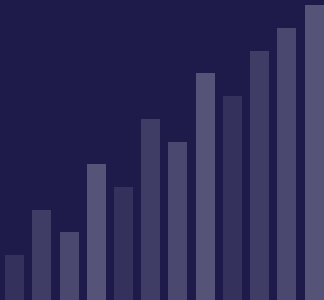

BASE 5 · ANEXO: BASE PRÁTICA EM C

Arquivos, argumentos e erros

Kernighan e Ritchie, *C: a linguagem de programação*,
capítulo 7 e seção 5.10

Estruturas de dados e seus algoritmos · José Dihego · IFBA



Roteiro da aula

Ao final, você será capaz de

- ▶ Ler e escrever arquivos texto e binários
- ▶ Receber parâmetros pela linha de comando (argc, argv)
- ▶ Tratar erros com valores de retorno, perror e assert
- ▶ Medir o tamanho e o tempo de arquivos texto contra binários



Teoria

FILE *, modos de abertura, texto contra binário, argc/argv, erros



Prática em C

Gravar e ler números em texto e em binário



Experimento

Bytes e nanossegundos por número nos dois formatos



Conexões

Bancos de dados, redes, ciência de dados e ferramentas de linha de comando

Arquivos em C

Chamada	Uso
<code>FILE *f = fopen("x.txt", "r")</code>	abre; modos "r", "w", "a", e "rb"/"wb" para binário
<code>fprintf(f, ...), fscanf(f, ...)</code>	texto formatado (como printf/scanf)
<code>fgets(s, tam, f)</code>	uma linha inteira, com limite de tamanho
<code>fwrite(v, tam, n, f), fread(...)</code>	bytes da memória, sem conversão
<code>fseek, ftell</code>	posicionar e consultar a posição
<code>fclose(f)</code>	fecha e grava o que estava no buffer

`stdin`, `stdout` e `stderr` são `FILE *` já abertos: `./prog < entrada.txt > saida.csv` redireciona sem mudar o código.

Texto ou binário?

Texto

- ▶ Legível e portátil (CSV, JSON).
- ▶ Converter número ↔ dígitos custa caro.
- ▶ Tamanho variável: 123456 ocupa 7 bytes com o `\n`.

Binário

- ▶ Os bytes da memória, como estão: `fwrite` de um vetor inteiro de uma vez.
- ▶ Rápido e de tamanho fixo (4 bytes por `int`).
- ▶ Depende da máquina (tamanho dos tipos, ordem dos bytes).

Os experimentos do curso imprimem CSV em texto: lento, mas qualquer ferramenta (planilha, Python, o gerador destes slides) consegue ler.

Argumentos e erros

- ▶ `int main(int argc, char *argv[])`: `argv[0]` é o nome do programa, `argv[1]` o primeiro argumento. Ex.: `./ordena 1000000 aleatorio`.
- ▶ `atoi`, `strtol`: de texto para número (`strtol` detecta erros).
- ▶ **Toda** chamada de E/S pode falhar: `fopen` devolve `NULL`, `fscanf` devolve quantos itens leu. Teste sempre.
- ▶ `perror("x.txt")` imprime o motivo ("No such file or directory") a partir de `errno`.
- ▶ `assert(cond)` aborta se `cond` for falsa: para erros **de programação**, não para entrada do usuário. Some com `-DNDEBUG`.
- ▶ Devolva 0 de `main` em caso de sucesso e outro valor em caso de erro: scripts e o `make` dependem disso.



Prática em C

- ▶ Nome do arquivo opcional em `argv[1]`
- ▶ Grava em texto e em binário; relê e confere
- ▶ `perror`, `assert` e tamanho com `fseek/ftell`

Arquivos e erros em C (1/3)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <assert.h>
4
5  /* Lê n e n inteiros; grava-os num arquivo texto e num binário; lê de volta.
6     Opcionalmente, o nome-base dos arquivos vem da linha de comando (argv[1]). */
7  int main(int argc, char *argv[]) {
8     const char *base = argc > 1 ? argv[1] : "dados";    // argv[0] é o nome do programa
9     char nome_txt[256], nome_bin[256];
10     snprintf(nome_txt, sizeof nome_txt, "%s.txt", base);
11     snprintf(nome_bin, sizeof nome_bin, "%s.bin", base);
12     int n;
13     if (scanf("%d", &n) != 1 || n < 1) return 1;
```

Arquivos e erros em C (2/3)

```
14  int *v = malloc(n * sizeof *v);
15  for (int i = 0; i < n; i++)
16      if (scanf("%d", &v[i]) != 1) return 1;
17  FILE *f = fopen(nome_txt, "w");
18  if (!f) { perror(nome_txt); return 1; } // perror explica o motivo (errno)
19  for (int i = 0; i < n; i++) fprintf(f, "%d\n", v[i]);
20  fclose(f);
21  FILE *g = fopen(nome_bin, "wb");
22  if (!g) { perror(nome_bin); return 1; }
23  size_t escritos = fwrite(v, sizeof *v, n, g); // os bytes da memória, como estão
24  assert(escritos == (size_t)n); // falha aqui = erro de programação
25  fclose(g);
26  f = fopen(nome_txt, "r");
27  long soma = 0;
```

Arquivos e erros em C (3/3)

