Faculdade de Computação - FACOM

Bacharelado em Sistemas de Informação

FACOM32201 - Algoritmos e Programação II

Prof. Thiago Pirola Ribeiro

ARQUIVOS

Parte 1

Arquivos

- Por que usar arquivos?

Arquivos

- Por que usar arquivos?
  - Permitem armazenar grande quantidade de informação;

Arquivos

- Por que usar arquivos?
  - Permitem armazenar grande quantidade de informação;
  - Persistência dos dados (disco);

Arquivos

- Por que usar arquivos?
  - Permitem armazenar grande quantidade de informação;
  - Persistência dos dados (disco);
  - Acesso aos dados poder ser não sequencial;

Arquivos

- Por que usar arquivos?
  - Permitem armazenar grande quantidade de informação;
  - Persistência dos dados (disco);
  - Acesso aos dados poder ser não 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:
  - **Textos** e
  - **Binários**.

Tipos de Arquivos

- **Arquivo texto**

Tipos de Arquivos

- **Arquivo texto**
  - armazena caracteres que podem ser mostrados diretamente na tela ou modificados por um editor de textos simples como o **Bloco de Notas**.

Tipos de Arquivos

- **Arquivo texto**
  - armazena caracteres que podem ser mostrados diretamente na tela ou modificados por um editor de textos simples como o **Bloco de Notas**.
  - Os dados são gravados como caracteres de 8 bits.

    Ex.: Um número inteiro de 32 bits com 8 dígitos ocupará 64 bits no arquivo (8 bits por dígito).

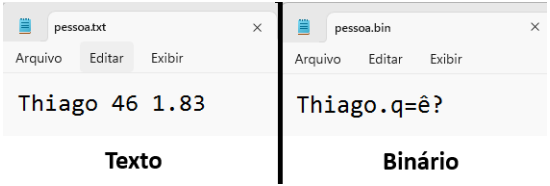
Tipos de Arquivos

- **Arquivo binário**
  - armazena uma sequência de bits que está sujeita as convenções dos programas que o gerou. Ex: **arquivos executáveis, arquivos compactados, arquivos de registros**, etc.
  - os dados são gravados na forma binária (do mesmo modo que estão na memória). Ex.: um número inteiro de 32 bits com 8 dígitos ocupará 32 bits no arquivo.

Tipos de Arquivos

Ex: Os dois trechos de arquivo abaixo possuem os mesmo dados:

```
1    char nome[20] = "Thiago";
2    int i = 46;
3    float a = 1.83;
```



Manipulando arquivos em C

- A linguagem C possui uma série de funções para manipulação de arquivos, cujos protótipos estão reunidos na biblioteca padrão de entrada e saída: `stdio.h`.

Manipulando arquivos em C

- A linguagem C possui uma série de funções para manipulação de arquivos, cujos protótipos estão reunidos na biblioteca padrão de entrada e saída: `stdio.h`.
- A linguagem C não possui funções que automaticamente leiam todas as informações de um arquivo.

Manipulando arquivos em C

- A linguagem C possui uma série de funções para manipulação de arquivos, cujos protótipos estão reunidos na biblioteca padrão de entrada e saída: `stdio.h`.
- A linguagem C não possui funções que automaticamente leiam todas as informações de um arquivo.
- Suas funções se limitam a:

Manipulando arquivos em C

- A linguagem C possui uma série de funções para manipulação de arquivos, cujos protótipos estão reunidos na biblioteca padrão de entrada e saída: `stdio.h`.
- A linguagem C não possui funções que automaticamente leiam todas as informações de um arquivo.
- Suas funções se limitam a:
  - abrir/fechar e

Manipulando arquivos em C

- A linguagem C possui uma série de funções para manipulação de arquivos, cujos protótipos estão reunidos na biblioteca padrão de entrada e saída: `stdio.h`.
- A linguagem C não possui funções que automaticamente leiam todas as informações de um arquivo.
- Suas funções se limitam a:
  - abrir/fechar e
  - ler caracteres/bytes

Manipulando arquivos em C

- A linguagem C possui uma série de funções para manipulação de arquivos, cujos protótipos estão reunidos na biblioteca padrão de entrada e saída: `stdio.h`.
- A linguagem C não possui funções que automaticamente leiam todas as informações de um arquivo.
- Suas funções se limitam a:
  - abrir/fechar e
  - ler caracteres/bytes
- É tarefa do programador criar a função que lerá um arquivo de uma maneira específica.

