

CP63B-DPGR3A

COMPUTAÇÃO 2

APNP 08 - Arquivos

Prof. Rafael Gomes Mantovani

Licença

Este trabalho está licenciado com uma Licença CC BY-NC-ND 4.0:



maiores informações:

https://creativecommons.org/licenses/by-nc-nd/4.0/deed.pt_BR

Roteiro



- 1** Introdução
- 2** Arquivos
- 3** Leitura/Escrita
- 4** Exercícios
- 5** Referências

Roteiro



- 1** Introdução
- 2** Arquivos
- 3** Leitura/Escrita
- 4** Exercícios
- 5** Referências

Introdução

- O padrão C ANSI define um conjunto completo de funções de E/S para ler e escrever qualquer tipo de dados
 - Este padrão será o utilizado neste material;
 - Maior compatibilidade com os compiladores;
- O padrão C UNIX contém dois sistemas de rotinas para realizar E/S:
 - Sistema de arquivo com *buffer*, formatado ou alto nível;
 - Sistemas de arquivos tipo UNIX, não formatado ou sem *buffer*;
- *Buffer*
 - É uma região de memória física utilizada para armazenar TEMPORARIAMENTE os dados enquanto eles estão sendo movidos de um lugar para outro;
 - Normalmente, os dados são armazenados em um *buffer* enquanto eles são recuperados de um dispositivo de entrada ou pouco antes de serem enviados para um dispositivo de saída (*wikipedia*, 2018).

Introdução

- Arquivos são dados manipulados fora do ambiente do programa (memória principal);
- Um arquivo pode ser qualquer coisa, desde um arquivo em disco (mais comum) até um terminal ou uma impressora;
- Um arquivo é armazenado em um dispositivo de memória secundária e pode ser lido ou escrito por um programa;
- Porque usar arquivos?
 - Permitem armazenar grande quantidade de informação;
 - Persistência de dados (armazenamento em disco);
 - Acesso aos dados podem ou não ser sequencial;
 - Acesso concorrente aos dados (mais de um programa “pode” usar os dados ao mesmo tempo);

Roteiro



- 1 Introdução
- 2 Arquivos
- 3 Leitura/Escrita
- 4 Exercícios
- 5 Referências

Tipos de Arquivo

- Basicamente, a linguagem C trabalha com dois tipos de arquivos:

Arquivo texto

- Armazena caracteres que podem ser mostrados diretamente na tela ou modificados por um editor de textos;
- Os dados são gravados como caracteres de 8 bits. Ex.: Um número inteiro de 32 bits com 8 dígitos ocupará 64 bits no arquivo.

Arquivo binário

- Armazena uma sequência de bits que está sujeita as convenções do programa que o gerou. Ex.: arquivos compactados;
- Os dados são gravados em binário, ou seja, do mesmo modo que estão na memória. Ex.: um número inteiro de 32 bits com 8 dígitos ocupará 32 bits no arquivo.

A manipulação de arquivos se dá por meio de fluxos (*streams*).

Manipulação de Arquivos

- A biblioteca `stdio.h` dá suporte à utilização de arquivos em C.
 - Renomear e remover;
 - Garantir acesso ao arquivo;
 - Ler e escrever;
 - Alterar o posicionamento dentro do arquivo;
 - Manusear erros;
 - Para mais informações: <http://www.cplusplus.com/reference/cstdio/>
- A linguagem C não possui funções que leiam automaticamente toda a informação de um arquivo:
 - Suas funções limitam-se em **abrir/fechar** e **ler/escrever** caracteres ou bytes;
 - O programador deve instruir o programa na leitura do arquivo de uma maneira específica;

Manipulação de Arquivos

- Todas as funções de manipulação de arquivos trabalham com o conceito de “ponteiro de arquivo”;
- Um ponteiro de arquivo é um ponteiro para informações que definem várias coisas sobre o arquivo, incluindo seu nome, *status* e a posição atual do arquivo;
- Um ponteiro de arquivo é uma variável ponteiro do tipo `FILE` (definido na biblioteca `stdio.h`);
- Pode-se declarar um ponteiro de arquivo da seguinte maneira:

```
FILE *arq
```

- `arq` é o ponteiro para arquivos que permite manipular um arquivo.

Abrindo um arquivo

- Para a abertura de um arquivo, usa-se a função `fopen`:

```
File *arq;
```

```
fopen(nome_arquivo, modo_de_abertura);
```

- O parâmetro *nome_arquivo* determina qual o arquivo ser aberto, incluindo a extensão
 - O nome deve ser válido no sistema operacional que estiver sendo utilizado;
 - **Caminho absoluto:** descrição de um caminho desde o diretório raiz
 - `C:\Programação\aula18\dados.txt`
 - **Caminho relativo:** descrição de um caminho desde o diretório corrente, ou seja, onde o programa está salvo
 - `dados.txt`
 - `..\dados.txt`

Abrindo um arquivo

```
FILE *arq;
```

```
arq = fopen(nome_arquivo, modo_de_abertura);
```

└─ A função fopen retorna um ponteiro do tipo FILE

- O modo de abertura determina que tipo de USO será feito do arquivo
 - Abrir para leitura;
 - Abrir para escrita;
 - Abrir para leitura e escrita.

