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

Lista de exercícios Módulo 07: Lambdas

Questões

Questão 1

Qual o bug do código A e como corrigí-lo mudando apenas o tipo da declaração?

main.cpp

```
#include <iostream>
#include <functional>

int main()
{
        std::function<int(int)> f = [&f] (int v) {
                if (!v)
                        return f(v - 1);
                return 0;
        };

   std::cout << "f(v): \t" << f(1) << std::endl;

   return 0;
}
```

Questão 2

Qual o bug no código B e como corrigí-lo modificando apenas o tipo de retorno especificado ?

main.cpp

```
#include <iostream>
#include <vector>
#include <iterator>

void f() {
```

```
        auto r = []() -> std::vector<int> {
                int a = 10, b = 26, c = 20;
                return {a, b, c};
        }();

        std::copy(
                begin(r), end(r),
                std::ostream_iterator<int>(std::cout, " ")
        );
}

int main()
{
   f();

   return 0;
}
```

Questão 3

lmplemente uma versão simplificada de uma classe Closure.

Closure.h

```
#ifndef CLOSURE_H
#define CLOSURE_H

#include <iostream>

template <class T>
class  Closure{
public:
   T value;

   Closure(T src) : value(src)
   {
      std::cout << "Closure(T)" << std::endl;
   }


   ~Closure(){
   }
};

// http://www.learncpp.com/cpp-tutorial/13-8-partial-template-specialization-for-pointers/
template <class T>
class  Closure<T*>{
public:
   T& value;

   Closure(T* src) : value(*src)
   {
      std::cout << "Closure(T*)" << std::endl;
   }

   ~Closure(){
   }
```

```
};

template <class T>
Closure<T> make_closure(T value){
   return Closure<T>(value);
}



#endif // CLOSURE_H
```

main.cpp

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

int main()
{
        int a = 0;

        auto closure = make_closure(a);
        auto ref_closure = make_closure(&a);

        a = 1;

        std::cout << "Closure = " << closure.value << std::endl;
        std::cout << "RefClosure = " << ref_closure.value << std::endl;

   return 0;
}
```

Questão 4

Na captura por copia, o = é necessário no modo default mas inválido no individual. lsso evita ambiguidade com a captura vazia, []. Seria melhor uma sintaxe alternativa (questão aberta)?

Uma opção seria a possibilidade de se omitir a lista de captura "[]" para realizar uma captura vazia e explicitá-la para realizar uma captura default.

Questão 5

O que há de "errado" na captura [&, x, &y] ?

A captura de &y é desnecessária, pois a captura default por referência foi habilitada.

Questão 6

Utilizando std::sort e um lambda, ordene os caracteres da string "eiAiLjaOQl" de acordo com o alfabeto, ignorando a capitalização (minúsculas e maiúsculas).

main.cpp

```
#include <iostream>
```

```cpp
#include <algorithm>

int main()
{
	std::string s = "eiAiLjaOQl";
	std::sort(s.begin(), s.end(),
		[] (char c1, char c2) {
			return tolower(c1) < tolower(c2);
	});

   std::cout << "s: \t" << s << std::endl;

   return 0;
}
```

Questão 7

Defina uma aplicação parcial que corresponda a multiplicação de um argumento por 10, combinado com sua some a 5. Utilize std::bind, std::multiplies, e std::plus.

main.h

```cpp
#include <iostream>
#include <functional>

int main()
{
	// http://stackoverflow.com/questions/13799550/how-to-compose-functors-with-stl
	auto f = std::bind(std::plus<int>(), std::bind(std::multiplies<int>(),
std::placeholders::_1, 10), 5);

   std::cout << "f(5): \t" << f(5) << std::endl;

   return 0;
}
```

Questão 8

Crie sua própria implementação de std::ref.

Ref.h

```cpp
#ifndef REF_H
#define REF_H

#include <iostream>

template <class T>
class  ReferenceWrapper{
public:
   T& value;

   ReferenceWrapper(T& src) : value(src)    {
```

```
    }

    ~ReferenceWrapper(){
    }

    operator T&() const { return value; }
};

template <class T>
ReferenceWrapper<T> ref(T& src){
    std::cout << "Custom ref!" << std::endl;
    return ReferenceWrapper<T>(src);
}

#endif // REF_H
```

main.cpp

```
#include <iostream>
#include <functional>
#include "Ref.h"

void mutate(int a, int& b) { b = a; }

int main()
{
        int i = 1, j = 0;
        std::bind(mutate, std::placeholders::_1, std::ref(j)) (i);

        std::cout << "j = " << j << std::endl;

        i = 2;
        std::bind(mutate, std::placeholders::_1, ref(j)) (i);

        std::cout << "j = " << j << std::endl;

    return 0;
}
```