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

Lista de exercícios Módulo 10: Programação genérica e STL

Questões

Questão 1

Itere mais uma vez sobre a função sum( ) e faça um lifting que a permita funcionar com listas encadeadas.

main.cpp

```
#include <iostream>
#include <string>
#include <vector>
#include <list>
#include <deque>
#include <type_traits>

// http://www.cplusplus.com/forum/articles/20881/
// https://stackoverflow.com/questions/18467190/a-template-function-that-takes-iterators-or-pointers-and-gets-the-pointed-to-typ
template<class ForwardIteratorType>
typename std::iterator_traits<ForwardIteratorType>::value_type sum( ForwardIteratorType begin,
ForwardIteratorType end) {
        auto result = typename std::iterator_traits<ForwardIteratorType>::value_type();
   while(begin != end)
   {
                result = result + *(begin);
      ++begin;
   }

        return result;
}

int main()
{
   const int LENGTH(6);
   int data[LENGTH] = { 5, 7, 8, 9, 1, 2 };
   std::vector<int> vData(data, data + LENGTH);
   std::list<int> lData(data, data + LENGTH);
   std::deque<int> dData(data, data + LENGTH);
```

```
    std::cout << "Array sum: " << sum(data, data + LENGTH) << std::endl;
    std::cout << "Vector sum: " << sum(vData.begin(), vData.end()) << std::endl;
    std::cout << "List sum: " << sum(lData.begin(), lData.end()) << std::endl;
    std::cout << "Deque sum: " << sum(dData.begin(), dData.end()) << std::endl;

    return 0;
}
```

Questão 2

Resolva o problema de invocar std::sort( ) em um container do tipo std::list.

Basta chamar o método *sort()* da própria std::list. Não é possível utilizar std::sort em uma std::list pois std::sort requer que os iteradores utilizados fornecem acesso aleatório.

Fonte: https://stackoverflow.com/questions/10652674/sorting-stdlists-using-stdsort

Questão 3

Coloque um default template argument em Accumulator e generalize-o para utilizar iteradores (ao invés de ponteiros). Qual conceito de iterador a implementação exige?

Os iteradores precisam implementar o conceito ForwardIterator, pois permite ser incrementado para obter o próximo elemento e dereferenciado para leitura do elemento atual.

Fonte: https://www.sgi.com/tech/stl/Iterators.html

main.cpp

```
#include <iostream>
#include <string>
#include <vector>
#include <list>
#include <deque>
#include <type_traits>

template<class T>
struct SumPolicy {
    static void accumulate(T& total, T v) { total += v; }
};

template<class T>
struct MultiplyPolicy {
    static void accumulate(T& total, T v) { total *= v; }
};

// https://stackoverflow.com/questions/12930653/getting-an-neutral-element-for-stdmultiply-and-stdplus
template <typename Policy> struct PolicyTraits;
template<>
struct PolicyTraits<SumPolicy<int> >
{
    static int const identity = 0;
};
template<>
struct PolicyTraits<MultiplyPolicy<int> >
{
    static int const identity = 1;
};

template <template<class> class Policy = SumPolicy>
struct Accumulator {
    template<class ForwardIteratorType>
    static typename std::iterator_traits<ForwardIteratorType>::value_type run(ForwardIteratorType cur,
ForwardIteratorType end) {
        auto acc = PolicyTraits<Policy<typename std::iterator_traits<ForwardIteratorType>::value_type>>::identity;
        // std::cout << "acc = " << acc << std::endl;
        while (cur != end){
```

```
            // std::cout << "cur = " << *cur << std::endl;
            Policy<typename std::iterator_traits<ForwardIteratorType>::value_type>::accumulate(acc, *cur);
            cur++;
            // std::cout << "acc = " << acc << std::endl;
        }
        return acc;
    }
};

int main()
{
    const int LENGTH(6);
    int data[LENGTH] = { 1, 2, 3, 4, 5, 6 };
    std::vector<int> vData(data, data + LENGTH);
    std::list<int> lData(data, data + LENGTH);
    std::deque<int> dData(data, data + LENGTH);

    // https://stackoverflow.com/questions/15373823/template-default-arguments
    std::cout << "Array sum_acc: " << Accumulator<>::run(data, data + LENGTH) << std::endl;
    std::cout << "Vector sum_acc: " << Accumulator<>::run(vData.begin(), vData.end()) << std::endl;
    std::cout << "List sum_acc: " << Accumulator<>::run(lData.begin(), lData.end()) << std::endl;
    std::cout << "Deque sum_acc: " << Accumulator<>::run(dData.begin(), dData.end()) << std::endl;

    std::cout << "Array mult_acc: " << Accumulator<MultiplyPolicy>::run(data, data + LENGTH) << std::endl;
    std::cout << "Vector mult_acc: " << Accumulator<MultiplyPolicy>::run(vData.begin(), vData.end()) << std::endl;
    std::cout << "List mult_acc: " << Accumulator<MultiplyPolicy>::run(lData.begin(), lData.end()) << std::endl;
    std::cout << "Deque mult_acc: " << Accumulator<MultiplyPolicy>::run(dData.begin(), dData.end()) << std::endl;

    return 0;
}
```

Questão 4

Pesquise na Internet, apenas para conhecimento, sobre o Curiously Recurring Template Pattern.

O CRTP ocorre quando uma classe tem uma classe base que é uma especialização de template da própria classe filha.

Fonte: https://stackoverflow.com/questions/4173254/what-is-the-curiously-recurring-template-pattern-crtp

Questão 6

Termine a implementação de PoolAlloc de maneira que a instanciação de std::vector<int, PoolAlloc<int>> compile sem erros. Não é permitido herdá-lo de std: :allocator.

main.cpp

```
#include <memory>
#include <vector>

template <class T>
struct PoolAlloc
{
    typedef T value_type;

    T* allocate(size_t n){
        void* p = malloc(n);
        return static_cast<T*>(p);
    }

    void deallocate(T* p, size_t){
        free(p);
    }
```

```
};

int main()
{
   std::vector<int, PoolAlloc<int>> v;

   return 0;
}
```

Questão 7

Verifique se o template IsClassType funciona conforme você espera.

main.cpp

```
#include <iostream>
#include <string>
#include <memory>
#include <vector>

template <class T>
struct IsClassType
{
   template <class C> static char size(int C::*);
   template <class C> static std::int16_t size(...);
   enum { Yes = sizeof(IsClassType<T>::size<T>(nullptr)) == 1 };
   enum { No = !Yes };
};

template <class T>
void f() {
   if (IsClassType<T>::Yes)
      std::cout << "yes" << std::endl;
   else
      std::cout << "no" << std::endl;
}

struct MyStruct{};

int main()
{
   std::cout << "IsClassType<int> = "; f<int>();
   std::cout << "IsClassType<MyStruct> = "; f<MyStruct>();

   return 0;
}
```

Result

```
IsClassType<int> = no
IsClassType<MyStruct> = yes
```

Sim, funciona como esperado.

Questão 8

Como você invocaria a função n_sum?

main.cpp

```cpp
#include <iostream>

template <class T>
T n_sum(T base) {
   return base;
}

template <class T, class ... Ts>
T n_sum(T first, Ts... rest) {
   return first + n_sum(rest...);
}

int main()
{
   std::cout << "n_sum(1,2.0,3L,4.0f,5) = " << n_sum(1,2.0,3L,4.0f,5) << std::endl;

   return 0;
}
```

Questão 9

Instancie, com o tipo Person, os conteiners std::map e std::unordered_map e veja quais operadores precisa em cada caso.

main.cpp

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

struct Person{
	std::string name_;
	int age_;

	Person(std::string name, int age)
	: name_(name), age_(age)
	{}
};

struct ComparePerson {
	bool operator() (const std::string& a, const std::string& b) const {
		return a.length() < b.length();
	}
};

int main()
{
	std::map<std::string, Person, ComparePerson> map;
	map.emplace("Arthur", Person("Arthur", 26));

	std::unordered_map<std::string, Person> umap;
	umap.emplace("Arthur", Person("Arthur", 26));

   return 0;
}
```

Questão 10

Experimente a interface de std::unordered_map, adicionando valores com insert, procurando-os com find e removendo-os com erase. Faça também alterações no loading factor da tabela de hash.

main.cpp

```
#include <iostream>
#include <string>
#include <vector>
#include <unordered_map>

void show(const std::unordered_map<std::string,std::string>& mymap){
	unsigned load_factor = mymap.load_factor();
	std::cout << "mymap load_factor is " << load_factor << ".\n";

	unsigned bucket_count = mymap.bucket_count();
	std::cout << "mymap has " << bucket_count << " buckets.\n";

	for (unsigned i=0; i<bucket_count; ++i) {
		std::cout << "bucket #" << i << " contains: ";
		for (auto it = mymap.begin(i); it!=mymap.end(i); ++it)
			std::cout << "[" << it->first << ":" << it->second << "] ";
			std::cout << "\n";
	}

}

// http://www.cplusplus.com/reference/unordered_map/unordered_map/bucket_count/
// http://www.cplusplus.com/reference/unordered_map/unordered_map/find/
// http://www.cplusplus.com/reference/unordered_map/unordered_map/erase/
int main ()
{
	std::unordered_map<std::string,std::string> mymap = {
		{"house","maison"},
		{"apple","pomme"},
		{"tree","arbre"},
		{"book","livre"},
		{"door","porte"},
		{"grapefruit","pamplemousse"}
	};

	mymap.insert(std::make_pair("I", "je"));
	mymap.insert(std::make_pair("knee", "genou"));

	mymap["flower"] = "fleur";

	show(mymap);

	// FIND
	std::string input;
	std::cout << "who? ";
	getline (std::cin,input);

	std::unordered_map<std::string,std::string>::const_iterator got = mymap.find(input);

	if ( got == mymap.end() )
		std::cout << "not found";
	else
		std::cout << got->first << " is " << got->second;

	std::cout << std::endl;

	// ERASE
	// erase examples:
	mymap.erase ( mymap.begin() );      // erasing by iterator
	mymap.erase ("tree");               // erasing by key
	mymap.erase ( mymap.find("knee"), mymap.end() ); // erasing by range
	show(mymap);

	return 0;
}
```

Questão 11

Qual o problema de se verificar a existência de um elemento em std::map com o operator [ ] ?

O operador[] de std::map realiza uma inserção se o elemento não existir, logo ele nunca indicará a não existência do elemento.

Fonte: http://en.cppreference.com/w/cpp/container/map/operator_at

Questão 12

Qual a diferença entre std::sort( ), std::partial_sort( ) e std::stable_sort( )? Teste-os com a seguintes elementos, usando como critério de ordenação apenas o inteiro de cada par: <4, "a">, <3, "a">, <9, “b”>, <9, “a”>, <1, “a”>, <10, “a”>, <1, “b”>, <2, “a”>, <7, “a”>.

std::sort não garante que a ordem de elementos iguais será preservada.

std::stable_sort garante que a ordem dos elementos iguais será preservada.

Std::partial_sort ordena apenas os n menores items e não garante a ordem de elementos iguais, nem a ordem dos elementos restantes.

Fonte: http://en.cppreference.com/w/cpp/algorithm/sort
http://en.cppreference.com/w/cpp/algorithm/stable_sort
http://en.cppreference.com/w/cpp/algorithm/partial_sort

main.cpp

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <tuple>
#include <algorithm>

void print(const std::vector<std::tuple<int, char>>& v){
   for(auto t : v)
      std::cout << "<" << std::get<0>(t) << ", " << std::get<1>(t) << ">" << std::endl;
}

int main()
{
   std::vector<std::tuple<int, char>> v = {
      std::make_tuple(4, 'a'),
      std::make_tuple(3, 'a'),
      std::make_tuple(9, 'b'),
      std::make_tuple(9, 'a'),
      std::make_tuple(1, 'a'),
      std::make_tuple(10, 'a'),
      std::make_tuple(1, 'b'),
      std::make_tuple(2, 'a'),
      std::make_tuple(7, 'a')
   };

   std::cout << "Unsorted:" << std::endl;
   print(v);
   std::cout << std::endl;

   auto compare = [](std::tuple<int, char> a, std::tuple<int, char> b) {
      return std::get<0>(a) < std::get<0>(b);
   };

   std::vector<std::tuple<int, char>> v1 = v;
   std::sort(v1.begin(), v1.end(), compare);
   std::cout << "std::sort():" << std::endl;
   print(v1);
   std::cout << std::endl;

   std::vector<std::tuple<int, char>> v2 = v;
   std::partial_sort(v2.begin(), v2.end(), v2.end(), compare);
   std::cout << "std::partial_sort():" << std::endl;
   print(v2);
```

```
    std::cout << std::endl;

    std::vector<std::tuple<int, char>> v3 = v;
    std::stable_sort(v3.begin(), v3.end(), compare);
    std::cout << "std::stable_sort():" << std::endl;
    print(v3);
    std::cout << std::endl;

    return 0;
}
```

Result

```
Unsorted:
<4, a>
<3, a>
<9, b>
<9, a>
<1, a>
<10, a>
<1, b>
<2, a>
<7, a>

std::sort():
<1, a>
<1, b>
<2, a>
<3, a>
<4, a>
<7, a>
<9, b>
<9, a>
<10, a>

std::partial_sort():
<1, a>
<1, b>
<2, a>
<3, a>
<4, a>
<7, a>
<9, a>
<9, b>
<10, a>

std::stable_sort():
<1, a>
<1, b>
<2, a>
<3, a>
<4, a>
<7, a>
<9, b>
<9, a>
<10, a>
```

Questão 13

Utilize a sobrecarga de std::regex_search que recebe um std::match_results para iterar e imprimir as partes da string *tag <foo> 123 </foo> gat* com o padrão R"(<(. *)>.*</\1>)".

main.cpp

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

int main()
{
    std::regex r(R"(<(.*)>.*</\1>)");
    std::string s = "tag <foo> 123 </foo> gat";
    std::smatch match;
    bool ok = std::regex_search(s, match, r);

    for(auto m : match){
        std::cout << "match = " << m << '\n';
    }

    return 0;
}
```

Result

match = <foo> 123 </foo>
match = foo