Introdução à Computação II – AULA 09

BCC Noturno - EMA896115B

Prof. Rafael Oliveira
olivrap@gmail.com

Universidade Estadual Paulista
“Júlio de Mesquita Filho”
UNESP

Rio Claro 2014 (Sem 2)

Relembrando

- Vimos como fazer para ler e gravar dados em arquivos
- Esses dados são sempre armazenados no formato texto
- Esses dados podem, por exemplo, ser vistos/modificados com um editor de testo
- Um arquivo precisa ser **aberto**, **manipulado** e depois **fechado**

Abrir um arquivo

- `fopen("C:\users\delamaro\dados.txt", "r");`
- Abre o arquivo `C:\users\delamaro\dados.txt`
- Segundo parâmetro indica como o arquivo vai ser manipulado
- No caso, arquivo é aberto para leitura

Manipular o arquivo

- Ao abrir o arquivo, é devolvido um identificador desse arquivo
- `f = fopen("C:\users\eu\dados.txt", "r");`
- Variável f é do tipo `File *`
- `fscanf(f, "%d %lf", &i, &x);`
- `fprintf(f, "%d %lf", i, x);`
- Essas duas funções retornam quantas variáveis foram lidas/escritas do/no arquivo

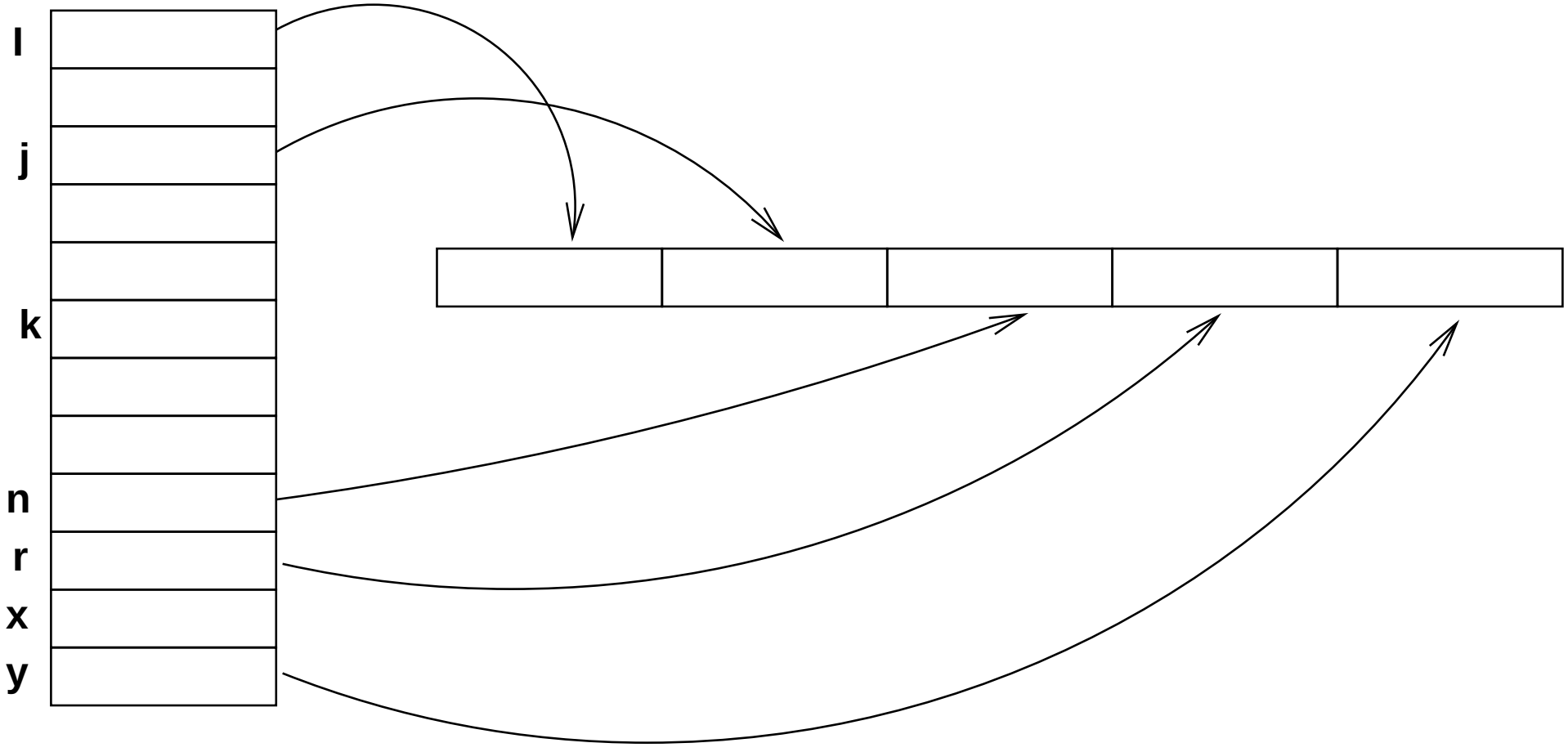
Fechar arquivo

- Para encerrar o uso do arquivo
- Garantir que todos os dados foram gravados
- A partir do fechamento do arquivo ele não pode ser mais usado, a não ser que seja aberto novamente
- `fclose(f);`

