre dos tipos de datos definidos por el usuario.

¿Quieres saber más?

<https://www.programiz.com/cpp-programming/operator-overloading>

Sobrecarga

- Mejor aún, es posible ocultar detalles como: qué tal que se desea sumar una coordenada polar con una rectangular.

```
Rect r1(1,2);  
Pol p1(3,45);  
Rect r3;  
  
r3 = r1 + p1;
```

Completamente válido y el compilador se encarga de realizar las instrucciones que le proporcionemos:

Para este ejemplo p1 será convertido a rectangular, se hace la suma y se regresa el valor esperado.

Código de Rect (.h)

```
1  #ifndef RECT_H_INCLUDED
2  #define RECT_H_INCLUDED
3
4
5  class Rect
6  {
7
8  private:
9      int x;
10     int y;
11
12 public:
13
14     Rect();
15     Rect(int, int);
16
17     Rect operator+(const Rect& val)
18     {
19         Rect temp;
20         temp.x = this->x + val.x;
21         temp.y = this->y + val.y;
22         return temp;
23     }
24
25     //no la mejor opcion para imprimir sus valores solamente
26     int getX(); |
27     int getY();
28
29 };
30
31
32
33 #endif // RECT_H_INCLUDED
34
```

Notar que la palabra reservada “operator” seguida del símbolo para suma indica que se está sobrecargado la suma.

Esta sobrecarga del “+” , permite que cuando dos tipos de datos Rect se sumen , su parte “x” se sumará con la parte “x”, la parte “y” con la parte “y” y se regresará la suma en un tipo de dato Rect.

Código de Rect (.cpp)

```
1  #include "Rect.h"
2
3  Rect::Rect ()
4  {
5
6  }
7
8  Rect::Rect (int valX, int valY)
9  {
10     x = valX;
11     y = valY;
12 }
13 int Rect::getX()
14 {
15     return x;
16 }
17
18 int Rect::getY()
19 {
20     return y;
21 }
22
```


Código de main (.cpp)

```
1  #include <iostream>
2  #include "Pol.h"
3  #include "Rect.h"
4
5  using namespace std;
6
7  int main()
8  {
9
10     Rect r1(1,2);
11     Rect r2(3,4);
12     Rect r3;
13
14     r3 = r1 + r2;
15
16     cout << r3.getX() << " " << r3.getY() << endl;
17
18     return 0;
19 }
20
21
```

La sobrecarga del “+” permite que esto sea posible de lo contrario sería un error de compilación (el compilador no sabría cómo interpretar la suma de dos tipos de datos Rect).