} Texto ou binário

Abrindo um arquivo

- Modos clássicos

Modo	Arquivo	Função
“r”	Texto	Leitura. Arquivo deve existir.
“w”	Texto	Escrita. Criar arquivo se não houver. Apaga o anterior se ele existir.
“a”	Texto	Escrita. Os dados serão adicionados no final do arquivo (<i>append</i>).
“rb”	Binário	Leitura. Arquivo deve existir.
“wb”	Binário	Escrita. Cria arquivo se não houver. Apaga o anterior se ele existir.
“ab”	Binário	Escrita. Os dados serão adicionados no fim do arquivo (<i>append</i>)
“r+”	Texto	Leitura/Escrita. O arquivo deve existir e pode ser modificado.
“w+”	Texto	Leitura/Escrita. Cria arquivo se não houver. Apaga o anterior se ele existir
“a+”	Texto	Leitura/Escrita. Os dados serão adicionados no fim do arquivo (<i>append</i>).
“r+b”	Binário	Leitura/Escrita. O arquivo deve existir e pode ser modificado.
“w+b”	Binário	Leitura/Escrita. Cria arquivo se não houver. Apaga o anterior se ele existir.
“a+b”	Binário	Leitura/Escrita. Os dados serão adicionados no fim do arquivo (<i>append</i>).

Abrindo um arquivo

- Um arquivo do tipo texto pode ser aberto para escrita utilizando o seguinte conjunto de comandos:
 - A condição `arq == NULL` testa se o arquivo foi aberto com sucesso;
 - No caso de erro a função `fopen` retorna um ponteiro nulo (`NULL`).

```
1 int main(){
2     FILE *arq;
3
4     arq = fopen("teste.txt", "w");
5     if(arq == NULL)
6         printf("Ocorreu um erro na abertura do arquivo");
7     ...
8 }
```

Erros comuns

- Vários erros podem acontecer e impedir a abertura de um arquivo. Os casos mais comuns são:
 - Disco cheio;
 - Disco protegido contra gravação;
 - Se o modo de abertura for especificado apenas como leitura e o arquivo não existir;
 - O caminho (relativo ou absoluto) do arquivo for informado errado;
 - O número de arquivo que pode ser aberto simultaneamente foi superado;
 - Etc.
- Para descobrir a quantidade máxima de arquivos que podem ser abertos, basta verificar o valor de `FOPEN_MAX`, disponível na biblioteca `stdio.h`:

```
printf("%d", FOPEN_MAX);
```

Erros comuns

- É comum parar a execução do programa caso o arquivo não seja aberto com sucesso
 - Provavelmente o programa não poderá continuar a executar;
 - Nesse caso, pode-se utilizar a função `exit()`, contida na biblioteca `stdlib.h`, para sair do programa;

```
void exit(int código_de_retorno);
```

- O código de retorno é semelhante ao valor que seria retornado pelo comando `return` da função `main()`;
 - Caso o valor zero seja usado como código, significa término normal do programa;
 - Um número diferente de zero, algum problema ocorreu.

Erros comuns

```
1 int main(){
2     FILE *arq;
3
4     arq = fopen("teste.txt", "w");
5     if(arq == NULL)
6         printf("Ocorreu um erro na abertura do arquivo");
7         system("pause");
8         exit(1);
9     }
10    ...
11
```

Fechando um arquivo

- Um arquivo pode ser fechado pela função `fclose()`;
 - Escreve no arquivo qualquer dado que ainda permanece no *buffer*;
 - Geralmente as informações só são gravadas no disco quando o *buffer* está cheio.
 - O ponteiro do arquivo é passado como parâmetro para `fclose()`;
 - **Esquecer de fechar** o arquivo pode gerar **inúmeros problemas**;

```
1 int main(){
2     FILE *arq;
3
4     arq = fopen("teste.txt", "w");
5     if(arq == NULL)
6         printf("Ocorreu um erro na abertura do arquivo");
7         system("pause");
8         exit(1);
9     }
10    ...
11    fclose(arq);
12
13    return 0;
14 }
```

Fechando um arquivo

- Por que utilizar um *buffer*? **EFICIÊNCIA!!!**
 - Para ler e escrever arquivos no disco temos que posicionar a cabeça de gravação em um ponto específico do disco;
 - Se tivermos que fazer isso para cada caractere lido/escrito, a leitura/escrita de um arquivo seria uma operação muito lenta;
 - Assim a gravação só é realizada quando existe um volume razoável de informações a serem gravadas ou quando o arquivo for fechado;

Roteiro



- 1** Introdução
- 2** Arquivos
- 3** Leitura/Escrita
- 4** Exercícios
- 5** Referências

Leitura/escrita de Arquivos

- Existe uma espécie de posição que indica a localização dentro do arquivo;
- É nessa posição onde será lido ou escrito o próximo caractere;
 - Quando utilizando o acesso sequencial, raramente é necessário modificar essa posição;
 - Isso porque, quando um caractere é lido, a posição no arquivo é automaticamente atualizada;
 - Leitura e escrita em arquivos são parecidos com escrever em uma **máquina de escrever**.

