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

```
class Complejo {
public:
  Complejo(double real, double imag);

  double get_real() const;
  double get_imag() const;
  void display() const;

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

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() };
}

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 };
}
```

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++ 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

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

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;
}
```

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)`

¿Qué operadores pueden sobrecargarse?

```
+   -   *   /   %   ^   &   |   <<   >>
==   <=   >=   !=   <   >   &&  ||  !
=  +=  -=  *=  /=
++  --
[]  ()  →
new delete
```

etc.

Sobrecarga del operador << para E/S

Generalizando el método `display()`

```
class Complejo {
public:
  …
  void display() const;

private:
  …
};
```

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

- 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:
  …
};
```

```
void Complejo::display(ostream &out) const {
  out << real << … << "i";
}
```

- 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

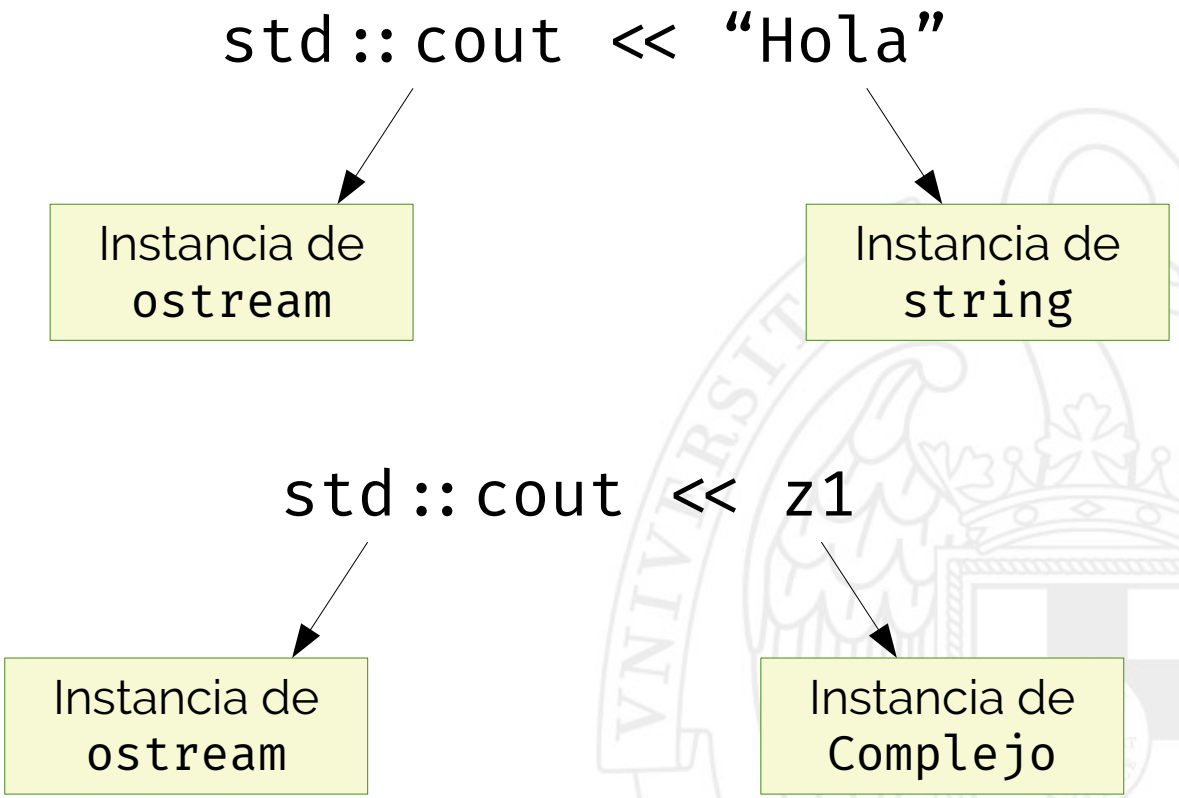
```
int main() {
  Complejo z1(2.0, -3.0), z2(1.0, 0.0);
  Complejo z3 = z1 + z2;
  Complejo z4 = z1 * z1 + z2;

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

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

  return 0;
}
```

El operador << para E/S



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::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;
}
```

Sobrecargando << para números complejos

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

```
std::cout << z3 << std::endl << z4 << 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:

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

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;
};
```

```
z1 + z2
```

equivale a

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

¿Podemos hacer lo mismo con...?

```
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;
}
```

¡No podemos añadir métodos a la clase `ostream`!