

CS1112: Programación II

Unidad 8: Sobrecarga de operadores

Sesión de Laboratorio - 13A

Profesores:

María Hilda Bermejo mbermejo@utec.edu.pe

Estanislao Contreras econtreras@utec.edu.pe

Jorge Villavicencio jvillavicencio@utec.edu.pe

Edson Mendiola emendiola@utec.edu.pe

Ian Paul Brossard ibrossard@utec.edu.pe

Jose Chavez jchaveza@utec.edu.pe

Julio Yarasca jyarascam@utec.edu.pe

Percy Quevedo pquevedo@utec.edu.pe

Wilder Nina wnina@utec.edu.pe

José Fiestas jfiestas@utec.edu.pe

Material elaborado por:

[María Hilda Bermejo](#)



8

Unidad 8: Sobrecarga de Operadores



Logro de la sesión:

Al terminar esta sesión el alumno se familiariza con el concepto de funciones amigas y de sobrecarga de operadores.

Ejemplo 1: Funciones Amigas

```
#include "Tipos.h"
class Rectangulo
{
private:
    TipoEntero m_largo;
    TipoEntero m_ancho;
public:
    Rectangulo() {
        m_largo=10;
        m_ancho=20;
    }
};
```

Rectangulo.h

```
#include <iostream>
#include "Rectangulo.h"
#include "Cuadrado.h"
int main() {
    Rectangulo r;
    Cuadrado c;
    c.mostrar(cout,r);
    return 0;
}
```

Largo : 10
Ancho : 20
Lado : 5

```
using namespace std;
class Cuadrado {
private:
    TipoEntero m_lado;
public:
    Cuadrado() {
        m_lado=5;
```

Cuadrado.h

Es posible acceder al atributo largo?

```
    }
    void mostrar(ostream &os,Rectangulo Rect)
    {
        os<<"\nLargo : "<<Rect.m_largo;
        os<<"\nAncho : "<<Rect.m_ancho;
        os<<"\nLado : "<<m_lado;
    }
};
```

Tipos.h

```
using TipoEntero = int; //desde c++ 11
```

Ejemplo 2: Funciones Amigas -Version2

```
#include "tipos.h"
```

```
class Rectangulo
```

```
{
```

```
private:
```

```
    TipoEntero m_largo;
```

```
    TipoEntero m_ancho;
```

```
public:
```

```
    Rectangulo() {
```

```
        m_largo=10;
```

```
        m_ancho=20;
```

```
    }
```

```
    friend class Cuadrado;
```

```
    //Se declara la clase amiga(friend)
```

```
};
```

Rectangulo.h

Tipos.h

```
using TipoEntero = int; //desde c++ 11
```

```
using namespace std;
```

```
class Cuadrado {
```

```
private:
```

```
    TipoEntero m_lado;
```

```
public:
```

```
    Cuadrado() {
```

```
        m_lado=5;
```

```
    }
```

```
    void mostrar(ostream &os, Rectangulo Rect)
```

```
    {
```

```
        os<<"\nLargo : "<<Rect.m_largo;
```

```
        os<<"\nAncho : "<<Rect.m_ancho;
```

```
        os<<"\nLado : "<<m_lado;
```

```
    }
```

```
};
```

Cuadrado.h

Es posible acceder al atributo largo?

```
#include <iostream>
```

```
#include "Rectangulo.h"
```

```
#include "Cuadrado.h"
```

```
int main() {
```

```
    Rectangulo r;
```

```
    Cuadrado c;
```

```
    c.mostrar(cout, r);
```

```
    return 0;
```

```
}
```

Largo : 10

Ancho : 20

Lado : 5

Sobrecarga de operadores

Ejemplo 3: Sumar 2 clases

Entero.h

```
#ifndef SOBRECARGA_ENTERO_H
#define SOBRECARGA_ENTERO_H
#include <iostream>
using namespace std;

class Entero {
private:
    int dato;
public:
    Entero(){}
    Entero(int _dato) {dato = _dato;}
    int get_dato(){return dato;}
};

#endif //SOBRECARGA_ENTERO_H
```

main.cpp

```
#include <iostream>
#include "Entero.h"
using namespace std;
int main()
{
    Entero a(10);
    Entero b(40);
    Entero c;

    c = a + b;

    return 0;
}
```

No es posible hacer la suma, porque a y b son clases y el operador + solo trabaja con tipos primitivos

