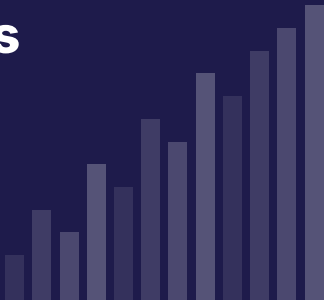

BASE 3 · ANEXO: BASE PRÁTICA EM C

Structs: registros, ponteiros para structs e nós

Kernighan e Ritchie, *C: a linguagem de programação*,
capítulo 6

Estruturas de dados e seus algoritmos · José Dihego · IFBA



Roteiro da aula

Ao final, você será capaz de

- ▶ Declarar structs com typedef e acessar campos com . e ->
- ▶ Construir vetores de structs e structs autorreferentes (nós)
- ▶ Entender alinhamento e preenchimento (sizeof, offsetof)
- ▶ Medir vetor de structs (AoS) contra struct de vetores (SoA)



Teoria

Registros, ->, structs dentro de structs, nós e alinhamento



Prática em C

Turma de alunos: vetor de structs e lista de nós



Experimento

Somar um campo: AoS contra SoA



Conexões

Redes, sistemas de arquivos, jogos e ciência de dados

Registros

- ▶ Uma struct agrupa campos de tipos diferentes: os “dados satélite” do Cormen ficam juntos da chave.
- ▶ `typedef struct { char nome[32]; double nota; } Aluno;` cria o tipo Aluno.
- ▶ `a.nota` acessa o campo; com um ponteiro, `p->nota` é o mesmo que `(*p).nota`.
- ▶ Atribuir uma struct a outra copia **todos** os bytes; passar por valor também. Para structs grandes, passe `const Aluno *`.
- ▶ Structs podem conter vetores, outras structs e ponteiros.

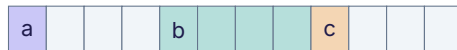
O nó: uma struct que aponta para si mesma

- ▶ `typedef struct No { int chave; struct No *prox; } No;`
- ▶ Dentro da declaração, o nome `No` do `typedef` ainda não existe: por isso `struct No *`.
- ▶ Uma struct não pode conter a si mesma (tamanho infinito), mas pode conter um **ponteiro** para si.
- ▶ Todo nó do curso é assim: listas (`prox`, `ant`), árvores (`esq`, `dir`, `pai`), grafos (listas de adjacência).
- ▶ Criar: `No *x = malloc(sizeof *x); x->chave = k; x->prox = NULL;`

No Cormen, $x.chave$ e $x.prox$ são exatamente `x->chave` e `x->prox` quando x é um ponteiro para nó.

Alinhamento e preenchimento

- ▶ O processador lê um `int` mais rápido (ou só consegue ler) num endereço múltiplo de 4.
- ▶ O compilador insere **buracos** entre os campos e no fim.
- ▶ `struct { char a; int b; char c; }`: 12 bytes. Reordenada como `{ int b; char a; char c; }`: 8 bytes.
- ▶ `offsetof(T, campo)` dá a posição do campo.



12 bytes (cinza = preenchimento)



8 bytes

Num milhão de registros, 4 bytes a menos por registro são 4 MB a menos para o cache carregar.



Prática em C

- ▶ Vetor de structs no heap e a melhor nota
- ▶ Lista de nós com uma struct dentro do nó
- ▶ `sizeof` e `offsetof` das duas ordens de campos

Structs em C (1/4)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stddef.h>
4
5  typedef struct {                               // typedef: dispensa escrever "struct"
6      char nome[32];
7      double nota;
8  } Aluno;
9
10 typedef struct No {                             // autorreferente: o nó de uma lista
11     Aluno dado;                                // struct dentro de struct
12     struct No *prox;                           // aqui ainda não existe o nome No
13 } No;
```

Structs em C (2/4)

```
15  /* Alinhamento: o compilador insere preenchimento para cada campo começar num
16     endereço múltiplo do seu tamanho. A ordem dos campos muda o sizeof. */
17  typedef struct { char a; int b; char c; } Mal;    // 1 + 3 (buraco) + 4 + 1 + 3 = 12
18  typedef struct { int b; char a; char c; } Bem;    // 4 + 1 + 1 + 2 = 8
19
20  void imprime(const Aluno *x) {                    // passar o endereço: não copia a struct
21      printf("%s (%.1f)\n", x->nome, x->nota);      // x->nota é (*x).nota
22  }
```

Structs em C (3/4)

