DOMINANDO A LINGUAGEM C EM 1 ANO

Caderno de Exercícios

52 aulas de prática, do #01 ao #52

C17 · gcc ou clang · 9 módulos · 71 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	Fundamentos da linguagem	#01–#06	Q1–Q9
MÓDULO 2	Ponteiros e strings	#07–#10	Q10–Q16
MÓDULO 3	Memória: pilha, heap e vazamentos	#11–#13	Q17–Q23
MÓDULO 4	Tipos próprios, pré-processador e compilação	#14–#20	Q24–Q32
MÓDULO 5	Entrada, saída e biblioteca padrão	#21–#25	Q33–Q40
MÓDULO 6	Ponteiros para função, recursão e bits	#26–#30	Q41–Q47
MÓDULO 7	Estruturas de dados	#31–#35	Q48–Q54
MÓDULO 8	Ferramentas, qualidade e comportamento indefinido	#36–#46	Q55–Q63
MÓDULO 9	Programação de sistemas	#47–#51	Q64–Q71
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

Fundamentos da linguagem

Por que C existe, tipos e memória, funções, decisões, laços e o primeiro vetor.

Artigos 01 a 06

#01 Por que C existe e o que ele quer de você
#02 Variáveis, Tipos e a Memória por Trás Deles
#03 Funções, Escopo e Passagem por Valor
#04 Quando o Programa Decide: if, else e switch
#05 A Arte da Repetição com while, do-while e for
#06 Vetores e o Primeiro Contato com a Memória Contígua

Perguntas conceituais

Q1 **O que C quer de você**

O artigo #01 diz que C "confia no programador". O que isso significa na prática? Cite duas coisas que a linguagem **não** faz por você e que linguagens como Python ou Java fazem.

Q2 **Tamanho dos tipos**

Por que o padrão C não fixa que `int` tem 32 bits? O que ele garante sobre `char`, `short`, `int` e `long`? Quando usar `int32_t` de `<stdint.h>`?

Q3 **Passagem por valor**

"Em C, toda passagem de argumento é por valor." Explique a frase e diga como, então, uma função consegue alterar uma variável de quem a chamou.

Q4 **switch e o break esquecido**

O que acontece quando um `case` de um `switch` não termina com `break`? Dê uma situação em que isso é intencional.

Q5 **Vetor como argumento**

Por que, dentro de uma função `void f(int v[])`, a expressão `sizeof v / sizeof v[0]` não dá o número de elementos? Como as funções da biblioteca resolvem isso?

Exercícios de código

Q6 Divisão inteira e conversões

Sem executar, diga o que o programa imprime. Depois execute e explique cada linha.

```c
#include <stdio.h>

int main(void) {
    int a = 7, b = 2;
    printf("%d\n", a / b);
    printf("%d\n", -7 / 2);
    printf("%d\n", -7 % 2);
    printf("%.2f\n", a / b * 1.0);
    printf("%.2f\n", (double) a / b);
    unsigned int u = 0;
    printf("%u\n", u - 1);
    return 0;
}
```

Q7 Troca com ponteiros

Escreva `void troca(int *a, int *b)` que troca os valores de duas variáveis e mostre, no mesmo programa, por que uma versão `void troca_errada(int a, int b)` não funciona.

Q8 Laços e o do-while

Escreva um programa que leia números inteiros da entrada até chegar um `0` e mostre quantos foram lidos (sem contar o zero), a soma e a média com uma casa decimal. Use `do-while` e trate o caso de nenhum número antes do zero.

Q9 Busca do maior e do segundo maior

Escreva uma função `void dois_maiores(const int v[], int n, int *maior, int *segundo)` que encontra, numa única passada, o maior e o segundo maior valor **distintos** do vetor. Teste com `{5, 9, 2, 9, 7}`.

MÓDULO 2

Ponteiros e strings

Endereços, aritmética de ponteiros, strings e argumentos de linha de comando.

Artigos 07 a 10

#07 Ponteiros: O que É um Endereço de Memória
#08 Aritmética de Ponteiros e a Ligação Profunda com Vetores
#09 Strings: Vetores de Caracteres e a Biblioteca string.h
#10 Ponteiro para Ponteiro e Argumentos de Linha de Comando

Perguntas conceituais

Q10 Vetor e ponteiro não são a mesma coisa

`int v[5]` e `int *p = v` permitem escrever `v[2]` e `p[2]`. Cite duas diferenças reais entre `v` e `p`.

