Apêndice
para
Programação Genérica

Levando a Abstração ao Limite

Fabio Galuppo, M.Sc.

http://fabiogaluppo.com e http://simplycpp.com/

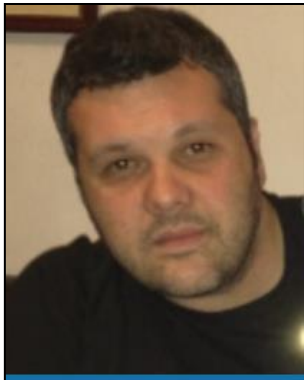
fabiogaluppo@acm.org

@FabioGaluppo

Microsoft MVP Visual Studio and Development Technologies

https://mvp.microsoft.com/en-us/PublicProfile/9529

http://bit.ly/prog_gen_qconsp_2016



Award Categories
Visual Studio and Development Technologies

First year awarded:
2002

Number of MVP Awards:
13

Palestra: Programação Genérica aplicada: levando a abstração ao limite

Track:
Desenvolvimento Poliglota: Funcionais, Dinâmicas e além

Sala: **Galeria**

Dia da semana: **Segunda feira**

6:15pm - 7:05pm

Elementos da Programação Genérica são conhecidos pela maioria de nós – afinal, técnicas como generics (Java, C#, F#) ou templates (C++, D) já estão presentes em diversas linguagens de programação modernas. Porém a Programação Genérica é muito mais do que isso: é um paradigma que permite construir algoritmos e estruturas de dados reutilizáveis e eficientes.

Nessa palestra iremos mostrar esses benefícios na prática. Serão abordados os seguintes tópicos:

- O que é mesmo a Programação Genérica
- Onde os princípios da Orientação a Objetos falham
- Algoritmos genéricos para resolução de problemas reais
- A relação com a Programação Funcional
- Standard Template Library (STL)
- Como reutilização e eficiência decorrem da Programação Genérica

Nessa palestra serão apresentadas técnicas e algoritmos da biblioteca padrão do C++, bem como exemplos em várias outras linguagens, incluindo C#, Java, Haskell e F#.

Slides da Palestra: http://bit.ly/prog_gen_qconsp_2016

http://qconsp.com/presentation/programacao-generica-aplicada-levando-abstracao-ao-limite

ZERO-COST ABSTRACTIONS

Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

não otimizado

```
        my_swap(a, b);
00007FF7B6992415  lea         rdx,[b]
00007FF7B6992419  lea         rcx,[a]
00007FF7B699241D  call        my_swap<int> (07FF7B69911BDh)
```

otimizado

```
    my_swap(a, b);
    std::cout << "after :" << a << " " << b << "\n";
00007FF6632D1051  mov         rcx,qword ptr [__imp_std:
00007FF6632D1058  lea         rdx,[string "after :" (07
00007FF6632D105F  call        std::operator<<<std::char
00007FF6632D1064  mov         rcx,rax
00007FF6632D1067  mov         edx,12Bh
00007FF6632D106C  call        qword ptr [__imp_std::bas
00007FF6632D1072  mov         rcx,rax
00007FF6632D1075  lea         rdx,[string " " (07FF6632
00007FF6632D107C  call        std::operator<<<std::char
00007FF6632D1081  mov         rcx,rax
00007FF6632D1084  mov         edx,0C7h
```

→ **b com valor 299**

→ **b com valor 199**

Microsoft (R) C/C++ Optimizing Compiler Version 19.00.23506 for x64

```
int a, b;
std::cout << "input a:"; std::cin >> a;
std::cout << "input b:"; std::cin >> b;
std::cout << "before:" << a << " " << b << "\n";
my_swap(a, b);
std::cout << "after :" << a << " " << b << "\n";
```