Manipulando arquivos em C

- Todas as funções de manipulação de arquivos trabalham com o conceito de "ponteiro de arquivo". Podemos declarar um ponteiro de arquivo da seguinte maneira:

```
FILE *p;
```

Manipulando arquivos em C

- Todas as funções de manipulação de arquivos trabalham com o conceito de "ponteiro de arquivo". Podemos declarar um ponteiro de arquivo da seguinte maneira:

```
FILE *p;
```

- p é o ponteiro para arquivos que permitirá manipular arquivos no C.

Abrindo um Arquivo

Abrindo um arquivo

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

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

Abrindo um arquivo

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

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

- O parâmetro `nome_arquivo` determina qual arquivo deverá ser aberto, sendo que o mesmo deve ser válido no sistema operacional que estiver sendo utilizado.

Abrindo um arquivo

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

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

- O parâmetro `nome_arquivo` determina qual arquivo deverá ser aberto, sendo que o mesmo deve ser válido no sistema operacional que estiver sendo utilizado.
- No parâmetro `nome_arquivo` pode-se trabalhar com caminhos absolutos ou relativos.

Abrindo um arquivo

- No parâmetro `nome_arquivo` pode-se trabalhar com caminhos absolutos ou relativos.
  - Caminho absoluto: descrição de um caminho desde o diretório raiz.

    `C:\Alg2\dados.txt`
  - Caminho relativo: descrição de um caminho desde o diretório corrente:

    `dados.txt` ou `..\dados.txt`

Abrindo um arquivo

- O modo de abertura determina que tipo de uso será feito do arquivo.
- A tabela a seguir mostra alguns dos modo válidos de abertura de um arquivo.

Modo	Arquivo	Função
"r"	Texto	Leitura. Arquivo deve existir.
"w"	Texto	Escrita. Cria arquivo se não houver. Apaga o anterior se ele existir.
"a"	Texto	Escrita. Os dados serão adicionados no fim 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").

Modo	Arquivo	Função
“r”	Texto	Leitura. Arquivo deve existir.
“w”	Texto	Escrita. Cria arquivo se não houver. Apaga o anterior se ele existir.
“a”	Texto	Escrita. Os dados serão adicionados no fim 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.
“a+b”	Binário	Leitura/Escrita. Os dados serão adicionados no fim do arquivo (“append”).

Abrindo um arquivo

- Um arquivo binário pode ser aberto para escrita utilizando o seguinte conjunto de comandos:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main() {
4     FILE *fp;
5     fp = fopen("dados.bin", "wb");
6     if (fp == NULL)
7         printf("Erro ao abrir o arquivo.\n");
8
9     fclose(fp);
10    return 0;
11 }
```

Abrindo um arquivo

- Um arquivo binário pode ser aberto para escrita utilizando o seguinte conjunto de comandos:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main() {
4     FILE *fp;
5     fp = fopen("dados.bin", "wb");
6     if (fp == NULL) // testa se o arquivo foi aberto com sucesso
7         printf("Erro ao abrir o arquivo.\n");
8
9     fclose(fp);
10    return 0;
11 }
```

Erro ao abrir um arquivo

- Caso o arquivo não tenha sido aberto com sucesso

Erro ao abrir um arquivo

- Caso o arquivo não tenha sido aberto com sucesso
  - Provavelmente o programa não poderá continuar a executar;

Erro ao abrir um arquivo

- Caso o arquivo não tenha sido aberto com sucesso
  - Provavelmente o programa não poderá continuar a executar;
  - Nesse caso, utiliza-se a função `exit()`, presente na biblioteca `stdlib.h`, para abortar o programa

```
void exit (int codigo_de_retorno);
```

Erro ao abrir um arquivo

- Caso o arquivo não tenha sido aberto com sucesso
  - Provavelmente o programa não poderá continuar a executar;
  - Nesse caso, utiliza-se a função `exit()`, presente na biblioteca `stdlib.h`, para abortar o programa

    ```
    void exit (int codigo_de_retorno);
    ```

  - A função `exit()` pode ser chamada de qualquer ponto no programa e faz com que o programa termine e retorne, para o sistema operacional, o `código_de_retorno`.

Erro ao abrir um arquivo

- Caso o arquivo não tenha sido aberto com sucesso
  - Provavelmente o programa não poderá continuar a executar;
  - Nesse caso, utiliza-se a função `exit()`, presente na biblioteca `stdlib.h`, para abortar o programa

    ```
    void exit (int codigo_de_retorno);
    ```

  - A função `exit()` pode ser chamada de qualquer ponto no programa e faz com que o programa termine e retorne, para o sistema operacional, o `código_de_retorno`.
  - A convenção mais usada é que um programa retorne **zero** no caso de um **término normal** e retorne um número não nulo no caso de ter ocorrido um problema.