Q11 Aritmética de ponteiros

Se `p` é um `int *` que vale `0x1000` e `int` tem 4 bytes, quanto vale `p + 3`? E se `p` fosse um `char *`? O que significa subtrair dois ponteiros?

Q12 O '\0' e o tamanho da string

Qual a diferença entre `sizeof` e `strlen` aplicados a `char s[20] = "casa";`? Por que `char t[4] = "casa";` é perigoso?

Q13 Literal de string

Por que `char *s = "texto"; s[0] = 'T';` pode derrubar o programa, enquanto `char s[] = "texto"; s[0] = 'T';` funciona?

Exercícios de código

Q14 Percorrendo com ponteiro

Escreva `size_t meu_strlen(const char *s)` sem usar índices, só ponteiros, e `void inverte(char *s)` que inverte a string no lugar. Teste com "ponteiro".

Q15 Contando palavras

Escreva `int conta_palavras(const char *s)` que conta palavras separadas por um ou mais espaços, tabs ou quebras de linha, usando `isspace` de `<ctype.h>`. Teste com `" uma frase\tcom\n cinco palavras "`.

Q16 Argumentos de linha de comando

Escreva um programa que receba números pela linha de comando e imprima a soma. Argumentos que não forem números inteiros devem ser avisados em `stderr` e ignorados. Use `strtol`, não `atoi`, e diga por quê.

MÓDULO 3

Memória: pilha, heap e vazamentos

Onde cada coisa vive, alocação dinâmica e a caça aos vazamentos.

Artigos 11 a 13

#11 A Pilha e o Heap: Onde Cada Coisa Vive
#12 Alocação Dinâmica: malloc, calloc, realloc e free
#13 Vazamentos de Memória e Como Caçá-los

Perguntas conceituais

Q17 **Pilha ou heap**

Para cada variável, diga onde ela vive e quando deixa de existir: (a) um `int` local de uma função; (b) uma variável `static` dentro de uma função; (c) o bloco devolvido por `malloc`; (d) uma variável global.

Q18 **realloc sem perder memória**

Por que `p = realloc(p, novo_tamanho);` é um padrão perigoso? Escreva a forma correta.

Q19 **calloc e malloc**

Quais as duas diferenças entre `malloc(n * sizeof(int))` e `calloc(n, sizeof(int))`?

Q20 **Dangling pointer**

O que é um *dangling pointer*? Dê dois jeitos de criar um e um hábito que reduz o estrago.

Exercícios de código

Q21 **Vetor que cresce**

Implemente um vetor dinâmico de `int` com `push` que dobra a capacidade quando enche (comece com capacidade 2). Insira os números de 1 a 10 e mostre a capacidade a cada realocação. Libere tudo no fim.

Q22 Duplicando uma string

Escreva `char *duplica(const char *s)`, equivalente ao `strdup` (que só entrou no padrão no C23), e um programa que duplica, modifica a cópia e mostra que o original não mudou.

Q23 Achando o vazamento

O programa abaixo vaza memória. Encontre os dois vazamentos, corrija e diga qual ferramenta do curso confirmaria a correção.

```
#include <stdlib.h>
#include <string.h>

char *saudacao(const char *nome) {
    char *s = malloc(strlen(nome) + 7);
    strcpy(s, "Olá, ");
    strcat(s, nome);
    return s;
}

int main(void) {
    for (int i = 0; i < 3; i++) {
        char *msg = saudacao("Ana");
        msg = saudacao("Bia");
        free(msg);
    }
    return 0;
}
```

MÓDULO 4

Tipos próprios, pré-processador e compilação

Structs, unions, enums, typedef, macros, cabeçalhos e classes de armazenamento.

Artigos 14 a 20

#14 Structs: Criando Seus Próprios Tipos
#15 Ponteiros para Struct e o Operador Seta
#16 Unions e Enums: Economia e Clareza
#17 typedef e o Design de Tipos Legíveis
#18 O Pré-processador: #define, Macros e Inclusões
#19 Compilação Separada: Cabeçalhos e Múltiplos Arquivos
#20 Classes de Armazenamento: static, extern e o Tempo de Vida

Perguntas conceituais

Q24 **Padding em structs**

Por que `sizeof` de `struct { char c; int i; char d; }` costuma ser 12 e não 6? Como reordenar os campos para gastar menos?

Q25 **union**

O que é uma `union` e qual o perigo de ler um campo diferente do último escrito? Dê um uso legítimo.

