Computação eletrônica: Arquivos

Gurvan Huiban
ghuiban@cin.ufpe.br

17 de julho de 2014

Plano de aula

1. Tipos de memória
2. Arquivos
3. Arquivos binários e arquivos de texto
4. Abrir arquivos
5. Ler e escrever em arquivos
6. Comandos úteis

Exercício: Registro de alunos

Escrever um programa que armazena o registro dos 23447 alunos dos cursos de graduação da UFPE (Campus Recife).

Exercício: Registro de alunos

Escrever um programa que armazena o registro dos 23447 alunos dos cursos de graduação da UFPE (Campus Recife).

Problemas

- E se o computador for desligado?
- E se faltar energia?
- E se quiser entrar com os dados em várias vezes?
- ...

Ou seja, precisamos armazenar dados de forma não volátil.

1 Tipos de memória
2 Arquivos
3 Arquivos binários e arquivos de texto
4 Abrir arquivos
5 Ler e escrever em arquivos
6 Comandos úteis

Um computador possui

- Um processador: coordena as atividades (entrada/saída, armazenamento em memória) e realiza os processamentos;
- Uma memória RAM: Armazena dados;
- Um disco rígido: Armazena dados.

Memória RAM

- Rápida;
- Cara;
  $\Rightarrow$Baixa capacidade;
- Volátil: Perde tudo ao desligar o computador.

Disco rígido

- Lento;
- Barato;
- Permanente: Não perde os dados quando o computador for desligado.

Necessidade dos arquivos

- Até agora, fizemos tudo usando apenas a memória RAM...
- Mas para alguns programas precisamos ler/armazenar dados no disco rígido;
- Usar o disco rígido é trabalhar com arquivos (armazenar/ler dados).

Não confunda

- Os arquivos que contém um programa em C:
  - Arquivo `.c` com o código fonte;
  - Arquivo `.exe` com o código compilado e que pode ser executados.
- Os arquivos que contém os dados do programa:
  - Serão lidos/escritos pelo programa;
  - Não podem ser executado;
  - É necessária a adequação entre o programa e o formato dos dados.

Arquivos binários

- Um arquivo é interpretado como uma sequência única de bits;
- Os bits lidos/escritos vão ser interpretados como sendo de um tipo único (`integer`, `char`, tipo definido pelo usuário ou ...);
- Não tem separação entre os dados, já que o tamanho (em bits) do dado a ser lido/escrito é fixo;
- Qualquer valor é lido ou escrito sem alteração entre a memória e o arquivo (bits<–>bits).

Arquivos de texto

A leitura/escrita num arquivo de texto é parecida com a leitura do teclado/escrita na tela:

- Um arquivo é interpretado como uma sequência de caracteres agrupados em linhas;
- Um mesmo arquivo pode misturar dados de diferentes tipos (números inteiros, cadeia de caracteres, . . .);
- Os dados são sequências de caracteres separados por espaços ou quebra de linha
  Ex: O número 1000 será armazenado como uma sequência de 4 caracteres.

Abrir arquivos

Permite ao programa ter acesso ao arquivo, para:

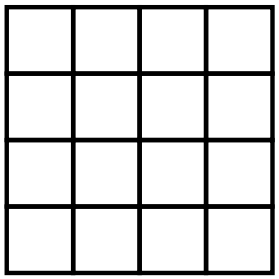
- Ler dados;
- Navegar no arquivo;
- Escrever dados.

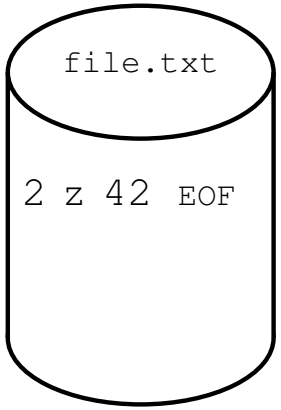
Ponteiro de arquivo

```
FILE *aptr;
```

- `FILE`: Tipo arquivo;
- `aptr`: Ponteiro de arquivo. Aponta para uma posição no arquivo.

