2º Semestre

Técnicas de Programação

Luiz Fernando Carvalho

luizfcarvalhoo@gmail.com

Arquivos

- 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).

Arquivos

- 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 dado podem ou não ser sequencial;
  - Acesso concorrente aos dados (mais de um programa “pode” usar os dados ao mesmo tempo);

Tipos de Arquivos

- 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 dede 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 (*append*).
"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 (*append*)
"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 (*append*).
"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.

Abrindo um Arquivo

- O padrão C11 possui novos modos de abertura para escrita:
  - O arquivo não terá seu conteúdo apagado caso o mesmo já exista;
  - A função irá falhar caso o arquivo já exista ou não possa ser criado;
  - O arquivo será criado para acesso exclusivo, ou seja, não compartilhado;

Modo	Arquivo	Função
“wx”	Texto	Cria um arquivo vazio para escrita.
“wbx”	Binário	Cria um arquivo vazio para escrita.
“w+x”	Texto	Cria um arquivo vazio para leitura/escrita.
“w+bx”	Binário	Cria um arquivo vazio para leitura/escrita.

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 ao abrir arquivos

- 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 ao abrir arquivos

- 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, término normal do programa;
    - Um número diferente de zero, algum problema ocorreu.

Erros ao abrir arquivos

```
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;

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 de caracteres em Arquivos

- 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);
```

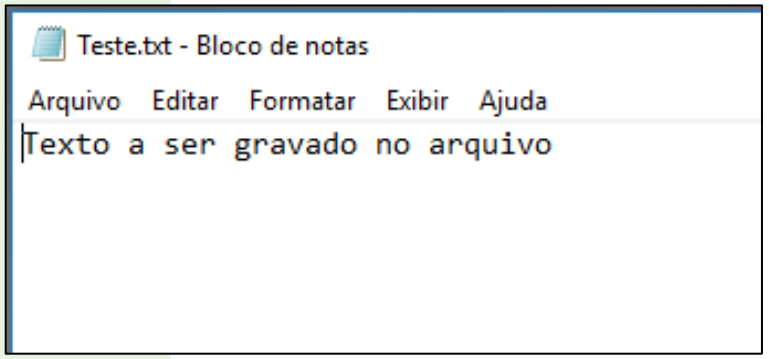
Usada também para impressão:

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

Escrita de caracteres em Arquivos

- 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 de caracteres em Arquivos

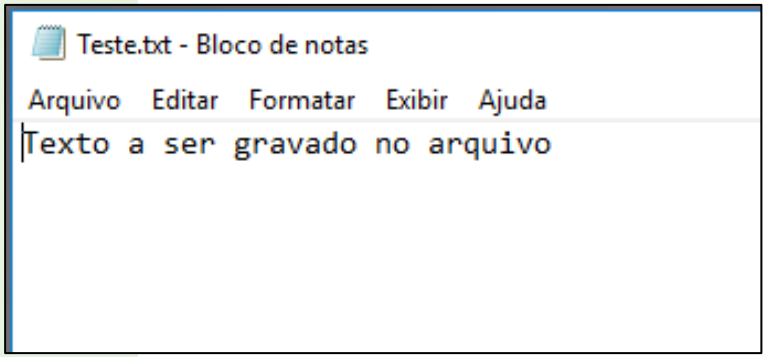
- Também podemos ler caracteres um a um do arquivo;
- A função usada para isso é a `fgetc` (*get character*);

```
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 de caracteres em Arquivos

- A assinatura da função é a seguinte:

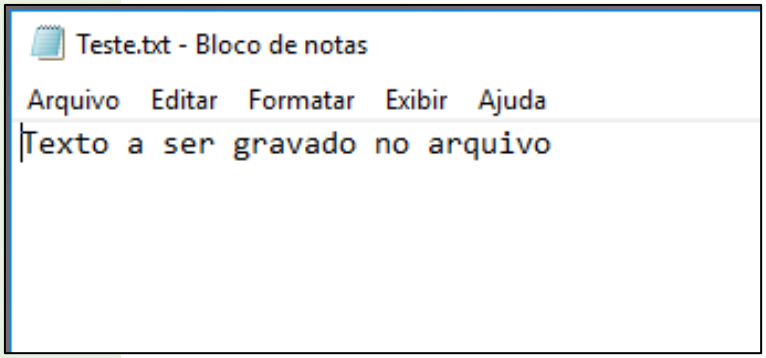
```
int fgetc(FILE *arq)
```

- 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 de caracteres em Arquivos

- 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);
```



