Curso: C++ Moderno
Período: 1º semestre/2017
Aluno: Arthur Nunes de Paiva Santos Queiroz

Lista de exercícios Módulo 05: Semântica de transferência e encaminhamento perfeito

Questões

Questão 1

lmplemente as funções de transferência para a classe BigInt.

BigInt.h

```
#ifndef BIGINT_H
#define BIGINT_H

#include <cstdint>
#include <algorithm>
#include <iostream>

struct BigInt{
    int sig_;
    int len_;
    uint8_t* mag_;

    BigInt(int sig, int len)
        : sig_(sig), len_(len), mag_( len ? new uint8_t[len] : nullptr)
    {
        // std::cout << "BigInt: Construtor" << std::endl;
    }

    BigInt(const BigInt& bi)
        : sig_(bi.sig_), len_(bi.len_), mag_( bi.len_ ? new uint8_t[bi.len_] : nullptr)
    {
        std::copy(bi.mag_, bi.mag_ + len_, mag_);
        // std::cout << "BigInt: Construtor de copia" << std::endl;
    }

    BigInt& operator=(const BigInt& bi){
        sig_ = bi.sig_;
        len_ = bi.len_;
        delete[] mag_;
        mag_ = len_ ? new uint8_t[len_] : nullptr;
```

```
        std::copy(bi.mag_, bi.mag_ + len_, mag_);
        // std::cout << "BigInt: Atribuicao por copia" << std::endl;
        return *this;
    }

    BigInt(BigInt&& bi)
        : sig_(bi.sig_), len_(bi.len_), mag_(nullptr)
    {
        mag_ = bi.mag_;
        bi.len_ = 0;
        bi.mag_ = nullptr;
        // std::cout << "BigInt: Construtor de transferencia" << std::endl;
    }

    BigInt& operator=(BigInt&& bi){
        sig_ = bi.sig_;
        len_ = bi.len_;
        delete[] mag_;
        mag_ = bi.mag_;
        bi.len_ = 0;
        bi.mag_ = nullptr;
        // std::cout << "BigInt: Atribuicao por transferencia" << std::endl;
        return *this;
    }

    ~BigInt(){
        delete[] mag_;
    }
};


#endif // BIGINT_H
```

Questão 2

Refatore o operator= para utilizar o copy-and-swap e compare a performance.

O benchmark foi realizado utilizando a biblioteca Nonius (v1.1.2).

Fonte: http://www.bfilipek.com/2016/01/micro-benchmarking-libraries-for-c.html
https://github.com/libnonius/nonius