Escrita: caracteres

- A maneira mais fácil de trabalhar com um arquivo é a leitura/escrita de **um único caractere por vez**;
- A função `fputc` (*put character*) pode ser utilizado para esse princípio;

```
1 FILE *arq;  
2 char str[] = "Texto a ser gravado no arquivo";  
3 int i;  
4 arq = fopen("Teste.txt", "w");  
5 if(arq == NULL){  
6     printf("Erro ao abrir o arquivo");  
7     system("pause");  
8     exit(1);  
9 }  
10 for(i=0;i<strlen(str);i++)  
11     fputc(str[i], arq);  
12  
13 fclose(arq);
```

```
fputc(caractere, ponteiro);
```

Equivale à:

```
putc(caractere, ponteiro);
```

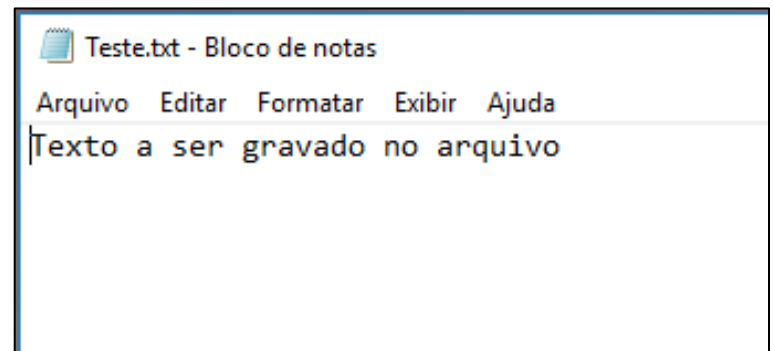
Usada também para impressão:

```
fputc('a', stdout);
```

Escrita: caracteres

- Agora usando **while**...

```
1 FILE *arq;  
2 char str[] = "Texto a ser gravado no arquivo";  
3 int i;  
4 arq = fopen("Teste.txt", "w");  
5 if(arq == NULL){  
6     printf("Erro ao abrir o arquivo");  
7     system("pause");  
8     exit(1);  
9 }  
10 i = 0;  
11 while(str[i] != '\0'){  
12     fputc(str[i], arq);  
13     i++;  
14 }  
15  
16 fclose(arq);
```



Leitura: caracteres

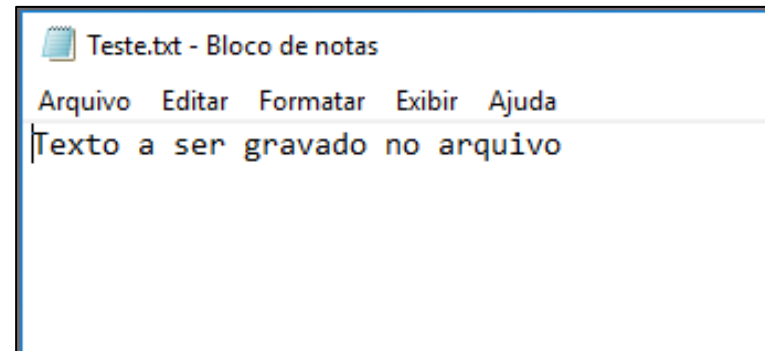
- Também podemos ler caracteres um a um do arquivo;
- A função usada para isso é a `fgetc` (*get character*);

```
1 FILE *arq;  
2 int i;  
3 char c;  
4 arq = fopen("Teste.txt", "r");  
5 if(arq == NULL){  
6     printf("Erro ao abrir o arquivo");  
7     system("pause");  
8     exit(1);  
9 }  
10 for(i=0;i<10;i++){  
11     c = fgetc(arq);  
12     printf("%c", c);  
13 }  
14  
15 fclose(arq);
```

`fgetc(ponteiro);`

Equivale à:

`getc(ponteiro);`



Saída: Texto a se

Leitura: caracteres

- A assinatura da função é a seguinte:

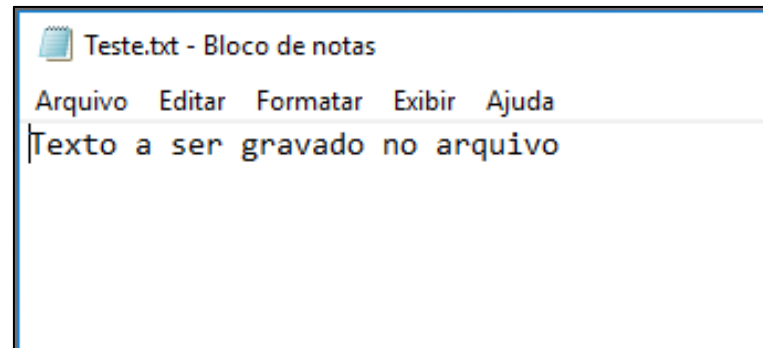
```
int fgetc(FILE *arq)
```

- Isso significa que na verdade ela não retorna um char, mas o valor inteiro que representa aquele símbolo (tabela ASCII);
- O que acontece quando a função fgetc tenta ler o próximo caractere de um arquivo que já acabou?
- Se o arquivo tiver acabado, a função devolve um valor int que não possa ser confundido com nenhum da tabela ASCII;
 - Por definição fgetc devolve o valor -1;
- Mais especificamente, fgetc devolve a constante EOF (*end of file*);

