

ESTRUCTURAS DE DATOS

NOTAS SOBRE C++

Sobrecarga de operadores

Manuel Montenegro Montes

Departamento de Sistemas Informáticos y Computación
Facultad de Informática – Universidad Complutense de Madrid



Ejemplo: números complejos

VAMOS A DESCANSAR UN POCO DE NODOS.

```
class Complejo { Clase distinta al resto
public:
    Complejo(double real, double imag);

    double get_real() const; métodos de acceso
    double get_imag() const;
    void display() const; mostrar por pantalla

private:
    double real, imag;
};
```

SOBRECARGA= ADAPTAR ELEMENTOS DE C++ A LAS CLASES QUE NOSOTROS DEFINAMOS

Él pone este ejemplo simplemente para ejemplificar pero no sería necesario crear la clase Complejo. Está ya definida en C++

Existe la clase `std::complex`, definida en `<complex>`

Aritmética con números complejos

```
Complejo suma(const Complejo &z1, const Complejo &z2) {  
    return { z1.get_real() + z2.get_real(),  
            z1.get_imag() + z2.get_imag() };  
}
```

Suman las partes reales y las partes imaginarias.

```
Complejo multiplica(const Complejo &z1, const Complejo &z2) {  
    double z1_real = z1.get_real(), z1_imag = z1.get_imag();  
    double z2_real = z2.get_real(), z2_imag = z2.get_imag();  
    return { z1_real * z2_real - z1_imag * z2_imag,  
            z1_real * z2_imag + z1_imag * z2_real };  
}
```

No hace falta saber como implementar esto y aprendernoslo, lo usa de ejemplo.

Ejemplo de uso

```
int main() {  
    Complejo z1(2.0, -3.0), z2(1.0, 0.0);  
    Complejo z3 = suma(z1, z2);  
    Complejo z4 = suma(multiplica(z1, z1), z2);  
  
    z3.display();  
    std::cout << std::endl;  
  
    z4.display();  
    std::cout << std::endl;  
  
    return 0;  
}
```

3-3i

-4-12i

Uso de operadores

- Con los tipos numéricos básicos (`int`, `double`, etc.) podemos expresar operaciones aritméticas utilizando los operadores `+` y `*` en forma infija.
 - Ejemplo: `x + y * z`
- Con nuestra clase `Complejo` no tenemos la misma suerte:
 - `suma(z1, z2)`
 - `suma(multiplica(z1, z1), z2)`
- Sería más legible poder escribir:
 - `z1 + z2`
 - `z1 * z1 + z2`

En c++ se puede hacer, SOBRECARGANDO LOS OPERADORES.
- En C++ es posible definir implementaciones personalizadas de los operadores, es decir, **sobrecargarlos**.

Sobrecargar operadores



Aritmética con números complejos

```
Complejo suma(const Complejo &z1, const Complejo &z2) {  
    return { z1.get_real() + z2.get_real(),  
            z1.get_imag() + z2.get_imag() };  
}
```

```
Complejo multiplica(const Complejo &z1, const Complejo &z2) {  
    double z1_real = z1.get_real(), z1_imag = z1.get_imag();  
    double z2_real = z2.get_real(), z2_imag = z2.get_imag();  
    return { z1_real * z2_real - z1_imag * z2_imag,  
            z1_real * z2_imag + z1_imag * z2_real };  
}
```

Aritmética con números complejos

Sobrecarga del operador suma

```
Complejo operator+(const Complejo &z1, const Complejo &z2) {  
    return { z1.get_real() + z2.get_real(),  
            z1.get_imag() + z2.get_imag() };  
}
```

sobrecarga del operador * multiplicación

```
Complejo operator*(const Complejo &z1, const Complejo &z2) {  
    double z1_real = z1.get_real(), z1_imag = z1.get_imag();  
    double z2_real = z2.get_real(), z2_imag = z2.get_imag();  
    return { z1_real * z2_real - z1_imag * z2_imag,  
            z1_real * z2_imag + z1_imag * z2_real };  
}
```

- Puede sobrecargarse un operador creando una función con nombre `operator[?]`, donde `[?]` es un operador de C++.

Ejemplo de uso

```
int main() {  
    Complejo z1(2.0, -3.0), z2(1.0, 0.0);  
    Complejo z3 = suma(z1, z2);  
    Complejo z4 = suma(multiplica(z1, z1), z2);  
  
    z3.display();  
    std::cout << std::endl;  
  
    z4.display();  
    std::cout << std::endl;  
  
    return 0;  
}
```

Ahora podremos cambiar esto por los operadores de suma y multiplicación respectivamente.



Ejemplo de uso

```
int main() {  
    Complejo z1(2.0, -3.0), z2(1.0, 0.0);  
    Complejo z3 = z1 + z2;  
    Complejo z4 = z1 * z1 + z2;  
  
    z3.display();  
    std::cout << std::endl;  
  
    z4.display();  
    std::cout << std::endl;  
  
    return 0;  
}
```

Equivale a
`operator+(z1, z2)`

Equivale a
`operator+(operator*(z1, z1), z2)`

Gracias a la sobrecarga lo podemos utilizar.

¿Qué operadores pueden sobrecargarse?

+ - * / % ^ & | << >>

== <= >= != < > && || !

Los que he rodeado yo son los que según él nos van a interesar.

= += -= *= /=

++ --
[] ()

→

new delete

etc.