Ejemplo 4: operator '+' y '<<'

```
#include <iostream>
#include "Tipos.h"
using namespace std;
class Entero {
private:
    TipoEntero m_dato;
public:
    Entero() { }
    Entero(TipoEntero _dato) { m_dato= _dato; }
    int getData() {return m_dato;}

    friend Entero operator+(Entero& x, Entero& y);
    friend ostream& operator<<(ostream& o, Entero e);
};
```

Entero.h

```
#include "Entero.h"

Entero operator+(Entero &x, Entero &y) {
    return Entero(x.m_dato + y.m_dato);
}

ostream& operator<<(ostream &o, Entero e) {
    o << e.m_dato + 5;
    return o;
}
```

Entero.cpp

```
#include <iostream>
#include "Entero.h"
using namespace std;
int main() {
    Entero a(10);
    Entero b(40);
    Entero c;
    c = a + b;
    cout << "c = " << c.getData() << "\n";
    cout << "c = " << c << "\n";

    return 0;
}
```

main.cpp

Salida:

c = 50
c = 55

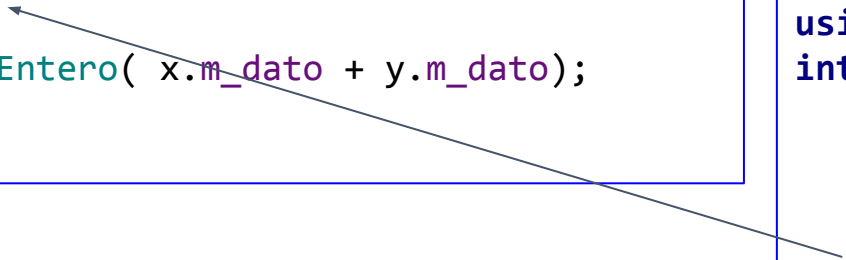
Tipos.h

```
using TipoEntero = int; //desde c++ 11
```


Ejemplo 5: Sobrecarga con función

```
#include "Entero.h"
Entero suma(Entero &x, Entero &y)
{
    return Entero( x.m_dato + y.m_dato);
}
```

Entero.cpp



```
#include <iostream>
#include "Tipos.h"
using namespace std;
class Entero {
private:
    TipoEntero m_dato;
public:
    Entero() { }
    Entero(TipoEntero dato) { m_dato= dato; }
    int getData() {return m_dato;}
    friend Entero suma(Entero& x, Entero& y);
};
```

Entero.h

```
#include <iostream>
#include "Entero.h"
using namespace std;
int main() {
    Entero a(10);
    Entero b(20);
    Entero c;
    c = suma(a,b);
    cout << "c = " <<c.getData() << "\n";
    return 0;
}
```

main.cpp

Tipos.h

```
using TipoEntero = int; //desde c++ 11
```

Ejemplo 6: Sobrecarga con función

Entero.h

```
#include <iostream>
using namespace std;
typedef int TipoEntero ;
class Entero {
private:
    TipoEntero dato;
public:
    Entero() { }
    Entero(TipoEntero _dato) { dato= _dato; }
    int getDato() {return dato;}
    friend Entero suma(Entero& x, Entero& y);
};
```

```
#include "Entero.h"
Entero suma(Entero &x, Entero &y)
{
    return Entero( x.dato + y.dato);
}
```

Entero.cpp

main.cpp

```
#include <iostream>
#include "Entero.h"

using namespace std;

int main() {
    Entero a(10);
    Entero b(20);
    Entero c;

    c = suma(a,b);
    cout << "c = " << c.getDato() <<
    "\n";

    return 0;
}
```

Salida:

c = 50

Ejemplo 7: Clase complejo

CComplejo.h

```
#include <iostream>
#include "Tipos.h"
/**
 * Un número complejo.
 * Sobrecarga el operador + para poder sumar otras clases CComplejo, arrays de dos
 * doubles y doubles sueltos.
 */
using namespace std;

class CComplejo {
private:
    /** Parte real del complejo */
    TipoDoble m_x;

    /** Parte imaginaria del complejo */
    TipoDoble m_y;
public:
    /** Constructor por defecto. Rellena los atributos x e y a 0.0 */
    CComplejo ();

    /** Constructor con array de doubles. Mete el primer double del array en
     * x y el segundo en y */
    CComplejo (const TipoDoble origen[]);

    /** Constructor copia. Copia los atributos del CComplejo recibido en los internos
     * de la clase. */
    CComplejo (const CComplejo &origen);
```