Q26 Macros com efeito colateral

Por que `#define QUADRADO(x) x * x` está errado duas vezes? Corrija e mostre um caso que nem a versão corrigida resolve.

Q27 static e extern

Explique os dois significados de `static`: numa variável dentro de uma função e numa função ou variável no escopo do arquivo. E para que serve `extern`?

Q28 Include guard

Para que serve o trecho `#ifndef PILHA_H / #define PILHA_H / ... / #endif` num cabeçalho? O que acontece sem ele?

Exercícios de código

Q29 Ponto e retângulo

Defina `typedef struct { double x, y; } Ponto;` e `typedef struct { Ponto canto; double largura, altura; } Retangulo;`. Escreva `int contem(const Retangulo *r, Ponto p)` e teste três pontos, usando o operador seta.

Q30 União etiquetada

Escreva um tipo `Valor` que guarda um inteiro **ou** um texto (até 31 caracteres), com um `enum` dizendo qual. Escreva `void imprime(const Valor *v)` e teste os dois casos. Mostre também o `sizeof` do tipo.

Q31 Contador com static

Escreva uma função `int proximo_id(void)` que devolve 1, 2, 3… a cada chamada, sem variável global. Depois explique por que ela não serve, como está, para um programa com várias threads.

Q32 Macro de depuração

Escreva uma macro `LOG(fmt, ...)` que imprime em `stderr` o arquivo, a linha e a mensagem formatada, e que some completamente quando o programa é compilado com `-DNDEBUG`.

MÓDULO 5

Entrada, saída e biblioteca padrão

printf e scanf a fundo, arquivos de texto e binários, stdlib, math, ctype, time e strings.

Artigos 21 a 25

#21 Entrada e Saída Formatada: printf e scanf a Fundo
#22 Lendo e Gravando Arquivos de Texto
#23 Arquivos Binários e Posicionamento no Disco
#24 Explorando a Biblioteca Padrão: stdlib, math, ctype, time
#25 Manipulação de Strings sem Sofrimento

Perguntas conceituais

Q33 scanf e o '\n' que sobra

Por que, depois de `scanf("%d", &idade);`, um `fgets(nome, sizeof nome, stdin);` parece "pular" a leitura? Como resolver?

Q34 Modo texto e binário

Qual a diferença entre abrir um arquivo com `"r"` e com `"rb"`? Em que sistema ela aparece de verdade?

Q35 feof no lugar errado

Por que o laço `while (!feof(f)) { fgets(linha, sizeof linha, f); ... }` costuma processar a última linha duas vezes?

Q36 **rand e a semente**

O que faz `srand(time(NULL))`? Por que chamá-lo dentro de um laço, antes de cada `rand()`, é um erro?

Exercícios de código

Q37 **Gravando e lendo texto**

Escreva um programa que grava três linhas `nome;nota` num arquivo `notas.txt`, fecha, reabre para leitura e imprime a média, tratando a falha de `fopen`.

Q38 Arquivo binário com fseek

Grave 5 structs `{int id; double preco;}` com `fwrite`, depois leia **só a quarta** usando `fseek`, sem ler as anteriores. Mostre também o tamanho do arquivo com `ftell`.

Q39 Estatísticas com math.h

Escreva um programa que calcula média, desvio padrão (populacional) e mediana do vetor `{4, 8, 15, 16, 23, 42}`, usando `qsort` para a mediana e `sqrt` de `<math.h>`.

Q40 Texto em maiúsculas e sem espaços duplicados

Escreva `void normaliza(char *s)` que, no próprio vetor, converte para maiúsculas, remove espaços do início e do fim e reduz espaços repetidos a um só. Teste com `" olá mundo em C "` e explique o que acontece com o `á`.

MÓDULO 6

Ponteiros para função, recursão e bits

Comportamento como argumento, qsort e bsearch, recursão, máscaras e campos de bits.

Artigos 26 a 30

#26 Ponteiros para Função: Passando Comportamento como Argumento
#27 qsort e bsearch: Ordenando e Buscando com a Biblioteca
#28 Recursão: Quando uma Função Chama a Si Mesma
#29 Manipulação de Bits e Máscaras
#30 Campos de Bits e Estruturas Compactas

Perguntas conceituais

Q41 Lendo declarações

Explique em palavras o que declara cada linha: (a) `int (*f)(int, int);` (b) `int *g(int, int);` (c) `void (*tabela[3])(void);`

Q42 Recursão e a pilha

O que é estouro de pilha (*stack overflow*) numa função recursiva? O que é recursão de cauda e por que o padrão C não garante que ela economize pilha?