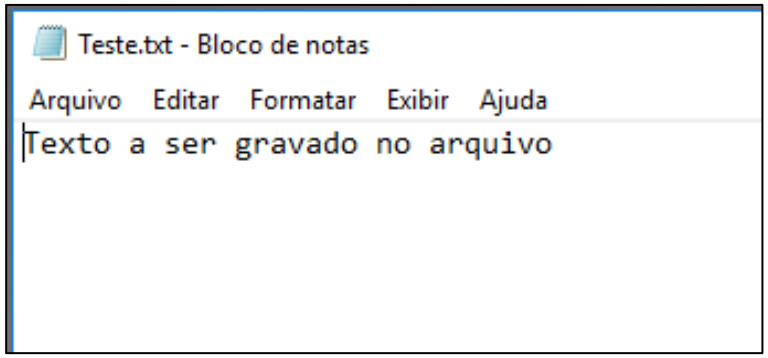
Saída:

Texto a ser gravado no arquivo

Leitura de caracteres em Arquivos

- 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?

Equivale

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

Leitura de caracteres em Arquivos

- 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` verifica o indicador de erro

`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);
```

Escrita de strings em Arquivos

- 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 de strings em Arquivos

```
 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 fgets(str, 100, stdin);
12 fputs(str, arq);
```

Leitura de strings deArquivos

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

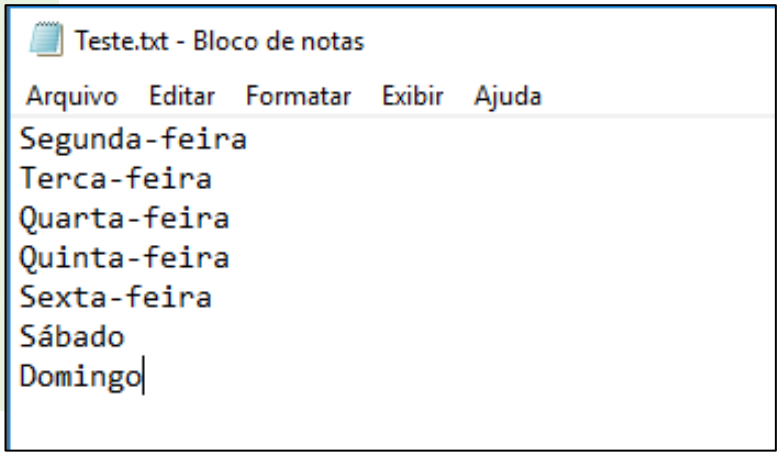
```
char *fgets(string_destino, n, ponteiro_arquivo)
```

- 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 de strings de Arquivos

```
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);
```

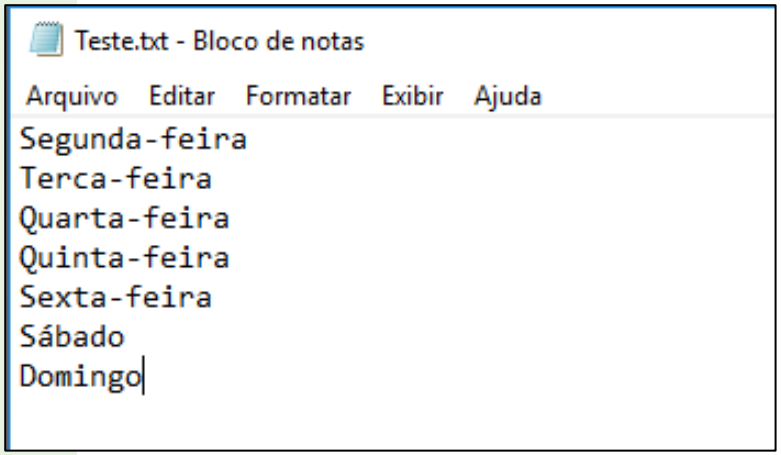


Saída:

Segunda-feira

Leitura de strings de Arquivos