Erro ao abrir um arquivo

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
    FILE *fp;
    fp = fopen("dados.bin", "wb");
    if (fp == NULL){
        printf("Erro ao abrir o arquivo.\n");
        exit(1);
   }
    fclose(fp);
    return 0;
}
```

Posição um Arquivo

Posição do arquivo

- Ao se trabalhar com arquivos, existe uma espécie de posição onde estamos dentro do arquivo. É nessa posição onde será lido ou escrito o próximo caractere.

Posição do arquivo

- Ao se trabalhar com arquivos, existe uma espécie de posição onde estamos 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 lemos um caractere, a posição no arquivo é automaticamente atualizada.

Fechando um Arquivo

Fechando um arquivo

- Sempre que se termina de usar um arquivo que abrimos, deve-se fechá-lo.

Fechando um arquivo

- Sempre que se termina de usar um arquivo que abrimos, deve-se fechá-lo.
- Para isso usa-se a função `fclose()`

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

Fechando um arquivo

- Sempre que se termina de usar um arquivo que abrimos, deve-se fechá-lo.
- Para isso usa-se a função `fclose()`

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

- O ponteiro `fp` passado à função `fclose()` determina o arquivo a ser fechado. A função retorna zero no caso de sucesso.

Fechando um arquivo

- Sempre que se termina de usar um arquivo que abrimos, deve-se fechá-lo.
- Para isso usa-se a função `fclose()`

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

- O ponteiro `fp` passado à função `fclose()` determina o arquivo a ser fechado. A função retorna zero no caso de sucesso.
- Ao fechar um arquivo, todo caractere que tenha permanecido no "buffer" é gravado.

Fechando um arquivo

- Sempre que se termina de usar um arquivo que abrimos, deve-se fechá-lo.
- Para isso usa-se a função `fclose()`

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

- O ponteiro `fp` passado à função `fclose()` determina o arquivo a ser fechado. A função retorna zero no caso de sucesso.
- Ao fechar um arquivo, todo caractere que tenha permanecido no "buffer" é gravado.
- O "buffer" é uma região de memória que armazena temporariamente os caracteres a serem gravados em disco imediatamente. Apenas quando o "buffer" está cheio é que seu conteúdo é escrito no disco.

Fechando um arquivo

- Por que utilizar um “buffer”? **Eficiência!**

Fechando um arquivo

- Por que utilizar um "buffer"? **Eficiência!**
  - Para ler e escrever arquivos no disco deve-se que posicionar a cabeça de gravação em um ponto específico do disco.

Fechando um arquivo

- Por que utilizar um "buffer"? **Eficiência!**
  - Para ler e escrever arquivos no disco deve-se que posicionar a cabeça de gravação em um ponto específico do disco.
  - Se precisasse fazer isso para cada caractere lido/escrito, a leitura/escrita de um arquivo seria uma operação muita lenta.

Fechando um arquivo

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

Fechando um arquivo

- Por que utilizar um "buffer"? **Eficiência!**
  - Para ler e escrever arquivos no disco deve-se que posicionar a cabeça de gravação em um ponto específico do disco.
  - Se precisasse fazer isso para cada caractere lido/escrito, a leitura/escrita de um arquivo seria uma operação muita lenta.
  - Assim a gravação só é realizada quando há um volume razoável de informações a serem gravadas ou quando o arquivo for fechado.
- A função `exit()` fecha todos os arquivos que um programa tiver aberto.

Escrita/Leitura em Arquivos

Escrita/Leitura em Arquivos

- Uma vez aberto um arquivo, pode-se ler ou escrever nele.
- Para tanto, a linguagem C conta com uma série de funções de leitura/escrita que variam de funcionalidade para atender as diversas aplicações.

Escrita/Leitura de Caracteres

- A maneira mais fácil de se trabalhar com um arquivo é a **leitura/escrita de um único caractere**.

Escrita/Leitura de Caracteres

- A maneira mais fácil de se trabalhar com um arquivo é a **leitura/escrita de um único caractere**.
- A função mais básica de entrada de dados é a função `fputc (put character)`.

```
int fputc (int ch, FILE *fp);
```

Escrita/Leitura de Caracteres

- A maneira mais fácil de se trabalhar com um arquivo é a **leitura/escrita de um único caractere**.
- A função mais básica de entrada de dados é a função `fputc (put character)`.