Leitura: caracteres

- Exemplo do uso do EOF

```
1 FILE *arq;
2 char c;
3 int i;
4 arq = fopen("Teste.txt", "r");
5 if(arq == NULL){
6     printf("Erro ao abrir o arquivo");
7     system("pause");
8     exit(1);
9 }
10
11 c = fgetc(arq);
12 while(c != EOF){
13     printf("%c", c);
14     c = fgetc(arq);
15 }
16
17 fclose(arq);
```

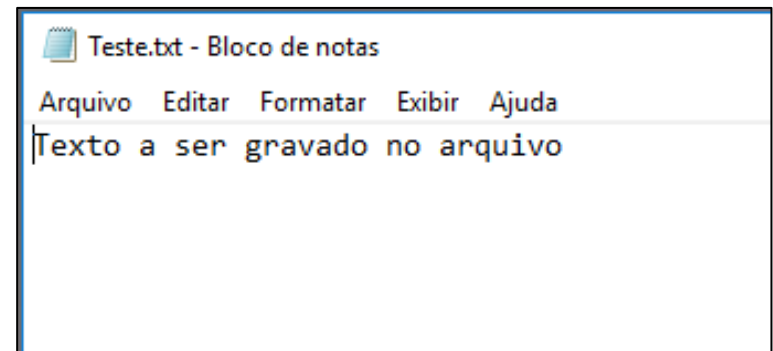


Saída: Texto a ser gravado no arquivo

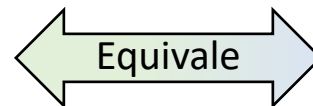
Leitura: caracteres

- Testando o valor de EOF

```
1 FILE *arq;  
2 char c;  
3 int i;  
4 arq = fopen("Teste.txt", "r");  
5 if(arq == NULL){  
6     printf("Erro ao abrir o arquivo");  
7     system("pause");  
8     exit(1);  
9 }  
10  
11 c = fgetc(arq);  
12 while(c != EOF){  
13     printf("%c %d\n", c, c);  
14     c = fgetc(arq);  
15 }  
16 printf("%c %d\n", c, c);  
17  
18 fclose(arq);
```



Qual será a saída?



```
while((c = fgetc(arq)) != EOF)  
    printf("%c %d\n", c, c);
```

Leitura: caracteres

- Podemos testar se chegamos ao final do arquivo por meio da função `feof()`;

`feof` na condição do loop: **mal uso!**

```
1 FILE *arq;
2 char c;
3
4 arq = fopen("Teste.txt", "r");
5 if(arq == NULL){
6     ...
7 }
8
9 while(!feof(arq)){
10     c = fgetc(arq);
11     printf("%c", c);
12 }
13 fclose(arq);
```

`feof` na condição do loop: **uso melhor!**

```
1 FILE *arq;
2 char c;
3
4 arq = fopen("Teste.txt", "r");
5 if(arq == NULL){
6     ...
7 }
8
9 while(1){
10     c = fgetc(arq);
11     if(feof(arq))
12         break;
13     printf("%c", c);
14 }
15 fclose(arq);
```

`feof` verifica o indicador de erro

Escrita: strings

- Para escrever uma *string* em um arquivo, pode-se utilizar a função `fputs()`

```
int fputs(string, ponteiro_arquivo)
```

- Essa função recebe como parâmetro um vetor de caracteres (*string*) e um ponteiro para o arquivo no qual se deseja escrever.
- Retorno
 - Se o texto for escrito com sucesso um valor não-negativo é retornado;
 - Se ocorrer erro na escrita, o valor **EOF** é retornado.

Escrita: strings

```
1 FILE *arq;  
2 char str[] = "Texto a ser escrito no arquivo";  
3 int resultado;  
4 arq = fopen("Teste.txt", "a");  
5 if(arq == NULL){  
6     printf("Erro ao abrir o arquivo");  
7     system ("pause");  
8     exit(1);  
9 }  
10  
11 resultado = fputs(str, arq);  
12 if(resultado == EOF)  
13     printf("Erro na escrita");
```

```
1 FILE *arq;  
2 char str[100];  
3 arq = fopen("Teste.txt", "a");  
4  
5 if(arq == NULL){  
6     printf("Erro ao abrir o arquivo");  
7     system ("pause");  
8     exit(1);  
9 }  
10  
11 scanf("%s", str);  
12 fputs(str, arq);
```

