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

Lista de exercícios Módulo 09: Projeto de classes e design de API

Questões

Questão 1

Faca testes através de instanciação/cópia de objetos e descubra quais construtores/operadores/destrutor são preservados/eliminados automaticamente pelo compilador para a classe Type.

O construtor por cópia é definido explicitamente.

O construtor por transferência não é gerado, de modo que o construtor por cópia é chamado mesmo quando são passados *rvalues*.

O operador de atribuição por cópia é eliminado pelo compilador.

O operador de atribuição por transferência é definido explicitamente.

O destrutor aparentemente é gerado, pois não se verificaram problemas quando o objeto saiu de escopo.

main.cpp

```
#include <iostream>
#include <string>

class Type {
public:
	Type() {
		std::cout << "Type: Default constructor\n";
	}
	Type& operator=(Type&& t){
		std::cout << "Type: Transfer assignement\n";
		return *this;
	}
	Type(const Type&){
		std::cout << "Type: Copy constructor\n";
	}
};

int main()
{
	Type t1;
	Type t2(t1);
	Type t3(std::move(t1));
```

```
        Type t4;
        //t4= t1;           // main.cpp:4:7: note:  'Type& Type::operator=(const Type&)'
                            // is implicitly declared as deleted because 'Type' declares
                            // a move constructor or move assignment operator
        Type t5;
        t5 = std::move(t1);
        Type t6 = std::move(t1); // Why does this call the copy constructor?
                                 // Implicitly conversion.

    return 0;
}
```

Questão 2

Corrija a implementação da classe Down de forma que a funcao main( ) compile corretamente.

main.cpp

```
#include <iostream>
#include <string>

struct Top {
        virtual void f() const;
        void g(int);
        void k();
};


struct Down : public Top {
        //void f() override;
        //main.cpp:12:7: error: 'void Down::f()' marked 'override', but does not override
        //  void f() override;
        //       ^
        void f() const override;

        using Top::g;
        void g(std::string);

        //void k(int) override;
        //main.cpp:20:7: error: 'void Down::k(int)' marked 'override', but does not override
        //  void k(int) override;
        //       ^
        using Top::k;
        void k(int);
};

int main()
{
    return 0;
}
```

Questão 3

A funcao std::swap( ) para arrays é mostrada abaixo. Adicione a ela o operador noexcept de forma que ela lance exceção somente se std::swap( ) dos elementos também lance.

main.cpp

```cpp
#include <iostream>
#include <string>

template <class T, size_t N>
void swap(T (&a) [N], T (&b) [N])
noexcept(noexcept(std::swap(a,b)))
{
}

int main()
{
   return 0;
}
```

Questão 4

Termine a implementação Pimpl da classe Employee. Teste seu funcionamento básico e simule a situação em que um novo membro é adicionado, sem que haja quebra de compatibilidade binária.

Compilando e testando com a versão 1 da biblioteca (“Employee.h” e “Employee.cpp”), o programa executa sem problemas. Entretanto, ao executar utilizando biblioteca dinâmica da versão 2, o seguinte erro é exibido:

./main: symbol lookup error: ./main: undefined symbol: _ZN6my_lib10getVersionEv

main.cpp

```cpp
#include <iostream>
#include <string>

#include "Employee.h"

int main()
{
	std::cout << "Lib version: " << my_lib::getVersion() << std::endl;

	my_lib::Employee employee("Joe");
	std::cout << "Employee name: " << employee.getName() << std::endl;

   return 0;
}

// Compile and test with v1
// g++ -c -Wall -Werror -fpic Employee.cpp
// g++ -shared -o libEmployee.so Employee.o
// g++ -L"/home/arthur/curso_cpp_moderno/module_09/exe04" -Wall -o main main.cpp -lEmployee
// export LD_LIBRARY_PATH=/home/arthur/curso_cpp_moderno/module_09/exe04:$LD_LIBRARY_PATH
// ./main

// Compile with v1 and test with v2
// g++ -c -Wall -Werror -fpic Employee.cpp
// g++ -shared -o libEmployee.so Employee.o
// g++ -L"/home/arthur/curso_cpp_moderno/module_09/exe04" -Wall -o main main.cpp -lEmployee
// g++ -c -Wall -Werror -fpic Employee_v2.cpp
```

```
// g++ -shared -o libEmployee.so Employee_v2.o
// export LD_LIBRARY_PATH=/home/arthur/curso_cpp_moderno/module_09/exe04:$LD_LIBRARY_PATH
// ./main

// Fonte:
// http://www.cprogramming.com/tutorial/shared-libraries-linux-gcc.html
```