```
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);
```

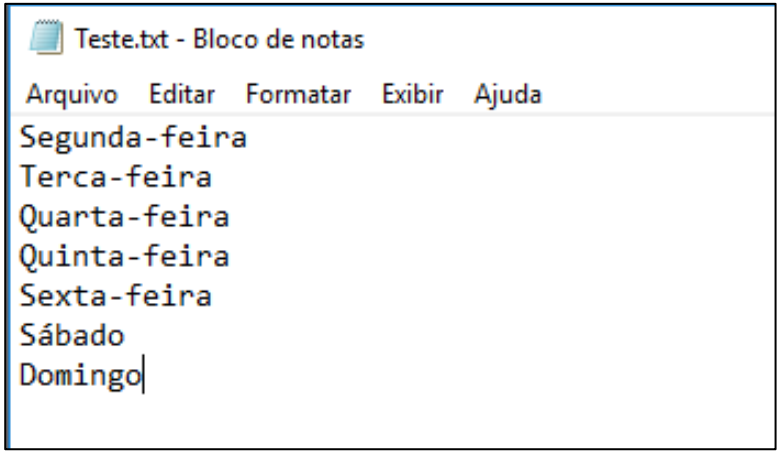


Saída:

Segunda-feira
Terca-feira

Leitura de strings de Arquivos

```
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/Escrita Formatada em Arquivos

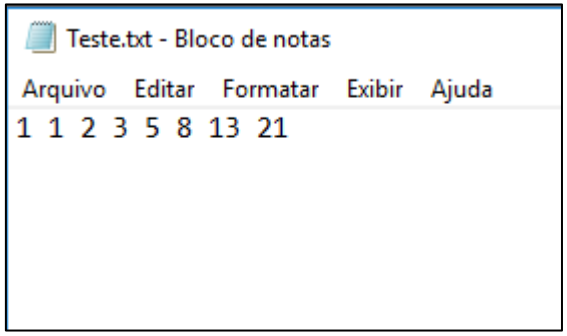
- Também podemos ler e escrever dados nos arquivos como fazemos com dados lidos do teclado ou impressos na tela;
- Isso permite maior flexibilidade, pois podemos trabalhar com os tipos de dados conhecidos: `int`, `float`, `char`, etc.
- Usamos para isso as funções `fprintf` e `fscanf`;

```
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/Escrita Formatada em Arquivos

```
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);
```



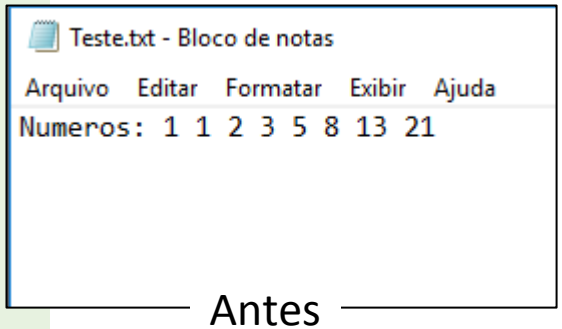
Saída:

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

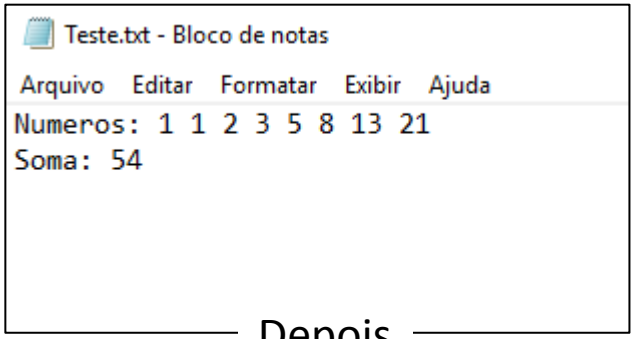
Leitura/Escrita Formatada em Arquivos

```
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/Escrita Formatada em Arquivos

- Embora seja mais fácil trabalhar com leitura e escrita dessa maneira, existem desvantagens
  - **DESEMPENHO:** Como os dados são escritos em ASCII e formatados como apareceriam na tela, um tempo extra é perdido a cada chamada de E/S;
  - Se a intenção é ter velocidade de execução, pode-se utilizar outras funções, como `fread` e `fwrite`.