```
int fputc (int ch, FILE *fp);
```

- Cada invocação dessa função grava um único caractere `ch` no arquivo especificado por `fp`.

Escrita/Leitura de Caracteres

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main() {
    FILE *arquivo;
    char info[100];
    arquivo = fopen("dados.txt", "w");
    if (arquivo == NULL) {
        printf("Erro ao abrir o arquivo");
        exit(1);
    }
    printf("Entre com a string a ser gravada no arquivo:");
    gets(info);
    for(int i=0; i < strlen(info); i++)
        fputc(info[i], arquivo);
    fclose(arquivo);
    return 0;
}
```

Escrita/Leitura de Caracteres

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  int main() {
5      FILE *arquivo;
6      char info[100];
7      arquivo = fopen("dados.txt", "w");
8      if (arquivo == NULL) {
9          printf("Erro ao abrir o arquivo");
10         exit(1);
11     }
12     printf("Entre com a string a ser gravada no arquivo:");
13     gets(info);
14     for(int i=0; i < strlen(info); i++)
15         fputc(info[i], arquivo);
16     fclose(arquivo);
17     return 0;
```

Escrita/Leitura de Caracteres

- A função `fputc` também pode ser utilizada para escrever um caractere na tela. Nesse caso, é necessário mudar a variável que aponta para o local onde será gravado o caractere:

Escrita/Leitura de Caracteres

- A função `fputc` também pode ser utilizada para escrever um caractere na tela. Nesse caso, é necessário mudar a variável que aponta para o local onde será gravado o caractere:
- Por exemplo, `fputc('*', stdout)` exibe um $*$ na tela do monitor (dispositivo de saída padrão).

Escrita/Leitura de Caracteres

- A função `fputc` também pode ser utilizada para escrever um caractere na tela. Nesse caso, é necessário mudar a variável que aponta para o local onde será gravado o caractere:
- Por exemplo, `fputc('*', stdout)` exibe um $*$ na tela do monitor (dispositivo de saída padrão).
- Da mesma maneira que gravamos um único caractere no arquivo, a leitura também é possível.

Escrita/Leitura de Caracteres

- A função `fputc` também pode ser utilizada para escrever um caractere na tela. Nesse caso, é necessário mudar a variável que aponta para o local onde será gravado o caractere:
- Por exemplo, `fputc('*', stdout)` exibe um * na tela do monitor (dispositivo de saída padrão).
- Da mesma maneira que gravamos um único caractere no arquivo, a leitura também é possível.
- A função correspondente de leitura de caracteres é `fgetc(get character)`.

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

Escrita/Leitura de Caracteres

- Cada chamada da função fgetc lê um único caractere do arquivo especificado.

Escrita/Leitura de Caracteres

- Cada chamada da função fgetc lê um único caractere do arquivo especificado.
- Se fp aponta para um arquivo então fgetc(fp) lê o caractere atual no arquivo e se posiciona para ler o próximo caractere do arquivo.

```
char c;
c = fgetc(fp);
```

Escrita/Leitura de Caracteres

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
    FILE *arquivo;
    char c;
    arquivo = fopen("dados.txt", "r");
    if (arquivo == NULL) {
        printf("Erro ao abrir o arquivo");
        exit(1);
    }
    for(int i=0; i < 5; i++){
      c = fgets(arquivo);
        printf("%c",c);
    }
    fclose(arquivo);
    return 0;
}
```

Escrita/Leitura de Caracteres

- Similar ao que acontece com a função `fputc`, a função `fgetc` também pode ser utilizada para a leitura do teclado (dispositivo de entrada padrão);

Escrita/Leitura de Caracteres

- Similar ao que acontece com a função `fputc`, a função `fgetc` também pode ser utilizada para a leitura do teclado (dispositivo de entrada padrão);
- Nesse caso, `fgetc(stdin)` lê o próximo caractere digitado no teclado.

Escrita/Leitura de Caracteres

- Similar ao que acontece com a função `fputc`, a função `fgetc` também pode ser utilizada para a leitura do teclado (dispositivo de entrada padrão);
- Nesse caso, `fgetc(stdin)` lê o próximo caractere digitado no teclado.
- O que acontece quando `fgetc` tenta ler o próximo caractere de um arquivo que já acabou?

Escrita/Leitura de Caracteres

- Similar ao que acontece com a função `fputc`, a função `fgetc` também pode ser utilizada para a leitura do teclado (dispositivo de entrada padrão);
- Nesse caso, `fgetc(stdin)` lê o próximo caractere digitado no teclado.
- O que acontece quando `fgetc` tenta ler o próximo caractere de um arquivo que já acabou?
- É necessário que a função retorne algo indicando o arquivo acabou.

Escrita/Leitura de Caracteres

- Similar ao que acontece com a função `fputc`, a função `fgetc` também pode ser utilizada para a leitura do teclado (dispositivo de entrada padrão);
- Nesse caso, `fgetc(stdin)` lê o próximo caractere digitado no teclado.
- O que acontece quando `fgetc` tenta ler o próximo caractere de um arquivo que já acabou?
- É necessário que a função retorne algo indicando o arquivo acabou.
- Porém, todos os 256 caracteres são "válidos"!