Employee.h

```
#include <string>

namespace my_lib{

class Employee {
public:
	Employee(const std::string&);
	~Employee();
	const std::string& getName();

private:
	class EmployeeImpl;
	EmployeeImpl* impl_;
};

std::string getVersion();

} // end namespace my_lib
```

Employee.cpp

```
#include "Employee.h"

namespace my_lib{

struct Date
{

};

class Employee::EmployeeImpl {
public:
	size_t id_;
	std::string name_;
#if  __cplusplus >= 201103L
	std::chrono::time_point date_;
#else
	Date date_;
#endif
	std::string nick_;
};

Employee::Employee(const std::string& name)
: impl_(new EmployeeImpl()){
	impl_->name_ = name;
}

Employee::~Employee(){
	delete impl_;
}

const std::string& Employee::getName(){
	return impl_->name_;
}

std::string getVersion(){
	return "v1";
}
```

```
} // end namespace my_lib
```

Employee_v2.h

```
#include <string>

namespace my_lib{

namespace my_lib_v1{

class Employee {
public:
        Employee(const std::string&);
        ~Employee();
        const std::string& getName();

private:
        class EmployeeImpl;
        EmployeeImpl* impl_;
};

std::string getVersion();

} // end namespace my_lib_v1

inline namespace my_lib_v2{

class Employee {
public:
        Employee(const std::string&);
        ~Employee();
        const std::string& getName();
        std::string getNameWithPrefix(const std::string& prefix);

private:
        class EmployeeImpl;
        EmployeeImpl* impl_;
};

std::string getVersion();

} // end namespace my_lib_v2

} // end namespace my_lib
```

Employee_v2.cpp

```
#include "Employee_v2.h"

namespace my_lib{

namespace my_lib_v1{

struct Date
{

};

class Employee::EmployeeImpl {
public:
        size_t id_;
        std::string name_;
#if  __cplusplus >= 201103L
        std::chrono::time_point date_;
#else
        Date date_;
#endif
```

```
        std::string nick_;
};

Employee::Employee(const std::string& name)
: impl_(new EmployeeImpl()){
        impl_->name_ = name;
}

Employee::~Employee(){
        delete impl_;
}

const std::string& Employee::getName(){
        return impl_->name_;
}

std::string getVersion(){
        return "v1";
}

} // end namespace my_lib_v1

namespace my_lib_v2{

struct Date
{

};

class Employee::EmployeeImpl {
public:
        size_t id_;
        std::string name_;
#if __cplusplus >= 201103L
        std::chrono::time_point date_;
#else
        Date date_;
#endif
        std::string nick_;
};

Employee::Employee(const std::string& name)
: impl_(new EmployeeImpl()){
        impl_->name_ = name;
}

Employee::~Employee(){
        delete impl_;
}

const std::string& Employee::getName(){
        return impl_->name_;
}

std::string Employee::getNameWithPrefix(const std::string& prefix){
        return prefix + " " + impl_->name_;
}

std::string getVersion(){
        return "v2";
}

} // end namespace my_lib_v2


} // end namespace my_lib
```

Questão 5

Ao implementar Employee com um std::unique_ptr, conforme nesse slide, você pode encontrar um erro relacionado ao destrutor. Explique-o e corrija-o. (A classe Employee deve estar em um header e você deve usá-la de um fonte.)

A implementação sugerida do destrutor tenta deletar o ponteiro, mas isso não é necessário ao se utilizar o “std::unique_ptr”.

main.cpp

```
#include <iostream>
#include <string>

#include "Employee.h"

int main()
{
	my_lib::Employee employee("Joe");
	std::cout << "Employee name: " << employee.getName() << std::endl;

  return 0;
}
```

Employee.h

```
#include <string>
#include <memory>

namespace my_lib{

class Employee {
public:
	Employee(const std::string&);
	~Employee();
	const std::string& getName();

private:
	class EmployeeImpl;
	std::unique_ptr<EmployeeImpl> impl_;
};

} // end namespace my_lib
```

Employee.cpp

```
#include "Employee.h"

namespace my_lib{

struct Date
{

};

class Employee::EmployeeImpl {
public:
	size_t id_;
	std::string name_;
#if __cp1usp1us >= 201103L
	std::chrono::time_point date_;
#else
	Date date_;
#endif
	std::string nick_;
};

Employee::Employee(const std::string& name)
: impl_(new EmployeeImpl()){
	impl_->name_ = name;
}
```

```
Employee::~Employee(){
}

const std::string& Employee::getName(){
	return impl_->name_;
}


} // end namespace my_lib
```

Questão 6

Há como evitar alocação dinâmica com o idioma Pimpl? Pesquise sobre "fast pimpl idiom".

