CS1112: Programación II

Unidad 8: Sobrecarga de operadores

Sesión de Laboratorio - 13B

Profesores:

María Hilda Bermejo mbermejo@utec.edu.pe
Estanislao Contreras econtreras@utec.edu.pe
Jorge Villavicencio jvillavicencio@utec.edu.pe
Edson Mendiolaza emendiolaza@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:
Maria Hilda Bermejo

Índice:

- Unidad 8: Sobrecarga de operadores
  - Casos particulares de sobrecarga

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

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.

En el ejemplo se sobrecargan operadores para realizar operaciones con conjuntos, tales como:

Unión

Diferencia

Intersección

con estos métodos, es posible realizar operaciones con conjuntos.

```
Conjunto a:  [10, 15, 20, 30, 40]
Conjunto b:  [3, 10, 25, 40]
Conjunto c:  [4, 5, 10, 60]

Conjunto b + a:   [3, 10, 15, 20, 25, 30, 40]
```

Conjunto a: [10, 15, 20, 30, 40]
Conjunto b: [3, 10, 25, 40]
Conjunto c: [4, 5, 10, 60]

Conjunto b + a: [3, 10, 15, 20, 25, 30, 40]

Conjunto (a - b) + (a * b) + (b - a): [3, 10, 15, 20, 25, 30, 40]

Diferencia simétrica: [3, 15, 20, 25, 30]

Ley distributiva: [10, 40] = [10, 40]

Ejemplo 1: Conjunto

main.cpp

```
#include <iostream>
#include "Conjunto.h"
using namespace std;
int main()
{  Conjunto a;
   a.agregar_elemento(30);
   a.agregar_elemento(10);
   a.agregar_elemento(40);
   a.agregar_elemento(20);
   a.agregar_elemento(15);
   // Usando el operador << sobrecargado
   Conjunto b;
   b << 3;
   b << 25;
   b << 10;
   b << 40;
   Conjunto c;
   c << 4;
   c << 5;
   c << 10;
   c << 60;
```

```
    cout << "Conjunto a: " << a << "\n";
    cout << "Conjunto b: " << b << "\n";
    cout << "Conjunto c: " << c << "\n";

    auto u1 = b + a;
    cout << "Conjunto b + a: " << u1 << "\n";

    auto u2 = (a - b) + (a * b) + (b - a);
    cout << "Conjunto (a - b) + (a * b) + (b - a): " << u2 << "\n";

    // Diferencia simetrica
    auto ds = (a + b) - (a * b);
    cout << "Diferencia simetrica: " << ds << std::endl;

    // Ley distributiva
    auto c1 = a * (b + c);
    auto c2 = a * b + a * c;
    cout << "Ley distributiva: " << c1 << " = " << c2 << "\n";

    return 0;
}
```

Util.h

```
#ifndef CONJUNTO_UTIL_H
#define CONJUNTO_UTIL_H

using TypeElemento = int;

#endif //CONJUNTO_UTIL_H
```

Conjunto.h

```h
#ifndef CONJUNTO_CONJUNTO_H
#define CONJUNTO_CONJUNTO_H

#include <iostream>
#include <vector>
#include "Util.h"
using namespace std;

class Conjunto
{
   vector<TypeElemento> elementos;
   friend ostream& operator<<(ostream& output, Conjunto& c);   //--- Para qué?
public:
   // Agregar Elementos
   void agregar_elemento(const TypeElemento& value);
   void operator<<(const TypeElemento& value);

   // Operaciones
   Conjunto operator+(const Conjunto& other); // Union        u1 = b + a;
   Conjunto operator-(const Conjunto& other); // Diferencia   u2 = b - a;
   Conjunto operator*(const Conjunto& other); // Interseccion u3 = b * a;
};
#endif //CONJUNTO_CONJUNTO_H
```

Conjunto.cpp

20

[3, 10, 15, 25, 30, 40]

[3, 10, 15, 20, 25, 30, 40]

```
#include <algorithm>
#include "Conjunto.h"

void Conjunto::agregar_elemento(const TypeElemento &value)
{//-------------------------------------------------------
   auto it = lower_bound(begin(elementos), end(elementos), value);
   elementos.insert(it, value);
}

void Conjunto::operator<<(const TypeElemento &value)
{//-------------------------------------------------------
   auto it = lower_bound(begin(elementos), end(elementos), value);
   elementos.insert(it, value);
}
```

```
Conjunto Conjunto::operator+(const Conjunto &other)
{//-------------------------------------------------
   vector<TypeElemento> vector_union {elementos};

   for (auto e: other.elementos) {
       auto it = lower_bound(begin(vector_union), end(vector_union), e);

       if (*it != e)
           vector_union.insert(it, e);
   }

   Conjunto cu;
   for (auto e: vector_union) {
       cu << e;
   }

   return cu;
}
```

elementos : [10, 15, 20, 30, 40]

other : [3, 10, 25, 40]

vector_union : [10, 15, 20, 30, 40]

vector_union : [3,10, 15, 20,25, 30, 40]

```
Conjunto Conjunto::operator-(const Conjunto &other)
{//-----------------------------------------------
   vector<TypeElemento> vector_diferencia;

   for (auto e: elementos) {
       auto it = find(begin(other.elementos), end(other.elementos), e);
       if (it == end(other.elementos))
           vector_diferencia.push_back(e);
   }

   Conjunto cd;
   for (auto e: vector_diferencia) {
       cd << e;
   }
   return cd;
}
```

elementos : [10, 15, 20, 30, 40]

other : [3, 10, 25, 40]

vector_diferencia : [ ]

vector_diferencia : [15, 20, 30, 40]

```
Conjunto Conjunto::operator*(const Conjunto &other)
{//------------------------------------------------
   vector<TypeElemento> vector_interseccion;

   for (auto e: elementos) {
       auto it = find(begin(other.elementos), end(other.elementos), e);
       if (it != end(other.elementos))
           vector_interseccion.push_back(e);
   }

   Conjunto ci;
   for (auto e: vector_interseccion) {
       ci << e;
   }

   return ci;
}
```

elementos : [10, 15, 20, 30, 40]

other : [3, 10, 25, 40]

vector_interseccion: [ ]

vector_interseccion : [10, 40]

```
ostream& operator<<(ostream& output, Conjunto& c)
{//------Para imprimir el conjunto-------------
    output << "[";
    auto it = begin(c.elementos);
    while (it != end(c.elementos)-1) {
        output << *it << ", ";
        ++it;
    }
    output << *it << "]";
    return output;
}
```

[10, 15, 20, 30, 40]

Resumen:

- Sobrecarga de operadores

¡Nos vemos en la siguiente clase!

UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA