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

Lista de exercícios Módulo 04: Preliminares de C++11 e C++14

Questões

Questão 1.....1
Questão 2.....1
Questão 3.....2
Questão 4.....2
Questão 5.....3
Questão 6.....3
Questão 7.....4
Questão 8.....4
Questão 9.....5
Questão 10.....6

Questão 1

Responda as perguntas deixadas em caixas cinzas nos slides.

Este exercicio não foi realizado.

p.2: Qual funcionalidade nenhum compilador implementou?
p.8: Qual a implementação típica de NULL em C e porquê ela não é compatível com C++?
p.9: Qual a diferença entre <cstdio> e <stdio.h>?
p.13: Qual o valor de NULL nesta plataforma, 0 ou 0L?
p.17: Mas e o std::for_each?
p.17: Por quê const&?
p.20: O que um iterador?
p.21: Qual a diferença entre diretivas e declarações using?
p.25: Realmente compilaria em C++98/03?
p.25: Por quê typename é necessário?
p.27: Qual a relação entre inteiros e enums em C++?
p.28: Por quê isso não é permitido?
p.28: Qual o problema de #include?
p.32: O que é um construtor explícito e um operador de conversão?
p.33: Qual a representação interna de um enum convencional?
p.36: Qual o nome desta técnica de programação?
p.38: Qual seria outra vantagem de constexpr?
p.47: Como escrever essa string em C++98/03?
p.48: Qual o prefixo de um literal std::wstring?
p.49: Qual o enconding de um literal std::wstring?
p.58: Como criar inteiros em octal e hexa em C++98/03?
p59: Quando um assert é avaliado?

Questão 2

Como NULL é implementado em sua plataforma?

Em <stddef.h>:

```
#define NULL 0
```

Fonte: http://en.cppreference.com/w/cpp/types/NULL

Questão 3

Modifique o vetor v da função g( ) incrementando cada elemento de uma unidade (in-place). Faça isso de duas maneiras diferentes: com o um for de intervalos e com std: :for_each.

main.cpp

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

void g(){
   std::vector<int> v = {1, 2, 3, 4};

   for(int& n : v){
      n++;
      std::cout << " " << n;
   }
   std::cout << std::endl;

   std::for_each(v.begin(), v.end(), [](int &n){ n++; });
   std::for_each(v.begin(), v.end(), [](const int& n) { std::cout << " " << n; });

   std::cout << std::endl;
}

int main(){
   g();

   return 0;
}
```

Questão 4

Reproduza o código com FaultProtection, ProcStatus e checkviolation( ), mas agora usando enumerações fortemente tipadas. Ambos enums devem ter char como tipo base.

main.cpp

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

#define STR(symbol) #symbol

enum class FaultProtection : char{
   Active,  Unavailable
};

enum class ProcStatus: char{
   Active,  Waiting, Cancelled
};
```

```
bool checkViolation(ProcStatus s){
   if(s == ProcStatus::Active){
      std:: cout << STR(ProcStatus::Active);
   } else if(s == ProcStatus::Waiting){
      std:: cout << STR(ProcStatus::Waiting);
   } else if(s == ProcStatus::Cancelled){
      std:: cout << STR(ProcStatus::Cancelled);
   } else return true;
}

int main(){
   checkViolation(ProcStatus::Waiting);

   std:: cout << std::endl;

   return 0;
}
```

Questão 5

Implemente a função de fatorial em tempo de compilação com constexpr. Qual o maior número com qual você consegue compilá-la?

O maior número foi 522, após o qual o compilador exibe o seguinte erro:

```
p.cpp:4:36:   in constexpr expansion of ‘factorial((x + 18446744073709551615ull))’
p.cpp:14:23: error: constexpr evaluation depth exceeds maximum of 512 (use -fconstexpr-depth= to increase the maximum)
    case factorial(522): break;
```

main.cpp

```
constexpr unsigned long long int factorial(unsigned long long int x){
   return  x > 1 ? x*factorial(x-1) : 1;
}

int main(){
   unsigned long long int i = 0;
   switch (i){
   case -1: break;
   case factorial(1): break;
   case factorial(2): break;
   case factorial(10): break;
   case factorial(521): break;
   case factorial(522): break;
   }
   return 0;
}
```

Questão 6

Escreva expressões regulares tanto com strings convencionais quando com strings literais puras que casem com o seguinte padrão: *leandro\"BR"\31-99990000*.

