



INF1037 - Programação em C

Files – Read & Write

Dept. de Informática, PUC-Rio



Funções de entrada e saída em arquivos

- **Motivação:** quando um programa precisa processar um volume de dados muito grande
 - Imagine ter que digitar n dados novamente por causa de um simples erro de digitação?
- A estratégia adequada para se trabalhar com grande volume de dados é fazer uso de arquivos de dados.
- Para que programas possam ler e salvar dados em um arquivo, é necessário ter acesso a um conjunto de funções que permitam realizar essas operações:
 - No nosso curso, usaremos um conjunto de funções presentes na biblioteca *stdio.h*.



Funções de entrada e saída em arquivos

- Para que possamos ler ou escrever em um arquivo é necessário que antes o **arquivo esteja aberto**.
- Cada arquivo é identificado pelo seu nome, que normalmente é o nome do arquivo mais a **extensão**: exemplo.**txt**, cap8.**ppt**, saida.**txt**
- Para abrir um arquivo é necessário definir se o mesmo será utilizado para leitura (*read*) ou escrita (*write*):
- Quando abrimos um arquivo para escrita criamos um novo arquivo onde dados de saída serão escritos, ou se o arquivo já existe, então estaremos sobrescrevendo.

Funções de entrada e saída em arquivos

- Para abrir um arquivo, a biblioteca padrão oferece a função **fopen**.
 - Esta função serve tanto para abrir um arquivo para leitura como para escrita.
 - A função recebe dois parâmetros, o nome do arquivo que se deseja abrir e o modo de abertura:
 - Leitura: "r" (*read*)
 - Escrita: "w" (*write*).
 - Esta função retorna um ponteiro para o tipo **FILE** definido na biblioteca. Você passa a usar este ponteiro para tudo que quiser fazer com o arquivo. E.g:

```
FILE * f;  
f = fopen(..., ...);
```
 - Exemplos no próximo slide

Declaração e Inicialização

- Declaração de uma variável do tipo ponteiro para FILE (no exemplo, o nome dado à variável é *fp*):
 - `FILE * fp;`
- Inicialização:
 - Abrindo o arquivo para leitura de dados:
 - `fp = fopen("entrada.txt", "r");`
 - Abrindo o arquivo para escrita de dados:
 - `fp = fopen("saida.txt", "w");`
 - “**rb**” e “**wb**” são para ler e escrever arquivos binários. Em Linux, Unix e MacOS, “r” serve tanto para texto como para binário. Mas em Windows isto é perigoso. Por isto, em programas “portáteis”, você deve sempre incluir o caractere ‘**b**’ na “*string* de modo” para abrir arquivos binários: `fp = fopen("image01.gif", "rb");`
 - OBS: **b** ignora caracteres de formatação (como o \n).

O valor retornado pela função `fopen` deve ser passado como parâmetro para as demais funções para identificar em qual arquivo se deseja fazer a operação de entrada ou saída.

Declaração e Inicialização

- Se a abertura do arquivo não for bem sucedida, a função retorna o valor definido pela constante simbólica **NULL** (o ponteiro nulo):

NULL está definido em vários *header files* padrão, tais como *stdio.h*, *stdlib.h*,

```
fp = fopen("entrada.txt", "r");  
if (fp == NULL)  
{  
    printf("Erro na abertura do arquivo.\n");  
    exit(1);  
}
```

Insucesso na abertura pode ser por inúmeras razões: arquivo não existe, modo de abertura não permite outros acessos, arquivo existe mas você não tem permissão,

exit(status) causa o término normal do programa e o status é retornado para o ambiente que chamou o programa (zero significa sucesso). Arquivos abertos são descarregados (*flushed*), *streams* são fechados, funções podem ser chamadas, e o controle retorna para o ambiente.