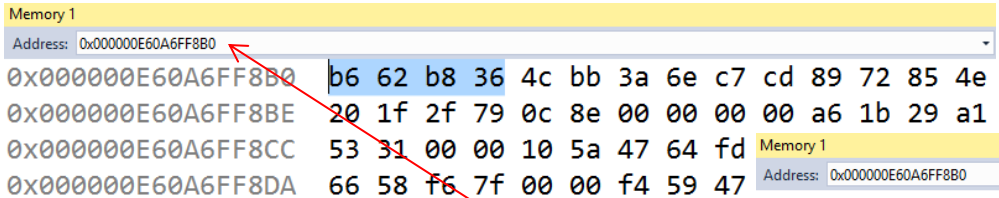
otimizado

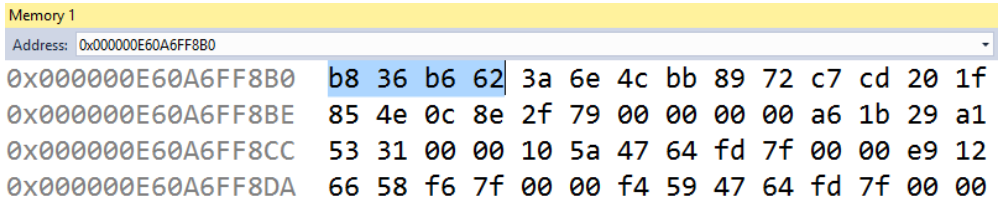
```
    my_swap(a, b);
00007FF7FB76109B  mov         eax,dword ptr [b]
    std::cout << "after :" << a << " " << b << "\n";
00007FF7FB76109F  lea         rdx,[string "after :" (07FF7
    my_swap(a, b);
00007FF7FB7610A6  mov         ebx,dword ptr [a]
    std::cout << "after :" << a << " " << b << "\n";
00007FF7FB7610AA  mov         rcx,qword ptr [__imp_std::co
    my_swap(a, b);
00007FF7FB7610B1  mov         dword ptr [a],eax
00007FF7FB7610B5  mov         dword ptr [b],ebx
    std::cout << "after :" << a << " " << b << "\n";
00007FF7FB7610B9  call        std::operator<<std::char_tr
```

Microsoft (R) C/C++ Optimizing Compiler Version 19.00.23506 for x64

before:25270 14008 47948 28218 52679 29321 20101 7968 31023 36364

62B6 36B8





after :14008 25270 28218 47948 29321 52679 7968 20101 36364 31023

otimizado

```
    for (size_t i = 1; i < xs.size(); i += 2)
00007FF661E111CD  mov         edx,1
00007FF661E111D2  nop         dword ptr [rax]
00007FF661E111D6  nop         word ptr [rax+rax]
        my_swap(xs[i - 1], xs[i]);
00007FF661E111E0  movzx       ecx,word ptr [rsp+rdx*2+6Eh]
00007FF661E111E5  movzx       eax,word ptr xs[rdx*2]
00007FF661E111EA  mov         word ptr [rsp+rdx*2+6Eh],ax
00007FF661E111EF  mov         word ptr xs[rdx*2],cx
    for (size_t i = 1; i < xs.size(); i += 2)
00007FF661E111F4  add         rdx,2
00007FF661E111F8  cmp         rdx,0Ah
00007FF661E111FC  jb          swap_from_array+0E0h (07FF661E111E0h)
```

my_swap

Microsoft (R) C/C++ Optimizing Compiler Version 19.00.23506 for x64

```cpp
template<RandomAccessIterator T>
/*
RandomAccessIterator required to [], copy, assign
*/
void my_swap_adjacent_pairs(T first, size_t N)
{
    for (size_t i = 1; i < N; i += 2)
        my_swap(first[i - 1], first[i]);
}
```

```cpp
void swap_from_array_no_raw_loops()
{
    randomizer<T> rnd;
    std::array<T, N> xs;

    //http://simplycpp.com/2015/11/06/mestre-iota/
    //fill with random numbers
    std::iota(xs.begin(), xs.end(), iota_random<T>(rnd));

    //print before
    std::cout << "before:";
    std::for_each(xs.begin(), xs.end(), [](const T& x) { std::cout << x << " "; });
    std::cout << "\n";

    //swap adjacent pairs
    my_swap_adjacent_pairs(xs.begin(), N);

    //print after
    std::cout << "after: ";
    std::for_each(xs.begin(), xs.end(), [](const T& x) { std::cout << x << " "; });
    std::cout << "\n";
}
```

