Arquivos

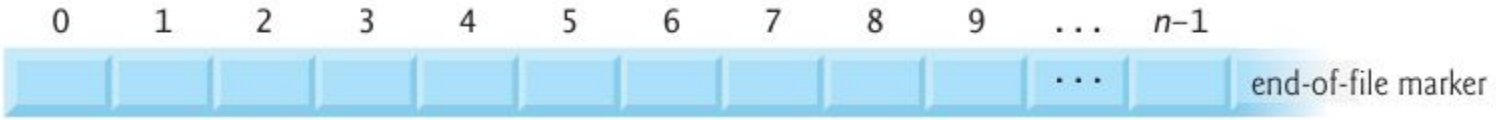
Linguagem de programação

Introdução

- O armazenamento de dados em variáveis e vetores é *temporário*
  - os dados são perdidos quando o programa é finalizado
- ***Arquivos*** são utilizados para armazenamento *permanente* de dados
- Nessa aula aprenderemos como arquivos de dados são criados, atualizados e processados por programas em C

Arquivos e *Streams* (fluxos)

- O C “enxerga” cada arquivo como uma stream (fluxo) de bytes
- Cada arquivo termina ou com um **marcador end-of-file (eof)** ou um número específico de byte armazenado em uma estrutura do SO.



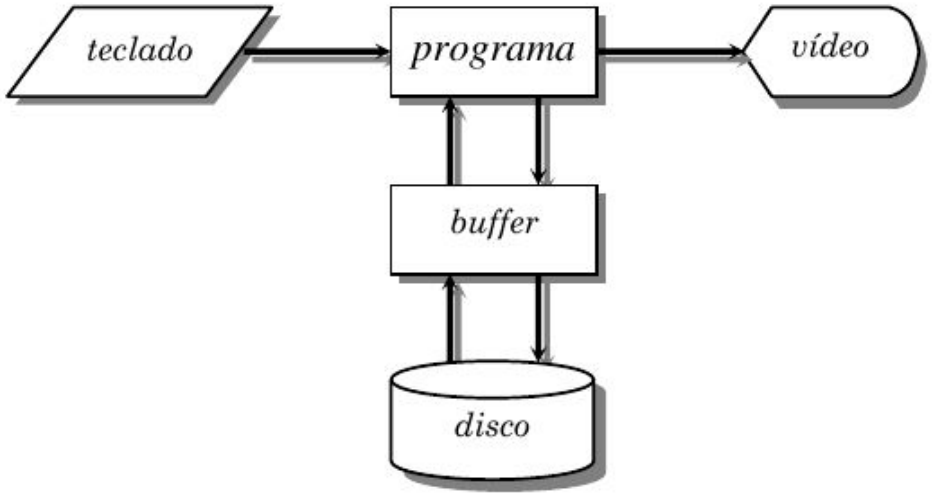
- Quando um arquivo é *aberto*, uma stream é associado a ele; e
- mais 2 arquivos e suas respectivas streams também são abertos: **standard input** e **standard output**

Arquivos e *Streams*

- As streams funcionam como canais de comunicação entre arquivos e programas
  - standard input: permite um programa ler dados do teclado
  - standard output: permite um programa mostrar dados na tela
- Streams melhoram a eficiência do uso de arquivos, no qual **stdin** e **stdout** funcionam como *buffers*
  - reduzindo o nº de acessos ao disco (HD), aumentando a velocidade de execução do programa

Arquivos e *Streams*

- **STDOUT**:
  - dados gravados pelo programa são temporariamente armazenados no *buffer* (**stdout**)
  - quando o buffer está cheio, o SO descarrega de uma vez seu conteúdo no disco
- De maneira análoga, durante a leitura, é utilizado o *buffer* **stdin**



Arquivos e *Streams*

- A abertura de um arquivo retorna um ponteiro para uma struct **`FILE`**
  - Essa struct é definida em **`<stdio.h>`** e contém informações para processar o arquivo, como:
    - I/O mode: indica se o arquivo será de leitura ou escrita
    - indicador end-of-file: indica se a leitura chegou ao final do arquivo
    - indicador de posição: indica a posição atual de (leitura/escrita) no arquivo
    - uma “id”, que é armazenada pelo SO em um vetor de **arquivos abertos**
- A abertura de um arquivo é realizada pela seguinte função:
  - `FILE *fopen( const char *filename, const char *mode );`
    - **`const:`** modificador indicando que o valor da variável não pode ser alterado
    - **`filename:`** string contendo o nome do arquivo a ser aberto
    - **`mode:`** string contendo o modo de abertura do arquivo: para leitura, escrita, etc.