- Após realizar as operações de entrada e saída no arquivo, este deve ser fechado com o uso da função **fclose**.

```
...  
fclose(fp);  
...
```

Modos de Abertura

Modo	Finalidade	OBS
"r"	Abrir para leitura	
"rb"	Abrir para leitura no modo binário	
"w"	Abrir para escrita	Se arquivo existe: conteúdos são sobrescritos Se o arquivo não existe, ele será criado
"wb"	Abrir para escrita no modo binário	Mesma observação feita para "w"
"a"	Abrir para <i>append</i> (juntar/anexar)	Dados são adicionados no fim do arquivo. Se arquivo não existe, ele será criado
"ab"	Abrir para <i>append</i> no modo binário	Mesma observação feita para "a"
"r+"	Abrir para leitura e escrita	
"rb+"	Abrir para leitura e escrita modo binário	
"w+"	Abrir para leitura e escrita	Mesma observação feita para "w"
"wb+"	Abrir para leitura e escrita modo binário	Mesma observação feita para "w"
"a+"	Abrir para leitura e <i>append</i>	Mesma observação feita para "a"
"ab+"	Abrir para leitura e <i>append</i> modo binário	Mesma observação feita para "a"

Leitura arquivos texto: fscanf

- A principal função da biblioteca padrão para leitura de arquivos texto chama-se *fscanf*, e é análoga à função *scanf* para captura de dados via teclado.

```
int a;  
float b;  
FILE * fp = fopen("entrada.txt", "r");  
.  
.  
.  
fscanf(fp, "%d %f", &a, &b);  
.  
.  
.  
fclose(fp);
```

- A função *fscanf* tem como valor de retorno o número de parâmetros que foram lidos com sucesso (isto também vale para a função *scanf*).



Escrita de arquivos texto: fprintf

- A principal função de saída em arquivo texto da biblioteca padrão chama-se fprintf, sendo análoga à função printf para saída na tela.
 - A diferença é que a função fprintf espera um primeiro parâmetro adicional que identifica o arquivo.

```
int a ;  
float b ;  
FILE * fp = fopen("saida.txt", "w");  
.  
.  
.  
fprintf(fp, "Valores : %d %f \n", a, b);  
.  
.  
.  
fclose(fp);
```

- Obviamente os valores de a e b precisam ser definidos antes da chamada à função fprintf.

Exemplo

```
int main(void)
{
    int np, nalunos;
    float p1, p2, media;
    FILE * fin = fopen("c:\\Temp\\notas.txt", "r");
    FILE * fout = fopen("c:\\Temp\\medias.txt", "w");
    nalunos = 0;
    while (!feof(fin))
    {
        np = fscanf(fin, "%f %f", &p1, &p2); // np= 0,1,2 ou EOF
        if (np >= 0 && np < 2)
        {
            printf("Erro lendo %d valores.", np);
            exit(2);
        }
    }
}
```

stdlib.h é o header da “*standard library*”, que contém funções de alocação de memória, controle de processo (e.g. **exit()**, **system()**), utilidades (e.g. **rand()**, **abs()**) e conversões (e.g. **atoi()**).

Somente quando uma operação de leitura encontra um “*end of file*” é que o **feof()** retorna verdade (i.e. retorna $\neq 0$). Então, o **while** ainda entra uma última vez para que **fscanf** encontre “*end of file*” e retorne **EOF** (usualmente -1). Esta não é a única maneira de ler o arquivo, mas é a que mais permite criticar os dados lidos. Veja outras formas no slide a seguir.

fscanf retorna o número de itens convertidos e atribuídos corretamente. Se ela ler 10.0 e A+, ao invés de 10.0 e 8.0, então ela lê corretamente apenas o 10.0. Outra característica é que o **fscanf** tentará buscar o segundo item nas outras linhas. Por exemplo,

linha 1: 10.0
linha 2: 2.0 3.0
linha 3: EOF

Na primeira leitura, p1 será 10.0 e p2 será 2.0 (np = 2). Na próxima leitura, **fscanf** lerá apenas o 3.0 (np = 1).

Continua no próximo slide

Exemplo

Ao invés de usar `while (!feof(fin))`, haveria duas outras possibilidades mais diretas:

(1) `while (fscanf(...) == 2)`

(2) `while (fscanf(...) != EOF)`

Mas isto dificultaria lidar com todos os casos de erro (veja, no exemplo, o uso que fazemos da variável `np`).

```
if (np == 2)
{
    nalunos++;
    media = (p1 + p2) / 2;
    fprintf(fout, "%.1f ", media);
    if (media >= 5.0)
        fprintf(fout, "Aprovado\n");
    else
        fprintf(fout, "Reprovado\n");
}
} // end of while
printf("%d alunos lidos\n", nalunos);
fclose(fin);
fclose(fout);
return 0;
}
```