main.cpp

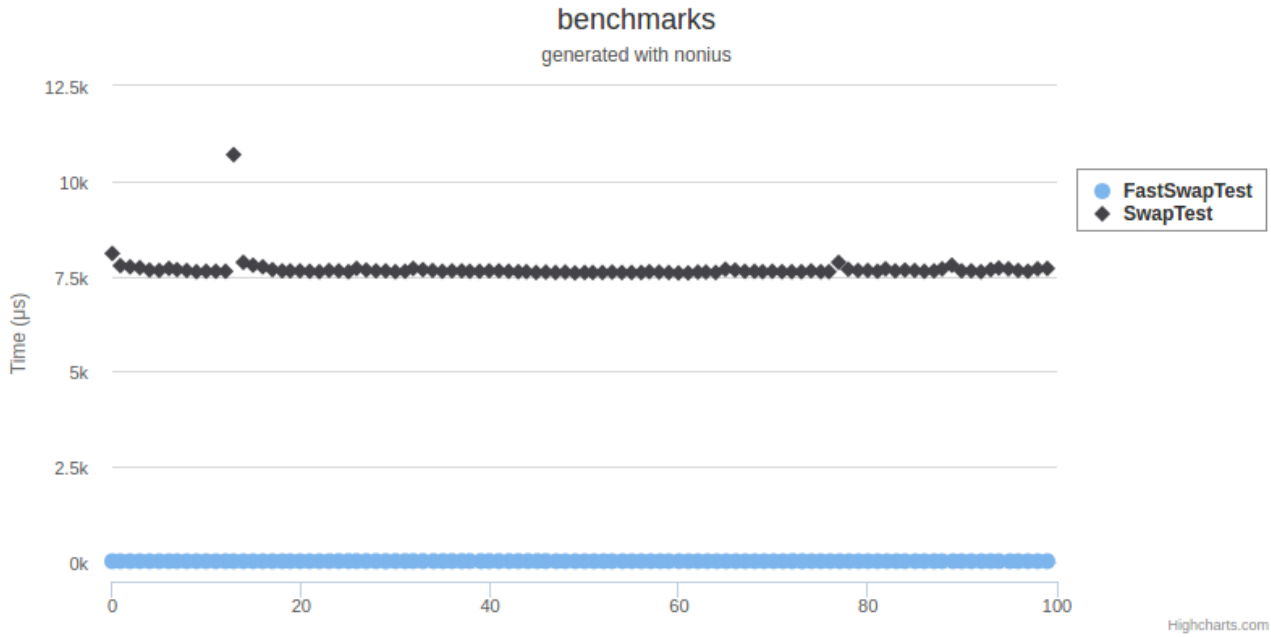
```
#include "nonius/main.h++"
#include "../exe01/BigInt.h"

template <class T>
void swap(T& a, T& b){
    T t(a);
    a = b;
    b = t;
}

template <class T>
void fast_swap(T& a, T& b){
    T t(std::move(a));
    a = std::move(b);
```

```
    b = std::move(t);
}


constexpr int BIG_INT_LEN = 100000;
constexpr int TEST_NUM_COUNT = 1000;

void test_swap(int count){
    BigInt a(1, BIG_INT_LEN);
    BigInt b(1, BIG_INT_LEN);

    for(int i = 0; i < count; i++){
        swap(a, b);
    }
}

void test_fast_swap(int count){
    BigInt a(1, BIG_INT_LEN);
    BigInt b(1, BIG_INT_LEN);

    for(int i = 0; i < count; i++){
        fast_swap(a, b);
    }
}

NONIUS_BENCHMARK("SwapTest", []
{
    test_swap(TEST_NUM_COUNT);
})

NONIUS_BENCHMARK("FastSwapTest", []
{
    test_fast_swap(TEST_NUM_COUNT);
})
```

makefile

```
TARGET = main
INCLUDE = ../../lib/nonius-1.1.2/include/
CXX = g++
CXXFLAGS = -std=c++11 -pthread -I$(INCLUDE)

.PHONY: default all clean

default: $(TARGET)
all: default

OBJECTS = $(patsubst %.cpp, %.o, $(wildcard *.cpp))
HEADERS = $(wildcard *.h)

%.o: %.c $(HEADERS)
	$(CXX) $(CXXFLAGS) -c $< -o $@

.PRECIOUS: $(TARGET) $(OBJECTS)
```

```
$(TARGET): $(OBJECTS)
	$(CXX) $(CXXFLAGS) $(OBJECTS) -o $@

clean:
	-rm -f *.o
	-rm -f $(TARGET)
	$(RM) tool
```



Questão 3

lmplemente um swap para BigInt e injete-o no namespace std.

main.cpp

```
#include "../exe01/BigInt.h"

namespace std{

void fast_swap(BigInt& a, BigInt& b){
    BigInt t(std::move(a));
    a = std::move(b);
    b = std::move(t);
}

}

constexpr int BIG_INT_LEN = 100000;
constexpr int TEST_NUM_COUNT = 1000;

int main(){
    BigInt a(1, BIG_INT_LEN);
    BigInt b(1, BIG_INT_LEN);

    for(int i = 0; i < TEST_NUM_COUNT; i++){
```

```
        std::swap(a, b);
    }
}
```

Questão 4

Substitua o array "puro" de BigInt por um std: :vector e revisite a implementação.