Q43 Operações com bits

Escreva as expressões para, num `unsigned` chamado `flags`: ligar o bit 3, desligar o bit 5, inverter o bit 0 e testar se o bit 7 está ligado. Por que usar tipos sem sinal para isso?

Exercícios de código

Q44 **Ordenando structs com qsort**

Ordene um vetor de alunos `{nome, nota}` por nota decrescente e, em caso de empate, por nome em ordem alfabética. Depois use `bsearch` num vetor de inteiros ordenado para achar o 23.

Q45 **Calculadora com tabela de funções**

Escreva uma tabela que associa os caracteres `+ - * /` a funções `double f(double, double)` e um laço que aplica cada operação a 12 e 4, procurando a função pelo símbolo.

Q46 Torres de Hanói

Escreva a solução recursiva das Torres de Hanói que imprime os movimentos para 3 discos e conta quantos foram. Quantos movimentos são necessários para n discos?

Q47 Contando bits e testando potência de 2

Escreva `int conta_bits(unsigned x)` com o truque `x &= x - 1` e `int potencia_de_2(unsigned x)`. Mostre os resultados para 0, 1, 12, 64 e 255, e explique o truque.

MÓDULO 7

Estruturas de dados

Listas simples, duplas e circulares, pilhas, filas, árvores de busca e tabelas hash.

Artigos 31 a 35

#31 Listas Encadeadas: A Estrutura que Só Existe com Ponteiros
#32 Listas Duplamente Encadeadas e Circulares
#33 Pilhas e Filas na Prática
#34 Árvores Binárias de Busca
#35 Tabelas Hash: Busca em Tempo Constante

Perguntas conceituais

Q48 Lista ou vetor

Compare lista encadeada e vetor dinâmico em: acesso ao i-ésimo elemento, inserção no início, uso de memória e aproveitamento do cache do processador. Quando a lista ganha?

Q49 Árvore desbalanceada

Em que ordem de inserção uma árvore binária de busca vira uma "lista"? Qual o custo da busca nesse caso e que tipos de árvore evitam o problema?

Q50 Colisões numa tabela hash

O que é uma colisão numa tabela hash? Explique o encadeamento separado e diga o que é o fator de carga e por que ele importa.

Exercícios de código

Q51 **Lista encadeada com inversão**

Implemente uma lista simplesmente encadeada de `int` com `insere_inicio`, `imprime`, `inverte` (no lugar, sem alocar nós) e `libera`. Teste inserindo 1, 2, 3 e 4.

Q52 **Verificando expressões com uma pilha**

Usando uma pilha (vetor e índice de topo), escreva `int balanceado(const char *s)` que verifica se `()`, `[]` e `{}` estão corretamente aninhados. Teste com quatro expressões.

Q53 Árvore binária de busca

Implemente uma BST de inteiros com `insere`, `busca`, percurso em ordem e `altura`. Insira 50, 30, 70, 20, 40, 60, 80 e mostre a altura; depois mostre a altura inserindo de 1 a 7 em ordem.

Q54 Tabela hash de contagem de palavras

Implemente uma tabela hash com encadeamento (31 posições, função djb2) que conta quantas vezes cada palavra aparece em "o rato roeu a roupa do rei de roma e o rei riu do rato". Imprima as palavras que aparecem mais de uma vez.

MÓDULO 8

Ferramentas, qualidade e comportamento indefinido

GDB, Valgrind, sanitizers, Make, CMake, UB, errno, testes, código limpo e desempenho.

Artigos 36 a 46

#36 Depurando de Verdade com o GDB
#37 Caçando Bugs de Memória com Valgrind e Sanitizers
#38 Automação de Build com Make
#39 Projetos Maiores com CMake
#40 Undefined Behavior: O Território Perigoso do C
#41 Tratamento de Erros Idiomático: errno e Códigos de Retorno
#42 Testes Automatizados em C
#43 Código Limpo em C: Nomes, Funções e Legibilidade
#44 Programação Defensiva e Contratos
#45 Organização de Projetos e Convenções da Comunidade
#46 Otimização Consciente e Noções de Desempenho

Perguntas conceituais

Q55 **Comportamento indefinido**

O que é comportamento indefinido (UB)? Por que "no meu computador funciona" não prova nada? Cite três exemplos.

Q56 **Sanitizers ou Valgrind**

Qual a diferença de funcionamento entre o AddressSanitizer e o Valgrind? Cite uma vantagem de cada.