No contexto da chamada da função `swap_from_array_no_raw_loops`, o corpo da função utilizada encontra-se *inline*

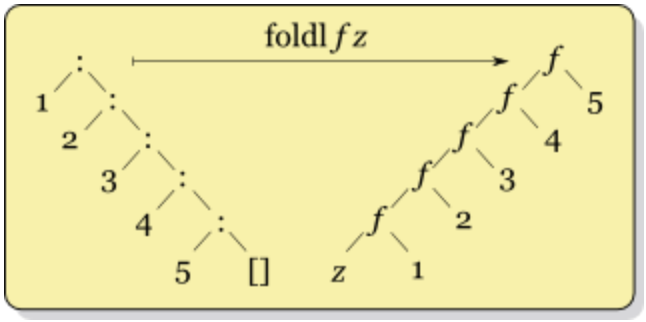
otimizado

```
    my_swap_adjacent_pairs(xs.begin(), N);
00007FF74B7618CC  mov         edx,1
00007FF74B7618D1  movzx       ecx,word ptr [rsp+rdx*2+36h]
00007FF74B7618D6  movzx       eax,word ptr xs[rdx*2]
00007FF74B7618DB  mov         word ptr [rsp+rdx*2+36h],ax
00007FF74B7618E0  mov         word ptr xs[rdx*2],cx
00007FF74B7618E5  add         rdx,2
00007FF74B7618E9  cmp         rdx,rsi
00007FF74B7618EC  jb          swap_from_array_no_raw_loops<unsigned short,10>+121h (07FF74B7618D1h)
```

Microsoft (R) C/C++ Optimizing Compiler Version 19.00.23506 for x64

ABSTRACTING FROM CONCRETE

```
template<typename InputIterator, typename T, typename Function>
T fold(InputIterator first, InputIterator last, T init, Function f)
{
        T acc{ init };
        while (first != last)
        {
                acc = f(acc, *first);
                ++first;
        }
        return acc;
}
```



…f(f(f(f(f(acc, 1),2),3),4),5)…,

sum

```
unsigned long initial = 0;
unsigned long total = std::accumulate(xs.begin(), xs.end(), initial); //implict operator+
```

append

```
List<char> ys{ 'a', 'b', 'c' }, zs{ 'd' };
zs = std::accumulate(ys.rbegin(), ys.rend(), std::move(zs),
    [](List<char>& acc, char x) -> List<char>& {
        return my_push_front(acc, x);
    });
```

hashing

```
std::string plain_data = "Simply C++";
size_t seed = 0;
size_t mask = 0x7FFFFFFFFFFFFFFF;
size_t N = 100; //some map to vector/array of size N
size_t plain_data_hash = std::accumulate(std::begin(plain_data), std::end(plain_data), seed,
                                         [](size_t hash, char c) { return (19 * hash + c) - hash; });
std::cout << "'" << plain_data << "' hash is " << plain_data_hash
          << " send to slot #" << ((plain_data_hash & mask) % N) << "\n";
//'Simply C++' hash is 17691675298189 send to slot #89
```

http://simplycpp.com/2015/12/08/por-quem-os-ponteiros-dobram-estrelando-stdaccumulate/

Strict Totally and Strict Weak

ORDERING

Fabio Galuppo - http://member.acm.org/~fabiogaluppo - fabiogaluppo@acm.org

```cpp
template<typename T>
//T - Strict Totally Ordering
inline const T& my_min(const T& a, const T& b)
{
    if (b < a) return b;
    return a;
}

template<typename T, typename R>
//T - Any Type
//R - Strict Weak Ordering on T
inline const T& my_min(const T& a, const T& b, R relation)
{
    if (relation(b, a)) return b;
    return a;
}
```

```cpp
template<typename T>
//T - Strict Totally Ordering
inline void stable_sort_2(T& a, T& b)
{
    if (b < a) my_swap(a, b);

    //post-condition:
    assert(a == my_min(a, b)); //equality and min
}
```

https://github.com/fabiogaluppo/samples/blob/master/events/qconsp2016/code/cpp/ordering_sample.hpp

https://en.wikipedia.org/wiki/Group_(mathematics)