main.cpp

```
#ifndef BIGINT_H
#define BIGINT_H

#include <cstdint>
#include <algorithm>
#include <vector>
#include <iostream>

struct BigInt{
    int sig_;
    int len_;
    std::vector<uint8_t> mag_;


    BigInt()
        : sig_(1), len_(0), mag_(0){
        // std::cout << "BigInt: Construtor" << std::endl;
    }

    BigInt(int sig, int len)
        : sig_(sig), len_(len), mag_( len)
    {
        // std::cout << "BigInt: Construtor" << std::endl;
    }

    BigInt(const BigInt& bi)
        : sig_(bi.sig_), len_(bi.len_), mag_(bi.mag_)
    {
        // std::cout << "BigInt: Construtor de copia" << std::endl;
    }

    BigInt& operator=(const BigInt& bi){
        sig_ = bi.sig_;
        len_ = bi.len_;
        mag_ = bi.mag_;
        // std::cout << "BigInt: Atribuicao por copia" << std::endl;
        return *this;
    }

    BigInt(BigInt&& bi)
        : sig_(bi.sig_), len_(bi.len_), mag_(std::move(bi.mag_))
    {
        bi.len_ = 0;
        // std::cout << "BigInt: Construtor de transferencia" << std::endl;
    }

    BigInt& operator=(BigInt&& bi){
```

```
        sig_ = bi.sig_;
        len_ = bi.len_;
        mag_ = std::move(bi.mag_);
        bi.len_ = 0;
        // std::cout << "BigInt: Atribuicao por transferencia" << std::endl;
        return *this;
    }

    ~BigInt(){
    }
};


#endif // BIGINT_H
```

Questão 5

Defina uma classe Point3D, composta por BigInts, e implemente suas funções especiais.

Point3D.h

```
#ifndef POINT3D_H
#define POINT3D_H

#include <cstdint>
#include <algorithm>
#include <vector>
#include "../exe04/BigInt.h"

class Point3D {
    std::vector<BigInt> coord_;

public:
    Point3D() : coord_({BigInt(),BigInt(),BigInt()})
    {
        std::cout << "Point3D: Construtor" << std::endl;
    }

    Point3D(const BigInt& x, const BigInt& y, const BigInt& z) : coord_(3)
    {
        coord_.push_back(x);
        coord_.push_back(y);
        coord_.push_back(z);
    }

    Point3D(const Point3D& bi)
        : coord_(bi.coord_)
    {
        std::cout << "Point3D: Construtor de copia" << std::endl;
    }

    Point3D& operator=(const Point3D& bi){
        coord_ = bi.coord_;
        std::cout << "Point3D: Atribuicao por copia" << std::endl;
        return *this;
    }
```

```
    Point3D(Point3D&& bi)
        : coord_(std::move(bi.coord_))
    {
        bi.coord_ =  {BigInt(),BigInt(),BigInt()};
        std::cout << "Point3D: Construtor de transferencia" << std::endl;
    }

    Point3D& operator=(Point3D&& bi){
        coord_ = std::move(bi.coord_);
        bi.coord_ =  {BigInt(),BigInt(),BigInt()};
        std::cout << "Point3D: Atribuicao por transferencia" << std::endl;
        return *this;
    }

    ~Point3D(){
    }

    const BigInt& x(){
        return coord_[0];
    }

    const BigInt& y(){
        return coord_[1];
    }

    const BigInt& z(){
        return coord_[2];
    }
};


#endif // POINT3D_H
```

Questão 6

Simule uma fábrica que invoca construtores diferentes de Point3D.

Point3DFactory.h

```
#ifndef POINT3DFACTORY_H
#define POINT3DFACTORY_H

#include "../exe05/Point3D.h"

struct Point3DFactory {
    template<class T = Point3D>
    static T create(T&& p){
        return T(std::forward<T>(p));
    }
};


#endif // POINT3DFACTORY_H
```

Questão 7

Verifique se as sobrecargas chamadas são as que você espera.

main.cpp

```
#include "../exe06/Point3DFactory.h"

int main()
{

    Point3D source;
    Point3D target1 = Point3DFactory::create(source);
    Point3D target2 = Point3DFactory::create(std::move(source));

    return 0;
}
```