Cuidado quando chamar `fscanf` várias vezes, porque o *stream* pode estar em um caractere não desejado porém ainda não lido. E.g., se o arquivo for "valor= 5", o seguinte código não lê `x` como 5, porque o stream está no '='. Use um `getc(fin)` para se livrar do '='.

```
char s[81];
int x;
fscanf(fin, "%[^=]", s);
fscanf(fin, "%d", &x);
```

`getc(fin);`

notas.txt - Notepad

File Edit Format View Help

```
7.0 5.0
10.0 8
2.0 3.0
5.0 5.0
|
```

medias.txt - Note

File Edit Format View Help

```
6.0 Aprovado
9.0 Aprovado
2.5 Reprovado
5.0 Aprovado
```

Outros Exemplos

- Ler caracter a caracter de um arquivo e imprimi-los:

```
int main(void)
{
    int c;
    FILE * fin = fopen("texto.txt", "r");
    while (1) // loop infinito!!!
    {
        c = getc(fin); // ou: c = fgetc(fin);
        if (feof(fin)) break;
        printf("%c", c);
    }
    fclose(fin);
}
```

getc() ou *fgetc()* são melhores do que usar *fscanf(fin, "%c", &c)*; porque *fscanf* e *scanf* não são bem comportadas para lidar com caracteres.

A diferença entre *getc* e *fgetc* é que *getc* pode ser implementada como macro enquanto que *fgetc* é sempre implementada como sendo uma função. Chamadas a macros são bem mais rápidas, mas macros perdem as potencialidades de uma função (e.g. *side effects* e ponteiro para função¹). Portanto, recomendamos o uso de *fgetc*. O mesmo vale para *putc* e *fputc*.

```
int main(void)
{
    int c;
    FILE * fin = fopen("texto.txt", "r");
    do
    {
        c = getc(fin); // ou: c = fgetc(fin);
        if (!feof(fin))
            printf("%c", c);
    } while (c != EOF);
    fclose(fin);
}
```

Uma macro é um fragmento de código ao qual se dá um nome. E toda vez que este nome é usado, o nome é substituído pelo conteúdo da macro. A mais simples é a que define um objeto, e.g. `#define PI 3.1416`

1. Pontoeiro para função não será visto no presente curso



ARQUIVOS BINÁRIOS

Escrita de arquivos *binary*: fwrite

- A sintaxe de fwrite é:

```
size_t fwrite(const void * ptr, size_t size, size_t n, FILE * fp);
```

`ptr` aponta para o bloco de memória que contém os itens a serem escritos

`size` especifica o número de bytes de cada item a ser escrito

`n` é o número de itens a serem escritos

`fp` é o ponteiro para o arquivo

- Supondo aberto um arquivo: `fout = fopen("xx.dat", "wb");`

- Escrever uma variável

```
int x = 5;
fwrite(&x, sizeof(int), 1, fout);
```

- Escrever um vetor

```
int a[3] = {10, 20, 30};
fwrite(a, sizeof(int), 3, fout);
```

Dependendo da implementação, `size_t` é *unsigned int* ou *unsigned long*. Para uma programação absolutamente segura, ou você faz um *cast* para *unsigned long* ou usa o tipo `size_t` provido pelo `<stddef.h>`. Para o nível deste curso, considere como `int`.

Leitura de arquivos *binary*: fread

- A sintaxe de fread é similar ao fwrite:

```
size_t fread(const void * ptr, size_t size, size_t n, FILE * fp);
```

- Supondo aberto um arquivo: fin = fopen(..., "rb");

- Ler uma variável (escrita no slide anterior)

```
int y;  
fread(&y, sizeof(int), 1, fin);  
printf("%d\n", y); // imprime 5
```

- Ler um vetor (escrito no slide anterior)

```
int b[3];  
fread(b, sizeof(int), 3, fin);  
printf("%d %d %d\n", b[0], b[1], b[2]); // imprime 10 20 30
```

- Invente exercícios que escreve e lê arquivos binários. Se fizer escrita e leitura no mesmo programa, não se esqueça de fazer *fclose()* e de abrir novamente o arquivo.

A função retorna o número total de itens lidos com sucesso. Se esse número difere do parâmetro *n*, então ou ocorreu um erro ou o EOF foi alcançado. Se *size* é 0, *fread* retorna 0.