Programação Orientada a Objeto com C++ Moderno

C++ Guia de Referência

Professor André Duarte Bueno
https://sites.google.com/view/professorandreduartebueno/
email: bueno@lenep.uenf.br
UENF-CCT-LENEP-LDSC

May 26, 2022

- Este guia foi desenvolvido na UENF/CCT/LENEP/LDSC
- Laboratório de Desenvolvimento de Software Científico - LDSC
  - `http://www.lenep.uenf.br/~ldsc`
- do Laboratório de Engenharia e Exploração de Petróleo - LENEP
  - `http://www.lenep.uenf.br`
- do Centro de Ciências e Tecnologia - CCT
  - `http://www.cct.uenf.br`
- da Universidade Estadual do Norte Fluminense - Darcy Ribeiro - UENF
  - `http://www.uenf.br`
- Site do Professor André Duarte Bueno
  - `https://sites.google.com/view/professorandreduartebueno/`

Chapter 1

Guia de Referência C++11/C++14/C++17/C++20/C++23

Prof. André D. Bueno

A listagem 1.1 apresenta um resumo dos principais comandos de C++11/C++14/C++17/C++20/C++23.

Listing 1.1: Um resumo de c++20 - C++14.

```
/**
================================================================================
Lista com as principais novidades de C++11/14/17/20
Autor: André Duarte Bueno.
Nota: procure nos diretórios por C++11-14-17-20.cpp
Referencias: livros de C++11/14/17/20
Sites: isocpp.org herbsutter.com en.cppreference.com

================================================================================
Lista com os principais arquivos #include
================================================================================
*/
#include <algorithm>    // Algoritmos da STL
#include <functional>   // Funções
#include <tuple>        // Tuplas
#include <chrono>       // Biblioteca date time no C++11
#include <regex>            // Biblioteca regex, replace, match_results
#include <random>       // Biblioteca números randomicos e estatística
#include <thread>       // Biblioteca threads
#include <mutex>                // Bibliotecas processamento paralelo
#include <condition_variable>
#include <filesystem>
namespace fs = std::filesystem;

================================================================================
Lista com as principais novidades de C++11 -> Núcleo da linguagem
================================================================================
// [ ] ------------------------------------------------------------------------> auto
auto x = 3;
auto x { 1 };    // inteiro no C++17, lista inicialização no  C++11

// [ ] ------------------------------------------------------------------------> decltype
decltype(x) y;

// [ ] ------------------------------------------------------------------------> nullptr
char *pc = nullptr;     // OK
int  *pi = nullptr;     // OK

// [ ] ------------------------------------------------------------------------> rvalue reference
template<typename T> void f(T&& param);                 // parâmetro do tipo rvalue reference
template<typename T> void f(std::vector<T>&& param);    // parâmetro do tipo rvalue reference
int main() { int&& var1 = 10;                           // rvalue reference (nome para uma
    constante)
cout << "\nvar1 = " << var1 ; }

// [ ] ------------------------------------------------------------------------> novas strings
char     s8[]  = u8"UTF-8 cstring. ";                   // const char[].
char16_t s16[] = u"UTF-16 cstring. ";                   // const char16_t[].
char32_t s32[] = U"UTF-32 cstring. ";                   // const char32_t[].
cout << u8"This is a Unicode Character: \u2018"       << endl;

// [ ] ------------------------------------------------------------------------> raw strings
// C++11 fornece a opção de raw strings literal: R("string \t 1"), não é interpretada
cout <<  "(\a xx \b yy \t zz \n)" << endl;
cout << R"(\a xx \b yy \t zz \n)" << endl;
```

```
55
56 // [ ] ------------------------------------------------------------------> sizeof
57 // C++11 permite obter o sizeof de membro da classe
58 class CPonto { public: double x; double y;};
59 cout << "sizeof(CPonto) = "     << sizeof(CPonto)    << endl;
60 cout << "sizeof(CPonto::x) = " << sizeof(CPonto::x) << endl;
61
62 // [ ] ------------------------------------------------------------------> enum class
63 // A vantagem de enum class é que a mesma não pode ser convertida para int;
64 enum class EDiaSemanaCpp11 { segunda = 2, terca = 3, quarta = 4, quinta = 5, sexta = 6 };
65
66 // [ ] ------------------------------------------------------------------> enum class: tipo
67 // Também podemos definir o tipo usado pela enumeração
68 enum class EMesesAnoCpp11: unsigned int { janeiro = 1, fevereiro, marco, abril,
69 maio, junho, julho, agosto, setembro, outubro, novembro, dezembro };
70
71 // [ ] ------------------------------------------------------------------> union
72 union UMiscelanea {   bool b;  int i;  double d;
73   CPonto p;   // C++11 retirou restrição dos tipos aceitos em uniões
74   UMisccelanea() { new( &p ) CPonto(); }
75 };
76 // [ ] ------------------------------------------------------------------> static_assert
77 // Mensagem de erro em tempo de compitalção
78 static_assert(constant-expression, error-message);
79 static_assert(sizeof(int) <= sizeof(T), "A dimensão de T não é suficiente!");
80 static_assert(std::is_integral<T>::value, "O parâmetro de f deve ser do tipo integral.");
81 static_assert((pi < 3.14) && ( pi > 3.15), "Aumentar precisão de pi!");
82 static_assert(constant-expression); // C++17 sem mensagem
83
84 // [ ] ------------------------------------------------------------------> constexpr
85 constexpr int FuncaoConstante() { return 5; }
86 constexpr int XY (int x, int y) { return x * y; }
87 constexpr double aceleracaoGravidade = 9.8;
88
89 // [ ] ------------------------------------------------------------------> range based for
90 int vetor_estilo_c[5] = { 1, 2, 3, 4, 5 };
91 for (int &elemento_vetor : vetor_estilo_c) cout << elemento_vetor << endl;
92 vector<int> v(5);
93 for( auto elemento_vetor : v )              cout << elemento_vetor << endl;
94 for (int i = 0; auto aluno_i: aluno) {      cout << "\nAluno " << i++ << '\n';
95       aluno_i.Entrada ();    }
96
97 // [ ] ------------------------------------------------------------> Inicialização
       uniforme
98 int x3{5.0}, x4 ={5.3}; // Com inicialização uniforme de C++11 aponta erro pois 5.0 é double
99 int x5, x6{};           // x5 valor indefinido, x6 valor padrão = 0
100 int *ptr1,*ptr2{};      // ptr1 valor indefinido, ptr2 em C++11 assume valor nullptr
101 char cx{14},cy{35000};  // cx OK, 14 é do tipo inteiro; cy Erro, estouro do intervalo
102 std::vector<int> vx { 0, 1, 2, 4, 5 };    // OK
103 std::vector<int> vy { 1, 2.3, 4, 5.6 };   // Erro
104 int vc1[3];   // Cria vetor estilo de C, com 3 elementos [0->2], valores indefinidos
105 int vc2 []= {1,2,3};  // Cria vetor estilo de C, com 3 elementos [0->2], valores definidos
106
107 // [ ] ------------------------------------------------------------------> Lista inicialização
108 class CLista { private:  vector< float > v;
109                public:   CLista( std::initializer_list<float> lista ): v(lista){};
110                          vector< float > V()    { return v; }; };
111
112 class CPonto {  double x;   double y;        // Construtor
113 CPonto(double _x, double _y): x(_x), y(_y) { std::cout << "Passou pelo construtor de SPonto;"
       ;} };
114 CPonto GetCPonto() { return { 0.0, 0.9 }; } // O objeto é criado sem usarmos o tipo CPonto
115
116 void Saida ( std::initializer_list<int> l)  // Note que recebe como parâmetro a lista l.
117 {    for (auto it = l.begin(); it != l.end(); ++it) std::cout << *it << "\n"; }
118
119 int main() {                                // Chama função saida, imprime valores na tela
120   Saida ({0,1,2,3,4,5,6,7,8,9});            // Usando initializer_list
121   CLista lista{ 5.1, 5.2, 5.3, 5.4 };       // Cria objeto lista
122   vector<int> v{ 1,2,3,4,5 };               // Usando initializer_list com biblioteca padrão
123
124   CPonto p2  { 5.1 , 6.1 };                 // Usando inicialização uniforme padrão C++11
125   // Criando uma lista de pontos usando inicialização uniforme padrão C++11
126   CPonto lista_pontos[4] =  { { 5.2,6.2 }, {5.3,6.3}, {5.4,6.4}, {5.5,6.5} };
127 }
128
129 // [ ] ------------------------------------------------------------------> Funções lambda
130 /*auto nomeFuncao = [captura](parametros)->tipoRetorno {corpo da funcao}
131 []: Não capturar nada.
132 [=]: Todas as variáveis externas são capturadas por valor.
133 [&]: Todas as variáveis externas são capturadas por referência.
```

```
[x, &y]: capturar x por valor(cópia) e y por referência.
[&, x]: Todas as variáveis externas são capturadas por referência, exceto x que é por valor.
[=, &z]: Todas as variáveis externas são capturadas por valor, exceto z que é por referência.
    */

// Função lambda anônima é criada e já executada. O () executa a função.
[] { std::cout << "Função lambda criada e já executada" << std::endl;   } ();

// Função lambda criada e chamada a seguir
auto l = [] { std::cout << "Função lambda criada e chamada a seguir" << std::endl; };
l();                                                                    // Chama função lambda

// Definição de função lambda que não captura nada e que não recebe parâmetros.
auto ptr_funcao = [] () { cout << "Olá mundo!\n"; };
ptr_funcao();

// Definição de função lambda que não captura nada e que recebe os parâmetros x e y.
auto ptr_funcao2 = [](int x, int y) { return x + y; }
cout << " x + y = " << ptr_funcao2(3,4) << endl;

// Usando função lambda com captura por referencia
int soma = 0;
auto Soma = [&soma]( int x ) { soma += x; cout << "Soma = " << soma << endl; };
Soma(10);

// [ ] ------------------------------------------------------------------> novo formato funções
auto X(int _x) -> void          { x = _x; }   // Nova versão de auto elimina necessidade -> tipo
auto X() -> int                 { return x; }
auto Set(int _x, int _y) -> void;
auto CPonto::Set(int _x, int _y) -> void{ x = _x;  y = _y; }

// [ ] ------------------------------------------------------------------> Ponteiro Função
void FC(int x)       { cout << "FC x=" << x << endl; } // // Declara e define função
void (*pFC)(int) = &FC;                         // ponteiro para função C++98
typedef void (*PonteiroFuncao)(double);         // ponteiro para função C++98

using PonteiroFuncao = void (*)(double);        // ponteiro para função C++11
std::function<void(int)> pF11 = &F11;           // ponteiro para função C++11
auto autopF11 = &F11;                           // ponteiro para função C++11

class C {    void F11(int x)   { cout << "F11 x=" << x << endl; } };
void (C::*pF03)(int) = &C::F03;                 //  ponteiro para função C++98
std::function<void (C&, int)> pF11_ref = &C::F11; // funciona como referencia
std::function<void (C*, int)> pF11_ptr = &C::F11; // funciona como ponteiro


// [ ] ------------------------------------------------------------------> delegação construtor
explicit CPonto (int _x, int _y):x(_x),y(_y)    {}
CPonto(int xy) : CPonto(xy,xy)  {}  // um construtor chama o outro

// [ ] ------------------------------------------------------------------> default e delete
class NonCopyable { public:  // Diz para o compilador desabilitar o operator= (não criar)
  NonCopyable& operator=(const NonCopyable&) = delete;

  // Diz para o compilador desabilitar o construtor de cópia (não criar)
  NonCopyable(const NonCopyable&) = delete;

  // Diz para o compilador criar o construtor default
  NonCopyable() = default;

// [ ] ------------------------------------------------------------------> override e final
class CPonto { virtual auto Entrada() -> void;
               virtual auto Saida() -> void; };

class CCirculo: public CPonto {
    virtual void Entrada() override ; // sobrecreve método virtual da classe base
    virtual void Saida() final; };    // última atualização de Saida

// [ ] ------------------------------------------------------------------> for_each
char s[] = "Olá Mundo!";
for_each( s, s + sizeof(s), [] (char c){ cout << c << endl; });
int vc[] = { 1 , 2 , 3 , 4 , 5 , 6 , 7 };
for_each( begin(vc), end(vc), [](int x) { cout << x << ' ';});
int soma = 0;
for_each( begin(vc), end(vc), [&soma] (int x) { soma += x;});

================================================================================
Lista com as principais novidades de C++11 -> Biblioteca std
================================================================================
// [ ] ------------------------------------------------------------------> array
```

```cpp
#include <array>
std::array<int, 4> array_4_int { {1,2,3,4} };            // Precisa de duplo {}
array<int, 3> array_3_int = {1, 2, 3};                      // Apos = precisa {} simples
array<string, 2> array_2_string = {"a", "b"} ;
sort(array_4_int.begin(), array_4_int.end());
for(auto& ae: array_4_int)          cout << ae << ' ';

// [ ] ------------------------------------------------------------------> all_of any_of none_of
void Teste( vector<int> &v , string msg )
{ cout << "Vetor " << msg << endl;
  if ( all_of(v.begin(), v.end(), [](int ev) { return ev > 0;} ) )   // Todos positivos?
    cout << "Todos positivos\n";
  if ( any_of(v.begin(), v.end(), [](int ev) { return ev > 0;} ) )   // Pelo menos um positivo
      ?
    cout << "Pelo menos um positivo\n"; }
int main() {  vector<int> v1{   1, 2, 3, 4, 5};    Teste( v1 , "v1"  );
              vector<int> v2{ 0,-1, 2, 3, 4, 5};   Teste( v2  , "v2" ); }

// [ ] ------------------------------------------------------------------------> unique_ptr
std::unique_ptr<int> ptr_int3( new int(3) );      // Cria ponteiro e objeto
cout << *ptr_int3 << endl;                        // Usa
unique_ptr<int> ptr_int5  = std::move(ptr_int3); // Transfere propriedade
cout << *ptr_int5 << endl;                        // Usa
ptr_int5.reset();                                 // Destrõe
ptr_int3.reset();                                 // Não faz nada.

// [ ] ------------------------------------------------------------------------> shared_ptr
// Use shared_ptr? quando quizer vários ponteiros apontando para mesmo objeto,
// somente quando o último for deletado o objeto será efetivamente deletado.
shared_ptr<int> ptr_int6(new int(6));             // Cria ponteiro e objeto
cout << *ptr_int6 << endl;                        // Usa
shared_ptr<int> ptr_int7 = ptr_int6;              // ptr7 aponta p/ mesmo objeto que ptr6
cout << *ptr_int7 << endl;
ptr_int6.reset();                                 // Não destrõe e objeto
cout << *ptr_int7 << endl;                        // Usa objeto
ptr_int7.reset();                                 // Agora deleta objeto

// [ ] ------------------------------------------------------------------------> weak_ptr
shared_ptr<int> ptr_int8(new int(8));
cout << *ptr_int8 << endl;
weak_ptr<int> wptr_int8 = ptr_int8;             // ptr_int8 owns the memory.
  {
  shared_ptr<int> ptr_int9 = wptr_int8.lock(); // Agora p8 e p9 acessam a mesma memória.
  cout << *ptr_int9 << endl;
  if( ptr_int9 )                                // Sempre verifique o ponteiro
    cout << *ptr_int9 << endl;                  // Faça algo com ptr_int9
  }     // ptr_int9 é destruído; ptr_int8 volta a ter a propriedade.
  cout << *ptr_int8 << endl;
  ptr_int8.reset();                             // A memoria é deletada.
}

// [ ] ------------------------------------------------------------------------> function
#include <functional>
function<double(double)> fx2 = [](double x) { return x*x;};  // funcao f
function<double(double)> f2x = [](double x) { return 2.0*x;};// funcao g
// Cada vez mais perto da notação matemática! Agrupando funções, como g(f(x));
std::function<double(double)> gf(function<double(double)> f, function<double(double)> g )
     { return [=](double x) { return g(f(x)); };
// Uso de gf, cria funcao fx4, retorna double, recebe funcao
function<double(double)> fx4 = gf(fx2, fx2);
int main() {    double x = 3;
   cout << "x = " << x << end;
   cout << "fx2 = " << fx2(x) << endl;
   cout << "fx4 = " << fx4(x) << endl; }

// [ ] ------------------------------------------------------------------------> bind
// Declara função f que recebe dois parâmetros, um int e uma string
void fis( int x, string s) { cout << "int x = " << x << " string s = " << s << endl; }
int main() {
  fis( 2, " oi tudo bem ");
  // Cria ponteiro para função fis que recebe apenas a string
  std::function<void( string )> fs = std::bind(&fis, 3 , std::placeholders::_1);
  fs("Usando fs, passando apenas a string");
  // Cria ponteiro para função alternativa que recebe apenas o inteiro
  function<void( int )> fi = std::bind(&fis, "Usando fi" ,std::placeholders::_2);
  fi(7); }

// [ ] ------------------------------------------------------------------------> pair
// Mostra uso de tie com pair<> e equivalencia de pair com tuple
  pair<double,double> p = make_pair(1.1,2.2);
  cout << "get<0>(p) = " << get<0>(p)   << " , get<1>(p) = "    << get<1>(p) << endl;
```

```
293   cout << "p.first␣␣␣=␣" << p.first      << "␣,␣p.second␣␣=␣"    << p.second << endl;
294
295 // [ ] ---------------------------------------------------------------------> tuple
296 // Mostra uso de tuple, get<>, tie, pair
297 #include <tuple>
298 // Cria tuple com 3 doubles
299 tuple<double, double, double> notasJoao(8.7,4.2,5.7);
300 cout<< "\nJoao\n"
301       << "P1:␣" << get<0>(notasJoao) << ",␣"     // Acesso aos elementos da tuple
302       << "P2:␣" << get<1>(notasJoao) << ",␣"     // usando funcao get<indice>(objeto_tuple)
303       << "P3:␣" << get<2>(notasJoao) << '\n';
304 std::get<2>(notasJoao) = 6.3;                    // Nota p3 corrigida, usa referencia.
305
306 // Mostra uso da funcao tie() para obter, separadamente, os valores da tuple
307 double n1,n2,n3;
308 tie(n1, n2, n3) = notasJoao;
309 cout<< "\nJoao\n" << "n1:␣" << n1 << ",␣" << "n2:␣" << n2 << ",␣" << "n3:␣" << n3 << '\n';
310
311 auto [a,b,c] = notasJoao; // C++17
312 // [ ] ---------------------------------------------------------------------> forward_as_tuple
313 // forward_as_tuple cria objeto temporario que funciona como uma tupla
314 // para objetos rvalue (right value). Note que como sao rvalue, nao alocam espaco em disco;
315 #include <tuple>        // std::tuple e std::make_tuple
316 // Note que os parametros da tuple sao right value
317 void print_pack (std::tuple<std::string&&,double&&> pack)
318 {   std::cout << std::get<0>(pack) << ",␣" << std::get<1>(pack) << std::endl; }
319 int main() { print_pack (std::forward_as_tuple(string("Joao"), 8.7)); }
320
321 // [ ] ---------------------------------------------------------------------> remove_if
322 bool is_even(int N) { return N % 2 == 0; } // Retorna verdadeiro se for par
323 int main() {   vector<int> v{1,2,3,4,5,6};
324   for_each (v.begin(),v.end(),[](int ev){ cout << ev << '\t'; });  // Vetor v antes de
        remove_if
325   remove_if(v.begin(),v.end(),is_even);
326   for_each (v.begin(),v.end(),[](int ev){ cout << ev << '\t'; });  // Vetor v depois de
        remove_if
327
328   // Efetivamente remove elementos no intervalo final do vetor
329   v2.erase(remove_if(v2.begin(), v2.end(), is_even), v2.end());  }
330
331 // [ ] ---------------------------------------------------------------------> random
332 // O gerador números randomicos tem duas partes; um motor que gera números randomicos
333 // e uma distribuição matemática.
334 // Motores: linear_congruential_engine,subtract_with_carry_engine e mersenne_twister_enginee.
335 // Distribuições: uniform_int_distribution, uniform_real_distribution,
336 // bernoulli_distribution, binomial_distribution, geometric_distribution, poisson_distribution
      ,
337 // normal_distribution, student_t_distribution, chi_squared_distribution,
338 // exponential_distribution, gamma_distribution, lognormal_distribution,
339 // cauchy_distribution, lognormal_distribution, weibull_distribution,
340 // extreme_value_distribution, fisher_f_distribution, negative_binomial_distribution,
341 // discrete_distribution, piecewise_constant_distribution, piecewise_linear_distribution.
342 #include <random>
343 int main()
344 { uniform_int_distribution<int> distribuicao(-20, 20); // Cria distribuição uniforme
345   mt19937 motor;                                       // Cria motor "Mersenne twister MT19937
        "
346   int numeroRandomico = distribuicao(motor);           // Gera número aleatório
347
348   std::normal_distribution<double>  normal(0.0,1.0);   // Normal, media 0 e desvio padrao 1
349   cout << "␣media␣=␣" << normal.mean() << "␣desvio␣padrao␣=␣" << normal.stddev()
350        << "␣max␣=␣" << normal.max() << "␣min␣=␣" << normal.min() << endl;
351   normal = normal_distribution<double>(12,3);          // Seta media = 12 e desvio padrao = 3
352
353 std::default_random_engine motor2;           // Cria motor, usa default
354 auto Normal = std::bind(normal, motor2);     // Cria gerador de número aleatorio
355 vector<double> vna(500);                     // Cria vetor de numeros aleatorios
356 for( double &ev : vna ) ev = Normal();       // Gera números aleatóros
357 }
358
359 // [ ] ---------------------------------------------------------------------> chrono
360 #include <chrono> // Biblioteca date time no C++11
361 #include <ctime>  // Biblioteca date time no C++03 (<time> no C)
362 int main() { // Cria objeto time_point
363   chrono::time_point<chrono::system_clock> start;
364 // Define valor de start como sendo agora (antes do processamento)
365   start = chrono::system_clock::now();
366 // Chama função com determinado tempo de processamento
367   int result = sin(45);
368 // Define valor de end como sendo agora (depois do processamento)
369   auto end = chrono::system_clock::now();
```

```
// count() retorna numero ticks, a diferença é convertida em segundos.
  int elapsed_seconds = chrono::duration_cast<chrono::seconds>(end-start).count();
  time_t end_time = chrono::system_clock::to_time_t(end);
  cout << "Computação terminada em " << ctime(&end_time)
       << "tempo(s) decorrido : " << elapsed_seconds << "s\n";
}

// [ ] ------------------------------------------------------------------> regex (-lregex)
#include <regex>        // regex, replace, match_results
// regex - Classe que representa uma Expressão Regular - ER.
// match_results - representa as ocorrências, casos em que a ER foi encontrada.
// regex_search - função usada para localizar uma ocorrência da ER.
// regex_replace - função que substitue a ocorrência encontrada por outro texto.
// As funções regex_search e regex_replace recebem uma expressão regular e uma string e
// escreve as ocorrências encontradas na estrutura match_results.
int main(){
if (regex_match ("Palmeiras, Campeão Mundial 1951", regex("r") ) )
 cout << "\nA expressão regular \"(ras)\" foi encontrada em \"Palmeiras, Campeão Mundial
      1951\"";

// A procura pela expressao regular er, sera feita em s pela funcao regex_match.
string s ("Palmeiras campeão mundial 1951");     // string a ser pesquisada
regex er ("r)");                                 // expressao regular usada na pesquisa
if (regex_match (s,er))                  // faz a procura
 cout << "\nA expressão regular \"(ras)\" foi encontrada em \"Palmeiras, Campeão Mundial
      1951\"";

// Faz a procura usando iteradores
if ( regex_match ( s.begin(), s.end(), er ) )    cout << "range matched\n";

// o mesmo que match_results<const char*> cm;
cmatch cm;
regex_match ("Palmeiras, Campeão Mundial 1951",cm,er);
cout << "string literal with " << cm.size() << " matches\n";

// o mesmo que match_results<string::const_iterator> sm;
smatch sm;
regex_match (s,sm,er);
cout << "string object with " << sm.size() << " matches\n";

regex_match ( s.cbegin(), s.cend(), sm, er);
cout << "Usando intervalo, foram encontradas " << sm.size() << " ocorrências\n";

// usando os flags de forma explicita:
regex_match ( "subject", sm, er, regex_constants::match_default );
cout << "As ocorrências são: ";
for (unsigned i=0; i<sm.size(); ++i) {    cout << "[" << sm[i] << "] ";  }
}
---
#include <regex>
int main(){ std::string fnames[] = {"foo.txt", "bar.txt", "zoidberg"};
std::regex txt_regex("[a-z]+\\.txt");
for (const auto &fname : fnames)
        std::cout << fname << ": " << std::regex_match(fname, txt_regex) << '\n';}

// [ ] ------------------------------------------------------------------> threads (C
    ++11/14/17)
// g++ -std=c++17 thread-Resumo.cpp -o thread-Resumo -lpthread
#include <thread>
#include <mutex>                // Bibliotecas processamento paralelo
#include <condition_variable>
                                // Função e classe com sobrecarga operador().
void f(){  std::cout << "Olá Mundo - função f." << std::endl; }
class COlaMundo { public:
 COlaMundo()    { cout << "\nConstrutor"; }
 ~COlaMundo()   { cout << "\nDestrutor" ; }
 void operator()(){ Ola();  Mundo(); }
private:
 void Ola()     { cout << "\nOlá" ; }
 void Mundo()   { cout << "\nMundo - classe COlaMundo." ; }
 void Repeat(int n) { while (n--) {Ola();  Mundo();} }
};
void FuncaoDestacada() {                // Função destacada
cout << "Função destacada, id = " << this_thread::get_id() << endl;}
int main() {
std::cout   << "\n================THREADS======================";
  std::thread t1{ f };  // Cria e dispara a thread t1 função f
                        // ...códigos de main em paralelo...
                        // thread de função lambda
  thread t2([](){ cout << "Função lambda." << endl;});
  COlaMundo obj;        // Cria objeto
```

```
448   thread t3{obj};            // Cria a thread t3
449                              // ...códigos de main em paralelo...
450   thread t4( &FuncaoDestacada );
451   t4.detach();               // Torna a thread destacada, t4 não tem mais acesso a join(), usa
          sleep_for
452   t1.join();t2.join();  // Aguarda retorno das threads
453   if(! t4.joinable())   // Verifica se não foi destacada
454         this_thread::sleep_for ( chrono::seconds(1) );  // Aguarda 1 segundo
455   thread t5{&COlaMundo::Repeat,&obj,3}; // executa 3x
456
457   thread t6 = move(t1); // Move t1 para t6
458
459   cout  << "\nMain␣thread,␣this_thread::get_id()␣=␣" << this_thread::get_id()
460         << "\nMain␣thread,␣t.get_id()␣=␣" << t.get_id()
461         << "\nNúmero␣de␣threads,␣thread::hardware_concurrency()␣=␣"
462         << thread::hardware_concurrency() << endl;
463
464   return 0;
465 }
466 std::cout   << "\n================THREADS␣Classes␣Auxiliares======================";
467 /// Uma classe que chama join quando e destruida.
468 class thread_guard {
469     std::thread& t;      /// referencia para uma thread
470 public:
471     /// Construtor deve ser chamado explicitamente
472     explicit thread_guard(std::thread& t_): t(t_)       {}
473     /// Destrutor chama join automaticamente.
474     ~thread_guard() { if(t.joinable()) t.join(); }
475     /// Construtor de copia desativado
476     thread_guard(thread_guard const&)=delete;
477     /// Operador de atribuicao desativado
478     thread_guard& operator=(thread_guard const&)=delete;
479 };
480 /// Neste caso a classe é proprietaria da thread e recebe a mesma no ato de construcao.
481 class scoped_thread {
482     std::thread t;       /// uma thread
483 public:
484     /// Construtor deve ser chamado explicitamente
485     explicit scoped_thread(std::thread t_): t(std::move(t_)) {
486         if( !t.joinable())
487                 throw std::logic_error("Not␣joinable!");        }
488     /// Destrutor chama join automaticamente.
489     ~scoped_thread() { t.join(); }
490     /// Construtor de copia desativado
491     scoped_thread(scoped_thread const&)=delete;
492     /// Operador de atribuicao desativado
493     scoped_thread& operator=(scoped_thread const&)=delete;
494 };
495
496 int main() {
497     COlaMundo obj;
498     // dispara thread com objeto_funcao
499     thread t(obj);
500     // cria thread_guard que chama join automaticamente
501     thread_guard tg(t); // quando sai de escopo.
502     // dispara thread que executa f1, vai chamar join automaticamente
503     scoped_thread st( std::thread ( f1, "thread␣secundaria" ) );
504 }
505 std::cout   << "\n================THREADS␣com␣vector======================";
506 int main() {
507   std::vector<std::thread> vthreads;            // Cria vetor de threads
508   for( int i = 0; i < 4; ++i )
509         vthreads.push_back( std::thread(f) );   // Adiciona funções
510   for( int i = 0; i < 4; ++i ) {                // Adiciona funções lambda
511     vthreads.push_back( thread(
512         [](){   cout    << "Função␣lambda:␣this_thread::get_id()␣=␣"
513                         << this_thread::get_id() << endl;}));
514     }
515   for(auto& t : vthreads)
516       t.join();                                         // Aguarda finalização
517
518   // for_each(vthread.begin(), vthread.end(), std::mem_fun_ref(&std::thread::join));
519 }
520 std::cout   << "\n================THREADS␣Parâmetros======================";
521 void FuncaoParametrosPorCopia(string s) {
522     cout << "FuncaoParametrosPorCopia,␣s␣=␣" << s << endl;
523     s = "string␣s␣modificada␣em␣FuncaoParametrosPorCopia.";
524 }
525 void FuncaoParametroPorPonteiro(string* s) {
526     cout << "FuncaoParametroPorPonteiro,␣s␣=␣" << *s << endl;
527     *s = "string␣s␣modificada␣em␣FuncaoParametroPorPonteiro.";
```

```
528 }
529 void FuncaoParametrosPorReferencia(string& s) {
530     cout << "FuncaoParametrosPorReferencia, s = " << s << endl;
531     s = "string s modificada em FuncaoParametrosPorReferencia.";
532 }
533 int main() {
534     string s = "String passada como parâmetro";
535     thread t_copy(&FuncaoParametrosPorCopia, s);
536     cout << "main thread após FuncaoParametrosPorCopia, s = " << s << endl;
537     thread t_pointer(&FuncaoParametroPorPonteiro, &s);
538     cout << "main thread após FuncaoParametroPorPonteiro, s = " << s << endl;
539 
540     thread t_reference2(&FuncaoParametrosPorReferencia, ref(s));
541     cout << "main thread após FuncaoParametrosPorReferencia, ref(s) = " << s << endl;
542     t_copy.join(); t_pointer.join(); t_reference2.join();
543     return 0;
544 }
545 std::cout   << "\n================THREADS MUTEX======================";
546 class CContador  {
547     std::mutex mutex_c;     // Cria um mutex = mutual exclusion
548     int contador{0};
549 public:
550     void operator++() {
551         mutex_c.lock();     // Bloqueia
552         ++contador;
553         mutex_c.unlock();  // Libera
554       }
555     int Valor() { return contador; }
556 };
557 int main() {
558  CContador contador;
559  vector<thread> vthreads;
560  for( int i = 0; i < 40; ++i ) {
561   vthreads.push_back( thread(
562     [&contador]() { for(int i = 0; i < 100000; ++i){ ++contador; } }));}
563  for(auto& thread : vthreads)
564          thread.join();
565  cout << contador.Valor() << endl;
566  return 0;
567 }
568 int main() {
569         mutex mutex_cout; // Mutual Exclusion para acesso a cout
570         vector<thread> vt;
571         for (int i = 0; i < 5; i++) {
572                 auto t = thread([i,&mutex_cout]() {
573                         mutex_cout.lock();       // Uso de lock
574                         cout << "thread function: " << i << "\n";
575                         mutex_cout.unlock();     // Uso de unlock
576                 });
577                 vt.push_back(move(t));           // Uso de move
578         }
579         for (int i = -5; i < 1; i++) {
580           mutex_cout.lock();                     // Uso de lock
581           cout << "main thread i =" << i << "\n";
582           mutex_cout.unlock();                   // Uso de unlock
583         }
584         for_each(vt.begin(), vt.end(), [](thread &t) {
585                 assert(t.joinable());
586                 t.join();
587         });
588         return 0;
589 }
590 mutex mcout;                                    // mutex com std::lock_guard<std::mutex>
591 void f1() {   for (int k = 100; k > 0; k--) {
592         // lock_guard chama lock no construtor e unlock no destrutor
593         std::lock_guard<std::mutex> l(mcout);
594         ...código compartilhado... uso de cout
595         }
596 }
597 
598 std::cout   << "\n================THREADS VARIÁVEL CONDICIONAL======================";
599 std::mutex mut;                                          // Cria mutex
600 std::queue<CData> data_queue;
601 std::condition_variable data_cond;                 // Cria variavel condicional
602 void data_preparation_thread() {                   // Prepara dados
603     while(more_data_to_prepare()) {
604         CData const data = prepare_data();     // Gera dados
605         std::lock_guard<std::mutex> lk(mut);   // Bloqueia mutex
606         data_queue.push(data);                          // Usa base de dados
607         data_cond.notify_one();                         // Notifica variavel condicional
608     }   }
```

```
609 void data_processing_thread() {
610     while(true) {
611         std::unique_lock<std::mutex> lk(mut);      // Bloqueia mutex
612                                                    // Aguarda variavel condicional
613         data_cond.wait(lk,[]{return !data_queue.empty();}); // liberacao de dados
614         CData data=data_queue.front();             // Usa base de dados
615         data_queue.pop();
616         lk.unlock();                                          // Desbloqueia o mutex
617         process(data);                                        // Processa dados
618         if(is_last_chunk(data))                            // Encerra processamento
619                 break;
620     }   }
621 int main() {
622     std::thread t1(data_preparation_thread);  // Produz dados
623     std::thread t2(data_processing_thread);       // Consome dados
624     t1.join();    t2.join();
625 }
626
627 std::cout   << "\n================THREADS␣ASYNC␣FUTURE======================";
628 #include <future>
629 #include <iostream>
630 int  f1() {    return 42;}
631 void f2(std::string const& message)  { cout << (message + "\n"); }
632
633 int main() {
634 // Dispara função usando async
635 auto a1 = async(f2,"1␣-␣Olá␣mundo␣disparado␣usando␣async(write_message\n");
636 // Dispara função write_message usando std::launch::async
637 auto a2 = async( launch::async,f2,"1␣-␣Olá␣mundo␣disparado␣usando␣async(launch::async,
      write_message,..\n" );
638 f2("2␣-␣Olá␣mundo␣disparado␣de␣main\n");
639 // Dispara função usando async usando std::launch::deferred
640 auto a3 = async( launch::deferred, f2,"1␣-␣Olá␣mundo␣disparado␣usando␣async(launch::deferred,
      write_message,\n" );
641 a1.wait();  a2.wait(); a3.wait();  // Aguarda retorno de chamada a async.
642
643 std::future<int> resultado = std::async(processar1);
644 thread t (processar2);
645 std::cout << "The␣answer␣is␣"<< resultado.get() << std::endl;
646 t.join();
647 }
648 int find_the_answer() { throw std::runtime_error("Unable␣to␣find␣the␣answer");}
649 int main() {
650  auto f =  async( find_the_answer ) ;
651  try {      cout << "the␣answer␣is␣"<< f.get() << "\n";     }
652  catch( runtime_error const& e ) { cout << "\nCaught␣exception:␣" << e.what() << endl;}
653 return 0;
654 }
655 #include <condition_variable>
656 #include <mutex>
657 #include <chrono>
658 std::condition_variable cv;
659 bool done;
660 std::mutex m;
661 bool wait_loop(){
662     auto const timeout= std::chrono::steady_clock::now()+
663         std::chrono::milliseconds(500);
664     std::unique_lock<std::mutex> lk(m);
665     while(!done)    {
666         if(cv.wait_until(lk,timeout)==std::cv_status::timeout)
667             break;
668     }
669     return done;
670 }
671 std::cout   << "\n================THREADS␣PROMISE======================";
672 // Usando thread com future.
673 // AguardandoNotificacao é chamada duas vezes; e só continua quando
674 // usuário pressiona enter, executando o cin.get() e a seguir set_value que libera sf.get().
675 #include <future> #include <thread> #include <iostream> #include <sstream>
676 using namespace std;
677 // Recebe id da thread e shared_future
678 void AguardandoNotificacao ( int id , std::shared_future<int> sf ) {
679  ostringstream os;
680  os << "Thread␣" << id <<"␣waiting\n";
681  cout << os.str(); os.str("");           // Chama shared_future.get()
682  os << "Thread␣" << id <<"␣woken,␣val=" << sf.get() << "\n";
683  cout << os.str();
684 }
685 int main()  {
686   std::promise<int> p;                  // Cria um promise
687   auto sf = p.get_future().share();
```

```
688   thread t1(AguardandoNotificacao, 1 ,sf);
689   thread t2(AguardandoNotificacao, 2 ,sf);
690   cout << "Waiting\n"; cin.get();
691   p.set_value(42);
692   t2.join(); t1.join();
693   return 0;
694 }
695
696 ================================================================================
697 Lista com as principais novidades de C++14 -> Núcleo da linguagem
698 ================================================================================
699
700 --> extensão das possibilidades das funções lambda, por exemplo com o uso de auto.
701 auto lambda = [](auto x, auto y) {return x + y;}; //C++14
702 auto lambda = [](int x, int y) {return x + y;}; //C++11
703
704 --> dedução de retorno
705 [=]() -> int { return foo() * 42; } // ok
706 [=] { return foo() * 42; } // ok, deduces "-> some_type"
707
708 --> melhoria no uso de decltype
709 string look_up_a_string_1() { return lookup1(); } //C++11
710 string& look_up_a_string_2() { return lookup2(); }//C++11
711 decltype(auto) look_up_a_string_1() { return lookup1(); }//C++14
712 decltype(auto) look_up_a_string_2() { return lookup2(); }//C++14
713
714 --> extensão do uso de constexpr que agora suportam o uso de if, switch, e de loops (incluindo
       range-based for loops).
715
716 --> uso de auto para determinar o tipo de retorno de uma função:
717 auto Pi() { return 3.1415 };
718
719 --> uso de variable templates.
720
721 --> uso do atributo [[deprecated]].
722
723 --> uso de literais binários
724 auto a1 = 42; // ... decimal
725 auto a2 = 0x2A; // ... hexadecimal
726 auto a3 = 0b101010; // ... binary
727
728 ================================================================================
729 Lista com as principais novidades de C++14 -> Biblioteca std
730 ================================================================================
731 --> uso de literais definidos pelo usuário para tipos de biblioteca padrão (user-defined
      literals for standard library types).
732
733 --> Facilita uso de std::string
734 string s = string("C++11␣não␣permite␣")
735           + string("␣somar␣diretamente␣strings");
736 using namespace std::string_literals;
737 string s = "C++14␣permite,␣"s
738           + "␣basta␣informar␣que␣é␣uma␣string␣adicionando␣s"s;
739
740 --> Facilita uso de chrono
741 using namespace std::literals::chrono_literals;
742 auto duration = 1h + 2min + 3s + 4ms + 5us + 6ns;
743
744 --> uso de sequências de inteiros determinados em tempo de compilação.
745
746 --> uso de std::make_unique - substitiu new.
747 std::unique_ptr<ClasseX> v = std::make_unique<ClasseX>();
748 std::unique_ptr<ClasseX> v = std::make_unique<ClasseX>(p1,p2,p3);
749
750 --> uso de ’␣para␣facilitar␣a␣leitura␣de␣números.
751 double␣x␣=␣123’456’789;
752
753 ================================================================================
754 Lista␣com␣as␣principais␣novidades␣de␣C++17␣->␣Núcleo␣da␣linguagem
755 ================================================================================
756
757 -->␣uso␣de␣assert␣sem␣texto.
758
759 -->␣uso␣de␣typename␣em␣templates␣e␣não␣class.
760
761 -->␣uso␣de␣namespace␣X::Y␣{␣?␣}␣no␣lugar␣de␣namespace␣X␣{␣namespace␣Y␣{␣?␣)).
762
763 -->␣uso␣de␣atributos␣em␣namespaces␣e␣enumerações.
764
765 -->␣uso␣de␣unicialização␣em␣if␣e␣switch.
766 ␣if(inicialização,␣condição)␣{...}
```

```
767 if(int x=4; x+j<4) {...j----...}
768
769 -->uso de if constexpr(expression).
770 funciona como static-if para o C++. Reduz uso de SFINAE.
771 if constexpr (is_floating_point_v<T>) {}
772
773 -->uso de dedução de tipo em construtores permitindo std::pair(5.0, false) no lugar de std::
      pair<double,bool>(5.0, false).
774
775 -->uso de variáveis inline, permitindo definições de variáveis em arquivos de cabeçalho
776
777 -->fold expression, forma compacta para variadic template
778 template<typename... Args>
779 auto Soma(Args...args) {return args+...;}
780
781 -->expecificação de exceções passam a ser parte do tipo da função;
782
783 -->funções lambda declaradas dentro de métodos membro capturam automaticamente this;
784   a captura de *this cria uma cópia do objeto.
785
786 -->agora constexpr pode ser usada com lambda
787 constexpr auto N = [](int n) {return n;}
788 static_assert (N(5)==5);
789
790 -->adicionada variável de pré-processamento __has_include
791
792 -->uso facilitado de tuplas
793 auto [a,b,c] = notasJoao; //C++17 notasJoao é uma tupla
794
795 -->atributos/variáveis inline
796 class X {static inline const double pi = 3.1415; };
797
798 -->static_assert sem mensagem;
799
800 -->u8 characteres literal
801
802 -->hexadecimal floating point decimal
803
804 -->nova especificação para construtor de herança
805
806 -->atributos:
807 [[nodiscard]] Informa se o retorno da função foi desconsiderado!
808 [[maybe_unused]] compilador desconsidera se não for usado
809
810 -->Deprecated:
811   Uso de register foi descontinuado;
812   Removidos os trigraphs;
813   Removido operador operator++(bool);
814   Removidos auto_ptr, random_shufle e outros.
815
816 ================================================================================
817 Lista com as principais novidades de C++17 -> Biblioteca std
818 ================================================================================
819 -->Adiçao da biblioteca std::filesystem baseada em boost::filesystem.
820
821 -->uso de funções matemática especiais - Mathematical Special Functions.
822 bessel, laguerre, hermite, neumann, beta
823
824 -->uso de std::string_view, uma versão leve e rápida de string (somente leitura).
825
826 -->uso de std::optional, para representar objetos opcionais.
827
828 -->uso de std::any, para manter valores únicos de qualquer tipofor holding single values of
      any type.
829
830 -->uso de std::variant, um contêiner de união marcadoa tagged union container.
831
832 -->uso de versões paralelizadas dos containers da STL (mais desempenho).
833
834 -->uso de std::byte.
835
836 -->Remoção de conceitos ultrapassados como std::auto_ptr, std::random_shuffle, e versões de
      adaptadores de função.
837
838 -->uso de Logical operator traits: std::conjunction, std::disjunction and std::negation.
839
840 -->std::uncaught_exceptions, as a replacement of std::uncaught_exception
841
842 -->New insertion functions try_emplace and insert_or_assign for std::map and std::
      unordered_map
843
```

```
--> Uniform container access: std::size, std::empty and std::data

--> Definition of "contiguous iterators"

================================================================================
Lista com as principais novidades de C++17 -> Biblioteca filesystem (C++17)
================================================================================
// Resumo dos comandos da biblioteca filesystem.
// Para compilar: g++ -std=c++17 filesystem-00-resumo.cpp -o resumo -lstdc++fs
#include <filesystem>
namespace fs = std::filesystem;
using namespace std;

int main() {
auto p = fs::current_path();
std::cout    <<"\n================Informações da path======================"
             <<"\nPath completa: "<<p
             <<"\nNote que o Caminho corrente pode ser decomposto nas seguintes partes:"
             <<"\nroot_name()="       <<p.root_name()
             <<"\nroot_directory()="  <<p.root_directory()
             <<"\nroot_path()="       <<p.root_path()
             <<"\nroot_path().string() = "<<p.root_path().string()
             <<"\nparent_path()="     <<p.parent_path()
             <<"\nfileName()="        <<p.fileName()
             <<"\nextension()="       <<p.extension();
             //<<"\nnative_file_string()    = "<<p.native_file_string()
             //<<"\nnative_directory_string() = "<<p.native_directory_string()
             //<<"\nrelative_path().string() = "<<p.relative_path().string()
             //<<"\nbranch_path().string()   = "<<p.branch_path().string()<<endl;
std::cout    <<"\nis_block_file (device)      = "<<is_block_file(p)
             <<"\nis_character_file (device) = "<<is_character_file(p)
             <<"\nis_directory                = "<<is_directory(p)
             <<"\nis_fifo                     = "<<is_fifo(p)
             <<"\nis_other                    = "<<is_other(p)
             <<"\nis_regular_file             = "<<is_regular_file(p)
             <<"\nis_socket                   = "<<is_socket(p)
             <<"\nis_symlink                  = "<<is_symlink    (p)
             <<"\nis_empty                    = "  <<fs::is_empty(p)
             <<"\nexists                      = "  <<fs::exists(p);
    // Também posso criar uma path, um caminho para um arquivo
    fs::path p1 = "/tmp/teste/imagens/img1.pgm";
    ofstream img1(p1);                                  // Cria arquivo
    string spgm = "p4 \n2 2 4\n1 2\n3 4\n";    // Cria string com conteúdo arquivo processado
    img1 << spgm;                                       // Substitui conteúdo do arquivo
    img1.close();                                       // Fecha o arquivo antes de mudar seu nome
cin.get();
    // Arquivo com . no nome
    fs::path p2 = "/tmp/teste/imagens/img1.processada.pbm";
    // Caso em que o arquivo tem . só tem extensão (arquivo oculto no GNU/Linux)
    fs::path p3 = "/tmp/teste/imagens/.arquivoOcultoNoLinux";
std::cout    <<"\n================DISCO======================";
    // Informações de espaço em disco
    fs::space_info home = fs::space("/home");
    cout  <<"\nDiretório"  <<"Capacidade"  <<"Livre"  <<"Disponível\n"
          <<setw(20)<<"/home: "<<setw(20)<<home.capacity <<setw(20)
          <<home.free <<setw(20)<<" "<<home.available  <<'\n';
std::cout    <<"\n================DIRETÓRIOS======================";
    // Criando um diretório e tentando copiar
    fs::path dir1 = "/tmp/teste/imagens";
    fs::path dir2 = "/tmp/teste/imagens.backup";
    fs::create_directory("/tmp/teste/");
    fs::create_directory("/tmp/teste/imagens");
    try {
        fs::copy_file(p1, "/tmp/teste/imagens/img1.pbm");
    } catch(fs::filesystem_error& e) {
        cout <<"\nNão deu para copiar o arquivo /tmp/teste/imagens/img1.pgm: "<<e.what() <<
            '\n';
    }
    try {fs::copy(dir1, dir2);                          // Cópia sem recursão e com recursão
        fs::copy("/tmp/teste/imagens",
                 "/tmp/teste/imagens.backup2", fs::copy_options::recursive);
    } catch(fs::filesystem_error& e) {
        std::cout <<"Não deu para copiar o diretório: "<<e.what() <<'\n'; }
    //fs::remove_all("/tmp/teste"); // Removendo diretório
    // Iterando pelo diretório
    cout <<"\nVai iterar não recursivamente pelo diretório e mostrar as paths :";
    for(auto& arq: fs::directory_iterator("/tmp/teste")) // Itera pelo diretório
        cout <<arq <<'\n';                              // e mostra path dos arquivos
    // Iterando recursivamente pelo diretório
    cout <<"\nVai iterar recursivamente pelo diretório e mostrar as paths :";
    for(auto& arq: fs::recursive_directory_iterator("/tmp/teste")) // Itera recursivamente
```

```
924 ␣␣␣␣␣␣␣␣cout␣<<␣arq␣<<␣'\n';␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣//␣e␣mostra␣arquivos
925
926 std::cout␣␣␣<<␣"\n================ARQUIVOS======================";
927 ␣␣␣␣//␣Criação␣e␣manipulação␣de␣arquivos
928 ␣␣␣␣cout␣<<␣"\nConteúdo␣do␣arquivo:␣\n"␣<<␣ifstream(p1).rdbuf()␣<<␣'\n';
929 ␣␣␣␣auto␣p4␣=␣p1;
930 ␣␣␣␣p4.replace_fileName("img1_processada");␣␣␣␣␣␣//␣Substitui␣nome␣da␣path
931 ␣␣␣␣p4.replace_extension(".pbm");␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣//␣Substitui␣extensão␣da␣path
932 ␣␣␣␣p4␣+=".backup";
933 ␣␣␣␣fs::copy(p1,p4);␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣//␣Cópia␣de␣arquivos
934
935 std::cout␣␣␣<<␣"\n================LINKS=======================";
936 ␣␣␣␣cout␣<<␣"\n\nVai␣criar␣links:";␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣//␣Criando␣links
937 ␣␣␣␣fs::create_hard_link("/tmp/teste/imagens/img1.pbm",␣"/tmp/teste/imagens/hard_link_img1.pbm
        ");
938 ␣␣␣␣fs::path␣p5␣=␣"/tmp/teste/imagens/hard_link_img1.pbm";//␣hard␣link␣simbólico
939 ␣␣␣␣fs::create_symlink("/tmp/teste/imagens/img1.pbm",␣"teste/imagens/symlink_img1.pbm");
940 ␣␣␣␣fs::path␣p6␣=␣"/tmp/teste/imagens/symlink_img1.pbm";␣//␣link␣simbólico
941 ␣␣␣␣fs::create_directory_symlink("/tmp/teste/imagens/","symlink_imagens");
942 ␣␣␣␣fs::path␣p7␣=␣"symlink_imagens";␣␣␣␣␣␣␣␣␣␣␣␣␣␣//␣link␣simbólico␣diretório
943 ␣␣␣␣system("tree␣/tmp/teste");
944 ␣␣std::cout␣<<␣"Os␣caminhos␣para␣p1␣e␣p2␣são␣equivalentes?␣"
945 ␣␣␣␣␣␣␣␣␣␣␣␣␣<<␣equivalent(p1,p2)␣<<␣endl;
946 ␣␣std::cout␣<<␣"O␣tamanho␣do␣p1␣é␣de␣"␣<<␣␣␣file_size(p1)␣<<␣"bytes"␣<<␣endl;
947 ␣␣auto␣ftime␣=␣fs::last_write_time(p1);
948 ␣␣//std::cout␣<<␣"A␣ultima␣escrita␣no␣p1␣ocorreu␣às␣="␣<<␣ftime␣␣<<␣endl;␣//C++20␣verificar
949 ␣␣rename(␣p1,␣␣"/tmp//teste/arquivoRenomeado.pbm"␣);
950 ␣␣std::cout␣<<␣"O␣arquivo␣p1␣foi␣renomeado␣para␣arquivoRenomeado.pbm."␣<<␣endl;
951 ␣␣remove(␣p1␣);␣␣␣std::cout␣<<␣"O␣arquivo␣p1␣foi␣removido."␣<<␣endl;
952 ␣␣fs::remove_all("/tmp/teste");␣//␣Removendo␣diretório
953 return␣0;}
954
955 //␣[␣]␣----------------------------------------------------------------------->␣header␣cstring
956 Defined␣in␣header␣<cstring>
957 void*␣memset(␣void*␣dest,␣int␣ch,␣std::size_t␣count␣);
958 memset(␣ponteiroVetor,␣0,␣vetorSize␣*␣sizeof(tipo)␣);
959
960 ============================================================================
961 Lista␣com␣as␣principais␣novidades␣de␣C++20␣->␣Núcleo␣da␣linguagem
962 ============================================================================
963 -->␣adição␣de␣concepts␣ou␣conceitos,␣expandindo␣as␣possibilidades␣de␣uso␣dos␣templates.
964
965 -->␣adição␣de␣modules␣simplificando␣e␣melhorando␣a␣velocidade␣da␣compilação.
966
967 -->␣adição␣de␣coroutines
968 ␣co_await␣para␣suspender␣a␣execução␣até␣retomar.
969 ␣co_yield␣para␣suspender␣a␣execução␣retornando␣um␣valor.
970 ␣co_return␣para␣completar␣a␣execução␣retornando␣um␣valor.
971
972 -->␣uso␣de␣std::format␣semelhante␣a␣prinft.
973 ␣cout␣<<␣std::format("{}␣{}!",␣"Hello",␣"world",␣"algo␣mais");
974 ␣https://en.cppreference.com/w/cpp/utility/format
975
976 -->␣uso␣de␣[=,␣this]␣para␣capturar␣this.
977
978 -->␣uso␣de␣constinit,␣inicialização␣na␣compilação.
979 ␣const␣char*␣FS()␣{␣return␣"retorna␣cstring␣1";␣}
980 ␣constexpr␣const␣char␣*F(bool␣b)␣{␣return␣b␣?␣"retorna␣cstring␣2"␣:␣FS();␣}
981 ␣constinit␣const␣char␣*c␣=␣F(true);
982
983 -->␣Designated␣initializers[70]␣(based␣on␣the␣C99␣feature,␣and␣common␣G++␣extension)
984
985 -->␣template␣parameter␣lists␣on␣lambdas[72]
986
987 -->␣three-way␣comparison␣using␣the␣"spaceship␣operator",␣operator␣<=>
988
989 -->␣initialization␣of␣an␣additional␣variable␣within␣a␣range-based␣for␣statement[73]
990
991 -->␣lambdas␣in␣unevaluated␣contexts[74][75]
992
993 -->␣default␣constructible␣and␣assignable␣stateless␣lambdas[74][76]
994
995 -->␣allow␣pack␣expansions␣in␣lambda␣init-capture[74][77]
996
997 -->␣string␣literals␣as␣template␣parameters[74][78]
998
999 -->␣removing␣the␣need␣for␣typename␣in␣certain␣circumstances[79]
1000
1001 -->␣new␣standard␣attributes␣[[no_unique_address]],[80]␣[[likely]]␣and␣[[unlikely]][81]
1002
1003 -->␣conditional␣explicit,␣allowing␣the␣explicit␣modifier␣to␣be␣contingent␣on␣a␣boolean␣
```

```
    expression[82]
1004
1005 -->␣expanded␣constexpr:␣virtual␣functions,[83]␣union,[84]␣try␣and␣catch,[85]␣dynamic_cast␣and␣
    typeid,[86]␣std::pointer_traits[87]
1006
1007 -->␣immediate␣functions␣using␣the␣new␣consteval␣keyword[88]
1008
1009 -->␣signed␣integers␣are␣now␣defined␣to␣be␣represented␣using␣two's complement (signed integer
    overflow remains undefined behavior)[89]
1010
1011 --> a revised memory model[90]
1012
1013 --> various improvements to structured bindings (interaction with lambda captures, static and
    thread_local storage duration)[91][92]
1014
1015 --> using on scoped enums[94]
1016
1017 ============================================================================
1018 Lista com as principais novidades de C++20 -> Biblioteca
1019 ============================================================================
1020 --> ranges (The One Ranges Proposal)[96]
1021
1022 --> std::make_shared and std::allocate_shared for arrays[97]
1023
1024 --> atomic smart pointers (such as std::atomic<shared_ptr<T>> and std::atomic<weak_ptr<T>>)
    [98]
1025
1026 --> std::to_address to convert a pointer to a raw pointer[99]
1027
1028 --> calendar and time-zone additions to <chrono>[100]
1029
1030 --> std::span, providing a view to a contiguous array (analogous to std::string_view but span
    can mutate the referenced sequence)[101]
1031
1032 --> std::erase and std::erase_if, simplifying element erasure for most standard containers
    [102]
1033
1034 --> <version> header[103]
1035
1036 --> std::bit_cast<> for type casting of object representations, with less verbosity than
    memcpy() and more ability to exploit compiler internals[104]
1037
1038 --> feature test macros[105]
1039
1040 --> various constexpr library bits[106]
1041
1042 --> smart pointer creation with default initialization[107]
1043
1044 --> std::map::contains method[108]
1045
1046 --> bit operations, such as leading/trailing zero/one count,[109] and log2 operations
    [110][111][112]
1047
1048 --> std::bind_front[113]
1049
1050 --> Attributes in C++20:
1051     [[likely]],
1052     [[unlikely]],
1053     [[no_unique_address]]
1054
1055 ============================================================================
1056 Dicas performance:
1057 ============================================================================
1058 - Na saída para cout tente montar toda a saída e somente depois redirecione, evite diversas
    chamadas ao operator<<.
1059 Ex:
1060     std::cout  << "\n\nBotão␣Help:\n\tAjuda␣do␣aplicativo.\
1061 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣\n\nBotão␣Selecionar␣Material:\
1062 ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣\n\t␣vai␣plotar␣todas␣as␣curvas␣de␣condutividade\n";
1063 - Só use  << std::endl; quando absolutamente necessário.
1064
1065 ============================================================================
1066 Dicas estilo de código (código limpo)
1067 ============================================================================
1068 - Na hora que esta escrevendo a classe pode usar nomes abreviados e, antes de iniciar os
    testes, converter para nomes padrões usando search-replace. Mas isso apenas para objetos
    que terão nomes grandes.
1069
1070 - Nome classe CNomeClasse, ex: class CFuncao; class CFDarcy: public CFuncao;
1071 - Nome objeto CFFuncao funcao2G; CFDarcy functionDarcy
1072 - Nome dos métodos devem indicar ação (verbos);
```

```
ex: funcao->Salvar(fileName); funcao->Ler(fileName);
ex: funcao->SalvarDisco(fileName); funcao->LerDadosDisco(fileName);
ex: funcao2G->Read(fileName); funcao2G->Write(fileName);
ex: functionDarcy->Read(fileName); function_darcy->Plot(grafico);
Note que a codificação fica clara, auto explicativa.
Havendo necessidade pode-se colocar alguns comentários, mas a ideia é que ao usar nomes claros quase tudo fique autoexplicativo.
- Na medida do possível o nome do objeto deve dar uma indicação do nome da classe base da hierarquia a que pertence.


=====================================
Comandos git
=====================================
git add
git add temp.txt
git clone
git clone alex@93.188.160.58:/path/to/repository
git commit - registra alterações
git commit ?m ?coloque sua mensagem aqui?
git status
git push - envia alterações
git push origin master
git checkout - cria ou alterna entre ramos
command git checkout -b <branch-name>
git checkout <branch-name>
git remote - conecta com repositório remoto
git remote ?v
git remote add origin <93.188.160.58>
git branch - listar, criar ou excluir ramos.
git branch
Para excluir um ramo: git branch ?d <branch-name>
git pull - mesclar todas as alterações presentes no repositório remoto para o diretório de trabalho local
git pull
git merge - O comando git merge é usado para mesclar uma ramificação no ramo ativo.
git merge <branch-name>
git diff - listar os conflitos.
git diff --base <file-name>
O seguinte comando é usado para exibir os conflitos entre ramos about-to-be-merged antes de mesclá-los:
git diff <source-branch> <target-branch>
Para simplesmente listar todos os conflitos atuais, use:
git diff
git tag - marcação
git tag 1.1.0 <insert-commitID-here>
git log - exibe uma lista de compromissos em uma ramificação
commit 15f4b6c44b3c8344caasdac9e4be13246e21sadw
Author: Alex Hunter <alexh@gmail.com>
Date:   Mon Oct 1 12:56:29 2016 -0600
git reset - Para redefinir o índice e o diretório de trabalho para o estado do último commit, o comando git reset é usado. Uso:
git reset --hard HEAD
git rm- usado para remover arquivos do índice e do diretório de trabalho. Uso:
git rm filename.txt
git stash - ajuda a salvar as mudanças que não devem ser cometidos imediatamente, mas em uma base temporária. Uso:
git stash
git show - Para visualizar informações sobre qualquer objeto git
git show
git fetch - permite que um usuário obtenha todos os objetos do repositório remoto que atualmente não residem no diretório de trabalho local. Exemplo de uso:
git fetch origin
git ls-tree - exibir um objeto de árvore juntamente com o nome e o modo de cada item e o valor SHA-1 do blob, use o comando git ls-tree. Por exemplo:
git ls-tree HEAD
git cat-file
Usando o valor SHA-1, exiba o tipo de um objeto usando o comando git cat-file. Por exemplo:
git cat-file ?p d670460b4b4aece5915caf5c68d12f560a9fe3e4
git grep
```