

LP1

Prof. Luciano Bernardes de Paula



Funções úteis

Funções `atoi()` e `atof()` (precisam da biblioteca **`stdlib.h`**)

Converte uma string em inteiro (`atoi()`) ou uma string em float (`atof()`).

Exemplos:

```
int i;  
float f;
```

```
i = atoi("5");  
f = atof("4.3");
```



Utilização de arquivos

Um arquivo pode ser utilizado para salvar dados de um programa ou para o programa receber dados.

A função `fopen()` executa duas tarefas.

Em primeiro lugar, cria e preenche uma estrutura `FILE` com as informações necessárias para o programa e para o sistema operacional, de maneira que possam se comunicar.



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

Exemplo:

```
FILE *arq;
```

```
arq = fopen("texto.txt", "rw");
```

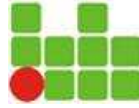
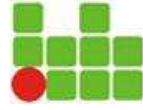
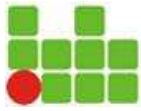


Tabela com os possíveis modos de abertura de arquivos:

"r"	Abre um arquivo para leitura em modo texto. Se o arquivo não existir, a operação irá falhar e fopen() retornará NULL.
"w"	Cria um arquivo em modo texto para gravação. Se o arquivo já existir, elimina seu conteúdo e recomeça a gravação a partir de seu início.
"a"	Abre um arquivo em modo texto para gravação, a partir de seu final. Se o arquivo não existir, ele será criado.
"r+"	Abre um arquivo em modo texto para atualização, ou seja, tanto para leitura como para gravação. Se o arquivo não existir, a operação irá falhar e fopen() retornará NULL.
"w+"	Cria um arquivo em modo texto para atualização, ou seja, tanto para leitura como para gravação. Se o arquivo já existir, seu conteúdo será destruído.
"a+"	Abre um arquivo em modo texto para atualização, gravando novos dados a partir do final do arquivo. Se o arquivo não existir, ele será criado.
"rb"	Abre um arquivo para leitura em modo binário. Se o arquivo não existir, a operação irá falhar e fopen() retornará NULL.
"wb"	Cria um arquivo em modo binário para gravação. Se o arquivo já existir, elimina seu conteúdo e recomeça a gravação a partir de seu início.
"ab"	Abre um arquivo em modo binário para gravação, a partir de seu final. Se o arquivo não existir, será criado.
"rb+"	Abre um arquivo em modo binário para atualização, ou seja, tanto para leitura como para gravação. Se o arquivo não existir, a operação irá falhar e fopen() retornará NULL.



"wb+" data-bbox="441 800 533 903">	<p>Cria um arquivo em modo binário para atualização, ou seja, tanto para leitura como para gravação. Se o arquivo já existir, seu conteúdo será destruído.</p>
"ab+" data-bbox="533 800 620 903">	<p>Abre um arquivo em modo binário para atualização, gravando novos dados a partir do final do arquivo. Se o arquivo não existir, ele será criado.</p>



Cuidados ao abrir um arquivo

Sempre que um arquivo é aberto, é necessário checar se houve sucesso na ação.

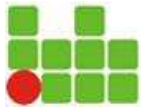
Algumas razões para falha na abertura de um arquivo:

- Abertura para escrita pode falhar se não houver mais espaço em disco;
- Abertura para leitura de um arquivo que não existe;
- O usuário não possui permissão para ler/escrever no arquivo;
- Etc



Se houve falha na abertura de um arquivo, a função fopen retorna um ponteiro com valor NULL.

Teste sempre antes de usar o arquivo pela primeira vez.



```
FILE *arq;
```

```
if((arq = fopen("arqteste.txt", "w")) == NULL){  
    printf("Falha ao abrir o arquivo...\n");  
    return 1;  
}
```

```
...
```

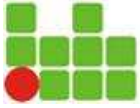


A função `fclose()` fecha o arquivo.

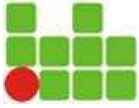
Essa função recebe como argumento o ponteiro para o arquivo aberto.

```
fclose(arq);
```

Fechar um arquivo significa, em primeiro lugar, esvaziar o seu buffer, gravando nele os caracteres remanescentes.