main.cpp

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

int main()
{
   std::string s = R"(leandro\"BR"\31-99990000)";

   std::regex conventional_regex("leandro\\\\\"BR\"\\\\31-99990000");
   if (std::regex_search(s, conventional_regex)) {
      std::cout << "Match found with conventional string.\n";
   }

   std::regex raw_regex(R"(leandro\\"BR"\\31-99990000)");
   if (std::regex_search(s, raw_regex)) {
      std::cout << "Match found with raw string.\n";
   }

   return 0;
}
```

Questão 7

Qual a diferença entre UTF-8, UTF-16 e UTF-32? Implemente três programas que mostram a string "olá" na tela, uma cada para encoding. O acento não deve estar literalmente no texto.

main.cpp

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

#include <codecvt>   // for codecvt_utf8
#include <locale>    // for wstring_convert

int main()
{
   std::string s_u8 = u8"ol\u00E1";
   std::u16string s_u16 = u"ol\u00E1";
   std::u32string s_u32 = U"ol\u00E1";

   std::cout << "UTF-8: " << s_u8 << "\n";

   std::wstring_convert<std::codecvt_utf8<char16_t>, char16_t> cv_u16;
   std::cout << "UTF-16: " << cv_u16.to_bytes(s_u16) << "\n";

   std::wstring_convert<std::codecvt_utf8<char32_t>, char32_t> cv_u32;
   std::cout << "UTF-32: " << cv_u32.to_bytes(s_u32) << "\n";

   return 0;
}
```

Questão 8

Projeta uma classe de números complexos em que a parte imaginária seja um literal de usuário com sufixo _i. A implementagfio é mínima, o suficiente para demonstrar sua

utilização.

main.cpp

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

struct Complex{
   double imaginary_ = 0.0;
   double real_ = 0.0;

   constexpr Complex(double real, double imaginary)
      : real_(real), imaginary_(imaginary){ }
};

constexpr Complex operator"" _i(long double value){
   return Complex(0, value);
}

std::ostream& operator<<(std::ostream& stream,
                  const Complex& complex) {
   stream << "(" << complex.real_ << ", " << complex.imaginary_ << "i)";
   return stream;
 }

Complex operator+(const Complex& a, const Complex& b)
{
   return Complex(a.real_ + b.real_, a.imaginary_ + b.imaginary_);
}

Complex operator+(long double a, const Complex& b)
{
   return Complex(a + b.real_, b.imaginary_);
}

int main()
{
   Complex a(1.0, 0.0);
   Complex b = 1.0_i;
   Complex c = 2.0 + 2.0_i;
   Complex d = a+b+c;

   std::cout << "a =  " << a << "\nb = " << b << "\nc = " << c << "\nd = " << d << "\n";

   return 0;
}
```

Questão 9

Quais os tipos exatos do números da função f( )?

main.cpp

```
#include <iostream>
#include <typeinfo>
```

```cpp
int main()
{
        std::cout   << "Type of 100: \t"      << typeid(100).name()       << std::endl
            << "Type of 100u: \t"     << typeid(100u).name()      << std::endl
            << "Type of 100UL: \t"    << typeid(100UL).name()     << std::endl
            << "Type of 100LL: \t"    << typeid(100LL).name()     << std::endl
            << "Type of 3.14: \t"     << typeid(3.14).name()      << std::endl
            << "Type of 3.14f: \t"    << typeid(3.14f).name()     << std::endl
            << "Type of 3.14L: \t"    << typeid(3.14L).name()     << std::endl;

   return 0;
}
```

```
Type of 100:        i
Type of 100u:       j
Type of 100UL:      m
Type of 100LL:      x
Type of 3.14:       d
Type of 3.14f:      f
Type of 3.14L:      e
```

Questão 10

Utilize static assert para verificar o tamanho dos tipos primitivos numéricos de sua plataforma.

main.cpp

```cpp
#include <iostream>
#include <typeinfo>

constexpr int threshold = 100;

int main()
{
   static_assert(sizeof(int) > threshold, "not portable");
   static_assert(sizeof(unsigned int) > threshold, "not portable");
   static_assert(sizeof(unsigned long) > threshold, "not portable");
   static_assert(sizeof(long long) > threshold, "not portable");
   static_assert(sizeof(double) > threshold, "not portable");
   static_assert(sizeof(float) > threshold, "not portable");
   static_assert(sizeof(long double) > threshold, "not portable");

   return 0;
}
```