

Manipulação de Arquivos e Strings

Aula 19

Às vezes é preciso se lembrar...



RAM vs Disco

- Quando rodamos um programa, ele fica armazenado na memória RAM
 - Dados existem apenas enquanto rodamos o programa
- E se quisermos guardar os dados para usar depois?
 - Persistência em Disco Rígido!
 - **Arquivos**, Banco de Dados...

X-Files

- Arquivos são sequências binárias de informações armazenadas no computador
 - Tratados pelo SO (Sistema Operacional, não Soulmate)
 - Podem codificar vários tipos de informações
 - Texto
 - Imagens
 - Sons
 - Vídeos
 - Executáveis



Entrada e Saída

- Podemos usar os arquivos:
 - Para **escrever informações** que nosso programa gera
 - Para **carregar informações** que nosso programa precisa
- Como usamos um arquivo no programa?

The Streams are flowing!

- Usamos **canais de comunicação** (streams) para manipular os arquivos



Fluxo de utilização do arquivo

- O fluxo natural de uso de um arquivo é:
 - **Abertura** do canal de comunicação, indicando o modo de operação dele
 - **Leitura e/ou escrita** de dados no arquivo, através do canal
 - **Fechamento** do canal de comunicação
- É muito importante que ocorra a abertura e fechamento adequado do stream!

FILE

- Em C, temos um registro padrão para manusear os arquivos: **FILE**
- Definido na biblioteca de IO (stdio), representa um arquivo

```
#include <stdio.h>
```

```
FILE *canal1;
```

```
FILE *canal2;
```


Abrir o arquivo

- Função: fopen

`fopen(char* caminho_do_arquivo, char* modo_abertura)`

```
#include <stdio.h>
```

```
int main() {
```

```
    FILE *arquivo;
```

```
    arquivo = fopen( "teste.txt", "r" );
```

```
    return 0;
```

```
}
```

Modos de abertura

Modo	Operação
r (rb para binários)	Leitura de arquivo
w (wb)	Escrita de arquivo. Cria se não existir, escreve a partir do início do arquivo.
a (ab)	Escrita de arquivo. Cria se não existir, escreve a partir do final do arquivo.
r+ (rb+ ou r+b)	Leitura e escrita. Escreve do começo, mas não trunca. Só abre se existir.
w+ (wb+ ou w+b)	Leitura e escrita. Escreve do começo, trunca o arquivo. (Igual a w)
a+ (ab+ ou a+b)	Leitura e escrita. Escreve do final do arquivo;

E se der pau?

- É importante **testar** se a abertura de stream deu certo.
- fopen retorna
 - Ponteiro pro arquivo em sucesso
 - Nulo caso contrário

```
FILE *arquivo = fopen( "teste.txt", "r" );  
if( arquivo == NULL ) {  
    printf( "Erro na abertura do arquivo!\n" );  
}
```



Fechando o arquivo

- É muito importante fechar o arquivo quando terminar de usá-lo, para evitar erros na gravação dos dados

```
fclose(FILE* arquivo_usado)
```

```
fclose(arquivo);
```

Escrita do arquivo

- Funções análogas às existentes para entrada padrão
 - **fprintf(arquivo, string_formatada, variáveis)**
 - **fputs (string, arquivo)**
 - **fputc(caractere (como inteiro), arquivo)**

Exemplo escrita

```
#include <stdio.h>
int main(){
    int i, alternativa;
    char letra;
    FILE *arquivo = fopen( "respostas.txt", "w" );
    if( arquivo == NULL ){
        printf( "Erro na abertura do arquivo" );
    }
```

Exemplo escrita

```
else{
    printf("Digite seu nome: ");
    scanf("%c", &letra);
    fprintf(arquivo, "Aluno: %c\n", letra);
    for(i=0; i<10; i++){
        printf("Questão %d: ", i+1);
        scanf("%d", &alternativa);
        fprintf(arquivo, "Resposta %d: %d\n", i+1, alternativa);
    }
    fclose( arquivo );
}
return 0;
}
```

Leitura do arquivo

- Funções análogas às existentes para entrada padrão
 - **fscanf(arquivo, string_formatação, variáveis)**
 - **fgets(string, tamanho, arquivo)**
 - **fgetc(arquivo)**

Exemplo de leitura

```
#include <stdio.h>
```

```
int main(){  
    int questao, resposta, acertos, i;  
    char letra;  
    int gabarito[10] = {3, 4, 2, 1, 3, 1, 1, 2, 4, 4};  
    FILE *arquivo = fopen( "respostas.txt", "r" );  
  
    if( arquivo == NULL ){  
        printf( "Erro na abertura do arquivo" );  
    }  
}
```

Exemplo Leitura

```
else{
    acertos = 0;
    fscanf(arquivo, "Aluno: %c\n", &letra);

    for(i=0; i<10; i++){
        fscanf(arquivo, "Resposta %d: %d\n", &questao, &resposta);
        if(gabarito[questao-1] == resposta)
            acertos++;
    }
    printf("Nota de %c\n", letra);
    printf("%.2f\n", (float)acertos/10.0);
    fclose( arquivo );
} return 0; }
```