Conforme o arquivo vai sendo escrito ou lido, o programa sabe a posição atual dentro do arquivo.



A função **feof()** retorna verdadeiro (diferente de 0) se o final do arquivo tiver sido atingido; caso contrário retorna falso (0).

EOF e o final de um arquivo

É importante entender que o EOF não é um caracter pertencente ao arquivo e sim uma constante do tipo **short int** enviada pelo SO para o programa quando o final de um arquivo é atingido.

Essa constante é definida na stdio.h com valor 0xFFFF ou -1.

Se utilizarmos uma variável do tipo char para receber os bytes lidos do arquivo, o caractere de código ASCII 255 decimal (0xFF), se existir dentro do arquivo, será interpretado como **EOF**, terminando a leitura precipitadamente.



Por causa disso, é recomendado utilizar um `short int` para ler caracteres de arquivos.

Dessa forma não haverá confusão entre `0xFF` e `EOF`.

O primeiro será armazenado em **ch** como `0x00FF`, e o segundo, como `0xFFFF`.



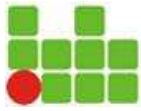
Há quatro grupos de funções para manipulação de arquivos

Grupo 1: gravar e ler um caracter por vez – funções `fputc()`, `fgetc()`;

Grupo 2: ler e gravar uma linha – funções `fputs()` e `fgets()`;

Grupo 3: ler e gravar dados formatados – funções `fprintf()` e `fscanf()`;

Grupo 4: ler e gravar blocos de bytes – funções `fwrite()` e `fread()`.



Grupo 1: gravar e ler um caracter por vez – fputc e fgetc

As funções `fputc()` e `fgetc()` gravam e lêem um caracter por vez.



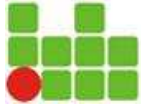
```
int main(){

    FILE *fptr;
    short int ch;

    fptr = fopen("arqteste.txt", "w");

    while((ch=getche())) != '\r') fputc(ch, fptr);

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

A função `fputc()` recebe dois argumentos: o caractere a ser gravado e o ponteiro para a estrutura `FILE` do arquivo a ser utilizado.

```
fputc(ch, fptr);
```



A função `fclose()` fecha o arquivo.

Essa função recebe como argumento o ponteiro para o arquivo aberto.

Fechar um arquivo significa, em primeiro lugar, esvaziar o seu buffer, gravando nele os caracteres remanescentes.



Uma das razões para fechar um arquivo é escrever tudo que tenha que ser escrito a partir do buffer no arquivo.

Outra razão é liberar as áreas de comunicação utilizadas.



A função **fgetc()** recebe um ponteiro para o arquivo aberto e lê um caracter.

```
ch = fgetc(ptr);
```

É preciso ter aberto o arquivo para leitura:

```
ptr = fopen("nomeArq", "r");
```

fgetc() retorna um caracter ou um **EOF** (*End Of File*) se o final do arquivo tiver sido atingido.



O valor do EOF pode mudar de SO para SO.

O ideal é testar com o nome da constante:

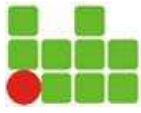
```
while(c != EOF){  
    ...  
}
```



Grupo 2: ler e gravar linha a linha – `fputs()` e `fgets()`

`fputs()` recebe dois argumentos: um ponteiro pra char, sendo a cadeia de caracteres a ser escrita e um ponteiro para o arquivo.

`fputs()` não coloca um `'\n'` no final da linha, sendo preciso colocar explicitamente.



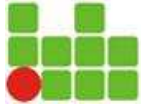
Exemplos de uso

```
fputs("bla bla bla\n", ptr);
```

Ou

```
char abc[10] = "ABCD";
```

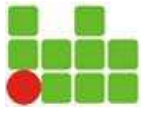
```
fputs(abc, ptr);
```



Lendo uma linha de um arquivo: A função **fgets()** recebe três argumentos:

- o ponteiro **char** para uma cadeia de caracteres em que os dados lidos do arquivo serão colocados;
- o número máximo de caracteres a serem lidos e
- o ponteiro para o arquivo.

Retorna um ponteiro para a cadeia de caracteres que foi lida ou NULL se chegou ao final do arquivo ou houve algum erro.

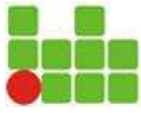


