DOMINANDO C++ EM 1 ANO

Caderno de Exercícios

59 aulas de prática, do #01 ao #59

C++23 · clang ou gcc · 10 módulos · 91 questões · 2 desafios integradores

Autor Desconhecido · theknowledgeisle.com

Nome do aluno: Início: Término:

Como usar este caderno

Cada módulo reúne um bloco de aulas do curso, com a lista dos artigos que ele cobre. As perguntas conceituais pedem explicações com as suas palavras; os exercícios de código pedem programas: escreva, compile e execute cada um.

Tente sempre resolver sozinho antes de abrir o **Caderno de Respostas**. É no erro que o conhecimento fixa.

Sumário

Módulo	Tema	Aulas	Questões
MÓDULO 1	O que muda ao sair do C	#01–#06	Q1–Q11
MÓDULO 2	Objetos e posse de recursos	#07–#12	Q12–Q22
MÓDULO 3	Herança, polimorfismo e operadores	#13–#16	Q23–Q32
MÓDULO 4	A biblioteca padrão: contêineres, iteradores e algoritmos	#17–#23	Q33–Q44
MÓDULO 5	Programação genérica	#24–#26	Q45–Q50
MÓDULO 6	Erros, valores opcionais e o C++ moderno	#27–#30	Q51–Q58
MÓDULO 7	Concorrência	#31–#34	Q59–Q65
MÓDULO 8	Ferramentas e qualidade	#35–#38	Q66–Q71
MÓDULO 9	Projeto final e extensões: o mini banco de dados, a web e a persistência	#39–#52	Q72–Q83
MÓDULO 10	Jogos	#53–#59	Q84–Q91
DESAFIOS	Desafios integradores	todas	D1–D2

Legenda: nas perguntas, `assim` indica código, nomes de tipos e comandos. As linhas pautadas são para a sua resposta; nos exercícios de código, use também o computador: **todo programa deve ser compilado e executado**.

MÓDULO 1

O que muda ao sair do C

Saída com std::print, referências, const, sobrecarga, std::string, nullptr e auto: o que o C++ muda para quem já domina C.

Artigos 01 a 06

#01 A Filosofia por Trás do C++ e a Saída Elegante do printf
#02 Referências e Ponteiros, Frente a Frente
#03 A Disciplina do const — Promessas que o Compilador Cobra
#04 Um Nome, Vários Sentidos — Sobrecarga de Funções
#05 O Fim do char* — Textos de Verdade com std::string
#06 Adeus NULL, Bem-vindo à Dedução — nullptr e auto

Perguntas conceituais

Q1 **Por que não printf**

O `printf` funciona em C++. Que dois problemas dele o `std::print` (C++23) e o `std::format` resolvem? Dê um exemplo de erro que o `printf` deixa passar e o `std::print` não.

Q2 **Referência ou ponteiro**

Cite três diferenças entre uma referência (`int&`) e um ponteiro (`int*`). Quando o ponteiro continua sendo a escolha certa?

Q3 **const em quatro lugares**

Explique o que cada `const` promete em `const char* const nome(const Pessoa& p) const;`, supondo que a função seja membro de uma classe.

Q4 **Sobrecarga e ambiguidade**

Por que a chamada `f(0)` é ambígua quando existem `void f(long)` e `void f(double)`, e não é quando existe também `void f(int)`?

Q5 **Por que nullptr**

Em C, `NULL` costuma ser `0` ou `((void*)0)`. Que problema isso cria com sobrecarga em C++, e como o `nullptr` o resolve?

Q6 Quando auto atrapalha

O `auto` deduz o tipo pela inicialização. Que tipo tem `x` em cada linha, e qual delas provavelmente é um erro?

```
const std::string nome = "Ana";
auto a = nome;
auto& b = nome;
auto c = {1, 2, 3};
auto d = 5u - 10;
```

Exercícios de código

Q7 Formatando uma tabela

Escreva um programa que imprima, com `std::println`, uma tabela de três produtos (nome, quantidade, preço) com o nome alinhado à esquerda em 12 colunas, a quantidade à direita em 4 e o preço com duas casas decimais à direita em 9. Na última linha, mostre o total.

Q8 Troca com referências

Escreva `void trocar(int& a, int& b)` e compare com a versão de C, que recebe ponteiros. Depois escreva `void dobrar_todos(std::vector<int>& v)` que dobre cada elemento usando um `for` com referência, e mostre o que acontece se você esquecer o `&` no laço.

Q9 Sobrecarga para áreas

Escreva três funções `area` sobrecarregadas: uma para um círculo (raio `double`), uma para um retângulo (largura e altura `double`) e uma para um quadrado de lado inteiro (`int`). Mostre qual é chamada para `area(2)`, `area(2.0)` e `area(2.0, 3.0)`, e explique o que muda se a versão `int` for removida.

Q10 Contando palavras com std::string

Escreva `std::string normalizar(std::string_view s)` que remova espaços do começo e do fim e reduza espaços repetidos a um só. Depois conte as palavras da frase normalizada. Não use `char*` nem `strtok`.

Q11 auto e o tipo verdadeiro

Escreva um programa que percorra um `std::map<std::string, int>` com `for (auto [nome, idade] : pessoas)` e mostre que alterar `idade` dentro do laço não muda o mapa. Corrija com a forma de referência e explique por que `nome` não pode ser alterado nem nela.

MÓDULO 2

Objetos e posse de recursos

Construtores e destrutores, RAII, a regra dos três e dos cinco, ponteiros inteligentes, invariantes e inicialização de membros.

Artigos 07 a 12

#07 Nascimento e Morte de um Objeto — Construtores, Destrutores e o RAII na Prática
#08 Quando Copiar Dá Errado — a Regra dos Três e dos Cinco
#09 Um Dono, Um Recurso — o std::unique_ptr
#10 Posse Compartilhada e seus Perigos — shared_ptr e weak_ptr
#11 Projetando Abstrações — Encapsulamento, Invariantes e o Ponteiro this
#12 A Vida Começa Aqui — Inicialização de Membros a Fundo

Perguntas conceituais

Q12 **RAII em uma frase**

O que significa RAII, e por que ele substitui com vantagem o padrão do C de "`goto fim` e liberar tudo no final"?

Q13 **Regra dos três, dos cinco e do zero**

Uma classe guarda um `char*` alocado com `new[]`. Quais funções especiais ela precisa escrever (regra dos cinco)? E o que diz a regra do zero?

Q14 unique_ptr não copia

Por que `std::unique_ptr` não pode ser copiado, mas pode ser movido? O que acontece com a origem depois de `auto b = std::move(a);`?

Q15 Ciclo de shared_ptr

Dois objetos `No` se apontam mutuamente com `std::shared_ptr`. Por que nenhum deles é destruído quando as variáveis locais saem de escopo? Como o `std::weak_ptr` resolve isso?

Q16 Ordem de inicialização dos membros

O que esta classe imprime para `Janela j(800);`, e por que o compilador avisa?

```
struct Janela {
    int area;
    int largura;
    Janela(int l) : largura(l), area(largura * 2) {}
};
```

Q17 **explicit**

Para que serve `explicit` num construtor de um argumento? Mostre o erro que ele previne com `struct Metros { Metros(double v); };` e uma função `void correr(Metros m)`.

Exercícios de código

Q18 **Um arquivo com RAII**

Escreva uma classe `Arquivo` que abra um `FILE*` no construtor (lançando `std::runtime_error` se falhar) e feche no destrutor, imprimindo uma mensagem em cada um. Proíba a cópia. Use-a numa função que retorna no meio e mostre que o arquivo é fechado mesmo assim.

Q19 Um buffer com a regra dos cinco

Escreva `class Buffer` que guarde `n` inteiros alocados com `new int[n]`, com os cinco membros especiais. Teste cópia, movimento e autoatribuição, e mostre o tamanho da origem depois de movida.

Q20 Uma fábrica com unique_ptr

Escreva uma hierarquia mínima `Forma` (com destrutor virtual e `area()` virtual pura), `Circulo` e `Quadrado`, e uma função `std::unique_ptr<Forma> criar(char tipo, double medida)` que devolva `nullptr` para tipo desconhecido. Guarde as formas num `std::vector<std::unique_ptr<Forma>>` e mostre que todas são destruídas sem nenhum `delete` no código.

Q21 Quebrando o ciclo com weak_ptr

Monte uma árvore com `struct No { std::string nome; std::vector<std::shared_ptr<No>> filhos; std::weak_ptr<No> pai; }` e um destrutor que imprima o nome. Crie uma raiz com dois filhos, mostre o caminho de um filho até a raiz usando `pai.lock()` e confirme que tudo é destruído no fim. Depois diga o que aconteceria com `pai` sendo `shared_ptr`.

Q22 Invariante protegida

Escreva `class Fracao` que mantenha sempre a invariante "denominador positivo e fração irredutível". O construtor deve lançar `std::invalid_argument` para denominador zero. Ofereça `numerador()`, `denominador()` e `somar(const Fracao&)`. Mostre `Fracao(2, -4)` e a soma de 1/6 com 1/3.

MÓDULO 3

Herança, polimorfismo e operadores

A cadeia de construção, funções virtuais, classes abstratas, destrutores virtuais e a sobrecarga de operadores.

Artigos 13 a 16

#13 Uma Classe que Nasce de Outra — Herança e a Cadeia de Construção
#14 Um Método, Vários Comportamentos — Funções Virtuais e Polimorfismo
#15 Contratos e Destruição Segura — Classes Abstratas e Destrutores Virtuais
#16 Ensinando Símbolos ao seu Tipo — Sobrecarga de Operadores

Perguntas conceituais

Q23 Ordem de construção e destruição

Numa hierarquia `Base → Meio → Folha`, em que ordem rodam os construtores e os destrutores ao criar e destruir uma `Folha`? Por que essa ordem é a única segura?

Q24 Fatiamento

O que é *object slicing*? Mostre como ele acontece com `void mostrar(Animal a)` recebendo um `Cachorro`, e como evitá-lo.

Q25 Destrutor virtual

Quando o destrutor da base precisa ser `virtual`? O que acontece em `Base* p = new Derivada; delete p;` quando ele não é?

Q26 override e final

O que `override` e `final` acrescentam, se o programa compila sem eles? Dê um erro que o `override` pega.

Q27 Operadores membros ou livres

Por que `operator+` de um tipo `Vetor2` costuma ser uma função livre, enquanto `operator+=` é membro? E por que `operator<<` para `std::ostream` não pode ser membro de `Vetor2`?

Exercícios de código

Q28 A cadeia de construção

Escreva `Base`, `Meio` e `Folha`, cada uma com um membro do tipo `Rastro` (uma classe que imprime ao nascer e ao morrer) e construtores e destrutores que imprimem. Crie uma `Folha` e explique a saída.

Q29 Polimorfismo e fatiamento

Escreva `Animal` com `virtual std::string som() const`, as derivadas `Cachorro` e `Gato`, e duas funções: `falar_valor(Animal a)` e `falar_ref(const Animal& a)`. Chame as duas com um `Cachorro` e mostre a diferença. Depois percorra um `std::vector<std::unique_ptr<Animal>>`.

Q30 Uma interface abstrata

Defina a interface `Notificador` com `virtual void enviar(const std::string& msg) = 0` e duas implementações, `Email` e `Sms` (que recusa mensagens com mais de 20 caracteres, lançando `std::length_error`). Escreva `void alertar(Notificador& n, const std::string& msg)` que trate o erro, e mostre que `Notificador n;` não compila.

Q31 Operadores de um vetor 2D

Escreva `struct Vetor2 { double x, y; }` com `+=`, `+`, `-`, multiplicação por escalar dos dois lados (`v * 2` e `2 * v`), `==` (use `= default`) e um `std::formatter` para que `std::println("{}", v)` imprima `(x, y)`.

Q32 Comparação com a nave espacial

Escreva `struct Versao { int maior, menor, patch; }` com `auto operator<=>(const Versao&) const = default;` e ordene um vetor de versões com `std::sort`. Explique por que a comparação gerada é a certa aqui, e quando não seria.

MÓDULO 4

A biblioteca padrão: contêineres, iteradores e algoritmos

vector a fundo, string e string_view, map, set e as versões hash, iteradores, <algorithm>, o for de intervalo e lambdas.

Artigos 17 a 23

#17 O Array que Cresce Sozinho — std::vector a Fundo
#18 Texto como Coleção — std::string Revisitada e o std::string_view
#19 Buscas Instantâneas — map, set e suas Versões Hash
#20 O Fio que Une Tudo — Iteradores como Conceito Central
#21 Pare de Escrever Laços — a Biblioteca <algorithm>
#22 Percorrendo com Elegância — o Range-Based for a Fundo
#23 Funções Anônimas e Capturas — Lambdas de Verdade

Perguntas conceituais

Q33 Capacidade do vector

Qual a diferença entre `size()` e `capacity()` num `std::vector`? Por que um `push_back` pode invalidar todos os ponteiros e iteradores para os elementos, e como o `reserve` ajuda?

Q34 O perigo do string_view

Por que esta função tem um bug, e por que ele não é detectado pelo compilador?

```
std::string_view saudacao(const std::string& nome) {
    std::string s = "Olá, " + nome;
    return s;
}
```

Q35 **map ou unordered_map**

Compare `std::map` e `std::unordered_map` em complexidade, ordem de iteração e requisitos sobre a chave. Quando o `map` é a melhor escolha mesmo sendo O(log n)?

Q36 **Categorias de iteradores**

Por que `std::sort` não funciona com `std::list`? O que isso diz sobre as categorias de iteradores?

Q37 **Erase-remove**

Por que `std::remove(v.begin(), v.end(), 0)` não diminui o tamanho do vetor? Como apagar de verdade, antes e depois do C++20?

Q38 Captura por referência

Esta função devolve uma lambda. Por que chamá-la depois é comportamento indefinido, e como corrigir?

```
auto contador() {
    int n = 0;
    return [&n] { return ++n; };
}
```

Exercícios de código

Q39 Observando a capacidade

Escreva um programa que faça 17 `push_back` num `std::vector<int>` vazio, imprimindo o tamanho e a capacidade sempre que a capacidade mudar. Repita com `reserve(17)` antes. O que o resultado diz sobre a estratégia de crescimento da sua biblioteca?

Q40 Frequência de palavras

Leia um texto (pode estar numa `std::string` no programa), conte a frequência de cada palavra em minúsculas com um `std::unordered_map<std::string, int>` e mostre as três mais frequentes, desempatando em ordem alfabética. Use `std::istringstream` para separar as palavras.

Q41 Agenda ordenada com map

Use um `std::map<std::string, std::string>` (nome → telefone) para uma agenda. Insira cinco nomes fora de ordem, liste em ordem alfabética, busque um nome com `find` e liste só os nomes que começam com "Ma" usando `lower_bound`.

Q42 Algoritmos no lugar de laços

Dado `std::vector<int> v{5, 3, 8, 1, 9, 2, 7}`, use só funções de `<algorithm>` e `<numeric>`, sem escrever nenhum `for`, para: (a) somar os elementos; (b) contar os pares; (c) achar o maior; (d) verificar se todos são positivos; (e) gerar um vetor com os quadrados; (f) ordenar de forma decrescente.

Q43 Removendo durante a iteração

Dado um `std::vector<std::string>` com nomes, apague todos os que têm menos de 4 letras de três formas: (a) o laço "ingênuo" com `erase` dentro de um `for` de intervalo (explique por que não fazer); (b) o laço correto com o iterador devolvido por `erase`; (c) `std::erase_if`.

Q44 Lambdas que guardam estado

Escreva `auto criar_contador(int inicio, int passo)` que devolva uma lambda que, a cada chamada, devolve o próximo valor da sequência. Crie dois contadores independentes. Depois use uma lambda com captura por referência para somar, com `std::for_each`, os elementos de um vetor numa variável local.

MÓDULO 5

Programação genérica

Templates de função e de classe, e concepts para dizer o que um tipo precisa oferecer.

Artigos 24 a 26

#24 Escreva uma Vez, Use com Todo Tipo — Templates de Função
#25 Estruturas de Dados para Qualquer Tipo — Templates de Classe
#26 Requisitos Explícitos — Concepts e o Fim dos Erros Ilegíveis

Perguntas conceituais

Q45 Template em vez de macro

Compare `#define MAX(a, b) ((a) > (b) ? (a) : (b))` com `template <typename T> const T& maximo(const T& a, const T& b)`. Cite dois problemas da macro que o template não tem.

Q46 Por que templates ficam no cabeçalho

Por que a definição de uma função template normalmente fica no `.h`, e não no `.cpp` como as funções comuns?

Q47 O ganho dos concepts

O que muda para quem usa a função quando `template <typename T> T somar(...)` vira `template <std::integral T> T somar(...)`? Mostre a diferença na mensagem de erro de `somar(std::string("a"), ...)`.

Exercícios de código

Q48 Um template de função com concept

Escreva `template <typename T> T limitar(T valor, T minimo, T maximo)` que prenda o valor ao intervalo, restrito a tipos com `operator<` (use `std::totally_ordered`). Teste com `int`, `double` e `std::string`, e mostre (em comentário) a chamada que não compila e por quê.

Q49 Uma pilha genérica

Escreva `template <typename T> class Pilha` sobre um `std::vector<T>`, com `empilhar`, `desempilhar` (que devolve `std::optional<T>`, vazio quando a pilha está vazia), `topo` e `tamanho`. Use-a com `int` e com `std::string`.

Q50 **Um concept próprio**

Defina o concept `Desenhavel`: um tipo que tem `desenhar()` devolvendo algo conversível para `std::string` e `area()` devolvendo um tipo de ponto flutuante. Escreva `void relatorio(const Desenhavel auto& f)` e teste com dois tipos sem herança. Mostre por que um terceiro tipo é rejeitado.

MÓDULO 6

Erros, valores opcionais e o C++ moderno

Exceções e RAII, std::optional, std::variant, std::expected e structured bindings.

Artigos 27 a 30

#27 Quando o Contrato se Quebra — Exceções e a Aliança com o RAII
#28 Um Valor que Pode Não Existir — std::optional
#29 Um de Vários, ou o Valor e a Razão da Falha — std::variant e std::expected
#30 O Conforto do C++ Moderno — Structured Bindings e Dedução de Tipos

Perguntas conceituais

Q51 **Exceção ou código de erro**

Em C, erros voltam como valor de retorno. Que dois problemas desse modelo as exceções resolvem, e qual é o custo delas?

Q52 **Garantias de segurança**

Explique as garantias básica, forte e *nothrow*. Qual delas o `std::vector::push_back` oferece?

Q53 **optional, variant ou expected**

Para cada situação, escolha entre `std::optional<T>`, `std::variant<A, B>` e `std::expected<T, E>`, justificando: (a) buscar um usuário por id; (b) converter texto em número, informando por que falhou; (c) um token de uma calculadora, que pode ser número, operador ou parêntese.

Q54 **Structured bindings**

O que `auto [it, inseriu] = mapa.insert({"a", 1});` declara? Por que essa forma é melhor que `std::pair` com `.first` e `.second`?

Exercícios de código

Q55 **Exceções e o desempilhar**

Escreva três funções aninhadas (`nivel1` chama `nivel2`, que chama `nivel3`), cada uma com um objeto local que imprime ao ser destruído. O `nivel3` lança `std::runtime_error`; só o `main` trata. Mostre a ordem das mensagens e explique o *stack unwinding*.

Q56 Convertendo texto com expected

Escreva `std::expected<int, std::string> para_int(std::string_view s)` usando `std::from_chars`, com mensagens diferentes para texto vazio, caractere inválido e número fora do intervalo do `int`. Teste com `"42"`, `""`, `"12a"` e `"99999999999"`, e use `value_or` num dos casos.

Q57 Encadeando com and_then

Usando o `para_int` do exercício anterior, escreva `std::expected<int, std::string> dividir(int a, int b)` (erro para `b == 0`) e calcule, com `and_then` e `transform`, a expressão "converter dois textos, dividir e dobrar o resultado", sem nenhum `if`. Teste um caminho de sucesso e dois de erro.

Q58 Tokens com variant

Escreva um tokenizador para expressões como `"3.5 * (2 + 10)"` que devolva um `std::vector<std::variant<double, char>>`, com números em `double` e operadores e parênteses em `char`. Imprima os tokens com `std::visit`, mostrando o tipo de cada um.

MÓDULO 7

Concorrência

std::thread, mutex e lock_guard, std::atomic, std::async e condition_variable.

Artigos 31 a 34

#31 Fazendo Duas Coisas ao Mesmo Tempo — std::thread
#32 O Cadeado que Protege — std::mutex e lock_guard
#33 Operações Indivisíveis — std::atomic
#34 Tarefas em Segundo Plano e a Arte de Esperar — std::async e condition_variable

Perguntas conceituais

Q59 **join ou detach**

O que acontece se um `std::thread` é destruído sem `join()` nem `detach()`? Por que o `std::jthread` (C++20) existe?

Q60 **Condição de corrida**

Duas threads executam `++contador` 100 mil vezes cada sobre um `int` compartilhado. Por que o resultado quase nunca é 200 mil? Cite duas correções.

Q61 **Deadlock**

Duas threads precisam travar os mutexes `a` e `b`. A primeira trava `a` e depois `b`; a segunda, `b` e depois `a`. Explique o *deadlock* e duas formas de evitá-lo.

Q62 **Espera espúria**

Por que a espera numa `std::condition_variable` deve usar um predicado (`cv.wait(lk, [&]{ return pronto; })`) em vez de um `cv.wait(lk)` simples?

Exercícios de código

Q63 **Corrida de dados medida**

Escreva um programa com quatro threads, cada uma incrementando 100 mil vezes: (a) um `std::atomic<int>`; (b) um `int` protegido por `std::mutex`; (c) um contador local por thread, somado no fim. Os três devem dar 400000. (Não inclua a versão com `int` sem proteção: é comportamento indefinido.)

Q64 Soma paralela com async

Some os números de 1 a 10 milhões dividindo o intervalo em quatro partes, cada uma calculada por um `std::async(std::launch::async, ...)`, e junte os `std::future<long long>`. Compare com a fórmula `n(n+1)/2`.

Q65 Produtor e consumidor

Escreva uma fila protegida por `std::mutex` e `std::condition_variable` (`push` e `pop` bloqueante) e use-a com um produtor que envia os números de 1 a 5 e depois um valor sentinela `-1`, e um consumidor que imprime o dobro de cada número até receber o sentinela.

MÓDULO 8

Ferramentas e qualidade

CMake, sanitizers, clang-tidy, testes com Catch2 e as Core Guidelines.

Artigos 35 a 38

#35 Do Comando Único ao Projeto Real — Construindo com CMake
#36 Redes de Segurança Automáticas — Sanitizers e clang-tidy
#37 Provando que Funciona — Testes Automatizados com Catch2
#38 A Sabedoria Codificada — Core Guidelines e Código Limpo em C++

Perguntas conceituais

Q66 Alvos e visibilidade no CMake

No CMake moderno, o que significam `PRIVATE`, `PUBLIC` e `INTERFACE` em `target_include_directories` e `target_link_libraries`?

Q67 Qual sanitizer

Para cada bug, diga qual sanitizer o encontra: (a) ler `v[10]` num vetor de 10 elementos; (b) `INT_MAX + 1`; (c) duas threads escrevendo a mesma variável sem trava; (d) usar um `string_view` para uma string já destruída.

Q68 O que um teste unitário deve ter

Cite três características de um bom teste unitário e explique por que um teste que depende da ordem de execução dos outros é um problema.

Exercícios de código

Q69 **Um CMakeLists para biblioteca e executável**

Escreva um `CMakeLists.txt` para um projeto com uma biblioteca `calc` (em `src/calc.cpp`, com o cabeçalho público em `include/calc/calc.h`), um executável `app` (`src/main.cpp`) que a usa e um executável de testes `testes` ligado ao Catch2 baixado com `FetchContent`. Exija C++23, ative os avisos e permita ligar o AddressSanitizer com uma opção.

Q70 **Testando sem framework**

Antes do Catch2, escreva um mini-framework de testes com uma macro `VERIFICA(expr)` que registre a falha com arquivo, linha e o texto da expressão, sem interromper os outros testes. Use-o para testar uma função `bool eh_palindromo(std::string_view)` (incluindo um caso que falha de propósito) e devolva um código de saída diferente de zero se algum teste falhar.

Q71 **Seguindo as Core Guidelines**

Reescreva este trecho seguindo as Core Guidelines e explique cada mudança com a regra correspondente (pelo tema, não é preciso decorar o número):

```
int* criar(int n) {
    int* v = (int*)malloc(n * sizeof(int));
    for (int i = 0; i <= n; i++) v[i] = 0;
    return v;
}
```

MÓDULO 9

Projeto final e extensões: o mini banco de dados, a web e a persistência

O mini banco de dados em memória (célula com variant, tabelas com RAII e STL, consultas com templates e lambdas, erros com expected) e as extensões: SQLite, HTTP, HTML gerado e dependências.

Artigos 39 a 52

#39 O Projeto Final Começa — Arquitetando um Mini Banco de Dados em Memória
#40 A Primeira Pedra — Uma Célula Poliforme com std::variant
#41 A Segunda Pedra — Linhas e Tabelas com RAII e STL
#42 A Terceira Pedra — Um Motor de Consultas com Templates e Lambdas
#43 A Quarta Pedra — Tratamento de Erros Coerente com std::expected
#44 A Pedra de Acabamento — Integração, Testes e Verificação do Sistema
#45 A Jornada Completa — Retrospectiva de Um Ano de C++
#46 Extensão — C++ no Mundo da Web e da Persistência
#47 Extensão — Dados que Sobrevivem: Integrando SQLite com RAII
#48 Extensão — O Terminal Vira Rede: Um Servidor HTTP Mínimo
#49 Extensão — Páginas Dinâmicas: HTML Gerado com Templates
#50 Extensão — Além do SQLite: PostgreSQL, MySQL e MongoDB
#51 Extensão — A Arte de Integrar: Dependências e Bibliotecas de Terceiros
#52 O Destino e o Novo Começo — Encerrando a Jornada Completa

Perguntas conceituais

Q72 variant ou herança para a célula

No projeto final, a célula da tabela (`Value`) guarda um inteiro, um real ou um texto num `std::variant`. Que vantagens isso tem sobre uma hierarquia `ValorBase` com `ValorInteiro`, `ValorReal` e `ValorTexto`? Quando a herança seria a melhor escolha?

Q73 Onde ficam as invariantes da tabela

A tabela tem um esquema (nomes e tipos das colunas). Por que validar cada linha **na inserção**, e não na consulta? Que invariante isso garante?

Q74 **Uma política de erros**

O artigo #43 adota `std::expected` para erros de uso (coluna inexistente, tipo errado) e mantém exceções para falhas de programação ou de recurso (memória esgotada). Justifique a divisão.

Q75 **RAII para uma API em C**

A API do SQLite é em C: `sqlite3_open` e `sqlite3_close`, `sqlite3_prepare_v2` e `sqlite3_finalize`. Como o `std::unique_ptr` com um *deleter* próprio transforma esses pares em RAII, sem escrever uma classe completa?

Q76 **Um servidor HTTP mínimo**

Um servidor HTTP feito à mão com *sockets* lê a requisição com `recv`. Por que é um erro assumir que uma chamada a `recv` traz a requisição inteira? E por que um servidor de estudo costuma responder com `Connection: close`?

Q77 **Bibliotecas de terceiros**

Compare três formas de trazer uma biblioteca para um projeto C++: copiar o código para o repositório, `FetchContent` do CMake e um gerenciador de pacotes (vcpkg ou Conan). Qual usar num projeto de equipe?

Exercícios de código

Q78 **Uma célula com variant e ordem entre tipos**

Escreva `class Value` sobre `std::variant<std::int64_t, double, std::string>`, com `para_texto()` usando `std::visit` e um `operator<=>` próprio em que inteiros e reais se comparam pelo valor numérico (2 é menor que 2.5) e todo número vem antes de todo texto. Ordene um vetor misto.

Q79 Tabela com esquema e expected

Escreva `struct Coluna { std::string nome; Tipo tipo; }` (com `enum class Tipo { Inteiro, Texto }`), `struct Erro { std::string mensagem; }` e `class Tabela` com `std::expected<void, Erro> inserir(std::vector<Valor> linha)`, que recuse linhas com número errado de células ou tipo errado. Use `using Valor = std::variant<long, std::string>;`. Mostre uma inserção válida e duas inválidas.

Q80 Um motor de consultas fluente

Escreva `template <typename T> class Consulta` que receba um `std::vector<T>` e ofereça `onde(pred)`, `ordenar_por(projecao)` e `limite(n)`, cada um devolvendo `Consulta&` para encadear, e `executar()` que devolve o resultado. Use-a para listar os dois livros mais antigos publicados depois de 1900.

Q81 SQLite com RAII e parâmetros

Usando a API C do SQLite (`<sqlite3.h>`, compile com `-lsqlite3`), crie um banco em memória com a tabela `livros(titulo, ano)`, insira três livros com um *statement* preparado e parâmetros `?` (um dos títulos deve ser uma tentativa de injeção de SQL) e liste-os em ordem de ano. Administre o banco e os *statements* com `std::unique_ptr` e *deleters* próprios.

Q82 Analisando a linha de requisição HTTP

Sem *sockets*, escreva `std::expected<Requisicao, std::string> analisar(std::string_view linha)` que separe uma linha como `GET /livros?autor=ramos HTTP/1.1` em método, caminho, *query string* e versão. Recuse métodos fora de GET, POST, PUT e DELETE, caminhos que não começam com `/` e versões que não começam com `HTTP/`.

Q83 HTML gerado com segurança

Escreva `std::string escapar(std::string_view)` que troque `& < > " '` pelas entidades HTML e `std::string renderizar(std::string_view modelo, const std::map<std::string, std::string>& vars)` que substitua `{{nome}}` pelo valor escapado. Mostre o resultado com um nome que contém `<script>`.

MÓDULO 10

Jogos

O game loop, raylib, o padrão ECS, colisões, recursos com RAII e estados de jogo: Pong, Snake e Breakout.

Artigos 53 a 59

#53 Um Novo Território — Por que Jogos, o Game Loop e as Três Bibliotecas
#54 O Primeiro Pixel — Janela, Desenho e Input com raylib
#55 Entidades de Verdade — Componentes e o Padrão ECS
#56 Quando as Coisas se Tocam — Colisão e o Primeiro Jogo: Pong
#57 Imagem e Som — Recursos de Jogo com RAII, e o Início do Snake
#58 Menu, Jogo, Fim — Estados de Jogo e o Snake Completo
#59 O Capstone dos Jogos — Breakout e o Fim da Jornada

Perguntas conceituais

Q84 O game loop

Descreva as três etapas de um *game loop* e explique por que usar o tempo do quadro (*delta time*) no movimento. Que problema o passo fixo resolve que o *delta time* variável não resolve?