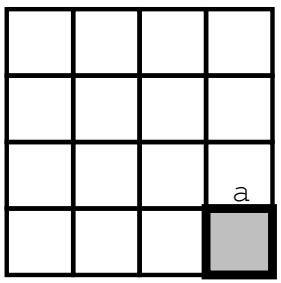
```
FILE* a;
char c; int i,j;
a = fopen("file.txt","r");
fscanf(a, "%d", &i);
fscanf(a, "%c", &c);
fscanf(a, "%d", &j);
printf("i=%d j=%d\n",i,j);
printf("c=%c\n",c);
fclose(a);
```

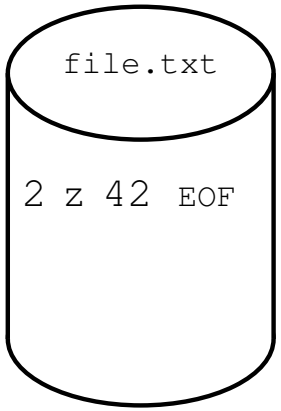




Tela

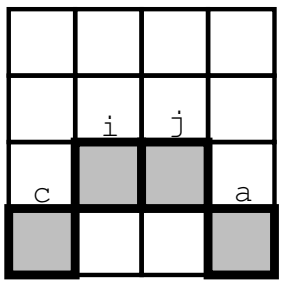
```
FILE* a;
char c; int i,j;
a = fopen("file.txt","r");
fscanf(a, "%d", &i);
fscanf(a, "%c", &c);
fscanf(a, "%d", &j);
printf("i=%d j=%d\n",i,j);
printf("c=%c\n",c);
fclose(a);
```

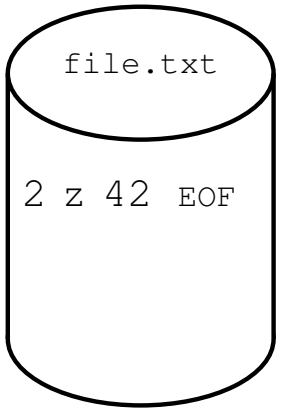




Tela

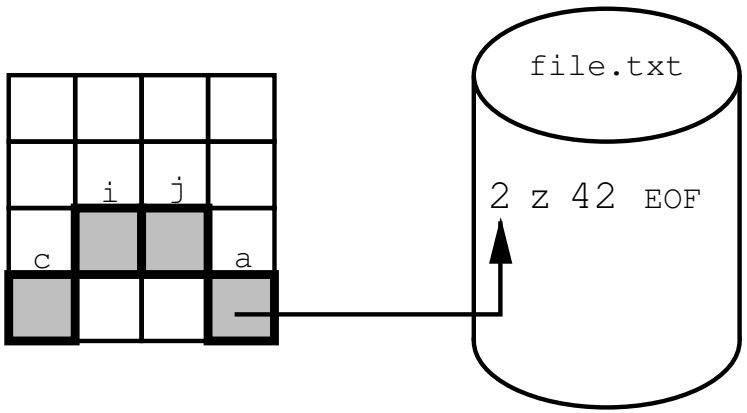
```
FILE* a;
char c; int i,j;
a = fopen("file.txt","r");
fscanf(a, "%d", &i);
fscanf(a, "%c", &c);
fscanf(a, "%d", &j);
printf("i=%d j=%d\n",i,j);
printf("c=%c\n",c);
fclose(a);
```





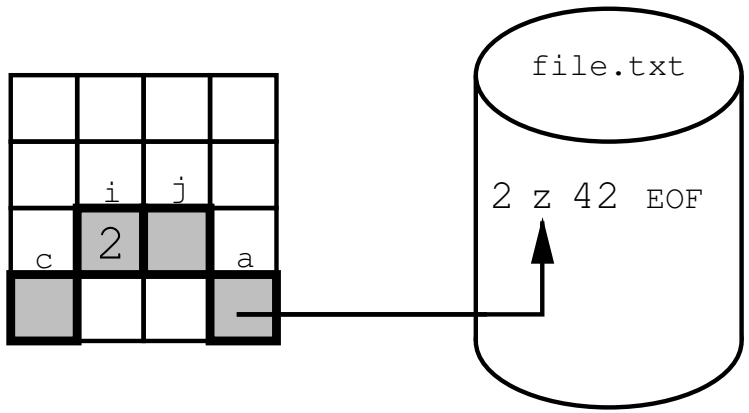
Tela

```
FILE* a;
char c; int i,j;
a = fopen("file.txt","r");
fscanf(a, "%d", &i);
fscanf(a, "%c", &c);
fscanf(a, "%d", &j);
printf("i=%d j=%d\n",i,j);
printf("c=%c\n",c);
fclose(a);
```