GROUP

```cpp
template <AdditiveGroup T>
struct Matrix2x2 final
{
    //default constructor
    Matrix2x2() :
        Matrix2x2(T(0)) {}

    //constructor
    Matrix2x2(T a) : //implicit
        Matrix2x2(a, a, a, a) {}

    //constructor
    Matrix2x2(T a1, T a2, T a3, T a4) :
        a1(a1), a2(a2), a3(a3), a4(a4) {}

    //copy constructor
    Matrix2x2(const Matrix2x2<T>& that) :
        a1(that.a1), a2(that.a2), a3(that.a3), a4(that.a4) {}

    //copy assignment
    Matrix2x2<T>& operator=(const Matrix2x2<T>& that)
    {
        a1 = that.a1; a2 = that.a2;
        a3 = that.a3; a4 = that.a4;
        return this;
    }
```

```cpp
    //equality
    bool operator==(const Matrix2x2<T>& that) const
    {
        if (this == &that)
            return true;

        return a1 == that.a1 && a2 == that.a2 &&
            a3 == that.a3 && a4 == that.a4;
    }
```

BINARY ASSOCIATIVE OPERATION

```cpp
    friend Matrix2x2<T> operator+(const Matrix2x2<T>& lhs, const Matrix2x2<T>& rhs)
    {
        return Matrix2x2<T>(lhs.a1 + rhs.a1, lhs.a2 + rhs.a2,
            lhs.a3 + rhs.a3, lhs.a4 + rhs.a4);
    }
};
```

GROUP AXIOMS:

- CLOSURE
- ASSOCIATIVITY
- IDENTITY
- INVERTIBILITY

```cpp
template <AdditiveGroup T>
void test_requirements(const T& x)
{
    T neutral_element { 0 };
    T x_inverse = -x;
    cout << x << " + " << x_inverse << " == " << neutral_element << "?\n";
    cout << boolalpha << (x + x_inverse == neutral_element) << "\n";
}
```

```cpp
template <AdditiveGroup T>
void test_associativity(const T& x, const T& y, const T& z)
{
    cout << "(" << x << " + " << y << ") + " << z << " == ";
    cout << x << " + (" << y << " + " << z << ")" << "?\n";
    cout << boolalpha << ((x + y) + z == x + (y + z)) << "\n";
}
```

```cpp
test_requirements(float(12.34)); //Additive Group of Reals
test_requirements(std::complex<double>(3.0, 8.0) /* 3.0 + 8.0i */); //Additive Group of Complex
test_requirements(Matrix2x2<short>(1, 2, 3, 4)); //Additive Group of Matrices

test_associativity(float(12.3), float(45.6), float(78.9));
test_associativity(std::complex<double>(1.0, 2.0), std::complex<double>(3.0, 4.0), std::complex<double>(5.0, 6.0));
test_associativity(Matrix2x2<short>(10, 20, 30, 40), Matrix2x2<short>(11, 22, 33, 44), Matrix2x2<short>(6, 7, 8, 9));
```