Escrita/Leitura de Caracteres

- Para evitar esse tipo de situação, `fgetc` não devolve um char mas um int:

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

Escrita/Leitura de Caracteres

- Para evitar esse tipo de situação, `fgetc` não devolve um char mas um int:

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

- O conjunto de valores do **`char`** está contido dentro do conjunto do **`int`**.

Escrita/Leitura de Caracteres

- Para evitar esse tipo de situação, `fgetc` não devolve um char mas um int:

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

- O conjunto de valores do **`char`** está contido dentro do conjunto do **`int`**.
- Se o arquivo tiver acabado, `fgetc` devolve um **`int`** que não possa ser confundido com um **`char`**.

Escrita/Leitura de Caracteres

- Para evitar esse tipo de situação, `fgetc` não devolve um char mas um int:

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

- O conjunto de valores do **`char`** está contido dentro do conjunto do **`int`**.
- Se o arquivo tiver acabado, `fgetc` devolve um **`int`** que não possa ser confundido com um **`char`**.
- Assim, se o arquivo não tiver mais caracteres, `fgetc` devolve -1.

Escrita/Leitura de Caracteres

- Para evitar esse tipo de situação, `fgetc` não devolve um char mas um int:

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

- O conjunto de valores do **char** está contido dentro do conjunto do **int**.
- Se o arquivo tiver acabado, `fgetc` devolve um **int** que não possa ser confundido com um **char**.
- Assim, se o arquivo não tiver mais caracteres, `fgetc` devolve -1.
- Mais exatamente, `fgetc` devolve a constante `EOF (end of file)`, que está definida na biblioteca `stdio.h`. Em muitos computadores o valor de `EOF` é -1.

```
if (c == EOF) printf ("\nO arquivo terminou!");
```

Escrita/Leitura de Caracteres

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
    FILE *arquivo;
    char c;
    arquivo = fopen("dados.txt", "r");
    if (arquivo == NULL) {
        printf("Erro ao abrir o arquivo");
        exit(1);
    }
    while((c = fgetc(arquivo)) != EOF)
        printf("%c",c);
    fclose(arquivo);
    return 0;
}
```

Escrita/Leitura de Caracteres

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
    FILE *arquivo;
    char c;
    arquivo = fopen("dados.txt", "r");
    if (arquivo == NULL) {
        printf("Erro ao abrir o arquivo");
        exit(1);
    }
    while((c = fgetc(arquivo)) != EOF)
        printf("%c",c);
    fclose(arquivo);
    return 0;
}
```

Fim do arquivo

- Como visto, `EOF ("End of file")` indica o fim de um arquivo. No entanto, podemos também utilizar a função `feof` para verificar se um arquivo chegou ao fim.

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

Fim do arquivo

- Como visto, `EOF ("End of file")` indica o fim de um arquivo. No entanto, podemos também utilizar a função `feof` para verificar se um arquivo chegou ao fim.

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

- Basicamente, a função retorna

Fim do arquivo

- Como visto, `EOF ("End of file")` indica o fim de um arquivo. No entanto, podemos também utilizar a função `feof` para verificar se um arquivo chegou ao fim.

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

- Basicamente, a função retorna
  - Diferente de zero: se o arquivo chegou ao fim

Fim do arquivo

- Como visto, `EOF ("End of file")` indica o fim de um arquivo. No entanto, podemos também utilizar a função `feof` para verificar se um arquivo chegou ao fim.

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

- Basicamente, a função retorna
  - Diferente de zero: se o arquivo chegou ao fim
  - Zero: se o arquivo NÃO chegou ao fim