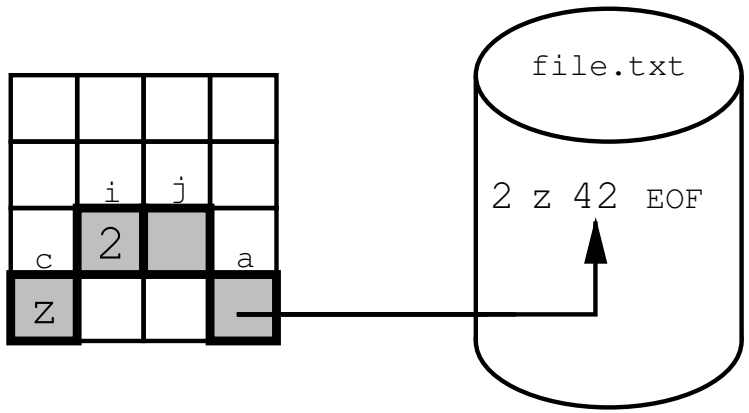
Tela

```
FILE* a;
char c; int i,j;
a = fopen("file.txt","r");
fscanf(a, "%d", &i);
fscanf(a, "%c", &c);
fscanf(a, "%d", &j);
printf("i=%d j=%d\n",i,j);
printf("c=%c\n",c);
fclose(a);
```



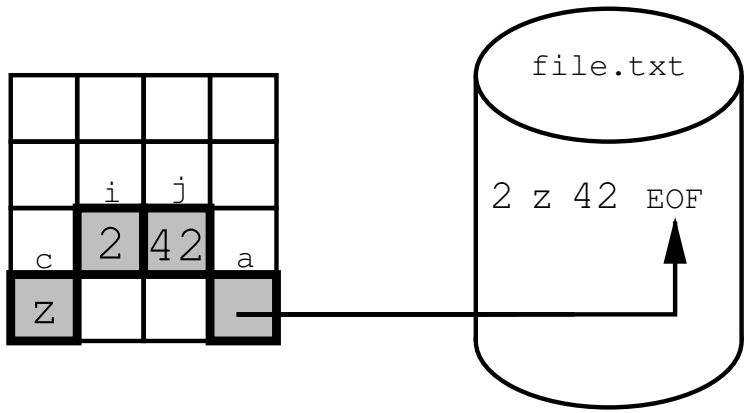
Tela

```
FILE* a;
char c; int i,j;
a = fopen("file.txt","r");
fscanf(a, "%d", &i);
fscanf(a, "%c", &c);
fscanf(a, "%d", &j);
printf("i=%d j=%d\n",i,j);
printf("c=%c\n",c);
fclose(a);
```



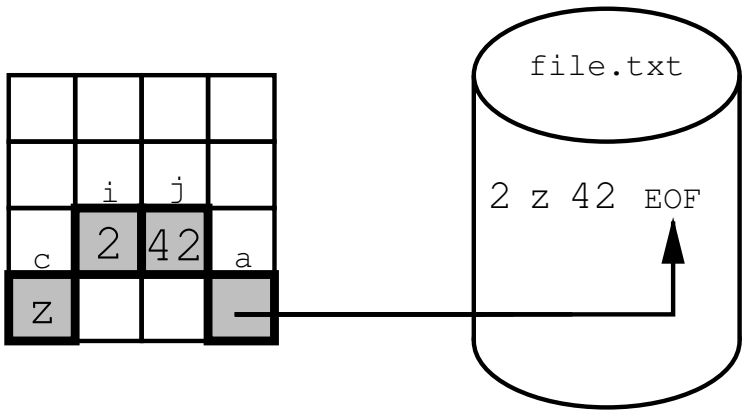
Tela

```
FILE* a;
char c; int i,j;
a = fopen("file.txt","r");
fscanf(a, "%d", &i);
fscanf(a, "%c", &c);
fscanf(a, "%d", &j);
printf("i=%d j=%d\n",i,j);
printf("c=%c\n",c);
fclose(a);
```