```
28     int x, lidos = 0;
29     while (fscanf(f, "%d", &x) == 1) { soma += x; lidos++; }
30     fclose(f);
31     printf("média lida do arquivo texto: %.2f\n", (double)soma / lidos);
32     g = fopen(nome_bin, "rb");
33     fseek(g, 0, SEEK_END); // tamanho = posição do fim
34     printf("arquivo binário: %ld bytes\n", ftell(g));
35     fclose(g);
36     remove(nome_txt);
37     remove(nome_bin);
38     free(v);
39     return 0;
40 }
```



Experimento

- ▶ n inteiros de 0 a 999.999
- ▶ Tamanho dos dois arquivos
- ▶ Tempo de gravar e reler, por número

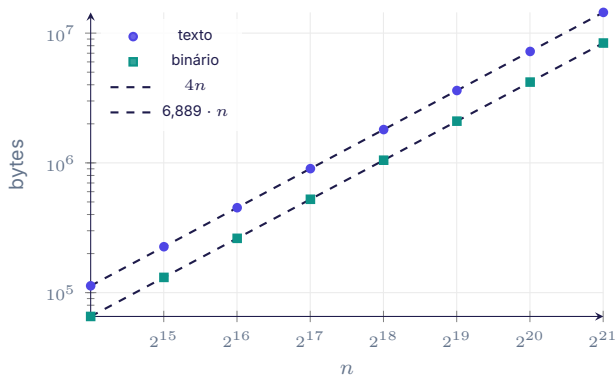
Texto contra binário

Hipótese. O binário ocupa **exatamente** $4n$ bytes. O texto ocupa em média $\approx 6,9$ bytes por número (quase todos têm 6 dígitos, mais o `\n`). O tempo do texto deve ser bem maior: `fprintf` e `fscanf` convertem dígitos um a um; `fwrite` e `fread` copiam blocos.

Método

1. $n = 2^{14}, \dots, 2^{21}$ números aleatórios
2. Gravação e leitura completas em cada formato
3. Tamanho com `fseek/ftell`
4. Tempo por número

Tamanho dos arquivos



n	medido	razão
16.384	65.536	1,000
32.768	131.072	1,000
65.536	262.144	1,000
131.072	524.288	1,000
262.144	1.048.576	1,000
2.097.152	8.388.608	1,000

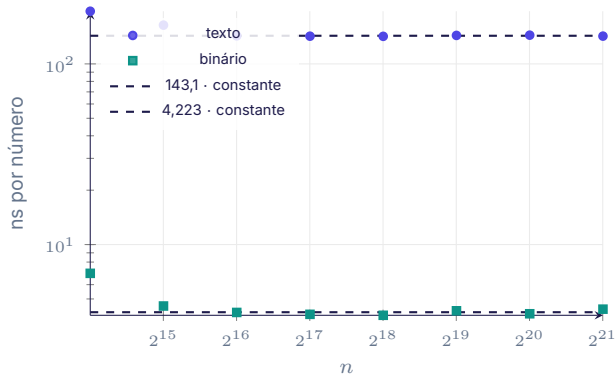
razão = medido $\div$ $4n$

✓ desvio máx. 0,0%

Binário: exatamente $4n$. Texto: cerca de $6,9 n$, 72% maior.

EXPERIMENTO

Tempo por número



n	medido	razão
16.384	195,7	195,7
32.768	163,9	163,9
65.536	142,9	142,9
131.072	142,3	142,3
262.144	142,1	142,1
2.097.152	142,5	142,5

razão = medido $\div$ constante

✓ desvio máx. 0,8%

Os dois são lineares (custo constante por número), mas o texto custa cerca de 85 ns e o binário, 2,5 ns: 35 vezes mais.

EXPERIMENTO

O que o experimento mostrou

Os dois formatos são lineares, mas com constantes muito diferentes: o texto é **72% maior** e **35 vezes mais lento**. O binário é exatamente $4n$ bytes. Por isso bancos de dados, imagens e modelos de IA são guardados em binário, e dados para trocar com pessoas e outras ferramentas, em texto.

➤ Reproduza: `gcc -O2 base5_experimento.c -o exp -lm && ./exp`



Conexões

- ▶ Bancos de dados
- ▶ Redes
- ▶ Ciência de dados
- ▶ Ferramentas de linha de comando

Onde isso aparece na computação

 Bancos de dados

Tabelas e índices ficam em arquivos binários de páginas de tamanho fixo (4 a 16 KB); fseek para a página k é $k * \text{tamanho}$.

 Redes

Protocolos binários (TCP/IP, gRPC/Protobuf) são compactos e rápidos; protocolos de texto (HTTP/1.1, JSON) são legíveis. A escolha é a mesma do experimento.

 Ciência de dados

CSV é universal, mas Parquet e Arrow (binários e colunares) são até dezenas de vezes menores e mais rápidos para grandes volumes.

 Ferramentas de linha de comando

grep, sort e gcc leem argv, leem stdin, escrevem stdout e devolvem um código de saída: é o que permite encadeá-los com |.

Para casa

Exercícios

1. Leia um CSV nome,nota com fgets e sscanf e imprima a média.
2. Faça ./gera N tipo que imprime N números (aleatório, ordenado ou invertido): use-o nos experimentos de ordenação.
3. Grave um vetor de Aluno em binário e leia o k -ésimo registro com fseek, sem ler os anteriores.
4. Abra um arquivo inexistente e mostre a mensagem do perror; depois trate o erro sem abortar.
5. No experimento, grave com fprintf para /dev/null: quanto do tempo é conversão e quanto é disco?

Leitura

Kernighan e Ritchie, *C: a linguagem de programação*, capítulo 7 (entrada e saída) e seção 5.10 (argumentos).

Código

josedihago.net/estruturas