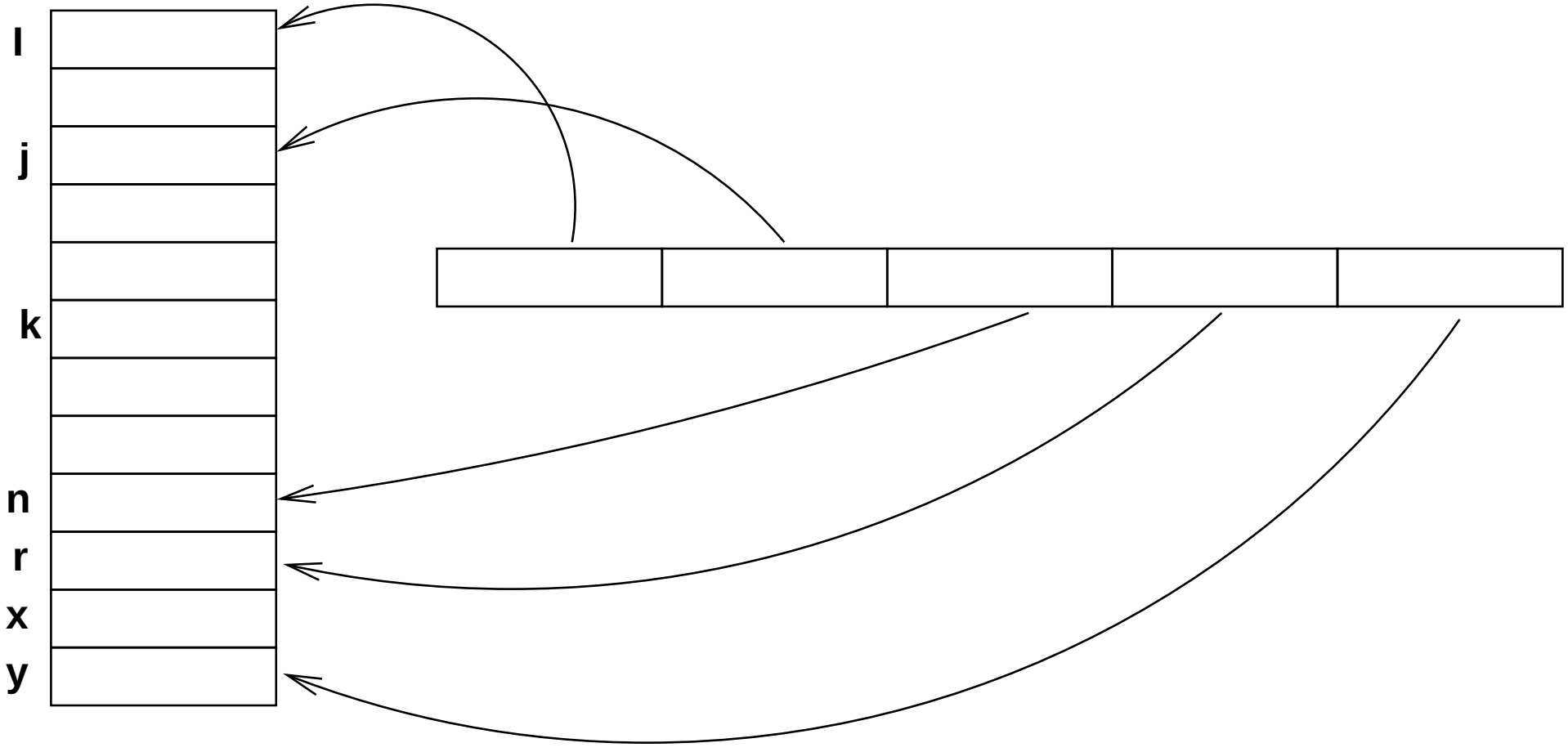
Arquivos binários

- São arquivos que contém dados que não podem ser visualizados diretamente pelo usuário
- Eles servem para armazenar dados no mesmo formato em que são manipulados pelo programa
- São “cópias” de algum trecho da memória do seu programa

Escrevendo variáveis



Lendo variáveis



Como usar

- O processo é o mesmo
- Abrir arquivo, indicando que trata-se de um arquivo binário
- Ler ou escrever, usando funções específicas para dados binários
- Fechar o arquivo usando `fclose`

Abrir binário

- Usá-se a mesma função `fopen`
- Apenas o modo de abertura do arquivo deve ser diferente
- Deve incluir a letra “b” para indicar um arquivo binário
- `f = fopen("C:\users\eu\dados.txt", "rb");`
- Modos: rb, wb, r+b, w+b ab, a+b

Fechar binário

- Nada muda
- `fclose(f);`

Escrever binário

- Utiliza-se o comando fwrite
- `fwrite(&i, 4, 1, f);`
  - qual é a variável que vai ser escrita
  - quantos bytes devem ser transferidos
  - quantas vezes esse tamanho
  - arquivo onde vai ser escrito
- Para saber o tamanho que uma variável ocupa pode-se utilizar **sizeof(i)**
- `fwrite(&i, sizeof(i), 1, f);`

Escrever – exemplo

```
#include <stdio.h>

int main()
{
int i;
double x;
FILE *f;

    f = fopen("ArquivoExemplo", "wb");
    printf("Digite um no. inteiro e um no. real: ");
    scanf("%d %lf", &i, &x);
    fwrite(&i, sizeof(i), 1, f);
    fwrite(&x, sizeof(x), 1, f);
    fclose(f);
}
```

Ler binário

- Utiliza-se o comando fread
- `fread(&i, 4, 1, f);`
  - qual é a variável que vai ser lida
  - quantos bytes devem ser transferidos
  - quantas vezes esse tamanho
  - arquivo de onde vai ser lido