Q85 ECS ou herança

Num jogo, por que uma hierarquia como `Entidade → Movel → Inimigo → InimigoVoador` tende a se complicar? Como o padrão ECS (*Entity Component System*) organiza o mesmo jogo?

Q86 **Recursos de jogo e a ordem de destruição**

Uma classe `Textura` com RAII chama `LoadTexture` no construtor e `UnloadTexture` no destrutor. Por que declarar uma `Textura` global quebra o programa, e como organizar a vida dos recursos em relação a `InitWindow` e `CloseWindow`?

Exercícios de código

Q87 **Passo fixo na prática**

Simule um *game loop* sem gráficos: os quadros chegam com tempos de 0,016 s, 0,033 s, 0,008 s, 0,050 s e 0,016 s. A física roda em passos fixos de 0,02 s, com um objeto a 100 unidades por segundo. Para cada quadro, mostre quantos passos de física rodaram, a posição e o tempo que sobrou no acumulador.

Q88 **Colisão AABB e o rebote do Pong**

Escreva `struct Retangulo { float x, y, w, h; }` e `bool colide(const Retangulo&, const Retangulo&)` (caixas alinhadas aos eixos). Simule uma bola de Pong que anda 4 unidades por passo em x e 1 em y e rebate numa raquete em x = 20: ao colidir, inverta a velocidade horizontal e empurre a bola para fora da raquete. Mostre os passos em que houve colisão.

Q89 Um ECS mínimo

Escreva um ECS em que as entidades são índices, os componentes `Posicao`, `Velocidade` e `Vida` ficam em `std::vector<std::optional<T>>`, e dois sistemas: `mover` (atua em quem tem posição e velocidade) e `envenenar` (tira 1 de vida por passo e marca para remoção quem chega a zero). Crie um jogador, uma moeda parada e um inimigo envenenado, e rode três passos.

Q90 A cobra do Snake

Represente a cobra do Snake como um `std::deque<Ponto>` (a cabeça na frente). Escreva `bool andar(Direcao d, bool comeu)` que acrescente a nova cabeça, remova a cauda quando a cobra não comeu e devolva `false` se a cabeça bater no próprio corpo. Simule uma sequência de movimentos que termina em colisão.

Q91 **Estados de jogo**

Modele os estados `Menu`, `Jogando`, `Pausado` e `FimDeJogo` com um `enum class` e uma função `Estado proximo(Estado atual, Evento e)` com as transições: Enter no menu começa o jogo; P pausa e despausa; perder vida leva ao fim de jogo; Enter no fim volta ao menu. Eventos que não se aplicam ao estado atual são ignorados. Rode uma sequência de eventos e mostre cada transição.

PARA IR ALÉM

Desafios integradores

Projetos maiores, que juntam vários módulos. Não há uma única resposta certa: o gabarito mostra um caminho.

D1

Uma calculadora de terminal

Escreva uma calculadora de terminal com sete operações: soma, subtração, multiplicação, divisão, potência (expoente inteiro), porcentagem (`15 % 200` é 15% de 200) e raiz quadrada (`raiz 2`). Ela deve: (a) aceitar vírgula ou ponto como separador decimal; (b) guardar o último resultado em `ans` , utilizável na conta seguinte; (c) recusar divisão por zero, raiz de negativo, expoente fracionário, resultado que estoura o `double` e texto que não é número com uma mensagem clara, **sem encerrar o programa**; (d) ter os comandos `hist` (histórico) e `sair` . Separe o núcleo (`calc.h` e `calc.cpp` , sem entrada nem saída) da interface (`main.cpp`) e use `std::expected` para os erros.

Sua solução (arquitetura, decisões e código principal):

D2

Um banco chave-valor persistente

Escreva um banco chave-valor de linha de comando com os comandos `set chave valor`, `get chave`, `del chave`, `list` e `sair`. Os dados devem sobreviver entre execuções: cada alteração é acrescentada a um arquivo de log (`set` e `del`), e ao iniciar o programa reconstrói o estado relendo o log. Acrescente um comando `compactar` que reescreve o log só com o estado atual. Teste o núcleo com Catch2.

Sua solução (arquitetura, decisões e código principal):

Bom estudo e bons exercícios!

Dominando C++ em 1 Ano · theknowledgeisle.com