Leitura: strings

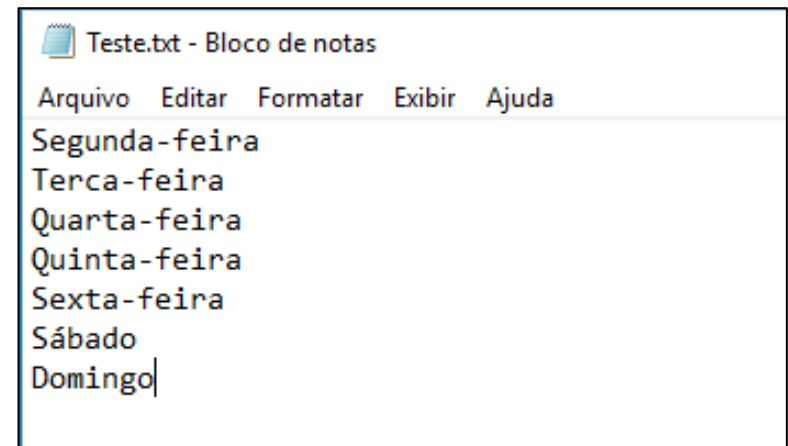
- Para ler uma *string* em um arquivo, pode-se utilizar a função `fgets()`

```
char *fgets(string_destino, n, ponteiro_arquivo)
```

- Lê até $n - 1$ caracteres do arquivo ou uma sequência terminada por uma nova linha;
 - A *string* resultante sempre terminará com `\0`, por isso somente $n - 1$ caracteres, no máximo, serão lidos;
 - **Lembre-se que:** se o caractere de nova linha `\n` for lido, ele fará parte da *string*;
- Retorno
 - Retorna NULL em caso de erro na leitura ou fim do arquivo;
 - O ponteiro para o primeiro caractere da *string* lida;

Leitura: strings

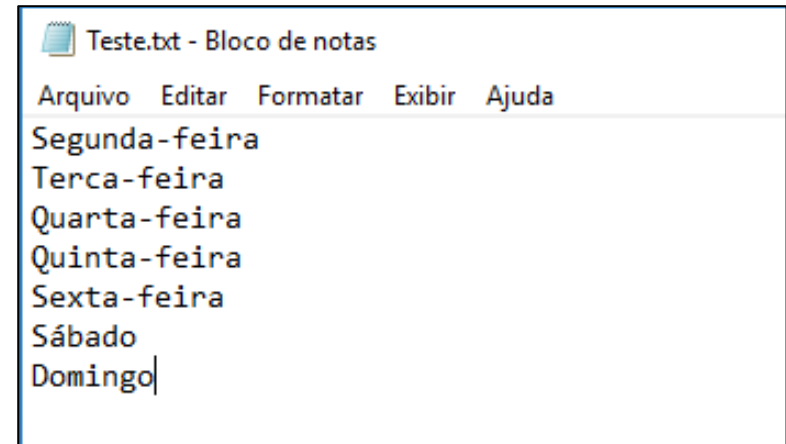
```
1 FILE *arq;
2 char str[100];
3
4 arq = fopen("Teste.txt", "r");
5 if(arq == NULL){
6     printf("Erro ao abrir o arquivo");
7     system("pause");
8     exit(1);
9 }
10
11 fgets(str, 100, arq);
12 printf("%s", str);
13
14 fclose(arq);
```



Saída: Segunda-feira

Leitura: strings

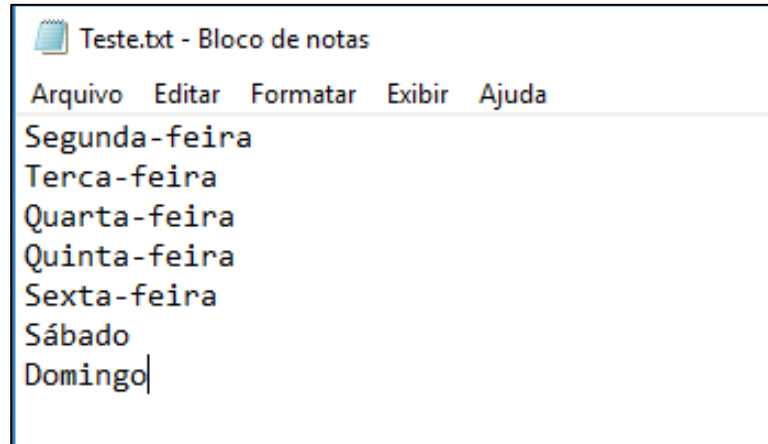
```
1 FILE *arq;  
2 char str[100];  
3  
4 arq = fopen("Teste.txt", "r");  
5 if(arq == NULL){  
6     printf("Erro ao abrir o arquivo");  
7     system("pause");  
8     exit(1);  
9 }  
10  
11 fgets(str, 100, arq);  
12 printf("%s", str);  
13  
14 fgets(str, 100, arq);  
15 printf("%s", str);  
16  
17 fclose(arq);
```



Saída: Segunda-feira
Terca-feira

Leitura: strings

```
1 FILE *arq;
2 char str[100];
3
4 arq = fopen("Teste.txt", "r");
5 if(arq == NULL){
6     printf("Erro ao abrir o arquivo");
7     system("pause");
8     exit(1);
9 }
10
11 while(1){
12     fgets(str, 100, arq);
13     if(feof(arq))
14         break;
15     printf("%s", str);
16 }
17 fclose(arq);
```



Saída:

Segunda-feira
Terca-feira
Quarta-feira
Quinta-feira
Sexta-feira
Sabado
Domingo

Leitura/Escreita formatada

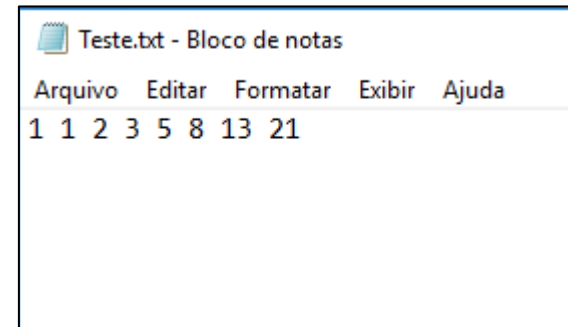
- Também podemos ler e escrever dados nos arquivos como fazemos com dados lidos do teclado ou impressos na tela;
- Isso permite maior flexibilidade, pois podemos trabalhar com os tipos de dados conhecidos: int, float, char, etc.
- Usamos para isso as funções fprintf e fscanf;

```
printf("Soma = %d", soma); //escreve na tela  
fprintf(arq, "Soma = %d", soma); //escreve no arquivo
```

```
scanf("%d", &soma); //lê do teclado  
fscanf(arq, "%d", &soma); //lê do arquivo arq
```

Leitura/Escreita formatada

```
1 FILE *arq;
2 int i, vet[8];
3 arq = fopen("Teste.txt", "r");
4 if(arq == NULL){
5     printf("Erro ao abrir o arquivo");
6     system("pause");
7     exit(1);
8 }
9
10 for(i=0;i<8;i++)
11     fscanf(arq, "%d", &vet[i]);
12
13 printf("Dados lidos do arquivo: ");
14 for(i=0;i<8;i++)
15     printf("%d ", vet[i]);
16
17 fclose(arq);
```



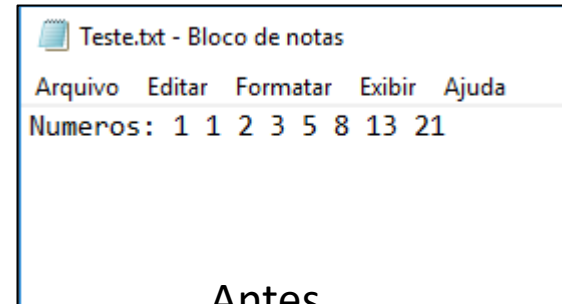
Saída:

Dados lidos do arquivo: 1 1 2 3 5 8 13 21

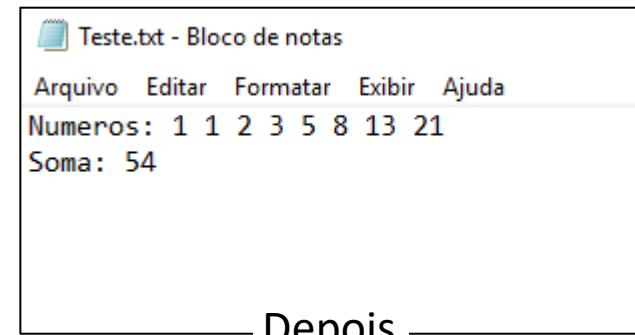
Leitura/Escreita formatada

```
1 FILE *arq;  
2 int i, soma = 0, vet[8];  
3 char str[20];  
4 arq = fopen("Teste.txt", "r+");  
5 if(arq == NULL){  
6     printf("Erro ao abrir o arquivo");  
7     system("pause");  
8     exit(1);  
9 }  
10  
11 fscanf(arq, "%s", str); //Lê a string inicial  
12 for(i=0;i<8;i++){  
13     fscanf(arq, "%d", &vet[i]);  
14     soma += vet[i];  
15 }  
16  
17 fprintf(arq, "\nSoma: %d", soma);  
18 fclose(arq);
```

Atenção ao modo de abertura!



Antes

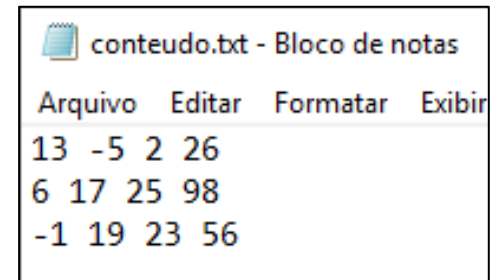


Depois

Leitura/Escreita formatada

Programa para gravar dados no arquivo

```
1 int i, j, matriz[3][4];
2 FILE *arq;
3
4 arq = fopen("conteudo.txt", "r");
5 if(arq == NULL){
6     printf("Erro ao abrir o arquivo");
7     exit(1);
8 }
9
10 for(i=0;i<3;i++){
11     for(j=0;j<4;j++){
12         fscanf(arq, "%d", &matriz[i][j]);
13     }
14
15     for(i=0;i<3;i++){
16         for(j=0;j<4;j++){
17             printf("%d ", matriz[i][j]);
18             printf("\n");
19         }
20
21     fclose(arq);
```



