# Sobrecarga de Operadores en C++

Facultad de Ingeniería
Universidad de Buenos Aires

# Tipos de Operadores

- Aritméticos
  - +, -, *, /, %, ++, --
- Lógicos
  - &&, ||, !
- Comparación
  - <, >, <=, >=, ==
- Bitwise
  - &, |, <<, >>, ~, ^
- Asignación
  - =, +=, -=, *=, /=, Etc.
- Misceláneos
  - Punteros y referencias: [], *, ->, &
  - Conversión de tipos: (int), (float), Etc. (casteo)
  - Invocación de funciones: ()

# Tipos según argumentos

- Unarios
  - Ej: i++;
- Binarios
  - Ej: i == j
- Ternarios
  - Ej: i < 0 ? 0 : i
- N-Arios
  - No empleados para tipos nativos en C/C++
  - Ej: arreglo[i, j, k, l, m]

# Caso de Estudio - std::string

```
1  std::string var1("Hola");
2  std::string var2 = "Mundo";
3  std::string var3 = var1;
4  var3 += " ";
5  var3 += var2;
6  std::count << var3 << std::endl;
7  std::count << var1 + " " + var3 << std::endl;
8  var3[0] = 'c'; var3[1] = 'h'; var3[2] = 'a'; var3[3] = 'u';
9  std::count << var3 << std::endl;
```

# Sobrecarga de Operadores en C++

- Aplica solamente a operaciones que utilicen clases/structs
- Permite sobrecargar los operadores default de clases/structs
- Utiliza el comportamiento del compilador para convertir "funciones de operador"

**Ej. Unario:**

```
var1.operator=(var2)          //equivale a:var1=var2
```

**Ej. Binario:**

```
var1 = operator+(var2, var3) //equivale a:var1=var2+var3
```

# Caso de Estudio - Clase Complejo

```
1   class Complex {
2     float re;
3     float im;
4   public:
5     Complex(float re, float im) : re(re), im(im) {
6     }
7     Complex(const Complex& other) : re(other.re), im(other.im) {
8     }
9     ... //operators overloading
10    float getRe() const { return re; }
11    float getIm() const { return im; }
12  };
```

# Operadores Predefinidos

- Para toda clase/struct existe un conjunto de operadores con comportamiento predefinido:
  - Operador de asignación (operator=):
    - Copia bit a bit del objeto asignado
  - Operador de referencia (operator&)
    - Retorna un puntero al objeto referenciado
  - Operador de secuencia (operator,)

```
1  Complex var1(1, 2);
2  Complex var2(0,0);
3  var2 = var1;
4  Complex* var3 = &var1;
```

# Ejemplo - Operator==

```
1   class Complex {
2     ...
3     bool operator==(const Complex& other) const {
4       return this->re == other.re && this->im == other.im;
5     }
6   };
7
8   ...
9   Complex var1(1,2);
10  Complex var2(1,3);
11  bool equals = var1 == var2;
```

# Ejemplo - Operator=

```
1   class Complex {
2     ...
3     Complex& operator=(const Complex& other) {
4       if (this != &other) {
5           this->re == other.re; this->im == other.im;
6       }
7       return *this;
8     }
9   };
10  ...
11  Complex var1(1,2);
12  Complex var2(0,0);
13  var2 = var1;
```

# Ejemplo - Operator+

```
1   class Complex {
2     ...
3     Complex operator+(const Complex& other) const {
4       Complex result(this->re + other.re,
5                      this->im + other.im);
6       return result;
7     }
8   };
9   ...
10  Complex var1(1,2);
11  Complex var2(1,3);
12  Complex var3 = var1 + var2;
```

# Ejemplo - Operator+ (modo binario)

```
1   class Complex {
2     ...
3   };
4   Complex operator+(const Complex& a, const Complex& b) {
5     Complex result(a.getRe() + b.getRe(),
6                    a.getIm() + b.getIm());
7     return result;
8   }
9   ...
10  Complex var1(1,2);
11  Complex var2(1,3);
12  Complex var3 = var1 + var2;
```

# Ejemplo - Operator<<

```
1   class Complex {
2     ...
3   };
4
5   std::ostream& operator<<(std::ostream& out, const Complex& a) {
6     out << a.getRe() << " + " << a.getIm() << "i";
7     return out;
8   }
9   ...
10  Complex var1(1,2);
11  std::cout << var1 << std::endl;
```

# Ejemplo - Conversión de Tipos

```
1   class Complex {
2     ...
3     operator float() const {
4       return sqrt(re*re + im*im);
5     }
6   };
7
8   ...
9   Complex var1(2, 0);
10  float var2 = var1;
```

# Ejemplo - Pre y Post incremento

```
1   class Complex {
2     ...
3     Complex& operator++() { //Pre-Incremento
4       ++re;
5       return *this;
6     }
7     Complex operator++(int) { //Post-Incremento
8       Complex copy(*this);
9       ++re;
10      return copy;
11    }
12  };
```

# Functors - Operador de llamada

```
1  class Impresor {
2    void operator()(int value) {
3      std::cout << "Valor: "
4                << value
5                << std::endl;
6    }
7  };
8
9  ...
10 Impresor impresor;
11 int numbers[] = {1, 2, 3, 4};
12 for (int i : numbers)
13   impresor(i);
```

```
1  class Sumador {
2    int suma;
3  public:
4    Sumador(): suma(0) {}
5    void operator()(int value) {
6      suma += value;
7    }
8  };
9  ...
10 Sumador sumador;
11 int numbers[] = {1, 2, 3, 4};
12 for (int i : numbers)
13   sumador(i);
```