ASSOCIATIVITY

```cpp
template<RandomAccessIterator I, SemigroupOperation F>
typename I::value_type divide_and_conquer_scan(I first, I last, F op)
{
    static size_t THRESHOLD = 2; //for demonstration purpose

    size_t n = std::distance(first, last);
    assert(n > 0);

    if (n >= THRESHOLD)
    {
        n /= 2; //divide into halves
        auto a = divide_and_conquer_scan(first, first + n, op);
        auto b = divide_and_conquer_scan(first + n, last, op);
        return op(a, b); //combine results
    }

    return sequential_scan(first, last, op);
}
```

```cpp
int min_elem2 = divide_and_conquer_scan(xs.cbegin(), xs.cend(), [](int a, int b) { return std::min(a, b); });
//((a + b) + (c + d)) + ((d + e) + (f + g)) == ((((((a + b) + c) + d) + e) + f) + g) - associativity
int acc2 = divide_and_conquer_scan(xs.cbegin(), xs.cend(), std::plus<int>());
```

https://github.com/fabiogaluppo/samples/blob/master/events/qconsp2016/code/cpp/associativity_sample.hpp

```
[<CompiledName("MyFoldAssociative")>]
//f : SemigroupOperation
let myfoldAssociative<'T> (f : 'T -> 'T -> 'T) (array:'T[]) =
    checkNonNull "array" array
    checkArrayNonZeroLen "array" array
    let f = OptimizedClosures.FSharpFunc<_,_,_>.Adapt(f)
    let THRESHOLD = 2 //for demonstration purpose
    let rec myfoldAssociativeRec (array2:'T[]) =
        let n = Array.length array2
        if (n >= THRESHOLD) then
            let lhs, rhs = array2 |> Array.splitAt (n / 2)
            //divide
            let a = myfoldAssociativeRec lhs //potential parallelism here (*)
            let b = myfoldAssociativeRec rhs //(*)
            //combine
            f.Invoke(a, b)
        else
            let mutable acc = array2.[0]
            for i = 1 to n - 1 do
                acc <- f.Invoke(acc,array2.[i])
            acc
    myfoldAssociativeRec array
```

```
let xs = [|10; 3; 17; 8; 2; 5; 1; 20; 9|]
printfn "min(%A) = %d" xs (xs |> myfoldAssociative (fun x y -> Math.Min(x, y)))
printfn "sum(%A) = %d" xs (xs |> myfoldAssociative (fun acc x -> acc + x))
```

```
min([|10; 3; 17; 8; 2; 5; 1; 20; 9|]) = 1
sum([|10; 3; 17; 8; 2; 5; 1; 20; 9|]) = 75
```

http://www.fssnip.net/7OM

Copy Constructor
Copy Assignment
Equality (and Inequality)
Destruction
Default Constructor

REGULAR TYPE

```
straight_line ab(2, 4); //constructor
straight_line cd; //default constructor
straight_line ef; //default constructor and
ef = straight_line(1, 6); //copy assignment
straight_line gh(straight_line(0, 2)); //copy constructor
```

Equality/Inequality

```
cout << ab << " == " << gh << "? " << boolalpha << (ab == gh) << "\n";
cout << cd << " == " << ef << "? " << boolalpha << (cd == ef) << "\n";
cout << cd << " != " << ef << "? " << boolalpha << (cd != ef) << "\n";
cout << ab << " <  " << gh << "? " << boolalpha << (ab < gh)  << "\n";
cout << ab << " <= " << gh << "? " << boolalpha << (ab <= gh) << "\n";
cout << gh << " >= " << ab << "? " << boolalpha << (gh >= ab) << "\n";
cout << gh << " >  " << ab << "? " << boolalpha << (gh > ab)  << "\n";
```

Ordering

```
using straight_lines = std::vector<straight_line>;
straight_lines xs { ab, cd, ef, gh };
```

```
auto m = immutable_median_of_3(xs[0], xs[1], xs[2]);

std::sort(xs.begin(), xs.end());

straight_lines::iterator bound = std::partition(xs.begin(), xs.end(),
    [&m](straight_line& line) { return line <= m; });
```