Leitura/Escreita formatada

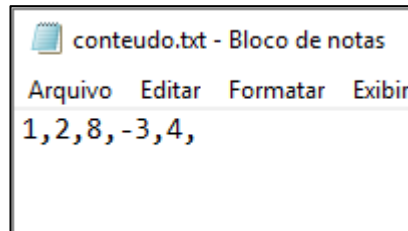
- Quando estiver usando a função de leitura fscanf:
 - Se existir delimitadores, deve-se manter o padrão usado no “tipos de saída” do fprintf(), e assim conseguir separar a leitura. Exemplos: ‘ ’, ‘\t’, ‘\n’
 - Colocar delimitadores no “tipos de entrada” significa ignorá-los na leitura e indica onde uma variável termina...
 - Veja o exemplo nos slides seguintes

Leitura/Escreita formatada

Programa para gravar dados no arquivo

```
int vet[5] = {1, 2, 8, -3, 4}, i;  
FILE *arq;  
  
arq = fopen("conteudo.txt", "w");  
  
if(arq == NULL){  
    printf("Erro ao abrir o arquivo");  
    exit(1);  
}  
  
for(i=0; i<5; i++){  
    fprintf(arq, "%d", vet[i]);  
}  
  
fclose(arq);
```

Arquivo gerado



Programa para ler dados no arquivo

```
int vet[5], i;  
FILE *arq;  
  
arq = fopen("conteudo.txt", "r");  
  
if(arq == NULL){  
    printf("Erro ao abrir o arquivo");  
    exit(1);  
}  
  
for(i=0; i<5; i++){  
    fscanf(arq, "%d", &vet[i]);  
}  
  
for(i=0; i<5; i++){  
    printf(" %d ", vet[i]);  
}  
  
fclose(arq);
```

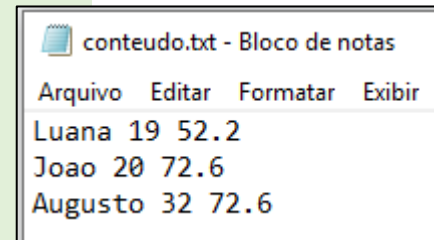

Leitura/Escreita formatada

Programa para ler dados no arquivo

Lendo string e números (3 linhas)

```
1 struct Pessoa{
2     char nome[30];
3     int idade;
4     double peso;
5 };
6
7 int main(){
8     struct Pessoa p[3];
9     int i;
10    FILE *arq;
11
12    arq = fopen("conteudo.txt", "r");
13    if(arq == NULL){
14        printf("Erro ao abrir o arquivo");
15        exit(1);
16    }
17    for(i=0;i<3;i++){
18        fscanf(arq,"%s %d %lf\n", p[i].nome, &p[i].idade, &p[i].peso);
19    }
20
21    for(i=0;i<3;i++){
22        printf("Nome: %s, Idade: %d, Peso: %lf\n", p[i].nome, p[i].idade, p[i].peso);
23    }
24
25    fclose(arq);
```

Arquivo a ser lido



Arquivo	Editar	Formatar	Exibir
Luana	19	52.2	
Joao	20	72.6	
Augusto	32	72.6	

Para ler valores do tipo **double**

Leitura/Escreita formatada

Lendo strings e números separados por vírgula (3 linhas)

Programa para ler dados no arquivo

```
1 struct Pessoa{
2     char nome[30];
3     int idade;
4     double peso;
5 };
6
7 int main(){
8     struct Pessoa p[3];
9     int i;
10    FILE *arq;
11
12    arq = fopen("conteudo.txt", "r");
13    if(arq == NULL){
14        printf("Erro ao abrir o arquivo");
15        exit(1);
16    }
17    for(i=0;i<3;i++){
18        fscanf(arq,"%[^,], %d, %lf\n", p[i].nome, &p[i].idade, &p[i].peso);
19    }
20
21    for(i=0;i<3;i++){
22        printf("Nome: %s, Idade: %d, Peso: %lf\n", p[i].nome, p[i].idade, p[i].peso);
23    }
24
25    fclose(arq);
```

conteudo.txt - Bloco de notas

Arquivo	Editar	Formatar	Exibir
Luana Cardoso, 19, 52.2			
Joao Pereira, 20, 72.6			
Augusto Rodrigo, 32, 72.6			

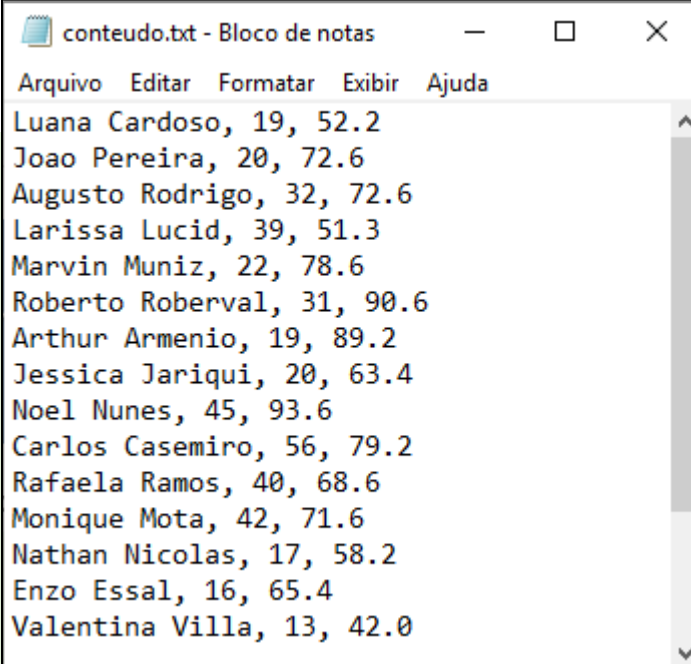
`%[^,]` faz com que todos os caracteres antes da vírgula sejam lidos. Eles vão ser armazenados na `string p[i].nome`