- Para saber o tamanho que uma variável ocupa pode-se utilizar **sizeof(i)**
- `fread(&i, sizeof(i), 1, f);`

Ler – exemplo

```
#include <stdio.h>

int main()
{
int i;
double x;
FILE *f;

    f = fopen("ArquivoExemplo", "rb");
    fread(&i, sizeof(i), 1, f);
    fread(&x, sizeof(x), 1, f);
    fclose(f);
    printf("Os numeros lidos foram: %d e %lf", i, x);
}
```

O que acontece se...

```
#include <stdio.h>

int main()
{
int i;
double x;
FILE *f;

    f = fopen("ArquivoExemplo", "rb");
    fread(&x, sizeof(x), 1, f); // <==
    fread(&i, sizeof(i), 1, f);
    fclose(f);
    printf("Os numeros lidos foram: %d e %lf", i, x);
}
```

Tipos estruturados

- Uma vantagem em se usar arquivos binários é que pode-se ler/grava tipos estruturados como vetores
- Com isso grandes quantidades de dados podem ser lidos ou escritos de uma vez
- Note que com os parâmetros das funções é fácil realizar essa operação

Tipos estruturados – exemplo

- Programa que escreve os n primeiros elementos de um vetor

Tipos estruturados – exemplo

- Programa que escreve os n primeiros elementos de um vetor

```
double v[100];
int main()
{
FILE *f;
int i, n;

    f = fopen("ArquivoExemploVetor", "wb");
    printf("Digite quantos números vão ser gravados: ");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        scanf("%lf", &v[i]);
    }

    fwrite(v, sizeof(v[0]), n, f);
    fclose(f);
}
```

O que acontece se

```
double v[100];
int main()
{
FILE *f;
int i, n;

    f = fopen("ArquivoExemploVetor", "wb");
    printf("Digite quantos números vão ser gravados: ");
    scanf("%d", &n);
    for (i = 0; i < n; i++)
    {
        scanf("%lf", &v[i]);
    }

    fwrite(v, sizeof(v), 1, f);
    fclose(f);
}
```

Como ler o vetor?