```
char str[81];
```

```
fgets(str, sizeof(str), ptr);
```

```
...
```

```
printf("%s", str);
```



Grupo 3: ler e gravar dados formatados – fprintf() e fscanf();

```
fprintf(ptr, “%i %c”, i, c);
```

fprintf() é igual ao printf, só que recebe um ponteiro para um arquivo como primeiro argumento.



`fscanf()` lê dados de um arquivo de acordo com o formato informado.

É similar a `scanf()`, só que recebe também um ponteiro para um arquivo.

```
fscanf(ptr, "%d", &a);
```



Grupo 4: ler e gravar blocos de bytes – fwrite() e fread()

Às vezes é preciso escrever ou ler um certo número de caracteres (mais ou menos de uma linha).

As funções para isso são a fwrite() e a fread().



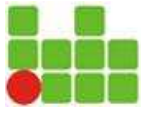
`fread(destino, tamanho da unidade, tamanho total
a ser lido, arquivo);`

Por exemplo, se queremos ler 1.024 bytes.

```
unsigned char buffer[1024];
```

...

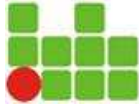
```
BytesLidos = fread(buffer, sizeof(char),  
sizeof(buffer), forigem);
```



E para escrever em um arquivo:

```
fwrite(buffer, sizeof(char), quantidade,  
        arquivo de destino);
```

`fwrite` ou `fread` só trabalham corretamente com arquivos em modo binário.



Modo texto e modo binário

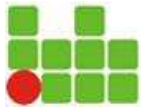
São três as diferenças entre trabalhar com um arquivo em modo texto e em modo binário:

- O caractere `\n` ou `\r`: Em uma operação de gravação em arquivos abertos em modo texto (padrão), o caractere `\n` ou `\r` é expandido em dois bytes, *carriage-return* e *linefeed* (`\r` e `\n`), antes de ser gravado. Em operações de leitura, o par de bytes `CR/LF` é convertido para um único byte `\n`. Se o arquivo é aberto em modo binário, não há essa conversão.
- O caractere `\x1A` (Ctrl-Z): Em modo texto, é interpretado como fim de arquivo. Em modo binário, é mais um caractere pertencente ao arquivo.
- Números: Em modo binário podem ser gravados ou lidos exatamente como se apresentam na memória. Por exemplo, o número 25.678 é armazenado, na memória do computador, ocupando dois bytes (**short int**). Em modo texto, ocuparia 5 bytes e seria armazenado na forma ASCII.



Ponteiros para os “arquivos” stdin, stdout e stderr.

Ponteiro	Stream
stdin	Entrada padrão (teclado)
stdout	Saída padrão (vídeo)
stderr	Saída de erro (vídeo)



A instrução

```
fputs(string, stdout);
```

Imprime a string na tela.



Gravando estruturas em um arquivo

É possível gravar uma estrutura em um disco com a função `fwrite()` e ler estruturas com a função `fread()`;

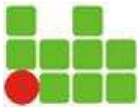
Sendo **var** uma estrutura:

```
fwrite(&var, sizeof(var), 1, fptr); // o valor 1 indica quantas  
//vezes será escrito
```

```
fread(&var, sizeof(var), 1, fptr);
```



Pode ser gravado cada membro da estrutura separadamente, como se fossem várias variáveis.



A função **rewind()** reposiciona o ponteiro de posição do arquivo para o começo deste.

Se for usado com o **stdin**, ele limpa qualquer caracter que houver no buffer de entrada.

```
rewind(ptr);
```



Gravando e lendo de um mesmo arquivo

É possível gravar e ler de um mesmo arquivo.

É preciso abri-lo para leitura e escrita, por exemplo com “ab+” .



Algumas funções úteis:

`fflush()` grava o conteúdo do buffer para o arquivo associado ao ponteiro que recebe como argumento.

Se não é passado parâmetro, `stdin` é usado como padrão.

Se for passado, vincula ao arquivo apontado.

```
fflush(ptr);
```