Tela

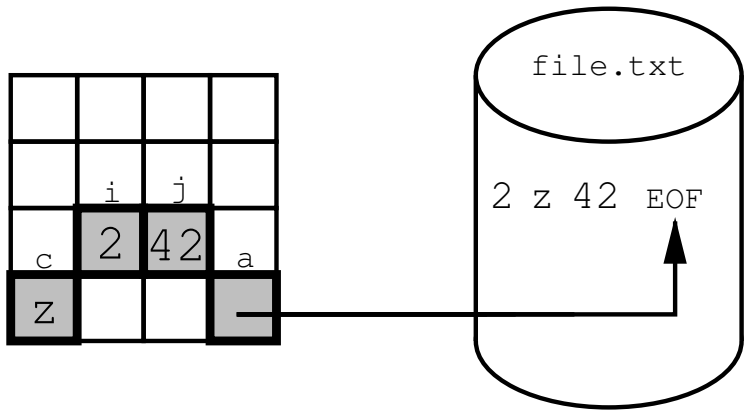
```
FILE* a;
char c; int i,j;
a = fopen("file.txt","r");
fscanf(a, "%d", &i);
fscanf(a, "%c", &c);
fscanf(a, "%d", &j);
printf("i=%d j=%d\n",i,j);
printf("c=%c\n",c);
fclose(a);
```



Tela

```
i=2 j=42
```

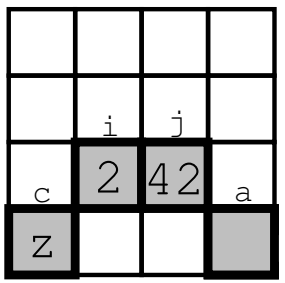
```
FILE* a;
char c; int i,j;
a = fopen("file.txt","r");
fscanf(a, "%d", &i);
fscanf(a, "%c", &c);
fscanf(a, "%d", &j);
printf("i=%d j=%d\n",i,j);
printf("c=%c\n",c);
fclose(a);
```

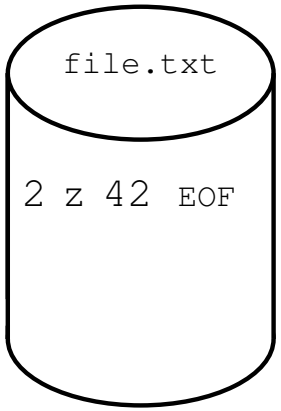


Tela

```
i=2 j=42
c=z
```

```
FILE* a;
char c; int i,j;
a = fopen("file.txt","r");
fscanf(a, "%d", &i);
fscanf(a, "%c", &c);
fscanf(a, "%d", &j);
printf("i=%d j=%d\n",i,j);
printf("c=%c\n",c);
fclose(a);
```





Tela

```
i=2 j=42
c=z
```

Sintaxe: Abrir um arquivo

```
FILE *fopen(const char *nome_arquivo,
            const char *modo);
```

- Abre o arquivo e retorna um ponteiro para o arquivo;
- `nome_arquivo`: Nome do arquivo;
- `modo`: Modo de acesso ao arquivo (leitura, escrita, ...).

Sintaxe: Fechar um arquivo

```
int fclose(FILE *aptr);
```

- Fecha o arquivo. Retorna `0`, salvo casos de erro;
- `aptr`: Ponteiro de arquivo que queremos fechar.

Exemplo

Abrir um arquivo de texto chamado `lista_alunos.txt` para leitura:

```
FILE *aptr;
aptr = fopen("lista_alunos.txt", "r");
...
fclose(aptr)
```

Observações

- A abertura do arquivo pode falhar
  (ex.: Arquivo que não existe);
- Neste caso, `fopen` retorna `NULL`.
  $\Rightarrow$Sempre verificar se a abertura deu certo!
- Sempre fechar o arquivo antes de encerrar o programa.

```c
FILE *aptr;
aptr = fopen("lista_alunos.txt", "r");
if (aptr == NULL)
{
  printf("Erro!\n");
}
else
{
  ...
  fclose(aptr)
}
```

```
FILE *fopen(const char *nome_arquivo,
            const char *modo);
```

Modos de acesso

- `r`: Arquivo de texto para leitura. O arquivo deve existir;
- `w`: Arquivo de texto para gravação. Se o arquivo existir, ele será reiniciado; caso contrário, será criado;
- `a`: Arquivo texto para gravação. Adiciona os dados no arquivo existente ou cria novo arquivo;
- `rb`: Arquivo binário para leitura;
- `wb`: Arquivo binário para gravação (reinicialização);
- `ab`: Arquivo texto para gravação (adição);
- e outros tipos: `r+`, `w+`, `a+`, ...