Sim, a solução envolve em utilizar uma especialização do operador “new” que instancia a implementação em um buffer de tamanho fixo na classe em questão.

Fonte: http://www.gotw.ca/gotw/028.htm

http://www.cleeus.de/w/blog/2017/03/10/static_pimpl_idiom.html

Questão 7

lmplemente uma versão do método push( ) da nossa Stack que ofereca a garantia strong. Utilize uma Stack temporária e, se tudo correr bem, realize um swap com a original. Obs: Note que o método push, quando bem sucedido, pode ainda causar invalidação de ponteiros/iteradores (essa situação também acontece com std: :vector : :push_back). 0 problema que queremos resolver é a garantia de "bom, comportamento" perante uma exceção.

main.cpp

```
#include <iostream>
#include <string>

class Type {
public:
	Type() {
		std::cout << "Type: Default constructor\n";
	}
	Type& operator=(Type&& t){
		std::cout << "Type: Transfer assignement\n";
		return *this;
	}
	Type(const Type&){
		std::cout << "Type: Copy constructor\n";
	}
};

template <class T>
class Stack{
private:
	T* p_;
	size_t size_;
	size_t used_;
public:
	void push(const T &v) {
		Stack<T> temp = *this;

		if (temp.used_ - temp.size_) {
			size_t size = temp.size_ * 2;
			T* p = new T[size];
			try{
```

```
                                std::copy(temp.p_, temp.p_ + temp.size_, p);
                        } catch (...) {
                                delete[] p;
                                throw;
                        }

                        delete[] temp.p_;
                        temp.p_ = p;
                        temp.size_ = size;
                }

                temp.p_[used_ - 1] = v;
                ++temp.used_;

                std::swap(this, temp);
        }
};

int main()
{
        Stack<int> stack;

  return 0;
}
```

Questão 8

lnvestigue o por quê de std::stack ter um método pop( ) que remove o elemento do topo e um método top( ) que retorna uma referéncia a ele, ao invés de um método único que faz as duas coisas, retornando o elemento do topo por cópia.

Se pop() retornasse o valor do topo da pilha, teria de fazê-lo por cópia, caso contrário o retorno seria um ponteiro inválido. De forma a evitar pagar um custo desnecessário (cópia adicional) em situações onde desejasse apenas remover o elemento do topo, preferiu-se criar um método adicional para inspecioná-lo.

Fonte: https://stackoverflow.com/questions/12206242/store-results-of-stdstack-pop-method-into-a-variable

http://www.sgi.com/tech/stl/stack.html

Questão 9

Crie uma classe ApiVersion, com funções major( ), minor( ), patch( ), que possa ser usada tanto em tempo de compilação quanto de execução.

Obs.: É possível instanciar objetos e definir major(), minor() e patch() como funções membro não estáticas, mas funções membro estáticas me pareceram mais adequadas para o caso em questão.

Fonte: https://softwareengineering.stackexchange.com/questions/165008/can-i-get-a-c-compiler-to-instantiate-objects-at-compile-time

main.cpp

```
#include <iostream>
#include <array>

class ApiVersion {
public:
        static constexpr int major(){
                return 1;
        }
        static constexpr int minor(){
                return 2;
```

```
        }
        static constexpr int patch(){
                return 3;
        }
};

int main()
{
        std::array<int, ApiVersion::major()> major;
        std::array<int, ApiVersion::minor()> minor;
        std::array<int, ApiVersion::patch()> patch;

        std::cout << "Version (compile time): "
                << major.size() << "."
                << minor.size() << "."
                << patch.size() << std::endl;

        std::cout << "Version (run time): "
                << ApiVersion::major() << "."
                << ApiVersion::minor() << "."
                << ApiVersion::patch() << std::endl;

   return 0;
}
```