https://github.com/fabiogaluppo/samples/blob/master/events/qconsp2016/code/cpp/regular_type_sample.hpp

```
var ab = new StraightLineSegment1d<int, Int32Trait>(42, 14); //constructor
var cd = new StraightLineSegment1d<int, Int32Trait>(); //default constructor
var ef = new StraightLineSegment1d<int, Int32Trait>(); //default constructor and
ef.Copy(new StraightLineSegment1d<int, Int32Trait>(31, 16)); //copy assignment
var gh = new StraightLineSegment1d<int, Int32Trait>(new StraightLineSegment1d<int, Int32Trait>(20, 0)); //copy constructor
```

```
Console.WriteLine("{0} == {1}? {2}", ab, gh, ab == gh);
Console.WriteLine("{0} == {1}? {2}", cd, ef, cd == ef);
Console.WriteLine("{0} != {1}? {2}", cd, ef, cd != ef);
Console.WriteLine("{0} <  {1}? {2}", ab, gh, ab < gh);
Console.WriteLine("{0} <= {1}? {2}", ab, gh, ab <= gh);
Console.WriteLine("{0} >= {1}? {2}", ab, gh, ab >= gh);
Console.WriteLine("{0} >  {1}? {2}", ab, gh, ab > gh);

var xs = new StraightLineSegment1d<int, Int32Trait>[] { ab, cd, ef, gh };
Array.Sort(xs, Util.StraightLineSegment1dComparison);
```

```
public static int StraightLineSegment1dComparison<TPositiveInteger, TPositiveIntegerTrait>
    (StraightLineSegment1d<TPositiveInteger, TPositiveIntegerTrait> lhs,
     StraightLineSegment1d<TPositiveInteger, TPositiveIntegerTrait> rhs)
    where TPositiveInteger : struct /* PositiveInteger */
    where TPositiveIntegerTrait : IPositiveIntegerTrait<TPositiveInteger>, new()
{
    if (lhs == rhs) return 0;
    if (lhs > rhs) return 1;
    /*if (lhs > rhs)*/
    return -1;
}
```

https://github.com/fabiogaluppo/samples/blob/master/events/qconsp2016/code/csharp/RegularTypeSampleProgram.cs

