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

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

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():
                                           3-3i
  std::cout << std::endl;</pre>
  z4.display();
                                          -4-12i
  std::cout << std::endl;</pre>
  return 0;
```

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++ es posible definir implementaciones personalizadas de los operadores, es decir, sobrecargarlos.

En c++ se puede hacer, SOBRECARGANDO LOS OPERADORES.

Sobrecargar operadores



Aritmética con números complejos

Aritmética con números complejos

 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;</pre>
  z4.display();
  std::cout << std::endl;</pre>
  return 0;
```

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

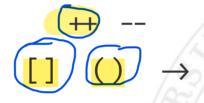


Ejemplo de uso

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

- 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.

Generalizando el método display()

```
class Complejo {
public:
    ...
    void display(ostream & out) const;

private:
    stream que recibo como parametro
}

void Complejo::display(ostream & out) const {
    out << real << ... << "i";
}
</pre>
```

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

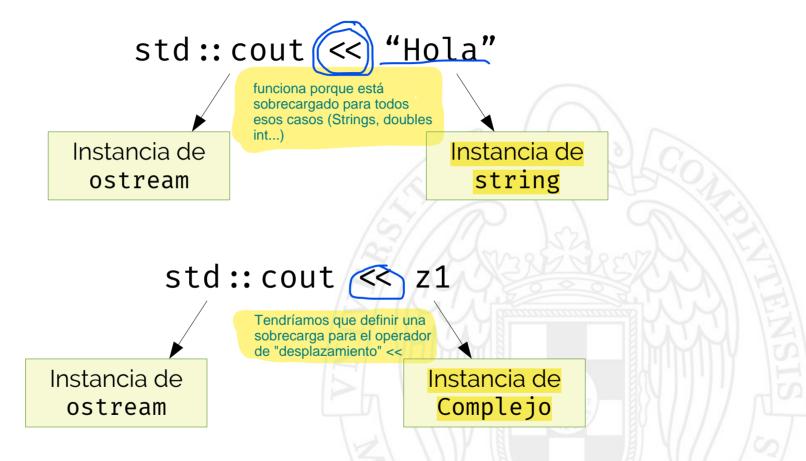
   z3.display(std::cout);
   std::cout << std::endl;

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

return 0;
}</pre>
```



El operador << para E/S



Sobrecargando << para números complejos

```
void operator<<(std::ostream &out, Complejo &z) {</pre>
  z.display(out);
                           ES LO MISMO QUE AMBAS
int main() {
                                    int main() {
                                      std:: cout << z3; notación del operador <<
  z3.display(std::cout);
  std::cout << std::endl;</pre>
                                      std::cout <</pre> std::endl;
  z4.display(std::cout);
                                      std::cout << z4;</pre>
                                      std::cout << std::endl;</pre>
  std::cout << std::endl;</pre>
  return 0;
                                      return 0;
```

Sobrecargando << para números complejos

```
void operator<<(std::ostream &out, Complejo &z) {</pre>
  z.display(out);
int main() {
                                  int main() {
                                    std::cout << z3 << std::endl << z4 << std::endl;</pre>
  z3.display(std::cout);
  std::cout << std::endl;</pre>
                                    return 0;
  z4.display(std::cout);
  std::cout << std::endl;</pre>
                                      NO PODRÍAMOS REUNIRLO ÚNICAMENTE EN UNO
  return 0;
```

Sobrecargando << para números complejos

```
std::ostream & operator<<(std::ostream &out, Complejo &z) {</pre>
  z.display(out);
                      Ahora si podríamos ponerlo todo en una sola línea. Antes era void. Devuelve el mismo ostream que recibe.
  return out;
        std::cout << z3 << std::endl << z4 << std::endl;</pre>
```

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.

```
z1 + z2
equivale a
z1.operator+(z2)
```

¿Podemos hacer lo mismo con...?

Sería this

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



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

Función aislada.

iNo podemos añadir métodos a la clase ostream!