Ejemplo 7: Clase complejo

CComplejo.h

```
/** Constructor con un double. Lo pone en la parte real x */
CComplejo (TipoDoble valor);

/** Operador igual para array de doubles. Mete el primer double del array
 * en x y el segundo en y */
CComplejo &operator = (const TipoDoble origen[]);

/** Operador igual para otro CComplejo. Copia los atributos del
 * CComplejo recibido. */
CComplejo &operator = (const CComplejo &origen);

/** Operador suma para un double. Suma el double recibido en x */
CComplejo operator + (TipoDoble sum) const;

/** Operador suma para un array de doubles. Suma el primer double del array en
 * x y el segundo en y */
CComplejo operator + (const TipoDoble sum[]);

/** Operador suma para otro CComplejo. Suma los atributos del CComplejo recibido
 * con los internos */
CComplejo operator + (const CComplejo &sum) const;

/** cast a double. Devuelve el módulo */
operator double ();

/** Devuelve X */
TipoDoble dameX() const;
```

Ejemplo 7: Clase complejo

CComplejo.h

```
/** Devuelve Y */
TipoDoble dameY() const;

/** Devuelve X */
TipoDoble tomaX(TipoDoble valor);

/** Devuelve Y */
TipoDoble tomaY(TipoDoble valor);
};

/*****
 * FUNCIONES EXTERNAS A LA CLASE
 *****/

/** Sobrecarga del operador suma global para sumar un double con un CComplejo */
CComplejo operator + (TipoDoble k, const CComplejo &origen);

/** Sobrecarga del operador suma global para sumar un array de doubles y un CComplejo */
CComplejo operator + (const TipoDoble sum1[], const CComplejo &sum2);

/** Sobrecarga del operador << global para que cout "sepa" escribir un CComplejo */
ostream &operator << (ostream &salida, const CComplejo &origen);

/** Sobrecarga del operador << global para que cout "sepa" escribir un array de doubles */
ostream &operator << (ostream &salida, const TipoDoble origen[] );
```

Ejemplo 7: Clase complejo

CComplejo.cpp

```
#include "Complejo.h"
#include <math.h>

/*****
 * METODOS DE LA CLASE
 *****/

/***** CONSTRUCTORES *****/

/* Constructor por defecto. Pone Los atributos x e y a 0.0 */
CComplejo::CComplejo () {
    m_x=0.0;
    m_y=0.0;
}

/* Constructor por defecto. Pone Los atributos x e y a 0.0 */
CComplejo::CComplejo (TipoDoble valor) {
    m_x=valor;
    m_y=0.0;
}

/* Constructor con un array de doubles. Pone el primer elemento del array en x y el
 * segundo en y */
CComplejo::CComplejo (const TipoDoble origen[]) {
    *this = origen;
}
```

Ejemplo 7: Clase complejo

CComplejo.cpp

```
/** Constructor copia. Copia los atributos de origen en los internos de la clase */
CComplejo::CComplejo (const CComplejo &origen) {
    m_x = origen.m_x;
    m_y = origen.m_y;
}

/***** OPERADORES SUMA EN LA CLASE *****/

/** operador suma para un double. Lo suma a la parte real y devuelve CComplejo para
 * poder concatenar sumas */
CComplejo CComplejo::operator + (TipoDoble sum) const {
    CComplejo aux(*this);
    aux.m_x = m_x + sum;
    return aux;
}

/** operador suma para array de doubles. Devuelve CComplejo para poder concatenar sumas */
CComplejo CComplejo::operator + (const TipoDoble sum[]) {
    CComplejo aux;
    aux.m_x = m_x + sum[0];
    aux.m_y = m_y + sum[1];
    return aux;
}
```

Ejemplo 7: Clase complejo

CComplejo.cpp

```
/** operador suma para otro CComplejo. Devuelve CComplejo para poder concatenar sumas */
CComplejo CComplejo::operator + (const CComplejo &sum) const {
    CComplejo aux;
    aux.m_x = m_x + sum.m_x;
    aux.m_y = m_y + sum.m_y;
    return aux;
}

/***** OPERADORES DE ASIGNACION EN LA CLASE *****/

/** operador = para arrays de doubles. Devuelve CComplejo para poder concatenar
 * operaciones de = estilo a = b = c; */
CComplejo &CComplejo::operator = (const TipoDoble origen[]) {
    m_x = origen[0];
    m_y = origen[1];
    return *this;
}

/** operador = para otro CComplejo. Devuelve CComplejo para poder concatenar
 * operaciones de =, estilo a = b = c; */
CComplejo &CComplejo::operator = (const CComplejo &origen) {
    m_x = origen.m_x;
    m_y = origen.m_y;
    return *this;
}
```