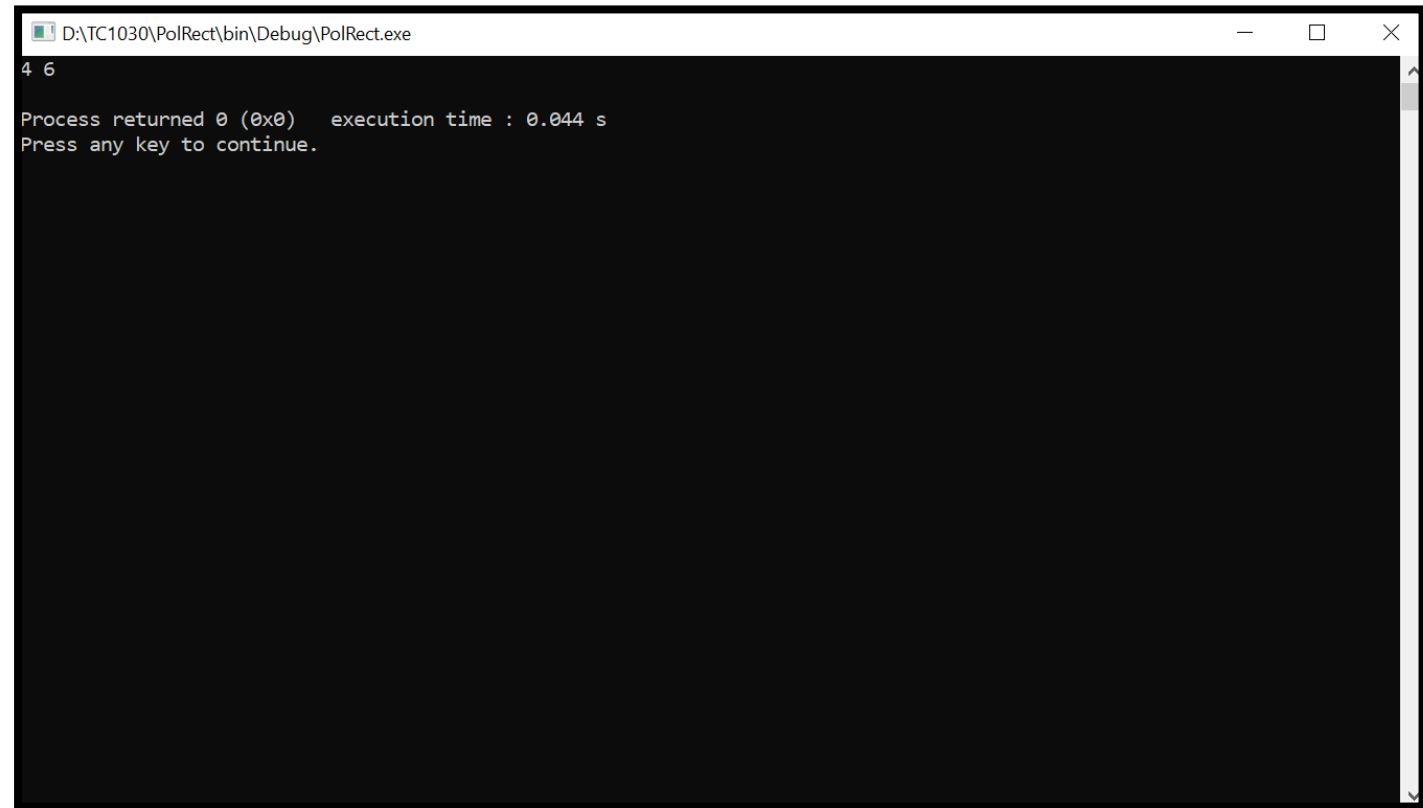
Se podría hacer de otra forma, declarando un método Suma, pero es mucho más claro y fácil de entender.

Resultado

Como se puede observar, se imprime la suma de rect1 y de rect2

$$1 + 3 = 4$$

$$2 + 4 = 6$$



The screenshot shows a Windows command prompt window with the title bar "D:\TC1030\PolRect\bin\Debug\PolRect.exe". The window content displays the output of a program: "4 6" on the first line, followed by "Process returned 0 (0x0) execution time : 0.044 s" and "Press any key to continue." on the next two lines. The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

```
D:\TC1030\PolRect\bin\Debug\PolRect.exe
4 6
Process returned 0 (0x0) execution time : 0.044 s
Press any key to continue.
```

Práctica

- Crear la sobre carga de “-” para Rect
- Crear la sobre carga de “*” y “/” para Pol

Reto:

- Crear la sobre carga para Rect de + **PERO** recibiendo un Polar.
- Hint: la sobrecarga de + para Rect con Pol debe recibir un Polar como parámetro, convertirlo a rectangular, hacer la suma, y regresarlo.



Breakout rooms





Quiz



Quiz: Sobrecarga de operadores

- Realizar Quiz rápido del tema: Sobrecarga de operadores en Canvas
- Contiene 5 preguntas

Cierre y avisos

Videos de sobrecargas de operadores

Los encuentras en esta lista de reproducción:

https://www.youtube.com/playlist?list=PLS4-ShDA9mk8GnsUZ_5A7Ktrr7npAkNb7