Retorno das funções

- Podemos verificar o retorno das funções para saber se ocorreu erro no manuseio dos arquivos
 - **EOF (End of File)** é o retorno padrão para erros nas operações
 - Exceção: fprintf retorna um número negativo quando ocorre erro na escrita

Buffering Stats

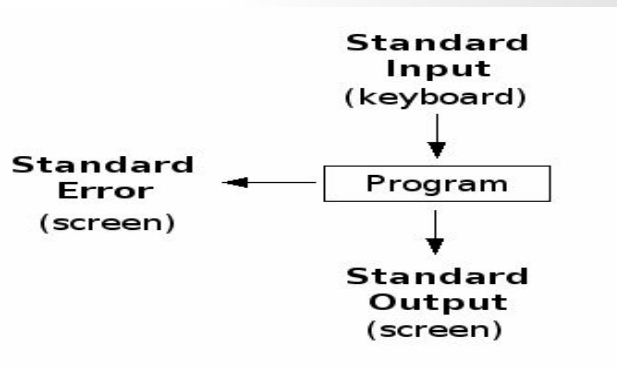


- A escrita não é imediata!
- A escrita ocorre quando
 - Nova linha
 - Canal é fechado
 - Buffer cheio

Canais padrão

- As entradas e saídas padrão são também interpretadas como canais de comunicação

- **stdin**
- **stdout**
- **stderr**



- Ou seja

- `printf("hadouken!") == fprintf(stdout, "hadouken!")`

Rasgando o arquivo

- A função `fseek` permite que você mova o ponteiro de leitura do arquivo de acordo com os seguintes parâmetros:

`fseek(arquivo, offset, partida)`

`arquivo` = ponteiro `FILE`

`offset` é diferença em bytes a partir do ponto `partida`

Algumas constantes

- Existem algumas constantes para facilitar o posicionamento do ponteiro
 - SEEK_SET: começo do arquivo
 - SEEK_CUR: posição atual de leitura
 - SEEK_END: final do arquivo

Exemplitcho

```
#include <stdio.h>

int main ()
{
    FILE *fp;

    fp = fopen("file.txt","w+");
    fputs("This is tutorialspoint.com", fp);

    fseek( fp, 7, SEEK_SET );
    fputs(" C Programming Language", fp);
    fclose(fp);

    return(0);
}
```


Not the Hard Way

- Muito útil para criar arquivos de teste, por exemplo
- `./executavel < entrada.txt`
- `./executavel > saida.txt`
- `./executavel 2> erros.txt`

O que é uma String?

- Representa um texto (vários caracteres).
- Logo, como são vários caracteres, usamos um **vetor de char**.
- Terminado com um caractere especial: **'\0'**.

Exemplo

```
char palavra[50];
```

'S'	'T'	'R'	'I'	'N'	'G'	'\0'
-----	-----	-----	-----	-----	-----	------

'S'	'T'	'R'	'\0'	'I'	'N'	'G'
-----	-----	-----	------	-----	-----	-----

Tratamento Especial!

- Em outras linguagens, Strings são tipos primitivos
- Em C não é, mas é tratada de forma diferenciada em algumas funções.



Leitura e escrita

```
printf("%s\n", string);
```

```
scanf("%s", string);
```

- As funções de leitura e escrita tratam a string em função do seu '\0'.

Outras formas de ler

Quando usamos `scanf("%s")`, a leitura será feita até **um espaço em branco!**

Ex: “Duas palavras”

```
scanf("%s", texto);  
printf("%s", texto);
```

“Duas”

Função gets

- A função gets permite que você leia uma linha inteira, terminando a leitura apenas com um \n

Ex: “Duas palavras”

```
    gets(texto);  
    printf(“%s”, texto);
```

“Duas palavras”

Outras funções - leitura

- **sscanf** - realiza um scanf em uma string, em vez de na entrada padrão
- **fscanf** - lê de um arquivo
- **getc** - lê um character do stream indicado
- **getchar** - lê um character da entrada padrão
- **ungetc** - “devolve” um character para o stream de leitura

Outras funções - escrita

- **puts** - imprime uma string, com \n no final.
- **sprintf** - realiza a impressão em uma string, em vez de na entrada padrão
- **fprintf** - realiza a impressão em um arquivo
- **putc** - imprime um caractere em um stream
- **putchar** - imprime um caractere na saída padrão

Definindo tamanhos



- Como todo vetor, é necessário que se aloque o tamanho antes de usar.
- Lembre-se de contar +1 para o caractere '\0'!

Definindo tamanho

- Pode-se colocar o tamanho direto na definição

```
char texto[100]
```

- Ou definir uma constante que guarda o valor

```
#define N 100  
char texto[N];
```

Está tudo pronto...

- Biblioteca string.h

- strlen
- strcpy
- strcmp
- strcat
- strstr

- E outras funções!



Dúvidas