Sobrecarga del operador << para E/S

Normalmente se utiliza cuando queremos imprimir un determinado objeto por pantalla.

Generalizando el método `display()`

```
class Complejo {  
public:
```

```
...  
void display() const;
```

```
private:
```

```
...  
};
```

```
void Complejo::display() const {  
    std::cout << real << ... << "i";  
}
```

En el lado izquierdo puedo poner cualquier cosa que sea instancia de la clase ostream o alguna de sus subclases

- El método `display()` envía una representación en cadena del objeto a la salida estándar (`std::cout`).
- ¿Y si quiero escribirla en un fichero (clase `ofstream`)?
- ¿Y si quiero escribirla un `string` (clase `ostringstream`)?

FP

Todas heredan de la clase `ostream`.

Generalizando el método `display()`

```
class Complejo {  
public:  
    ...  
    void display(ostream &out) const; → { void Complejo::display(ostream &out) const {  
private:                                out << real << ... << "i";  
    ...                                }  
};
```

stream que recibo como parametro

- El método `display()` envía una representación en cadena del objeto a la salida estándar (`std::cout`).
 - ¿Y si quiero escribirla en un fichero (clase `ofstream`)?
 - ¿Y si quiero escribirla un `string` (clase `ostringstream`)?
- Todas heredan de la clase `ostream`.

Actualizando el ejemplo

```
int main() {  
    Complejo z1(2.0, -3.0), z2(1.0, 0.0);  
    Complejo z3 = z1 + z2;  
    Complejo z4 = z1 * z1 + z2;  
  
    z3.display(std::cout);  
    std::cout << std::endl;  
  
    z4.display(std::cout);  
    std::cout << std::endl;  
  
    return 0;  
}
```



El operador << para E/S

`std::cout << "Hola"`

Instancia de
`ostream`

funciona porque está
sobrecargado para todos
esos casos (Strings, doubles
int...)

Instancia de
`string`

`std::cout << z1`

Instancia de
`ostream`

Tendríamos que definir una
sobrecarga para el operador
de "desplazamiento" <<

Instancia de
`Complejo`

Sobrecargando << para números complejos

```
void operator<<(std::ostream &out, Complejo &z) {  
    z.display(out);  
}
```

```
int main() {  
    ...
```

```
    z3.display(std::cout);  
    std::cout << std::endl;
```

```
    z4.display(std::cout);  
    std::cout << std::endl;
```

```
    return 0;
```

```
}
```

ES LO MISMO QUE AMBAS

```
int main() {
```

```
    ...
```

```
    std::cout << z3; notación del operador <<  
    std::cout << std::endl;
```

```
    std::cout << z4;  
    std::cout << std::endl;
```

```
    return 0;
```

```
}
```

Sobrecargando << para números complejos

```
void operator<<(std::ostream &out, Complejo &z) {  
    z.display(out);  
}
```

```
int main() {  
    ...  
  
    z3.display(std::cout);  
    std::cout << std::endl;  
  
    z4.display(std::cout);  
    std::cout << std::endl;  
  
    return 0;  
}
```

```
int main() {  
    ...  
  
    ( ( (std::cout << z3) << std::endl) << z4 ) << std::endl; ✖  
  
    return 0;  
}
```

NO PODRÍAMOS REUNIRLO ÚNICAMENTE EN UNO.

Sobrecargando << para números complejos

```
std::ostream & operator<<(std::ostream &out, Complejo &z) {  
    z.display(out);  
    return out;  
}
```

Ahora si podríamos ponerlo todo en una sola línea. Antes era void. Devuelve el mismo ostream que recibe.

`((std::cout << z3) << std::endl) << z4 << std::endl;` ✓

`std::cout <<`

`std::cout << z4`

`std::cout << std::endl`

Sobrecarga dentro de una clase



Sobrecarga fuera de una clase

- Las definiciones de sobrecarga vistas hasta ahora son funciones que no pertenecen a ninguna clase:

Queremos que pertenezcan a una clase.

```
Complejo operator+(const Complejo &z1, const Complejo &z2) {  
    return { z1.get_real() + z2.get_real(),  
            z1.get_imag() + z2.get_imag() };  
}
```

no podemos acceder directamente a los atributos,
sin utilizar los métodos de acceso.

Sobrecarga dentro de una clase

- También habríamos podido definirlas como métodos de la clase Complejo.
- Si lo hacemos así, el primer operando es this.
- Ventaja: podemos acceder a los atributos privados.

```
class Complejo {  
public:  
    ...  
  
    Complejo operator+(const Complejo &z2) const {  
        return { real + z2.real, imag + z2.imag };  
    }  
  
private:  
    double real, imag;  
};
```

Solo recibe un parámetro

$z1 + z2$


equivale a

$z1.operator+(z2)$


¿Podemos hacer lo mismo con...?

Sería this

```
Complejo operator*(const Complejo &z1, const Complejo &z2) {  
    double z1_real = z1.get_real(), z1_imag = z1.get_imag();  
    double z2_real = z2.get_real(), z2_imag = z2.get_imag();  
    return { z1_real * z2_real - z1_imag * z2_imag,  
            z1_real * z2_imag + z1_imag * z2_real };  
}
```



```
std::ostream & operator<<(std::ostream &out, Complejo &z) {  
    z.display(out);  
    return out;  
}
```



Tendría que incluir ese método como un método de la clase ostream.

Función aislada.

¡No podemos añadir
métodos a la clase
ostream!