Leitura/Escreita formatada

Lendo strings e números separados por vírgula
(número desconhecido de linhas)

Porque não sabemos informação de quantas pessoas estão no arquivo

```
1 struct Pessoa{
2     ...
3 };
4
5 int main(){
6     struct Pessoa p[100];
7     int i, cont=0;
8     FILE *arq;
9
10    arq = fopen("conteudo.txt", "r");
11    if(arq == NULL){
12        printf("Erro ao abrir o arquivo");
13        exit(1);
14    }
15    while(1){
16        fscanf(arq,"%[^,], %d, %lf\n", p[cont].nome, &p[cont].idade, &p[cont].peso);
17        cont++;
18        if(feof(arq))
19            break;
20    }
21    for(i=0;i<cont;i++){
22        printf("Nome: %s, Idade: %d, Peso: %lf\n", p[i].nome, p[i].idade, p[i].peso);
23    }
24
25    fclose(arq);
```



conteudo.txt - Bloco de notas

Arquivo Editar Formatar Exibir Ajuda

Luana Cardoso, 19, 52.2
Joao Pereira, 20, 72.6
Augusto Rodrigo, 32, 72.6
Larissa Lucid, 39, 51.3
Marvin Muniz, 22, 78.6
Roberto Roberval, 31, 90.6
Arthur Armenio, 19, 89.2
Jessica Jariquei, 20, 63.4
Noel Nunes, 45, 93.6
Carlos Casemiro, 56, 79.2
Rafaela Ramos, 40, 68.6
Monique Mota, 42, 71.6
Nathan Nicolas, 17, 58.2
Enzo Essal, 16, 65.4
Valentina Villa, 13, 42.0

Roteiro



- 1 Introdução
- 2 Arquivos
- 3 Leitura/Escrita
- 4 Exercícios
- 5 Referências

Exercícios

Universidade Tecnológica Federal do Paraná – Câmpus Apucarana
Computação 2 (CP63B) - DPGR3A – Arquivos
Prof. Dr. Rafael Gomes Mantovani

Instruções:

Moodle :))

- Antes de codificar, esboce em um papel a sequencia de passos necessários para criar o seu programa. Isso ajuda a programar a solução;
- Crie um arquivo .c para cada um dos exercícios. Por exemplo, na resolução do exercício 01, crie um arquivo chamado 'ex01.c'.
- em todos os exercícios faça uma função `main` para testar sua função.

Exercícios sobre Arquivos

Exercício 1. Crie manualmente um arquivo de texto com uma matriz 5 x 5 de números inteiros. Faça um programa que, a partir da leitura desse arquivo texto, preencha uma matriz usando o comando `fscanf`. Em seguida, mostre a matriz lida do arquivo.

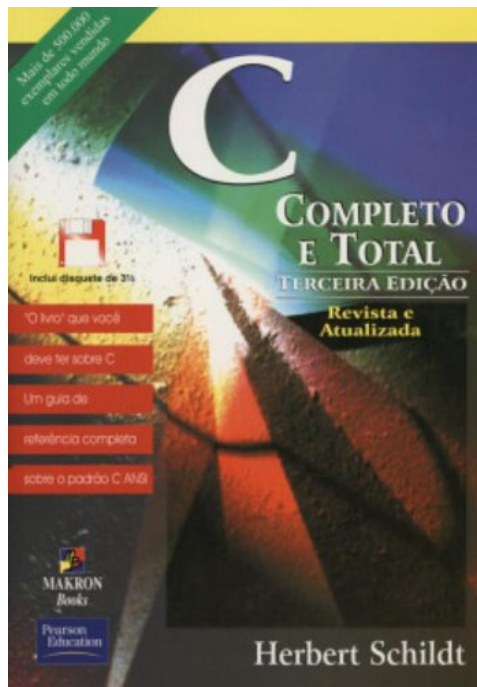
Exercício 2. Crie um programa que armazene em um arquivo, informações sobre 4 alunos. Receba as informações das pessoas por meio do teclado. Cada pessoa deve ter nome, número da matrícula, curso e media final. O arquivo deverá se chamar “Alunos.txt”

Roteiro



- 1** Introdução
- 2** Recursão
- 3** Exemplo
- 4** Exercícios
- 5** Referências

Referências



[Schildt, 1997]



[de Souza et al, 2011]

Referências

- [Schildt, 1997] SCHILDT, H. **C Completo e Total**. 3. ed. São Paulo: Pearson, 1997.
- [de Souza et al, 2011] DE SOUZA, M. A. F. et al. **Algoritmos e lógica de programação**. 2. ed. São Paulo: Cengage Learning, 2011.

Perguntas?

Prof. Rafael G. **Mantovani**

rafaelmantovani@utfpr.edu.br