Escrita/Leitura de Caracteres

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
    FILE *fp;
    char c;
    fp = fopen("dados.txt", "r");
    if (fp == NULL) {
        printf("Erro ao abrir o arquivo");
        exit(1);
    }
    while(!feof(fp)){
      c = fgetc(fp);
      printf("%c",c);
   }
    fclose(fp);
    return 0;
}
```

Arquivos Pré-Definidos

Arquivos pré-definidos

- Como visto anteriormente, os ponteiros `stdin` e `stdout` podem ser utilizados para acessar os dispositivo de entrada (geralmente o teclado) e saída (geralmente o vídeo) padrão.

Arquivos pré-definidos

- Como visto anteriormente, os ponteiros `stdin` e `stdout` podem ser utilizados para acessar os dispositivo de entrada (geralmente o teclado) e saída (geralmente o vídeo) padrão.
- Na verdade, no início da execução de um programa, o sistema automaticamente abre alguns arquivos pré-definidos, entre eles `stdin` e `stdout`.

Arquivos pré-definidos

- Como visto anteriormente, os ponteiros `stdin` e `stdout` podem ser utilizados para acessar os dispositivo de entrada (geralmente o teclado) e saída (geralmente o vídeo) padrão.
- Na verdade, no início da execução de um programa, o sistema automaticamente abre alguns arquivos pré-definidos, entre eles `stdin` e `stdout`.
- **stdin**: dispositivo de entrada padrão (geralmente o teclado)

Arquivos pré-definidos

- Como visto anteriormente, os ponteiros `stdin` e `stdout` podem ser utilizados para acessar os dispositivo de entrada (geralmente o teclado) e saída (geralmente o vídeo) padrão.
- Na verdade, no início da execução de um programa, o sistema automaticamente abre alguns arquivos pré-definidos, entre eles `stdin` e `stdout`.
- **stdin**: dispositivo de entrada padrão (geralmente o teclado)
- **stdout**: dispositivo de saída padrão (geralmente o vídeo)

Arquivos pré-definidos

- Como visto anteriormente, os ponteiros `stdin` e `stdout` podem ser utilizados para acessar os dispositivo de entrada (geralmente o teclado) e saída (geralmente o vídeo) padrão.
- Na verdade, no início da execução de um programa, o sistema automaticamente abre alguns arquivos pré-definidos, entre eles `stdin` e `stdout`.
- **stdin**: dispositivo de entrada padrão (geralmente o teclado)
- **stdout**: dispositivo de saída padrão (geralmente o vídeo)
- **stderr**: dispositivo de saída de erro padrão (geralmente o vídeo)

Arquivos pré-definidos

- Como visto anteriormente, os ponteiros `stdin` e `stdout` podem ser utilizados para acessar os dispositivo de entrada (geralmente o teclado) e saída (geralmente o vídeo) padrão.
- Na verdade, no início da execução de um programa, o sistema automaticamente abre alguns arquivos pré-definidos, entre eles `stdin` e `stdout`.
- **stdin**: dispositivo de entrada padrão (geralmente o teclado)
- **stdout**: dispositivo de saída padrão (geralmente o vídeo)
- **stderr**: dispositivo de saída de erro padrão (geralmente o vídeo)
- **stdaux**: dispositivo de saída auxiliar (em muitos sistemas, associado à porta serial)

Arquivos pré-definidos

- Como visto anteriormente, os ponteiros `stdin` e `stdout` podem ser utilizados para acessar os dispositivo de entrada (geralmente o teclado) e saída (geralmente o vídeo) padrão.
- Na verdade, no início da execução de um programa, o sistema automaticamente abre alguns arquivos pré-definidos, entre eles `stdin` e `stdout`.
- **stdin**: dispositivo de entrada padrão (geralmente o teclado)
- **stdout**: dispositivo de saída padrão (geralmente o vídeo)
- **stderr**: dispositivo de saída de erro padrão (geralmente o vídeo)
- **stdaux**: dispositivo de saída auxiliar (em muitos sistemas, associado à porta serial)
- **stdprn**: dispositivo de impressão padrão (em muitos sistemas, associado à porta paralela)

Escrita/Leitura de Strings

Escrita/Leitura de Strings

- Até o momento, apenas caracteres isolados puderam ser escritos em um arquivo.

Escrita/Leitura de Strings

- Até o momento, apenas caracteres isolados puderam ser escritos em um arquivo.
- Porém, existem funções na linguagem C que permitem ler/escrever uma sequência de caracteres, isto é, uma string.

`fputs()`

`fgets()`

Escrita/Leitura de Strings

- Até o momento, apenas caracteres isolados puderam ser escritos em um arquivo.
- Porém, existem funções na linguagem C que permitem ler/escrever uma sequência de caracteres, isto é, uma string.

  `fputs()`

  `fgets()`

- Basicamente, para se escrever uma string em um arquivo usamos a função `fputs`:

```
int fputs (char *str,FILE *fp);
```

Escrita/Leitura de Strings

- Até o momento, apenas caracteres isolados puderam ser escritos em um arquivo.
- Porém, existem funções na linguagem C que permitem ler/escrever uma sequência de caracteres, isto é, uma string.

  `fputs()`

  `fgets()`

- Basicamente, para se escrever uma string em um arquivo usamos a função `fputs`:

```
int fputs (char *str,FILE *fp);
```

- Esta função recebe como parâmetro um array de caracteres (string) e um ponteiro para o arquivo no qual se quer escrever.

Escrita/Leitura de Strings

- Retorno da função

Escrita/Leitura de Strings

- Retorno da função
  - Se o texto for escrito com sucesso um valor inteiro diferente de zero é retornado.

Escrita/Leitura de Strings

- Retorno da função
  - Se o texto for escrito com sucesso um valor inteiro diferente de zero é retornado.
  - Se houver erro na escrita, o valor `EOF` é retornado.

Escrita/Leitura de Strings

- Retorno da função
  - Se o texto for escrito com sucesso um valor inteiro diferente de zero é retornado.
  - Se houver erro na escrita, o valor `EOF` é retornado.
- Como a função `fputc`, `fputs` também pode ser utilizada para escrever uma string na tela: `fputs (str,stdout);`

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
    char str[20] = "Hello World!;
    int result;
    FILE *arquivo;
    arquivo = fopen("DadosGravar.txt", "w");
    if (arquivo == NULL) {
        printf("Erro na CRIACAO do arquivo");
        exit(1);
    }
    result = fputs(str, arquivo);
    if(result == EOF){
        printf("Erro na Gravacao\n");
        exit(1);
    }
    fclose(arquivo);
    return 0;
}
```

Escrita/Leitura de Strings

- Da mesma maneira que se grava uma cadeia de caracteres no arquivo, a sua leitura também é possível.

Escrita/Leitura de Strings

- Da mesma maneira que se grava uma cadeia de caracteres no arquivo, a sua leitura também é possível.
- Para se ler uma string de um arquivo pode-se usar a função `fgets()` cujo protótipo é:

```
char *fgets (char *str, int tamanho,FILE *fp);
```

Escrita/Leitura de Strings

- Da mesma maneira que se grava uma cadeia de caracteres no arquivo, a sua leitura também é possível.
- Para se ler uma string de um arquivo pode-se usar a função `fgets()` cujo protótipo é:

```
char *fgets (char *str, int tamanho,FILE *fp);
```

- A função fgets recebe 3 parâmetros:

Escrita/Leitura de Strings

- Da mesma maneira que se grava uma cadeia de caracteres no arquivo, a sua leitura também é possível.
- Para se ler uma string de um arquivo pode-se usar a função `fgets()` cujo protótipo é:

```
char *fgets (char *str, int tamanho,FILE *fp);
```

- A função fgets recebe 3 parâmetros:
  - **str**: aonde a lida será armazenada, str;

Escrita/Leitura de Strings

- Da mesma maneira que se grava uma cadeia de caracteres no arquivo, a sua leitura também é possível.
- Para se ler uma string de um arquivo pode-se usar a função `fgets()` cujo protótipo é:

```
char *fgets (char *str, int tamanho,FILE *fp);
```

- A função fgets recebe 3 parâmetros:
  - **str**: aonde a lida será armazenada, str;
  - **tamanho**: o número máximo de caracteres a serem lidos;

Escrita/Leitura de Strings

- Da mesma maneira que se grava uma cadeia de caracteres no arquivo, a sua leitura também é possível.
- Para se ler uma string de um arquivo pode-se usar a função `fgets()` cujo protótipo é:

```
char *fgets (char *str, int tamanho,FILE *fp);
```

- A função fgets recebe 3 parâmetros:
  - **str**: aonde a lida será armazenada, str;
  - **tamanho**: o número máximo de caracteres a serem lidos;
  - **fp**: ponteiro que está associado ao arquivo de onde a string será lida.

Escrita/Leitura de Strings

- Da mesma maneira que se grava uma cadeia de caracteres no arquivo, a sua leitura também é possível.
- Para se ler uma string de um arquivo pode-se usar a função `fgets()` cujo protótipo é:

```
char *fgets (char *str, int tamanho,FILE *fp);
```

- A função fgets recebe 3 parâmetros:
  - **str**: aonde a lida será armazenada, str;
  - **tamanho**: o número máximo de caracteres a serem lidos;
  - **fp**: ponteiro que está associado ao arquivo de onde a string será lida.
- E retorna

Escrita/Leitura de Strings

- Da mesma maneira que se grava uma cadeia de caracteres no arquivo, a sua leitura também é possível.
- Para se ler uma string de um arquivo pode-se usar a função `fgets()` cujo protótipo é:

```
char *fgets (char *str, int tamanho,FILE *fp);
```

- A função fgets recebe 3 parâmetros:
  - **str**: aonde a lida será armazenada, str;
  - **tamanho**: o número máximo de caracteres a serem lidos;
  - **fp**: ponteiro que está associado ao arquivo de onde a string será lida.
- E retorna
  - `NULL` em caso de erro ou fim do arquivo;

Escrita/Leitura de Strings

- Da mesma maneira que se grava uma cadeia de caracteres no arquivo, a sua leitura também é possível.
- Para se ler uma string de um arquivo pode-se usar a função `fgets()` cujo protótipo é:

```
char *fgets (char *str, int tamanho,FILE *fp);
```

- A função fgets recebe 3 parâmetros:
  - **str**: aonde a lida será armazenada, str;
  - **tamanho**: o número máximo de caracteres a serem lidos;
  - **fp**: ponteiro que está associado ao arquivo de onde a string será lida.
- E retorna
  - `NULL` em caso de erro ou fim do arquivo;
  - O ponteiro para o primeiro caractere recuperado em str.