Façam no caderninho

Como ler o vetor?

```
double v[100];
int main()
{
int n, i;
FILE *f;

    f = fopen("ArquivoExemploVetor", "rb");
    printf("Quantos elementos devem ser lidos? ");
    scanf("%d", &n);
    fread(v, sizeof(v[0]), n, f);

    fclose(f);
    printf("Os números lidos foram:");
    for (i = 0; i < n; i++)
    {
        printf("%lf ", v[i]);
    }
    printf("\n");
}
```

O que retornam

- Tanto `fread` quanto `fwrite` retornam o número de elementos que foram realmente lidos
- `fwrite(v, sizeof(v[0]), 5, f);`
  - Deveria devolver 5, ou algo estranho aconteceu
- Algo estranho
  - Fim do arquivo
  - Algum erro

Acesso sequencial

- Conforme vimos até agora, toda leitura/escrita é feita sequencialmente no arquivo
- Existe um “apontador” que indica qual parte do arquivo vai ser lida ou escrita
- A leitura de um bloco de bytes, move esse apontador para a frente
- A próxima leitura/escrita será feita nesse novo ponto
- Existe um único apontador para escrita e para leitura

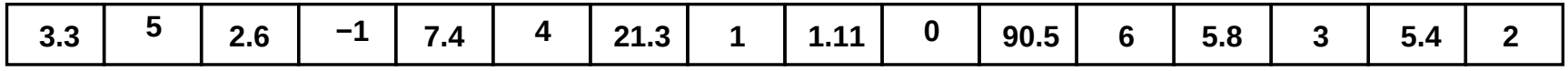
Acesso aleatório

- Existem funções que permitem mover o apontador para frente ou para trás
- Essas operações não modificam o arquivo e nem recuperam dados do arquivo
- Apenas mudam o ponto aonde as próximas operações de leitura/escrita vão ocorrer
- `fseek(arquivo, deslocamento, relativo)`
  - **arquivo** é o identificador do arquivo
  - **deslocamento** é um valor tipo *long* que indica a quantidade de bytes que o ponteiro deve ser movido
  - relativo indica a partir de onde os bytes são contados

fseek – exemplos

- `fseek(f, (long) 10, SEEK_CUR)`
  - avança 10 bytes a partir da posição corrente
- `fseek(f, (long) -10, SEEK_CUR)`
  - retrocede 10 bytes a partir da posição corrente
- `fseek(f, (long) 10, SEEK_SET)`
  - posiciona a 10 bytes a partir do início do arquivo
- `fseek(f, (long) -10, SEEK_END)`
  - posiciona a 10 bytes antes do fim do arquivo
- Retorna 0 se tudo ocorreu bem ou -1 se um erro ocorreu

fseek – exemplo

- Considere um arquivo como mostrado abaixo
- 

- Ele contém uma lista de números *double* seguidos de um número inteiro
- O primeiro número *double* da lista aparece no final do arquivo
- O número inteiro que vem depois do *double* indica qual é o próximo elemento da lista
- O valor inteiro -1 indica que não há mais números a serem lidos
- Escreva um programa que leia e mostre os números *double*, na ordem que está determinada no arquivo

Função *ftell*

- Retorna aonde está posicionado o ponteiro de escrita/leitura
- Devolve um valor do tipo *long*
- `l = ftell(f);`
- O valor devolvido refere-se ao deslocamento a partir do início do arquivo
- Como fazer para saber o tamanho de um arquivo?
  - Posicionar o ponteiro no final do arquivo
  - Usar o *ftell*

Função *ftell*

- Retorna aonde está posicionado o ponteiro de escrita/leitura
- Devolve um valor do tipo *long*
- `l = ftell(f);`
- O valor devolvido refere-se ao deslocamento a partir do início do arquivo
- Como fazer para saber o tamanho de um arquivo?
  - `fseek(f, (long) 0, SEEK_END);`
  - `l = ftell(f);`