Ejemplo 7: Clase complejo

CComplejo.cpp

```
/* ***** OPERADORES DE CAST EN LA CLASE ***** */

/** operador cast a double. Devuelve el módulo del complejo */
CComplejo::operator TipoDoble () {
    return sqrt (m_x*m_x + m_y*m_y);
}

/* ***** OTROS MÉTODOS DE LA CLASE ***** */

/** Devuelve X */
TipoDoble CComplejo::dameX () const {
    return m_x;
}

/** Devuelve Y */
TipoDoble CComplejo::dameY () const {
    return m_y;
}

/** Asigna X */
TipoDoble CComplejo::tomaX (TipoDoble valor) {
    m_x = valor;
}

/** Asigna Y */
TipoDoble CComplejo::tomaY (TipoDoble valor) {
    m_y = valor;
}
```

Ejemplo 7: Clase complejo

CComplejo.cpp

```
/* *****  
 * FUNCIONES EXTERNAS A LA CLASE  
 ***** */  
  
/* ***** OPERADOR << PARA EL COUT ***** */  
  
/* operador global << para escribir CComplejo en pantalla. */  
ostream &operator << (ostream &salida, const CComplejo &origen) {  
    // Se pone signo positivo por defecto  
    TipoChar signo='+';  
  
    // Si la y es negativa, no se pone signo, ya que al escribir la y ya sale un signo  
    // negativo.  
    if (origen.dameY() < 0.0)  
        signo = 0;  
  
    // Se escriben los campos separados por el signo  
    cout << origen.dameX() << signo << origen.dameY() << "j";  
}
```

Ejemplo 7: Clase complejo

CComplejo.cpp

```
/* operador global << para escribir array de doubles en pantalla. */
ostream &operator << (ostream &salida, const TipoDoble origen[]) {
    // Se pone signo positivo por defecto
    TipoChar signo='+';
    // Si la y es negativa, no se pone signo, ya que al escribir la y ya sale un signo
    // negativo.
    if (origen[1] < 0.0)
        signo = 0;
    // Se escriben los campos separados por el signo
    cout << origen[0] << signo << origen[1] << "j";
}

/***** OPERADORES SUMA EXTERNOS *****/
/* operador global + para sumar array de doubles con CComplejo.
 * Devuelve un CComplejo para poder encadenar sumas a+b+c+d */
CComplejo operator + (const TipoDoble sum1[], const CComplejo &sum2) {
    CComplejo aux;
    aux = sum2 + sum1;
    return aux;
}

/* operador global + para sumar un double con un complejo. Devuelve CComplejo
 * para poder encadenar sumas */
CComplejo operator + (TipoDoble sum, const CComplejo &origen) {
    CComplejo aux;
    aux = origen + sum;
    return aux;
}
```

Ejemplo 7: Clase complejo

Tipos.h

```
using TipoDoble = double;  
using TipoChar = char;
```

main.cpp

```
#include <iostream>  
#include "Complejo.h"  
using namespace std;  
  
int main() {  
    TipoDoble c1[] = {1.0, -1.0}; // Un array de doubles.  
    TipoDoble c2[] = {-1.0, 1.0}; // Otro array de doubles.  
    CComplejo complejoA(c1) ;    // complejoA, copia del array c1.  
    CComplejo complejoB;        // complejoB, por defecto.  
  
    // Se realizan varios tipos de sumas, escribiendo en pantalla los resultados.  
  
    // ComplejoC operator + (double sum);  
    // 1-j + 1 da 2-j  
    cout << complejoA << " + " << 1 << " = ";  
    cout << complejoA + 1 << endl;  
  
    // ComplejoC operator + (double sum[]);  
    // 1-j + 1-j da 2-2j  
    cout << complejoA << " + " << c1 << " = ";  
    cout << complejoA + c1 << endl;
```