Escrita/Leitura de Strings

- A função lê a string até que um caractere de nova linha seja lido ou tamanho-1 caracteres tenham sido lidos.

Escrita/Leitura de Strings

- A função lê a string até que um caractere de nova linha seja lido ou tamanho-1 caracteres tenham sido lidos.
- Se o caractere de nova linha (`'\n'`) for lido, ele fará parte da string, o que não acontecia com `gets`.

Escrita/Leitura de Strings

- A função lê a string até que um caractere de nova linha seja lido ou tamanho-1 caracteres tenham sido lidos.
- Se o caractere de nova linha (`'\n'`) for lido, ele fará parte da string, o que não acontecia com `gets`.
- A string resultante sempre terminará com `'\0'` (por isto somente tamanho-1 caracteres, no máximo, serão lidos).

Escrita/Leitura de Strings

- A função lê a string até que um caractere de nova linha seja lido ou tamanho-1 caracteres tenham sido lidos.
- Se o caractere de nova linha (`'\n'`) for lido, ele fará parte da string, o que não acontecia com `gets`.
- A string resultante sempre terminará com `'\0'` (por isto somente tamanho-1 caracteres, no máximo, serão lidos).
- Se ocorrer algum erro, a função devolverá um ponteiro nulo em str.

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
    char str[20];
    int result;
    FILE *arquivo;
    arquivo = fopen("Dados.txt", "r");
    if (arquivo == NULL) {
        printf("Erro na CRIACAO do arquivo");
        exit(1);
    }
    result = fgets(str, 13, arquivo);
    if(result == NULL){
        printf("Erro na Leitura\n");
        exit(1);
    }
    else
      printf("%s",str);
    fclose(arquivo);
    return 0;
}
```

Escrita/Leitura de Strings

- A função `fgets` é semelhante à função `gets`, porém, com as seguintes vantagens:

Escrita/Leitura de Strings

- A função `fgets` é semelhante à função `gets`, porém, com as seguintes vantagens:
  - pode fazer a leitura a partir de um arquivo de dados e incluir o caractere de nova linha (`'\n'`) na string;

Escrita/Leitura de Strings

- A função `fgets` é semelhante à função `gets`, porém, com as seguintes vantagens:
  - pode fazer a leitura a partir de um arquivo de dados e incluir o caractere de nova linha (`'\n'`) na string;
  - especifica o tamanho máximo da string de entrada. Evita estouro no buffer;

Escrita/Leitura de Strings

- A função `fgets` é semelhante à função `gets`, porém, com as seguintes vantagens:
  - pode fazer a leitura a partir de um arquivo de dados e incluir o caractere de nova linha (`'\n'`) na string;
  - especifica o tamanho máximo da string de entrada. Evita estouro no buffer;
- Vale lembrar que o ponteiro `fp` pode ser substituído por `stdin`, para se fazer a leitura do teclado:

```
fgets (str, tamanho, stdin);
```

Escrita/Leitura de Strings

- Existem funções na linguagem C que permitem ler/escrever uma sequência de caracteres, isto é, uma string, no modo formatado

  `fscanf()` e `fprintf()`

Função fprintf( )

- Grava um arquivo de modo formatado

```
fprintf(FILE *fp, char *s, ... )
```

fp: é o ponteiro devolvido por `fopen`

s: string formatada

...: variáveis

```c
int main() {
    FILE * pFile;
    int idade;
    char nome [100];

    pFile = fopen ("formatado.txt","w"); // Testar arquivo

    printf ("Digite seu nome: ");
    gets (nome);
    printf ("Digite sua idade:");
    scanf("%d", &idade);

    fprintf (pFile, "\nNome: %s, \nIdade: %d\n\n",nome, idade);

    fclose(pFile);
    return 0;
}
```

Função fscanf( )

- Lê dados formatados

```
fscanf(FILE *fp, char *s, ... )
```

fp: é o ponteiro devolvido por `fopen`

s: string formatada

...: endereço das variáveis

```c
int main() {
    char str [80];
    float f;
    FILE * pFile;

    pFile = fopen ("arq.txt","w+");

    fprintf (pFile, "%f %s", 3.1416, "PI");
    rewind (pFile);

    fscanf (pFile, "%f", &f);
    fscanf (pFile, "%s", str);

    fclose (pFile);
    printf ("Leitura do arquivo: %f e %s \n",f,str);
    return 0;
}
```

Faculdade de Computação - FACOM

Bacharelado em Sistemas de Informação

Prof. Thiago Pirola Ribeiro