Sintaxe: Ler um carácter

```
int fgetc(FILE *aptr);
```

- Retorna o carácter apontado por `aptr`, e faz `aptr` apontar para o carácter seguinte;
- Retorna `EOF` no fim do arquivo;
- `aptr`: Ponteiro de arquivo, apontando para o carácter a ser lido.

Sintaxe: Escrever um carácter

```
int fputc(char c, FILE *aptr);
```

- Escreve o carácter `c` no arquivo apontado por `aptr`;
- Retorna o carácter escrito ou `EOF` em caso de erro;
- `c`: Carácter a ser escrito;
- `aptr`: Ponteiro de arquivo, apontando para o arquivo onde o carácter será escrito.

Exercício: Leitura de arquivo existente

Criar com o Bloco de notas (*Notepad*) um arquivo com um texto. Escrever um programa que abra este arquivo e imprima o conteúdo dele na tela, e depois a quantidade de caracteres do arquivo.

Exercício: Atualização de arquivo existente

Escrever um programa que adiciona seu nome no fim do arquivo do exercício anterior.

Sintaxe: Ler uma cadeia de caracteres

```
char *fgets(char *cadeia, int tamanho, FILE *aptr)
```

- Armazena em `cadeia` a quantidade `tamanho` de caracteres a partir do carácter apontado por `aptr` e atualizando a posição apontada por `aptr`;
- Retorna `cadeia`, ou `NULL` em caso de erro ou fim de arquivo;
- `cadeia`: Cadeia de caracteres recebendo a cadeia lida;
- `tamanho`: Quantidade de caracteres a serem lidos;
- `aptr`: Ponteiro de arquivo, apontando para o arquivo onde a cadeia será lida;
- `cadeia` deve ter um tamanho suficiente ($\geq$`tamanho`).

Sintaxe: Escrever uma cadeia de caracteres

```
int fputs(const char *cadeia, FILE *aptr);
```

- Escreve a cadeia de caracteres `cadeia` no lugar apontado por `aptr`;
- Retorna um inteiro positivo, ou `EOF` em caso de erro;
- `cadeia`: Cadeia de caracteres a ser escrita;
- `aptr`: Ponteiro de arquivo, apontando para o arquivo onde a cadeia de caracteres será escrita.

Observação

- Não usar o valor de retorno de `fgets` para verificar se terminamos de ler o arquivo;
- Usar o comando:

  ```
  int feof(FILE *aptr);
  ```

  que retorna um valor não nulo quando `aptr` aponta para o final do arquivo.

```
fgets(s, 80, aptr);
while (!feof(aptr))
{
   printf("%s", s);
   fgets(s, 80, aptr);
}
```

Exercício: Leitura de arquivo existente

Criar com o Bloco de notas (*Notepad*) um arquivo com um texto. Escrever um programa que abra este arquivo e imprima o conteúdo dele na tela usando o comando `fgets`.

Exercício: Atualização de arquivo existente

Escrever um programa que adiciona seu nome no fim do arquivo do exercício anterior, usando o comando `fputs`.

Sintaxe: Ler e escrever valores com formato

```
int fscanf(FILE *aptr, const char *formato, ...);
int fprintf(FILE *aptr, const char *formato, ...);
```

- Mesmo funcionamento que os comandos `printf` e `scanf`, mas para ler/escrever em arquivos ao invés do teclado/da tela;
- `aptr`: Ponteiro de arquivo, apontando para o arquivo onde os valores serão lidos/escritos;
- `formato`: Formato de leitura/escrita.

Sintaxe: Ler e escrever dados

```
size_t fread(void *dptr, size_t tam,
             size_t quant, FILE *aptr);
size_t fwrite(const void *dptr, size_t tam,
             size_t quant,FILE *aptr);
```

- `dptr`: Ponteiro para o lugar da memória onde vão ser lidos/escritos os dados;
- `tam`: Tamanho de um elemento a ser lido/escrito;
- `num`: Quantidade de elementos a serem lidos/escritos;
- `aptr`: Ponteiro de arquivo, apontando para o arquivo onde os dados serão lidos/escritos;
- Retornam a quantidade de elementos lidos/escritos.

Voltar no início de um arquivo

```
void rewind(FILE *aptr)
```

Fim de arquivo

```
int feof(FILE *aptr);
```

que retorna um valor não nulo quando `aptr` aponta para o final do arquivo.

E ainda tem mais!

- Navegar no arquivo;
- Procurar por elementos;
- Atualizar valores;
- Inserir ou remover em qualquer lugar.
- ...