Q57 **O que o make sabe**

Como o `make` decide o que recompilar? O que acontece se o `Makefile` não declarar que `main.o` depende de `pilha.h`?

Q58 **errno**

Como funciona o `errno`? Por que é errado testar `errno` sem antes verificar o valor de retorno da função?

Q59 **assert não é tratamento de erro**

Para que serve o `assert`? Por que é errado usá-lo para validar a entrada do usuário ou o retorno de `malloc`?

Exercícios de código

Q60 Estouro detectado

O código abaixo calcula a média de dois `int` e falha para valores grandes. Explique o problema e escreva uma versão que nunca estoura. Teste com `INT_MAX` e `INT_MAX - 2`.

```
int media(int a, int b) { return (a + b) / 2; }
```

Q61 Erros por código de retorno

Escreva `int le_inteiro(const char *texto, int *saida)` que devolve 0 em caso de sucesso e códigos de erro distintos para "vazio", "não é número", "sobra texto" e "fora do intervalo de int". Escreva um `main` que testa cinco entradas e mostra a mensagem de cada erro.

Q62 Um teste automatizado mínimo

Sem biblioteca externa, escreva uma macro `CHECA(cond)` que conta testes executados e falhos, mostrando a linha de cada falha. Use-a para testar uma função `int fatorial(int n)` (com o caso n = 0) e mostre o resumo. Inclua de propósito um teste errado para ver a falha.

Q63 Makefile com dependências automáticas

Escreva um `Makefile` para um projeto com `main.c`, `pilha.c` e `pilha.h` que: compila com `-Wall -Wextra -g`, gera as dependências de cabeçalhos automaticamente, tem os alvos `all` e `clean` e não confunde `clean` com um arquivo.

MÓDULO 9

Programação de sistemas

Processos, threads, sincronização, sockets e sinais no Unix.

Artigos 47 a 51

#47 Processos: fork, exec e wait
#48 Threads com pthreads
#49 Sincronização: Mutexes e Variáveis de Condição
#50 Comunicação em Rede com Sockets
#51 Sinais Unix e Eventos do Sistema Operacional

Perguntas conceituais

Q64 **fork**

O que `fork()` devolve no processo pai, no filho e em caso de erro? Por que um `printf("oi")` sem `\n` antes do `fork` pode aparecer duas vezes?

Q65 **Zumbis**

O que é um processo zumbi e como evitá-lo? E um processo órfão?

Q66 **Mutex e variável de condição**

Por que o `pthread_cond_wait` precisa receber o mutex? E por que a condição é testada num `while`, não num `if`?

Q67 **O que fazer num tratador de sinal**

Por que é errado chamar `printf` ou `malloc` dentro de um tratador de sinal? Qual o padrão recomendado?

Exercícios de código

Q68 **fork, exec e wait**

Escreva um programa que cria um processo filho para executar `echo` com os argumentos `"olá do filho"`, espera o filho terminar e mostra o código de saída dele.

Q69 Somando em paralelo com threads

Some os números de 1 a 10.000.000 dividindo o trabalho entre 4 threads, cada uma com o seu intervalo e o seu resultado parcial, sem nenhum mutex. Explique por que não é preciso mutex.

Q70 Contador compartilhado com mutex

Crie 4 threads que incrementam um contador global 100.000 vezes cada. Mostre o resultado com o mutex protegendo o incremento e explique o que muda sem ele.

Q71 Encerrando com Ctrl+C

Escreva um programa que fica num laço "trabalhando" e, ao receber `SIGINT`, termina o laço de forma limpa e imprime quantas voltas deu. Para o teste ser automático, o próprio programa envia o sinal a si mesmo depois de 3 voltas com `raise(SIGINT)`.

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 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.c` e `calc.h`, que não fazem entrada nem saída) da interface (`main.c`).

Sua solução (arquitetura, decisões e código principal):

D2

Um mini-shell

Escreva um mini-shell que mostra um prompt, lê um comando com argumentos, executa-o num processo filho e espera o fim, e tem três comandos internos: `cd`, `historico` (os últimos 10 comandos, numa lista circular) e `sair`. Comandos inexistentes devem mostrar uma mensagem e devolver o prompt. Como extensão, implemente o redirecionamento `> arquivo`.

Sua solução (arquitetura, decisões e código principal):

Bom estudo e bons exercícios!

Dominando a Linguagem C em 1 Ano · theknowledgeisle.com