```
24 int main(void) {
25     int n;
26     if (scanf("%d", &n) != 1 || n < 1) return 1;
27     Aluno *turma = malloc(n * sizeof *turma); // vetor de structs no heap
28     No *lista = NULL;
29     for (int i = 0; i < n; i++) {
30         if (scanf("%31s %lf", turma[i].nome, &turma[i].nota) != 2) return 1;
31         No *no = malloc(sizeof *no);
32         no->dado = turma[i]; // atribuição copia a struct inteira
33         no->prox = lista;
34         lista = no;
35     }
36     int melhor = 0;
37     for (int i = 1; i < n; i++)
38         if (turma[i].nota > turma[melhor].nota) melhor = i;
39     printf("melhor: ");
```

Structs em C (4/4)

```
40     imprime(&turma[melhor]);
41     printf("sizeof(Mal) = %zu, sizeof(Bem) = %zu; b começa no byte %zu de Mal\n",
42           sizeof(Mal), sizeof(Bem), offsetof(Mal, b));
43     printf("lista:");
44     while (lista) {
45         No *x = lista;
46         printf(" %s", x->dado.nome);
47         lista = x->prox;
48         free(x);
49     }
50     printf("\n");
51     free(turma);
52     return 0;
53 }
```



Experimento

- ▶ Registro de 64 bytes; soma do campo nota (8 bytes)
- ▶ AoS: `aos[i].nota`; SoA: `notas[i]`
- ▶ n de 4 mil a 8 milhões de registros

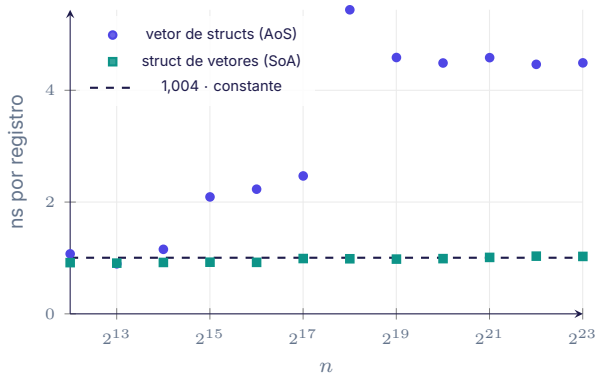
Vetor de structs ou struct de vetores?

Hipótese. No modelo RAM, os dois fazem n somas. Mas cada linha de cache traz 64 bytes: no AoS, só 8 deles são a nota (1/8 aproveitado); no SoA, todos. Enquanto tudo cabe no cache, empate; depois, o AoS deve ficar várias vezes mais lento, e o SoA, constante.

Método

1. Registro: `double nota; char nome[48]; int idade, turma;` (64 bytes)
2. Cerca de 33 milhões de leituras por medida
3. Tempo por registro
4. $n = 2^{12}, \dots, 2^{23}$

Tempo por registro



n	medido	razão
4.096	0,916	0,916
16.384	0,920	0,920
65.536	0,921	0,921
262.144	0,986	0,986
1.048.576	0,988	0,988
8.388.608	1,027	1,027

razão = medido $\div$ constante

✓ desvio máx. 2,8%

Até cerca de 100 mil registros (6 MB), empate. Fora do cache, o AoS fica cerca de 5 vezes mais lento; o SoA quase não muda.

EXPERIMENTO

O que o experimento mostrou

A organização dos dados na memória muda o desempenho em **5 vezes**, com o mesmo algoritmo. Agrupar em structs é o natural para programar; separar em vetores é o natural para o hardware. Bancos colunares, motores de jogos e bibliotecas de ciência de dados escolhem SoA pelo mesmo motivo.

> Reproduza: `gcc -O2 base3_experimento.c -o exp -lm && ./exp`



Conexões

- ▶ Redes
- ▶ Sistemas de arquivos
- ▶ Motores de jogos
- ▶ Ciência de dados

Onde isso aparece na computação

 Redes

Cabeçalhos IP, TCP e Ethernet são structs com campos em posições fixas; o preenchimento errado corrompe pacotes (`__attribute__((packed))`).

 Sistemas de arquivos

O *inode* do Linux e as entradas de diretório são structs gravadas no disco byte a byte.

 Motores de jogos

Arquiteturas *entity component system* guardam cada componente (posição, velocidade) num vetor próprio: SoA para atualizar milhões de objetos por quadro.

 Ciência de dados

NumPy, pandas e Apache Arrow guardam cada coluna num vetor contíguo: é o SoA do experimento, e a razão de serem tão rápidos.

Para casa

Exercícios

1. Defina Ponto e Retangulo (dois pontos) e escreva `double area(const Retangulo *r).`
2. Ordene um vetor de Aluno por nota com `qsort` (veja Base 4).
3. Use `offsetof` para imprimir a posição de cada campo de uma struct com `char`, `double` e `short`.
4. Implemente uma pilha de Aluno com nós (`struct No { Aluno dado; struct No *prox; }`).
5. Reduza o registro do experimento para 16 bytes. A diferença entre AoS e SoA diminui? Por quê?

Leitura

Kernighan e Ritchie, C: a linguagem de programação, capítulo 6 (estruturas).

Código

josedihego.net/estruturas