Ejemplo 7: Clase complejo

main.cpp

```
// ComplejoC operator + (ComplejoC sum);  
// 1-j + 0+0j da 1-j  
cout << complejoA << " + " << complejoB << " = ";  
cout << complejoA + complejoB << endl;  
  
// ComplejoC operator + (double k, ComplejoC &origen);  
// 1 + 1-j + 1-j + -1+j + 0+0j da 2-j  
cout << 1 << " + " << complejoA << " + " << c1 << " + " << c2 << " + " << complejoB \\  
    << " = ";  
cout << 1 + complejoA + c1 + c2 + complejoB << endl;  
  
// ComplejoC operator + (double sum1[], ComplejoC &sum2);  
// -1+j + 1-j da 0+0j  
cout << c2 << " + " << complejoA << " = ";  
cout << c2 + complejoA << endl;  
  
// ComplejoC operator double  
// El módulo de 1-j es 1.4142 (raiz de 2)  
cout << "|" << complejoA << "| = ";  
cout << (TipoDoble)complejoA << endl;  
  
// cast de double a ComplejoC  
// Debe dar 2.3+0j  
cout << (CComplejo)2.3 << endl;  
  
return 0;  
}
```

Ejemplo 8: producto por un escalar

CVector.h

```
using namespace std;

class Vector {
public: int m_x, m_y;
    Vector& operator= (const Vector& v) { // asignación V = V
        m_x = v.m_x;
        m_y = v.m_y;
        return *this;
    }

    Vector operator* (int i) {          // Producto V * int
        Vector vr;
        vr.m_x = m_x * i;   vr.m_y = m_y * i;
        return vr;
    }

    void showV();
    friend Vector operator* (int, Vector); // Producto int * V
};

void Vector::showV() {
    cout << "X = " << m_x << "; Y = " << m_y << endl;
}
```

Ejemplo 8: producto por un escalar

CMatrizVector.h

```
#include "CVector.h"
using namespace std;

class mVector {    // definición de clase mVector
    int m_dimension;
public:
    Vector* mVptr;
    mVector(int n = 1) {           // constructor por defecto
        m_dimension = n;
        mVptr = new Vector[m_dimension];
    }
    ~mVector() {
        delete [] mVptr;
    }
    Vector& operator[](int i) { return mVptr[i]; }
    void showmem (int);
};

void mVector::showmem (int i) {
    if((i >= 0) && (i <= m_dimension)) mVptr[i].showV();
    else cout << "Argumento incorrecto! pruebe otra vez" << endl;
}

Vector operator* (int i, Vector v) {
    Vector vr;
    vr.m_x = v.m_x * i;    vr.m_y = v.m_y * i;
    return vr;
}
```

Ejemplo 8: producto por un escalar

```
#include <iostream>
#include "CVector.h"
#include "CMatrizVector.h"
```

```
int main() {
    mVector mV1(5);
    mV1[0].m_x = 2;  mV1[0].m_y = 3;
    mV1.showmem(0);

    Vector v1;
    v1 = mV1[0];
    v1.showV();
    mV1[4] = mV1[0] * 5;
    mV1.showmem(4);

    mV1[2] = 5 * mV1[0];
    mV1.showmem(2);

    return 0;
}
```

main.cpp

Resultado

X = 2; Y = 3

X = 2; Y = 3

X = 10; Y = 15

X = 10; Y = 15

Ejemplo 9

```
class Vector {  
public:  
    float x, y;  
    void operator()() { // función-operador  
        cout << "Vector: (" << x << ", " << y << ") " << endl;  
    }  
};
```

Vector.h

```
#include <iostream>  
using namespace std;  
#include "CVector.h"
```

```
int main () {  
    Vector v1 = {2, 3};  
    v1(); // Ok. invocación de v1.operator()  
    v1.operator()(); // Ok. invocación clásica  
    return 0;  
}
```

main.cpp

Salida:

Vector: (2, 3)

Vector: (2, 3)

Ejemplo 10

```
class Vector {  
public: float x, y;  
    void operator()() {           // Versión-1  
        cout << "Vector: (" << x << ", " << y << ") " << endl;  
    }  
    void operator()(int i) {      // Versión-2  
        cout << "Coordenadas: (" << x << ", " << y << ") " << endl;  
    }  
};
```

Vector.h

El argumento (int) de la segunda definición se ha utilizado exclusivamente para permitir al compilador distinguir entre ambas.

```
#include <iostream>  
using namespace std;  
#include "CVector.h"
```

main.cpp

```
int main () {  
    Vector v1 = {2, 3};  
    v1();           // invocación de v1.operator()  
    v1.operator()(); // invocación clásica  
    return 0;  
}
```

Salida:

Vector: (2, 3)
Coordenadas: (2, 3)

Resumen:

- Funciones amigas
- Sobrecarga de operadores

¡Nos vemos en la
siguiente clase!