POLICY-BASED DESIGN

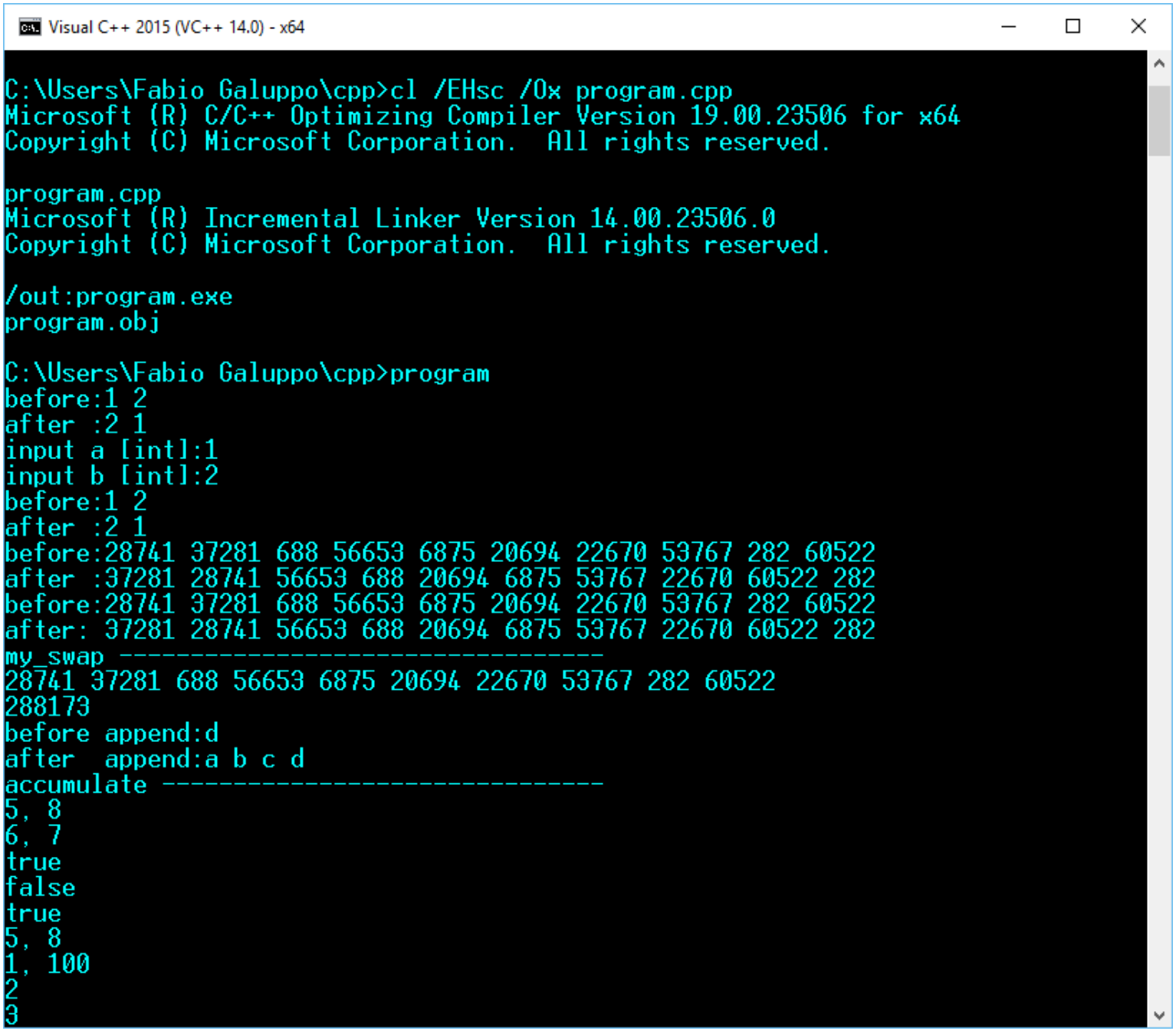
```cpp
template<typename TKey, typename TValue, typename LockPolicy = no_lock_policy>
struct kv_storage
{
    using storage_type = std::unordered_map<TKey, TValue>;

    kv_storage() = default;
    ~kv_storage() = default;
    kv_storage(const kv_storage<TKey, TValue>&) = delete;
    kv_storage<TKey, TValue>& operator=(const kv_storage<TKey, TValue>&) = delete;

    void add(const TKey& key, const TValue& value)
    {
        TKey k = key;
        TValue v = value;
        auto kv = std::make_pair<TKey, TValue>(std::move(k), std::move(v));

        my_lock_guard<LockPolicy> guard(my_lock);
        storage.insert(kv);
    }

    bool has_key(const TKey& key) const { ... }

    bool try_get_value(const TKey& key, TValue& value) const { ... }

private:
    storage_type storage;
    mutable LockPolicy my_lock;
};
```

```cpp
kv_storage<std::string, std::string, mutex_lock_policy> books;
```

https://github.com/fabiogaluppo/samples/blob/master/events/qconsp2016/code/cpp/policy_sample.hpp

https://github.com/fabiogaluppo/samples/tree/master/events/qconsp2016/code

SAMPLES



https://github.com/fabiogaluppo/samples/tree/master/events/qconsp2016/code/cpp

Apêndice
para
Programação Genérica

Levando a Abstração ao Limite

Fabio Galuppo, M.Sc.

http://fabiogaluppo.com e http://simplycpp.com/

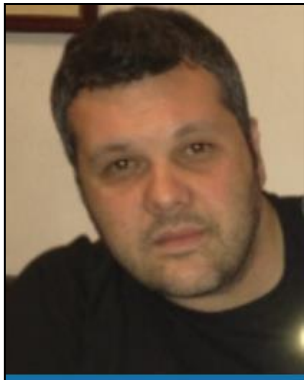
fabiogaluppo@acm.org

@FabioGaluppo

Microsoft MVP Visual Studio and Development Technologies

https://mvp.microsoft.com/en-us/PublicProfile/9529

http://bit.ly/prog_gen_qconsp_2016



Award Categories
Visual Studio and Development Technologies

First year awarded:
2002

Number of MVP Awards:
13