Modos de abertura

- Na abertura de arquivos há vários tipos de **`mode`**, alguns são:
  - **“r”** (leitura): abre um arquivo para leitura. O arquivo deve existir.
    - **“r+”** (leitura/update): abre um arquivo para update (leitura ou escrita)
  - **“w”** (escrita): Cria um arquivo para escrita. Se um arquivo com mesmo nome existir, seu conteúdo é descartado, e um novo arquivo em branco é criado.
    - **“w+”** (escrita/update): cria um arquivo para update
  - **“a”** (append): abre um arquivo para escrita no final. O arquivo é criado caso não exista.
    - **“a+”** (append/update): cria um arquivo para update. Escrita realizada no final

Exemplo com **`fopen`**

```
int main (){
 FILE * arq;
 arq = fopen ("meuarquivo.txt","w");
 if (arq!=NULL){
   fputs ("exemplo fopen",arq);
   fclose (arq);
 }
 return 0;
}
```

- se **`arq==NULL`** então a abertura de arquivo falhou
- **`fputs:`** escreve uma string no arquivo. Outras funções para escrita:
  - `fprintf (arq, "%s", exemplo);`
  - `fputc ('a', arq)`
- **`fclose`**: fecha o arquivo e o desassocia à *stream*.
  - todo conteúdo no *buffer* **stdin** é descartado
  - todo conteúdo no *buffer* **stdout** é escrito

Funções de leitura

- `int fgetc( FILE *stream );`
  - retorno: o caracter lido ou `EOF`
- `char* fgets( char* str, int num, FILE * stream );`
  - retorno: `str` ou `NULL` se o final do arquivo foi atingido sem nenhum caractere ter sido lido
- `int fscanf(FILE *stream, const char *format, ... );`
  - Assim como o scanf() lê uma entrada formatada, mas do arquivo.
  - retorno: nº de argumentos lidos corretamente ou `EOF`

Exemplo com `fgets`

- Nesse exemplo, o comando **`while`** é utilizado para “iterar” no arquivo;
- Começando do início do arquivo, a cada chamada do **`fgets`**, o ponteiro **`arq`** aponta para o início de uma nova linha

```
int main(){
 FILE * arq;
 char str [100];
 arq = fopen ("arquivo.txt" , "r");
 if (arq == NULL){
    printf ("Erro de abertura");
    return -1;
 }
 while(fgets(str,100,arq)!= NULL){
   printf("%s",str);
 }
 fclose (arq);
 return 0; }
```

Exemplo com `fscanf`

- A função `int feof()` acessa o indicador de end-of-file e verifica seu status
  - caso retorne 0, o final do arquivo não foi alcançado
  - caso contrário, atingiu-se o final do arquivo

```
int main(){
 FILE * arq;
 char str [100];
 arq = fopen ("arquivo.txt" , "r");
 if (arq == NULL){
    printf ("Erro de abertura");
    return -1;
 }
 while (feof(arq)==0){
   fscanf(arq,"%s", str);
   printf("%s ",str);
 }
 fclose (arq);
 return 0; }
```

Exercício

- Crie manualmente um arquivo chamado votos.txt com o nome e qtd de votos de 3 candidatos
    - Ex.:
      x 100
      y 56
      z 67

- Em seguida, crie uma função `void mostrar_votos(FILE* arq)` que leia o arquivo votos.dat e imprima na tela o nome dos candidatos com seus respectivos votos

Exercício

- Crie um programa em C para copiar o conteúdo de um arquivo para outro

Exercício

- Crie uma struct **Carro** que represente um carro com: **int ano, char modelo[100]** e **char marca[100];**
- Em seguida crie um catálogo de carros (vetor do tipo **Carro**) e o preencha com **n** carros;
- Por fim, crie uma função **void gerarCatalogo(Carros*, int)**, que grave todos os carros em um arquivo catalogo.txt