Linguagem de programação

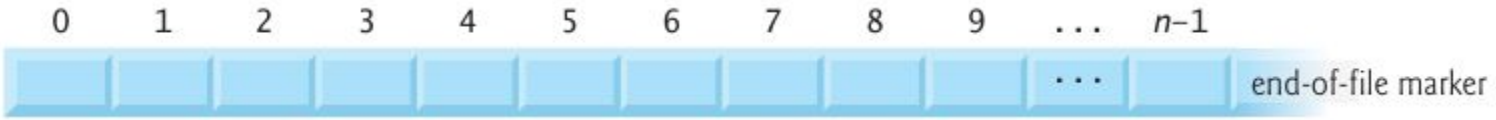
Arquivos

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.



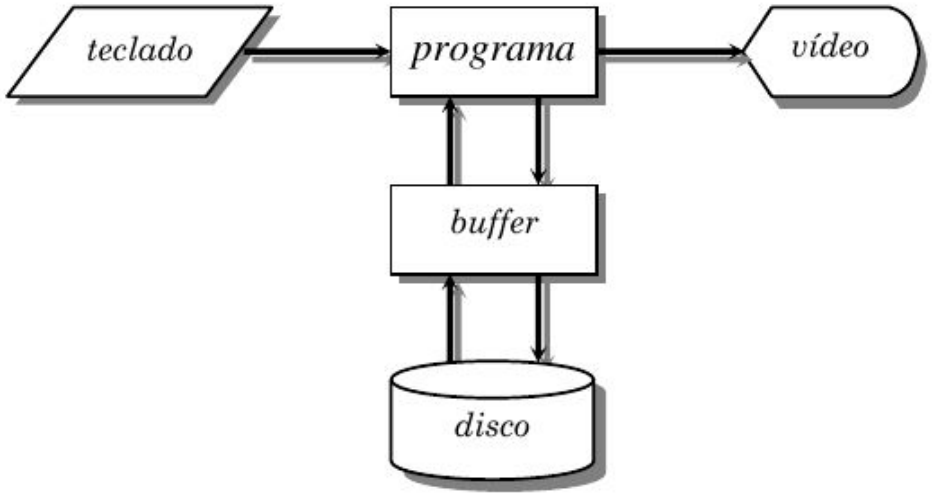
- Quando um arquivo é *aberto*, uma stream é associado a ele; e
- mais 2 arquivos e suas respectivas streams também são abertos: **standard input** e **standard output**

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
- *Buffers* melhoram a eficiência do uso de arquivos
  - São espaços de armazenamento temporário na memória
  - reduzem o nº de acessos ao disco (HD), aumentando a velocidade de execução do programa
  - Cada stream (**stdin** e **stdout)** possuem seus respectivos *buffers*

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/atualização): abre um arquivo para *atualização* (leitura ou gravação)
  - **“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/atualização): cria um arquivo para *atualização*
  - **“a”** (anexa): abre ou cria um arquivo para gravação no final.
    - **“a+”** (anexa/atualização): cria um arquivo para *atualização*. 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; }
```

Ler / escrever structs em um arquivo em C

- É fácil escrever string ou int em arquivo usando fprintf e putc,
- mas você pode enfrentar dificuldades ao escrever o conteúdo de uma struct.
- **fwrite** e **fread** tornam a tarefa mais fácil quando você deseja escrever e ler blocos de dados.

fwrite

```
size_t fwrite ( const void * ptr, size_t size, size_t count, FILE * stream );
```

Parâmetros:

- **ptr**: Ponteiro para a array de elementos a serem gravados
- **size**: Tamanho em bytes de cada elemento a ser escrito
  - **size_t** é um apelido para unsigned int
- **count:** Número de elementos, cada um com um tamanho de **size** bytes.
- **stream** : Ponteiro para um objeto FILE que especifica uma stream de saída.

Retorno:

- O número total de elementos gravados com sucesso é retornado.

fwrite - Exemplo

```c
// a struct to read and write
struct person{
   int id;  char fname[20]; char lname[20];
};

int main (){
   FILE *outfile;
   // open file for writing
   outfile = fopen ("person.dat", "w");
   if (outfile == NULL)  {
       fprintf(stderr, "\nError opend file\n");
       exit (1);   }

    struct person input1 = {1, "rohan", "sharma"};
    struct person input2 = {2, "mahendra", "dhoni"};
```

```c
// write struct to file
   fwrite (&input1, sizeof(struct person), 1,
outfile);
   fwrite (&input2, sizeof(struct person), 1,
outfile);

   if(fwrite != 0)
       printf("contents to file written
successfully !\n");
   else
       printf("error writing file !\n");
    // close file
   fclose (outfile);
    return 0; }
```

fread

```
size_t fread ( void * ptr, size_t size, size_t count, FILE * stream );
```

Parâmetros:

- **ptr:** ponteiro para um bloco de memória com um tamanho mínimo de **size*count** bytes.
- **size:** tamanho em bytes de cada elemento a ser lido
- **count:** número de elementos a serem lidos
- **stream :** ponteiro para um objeto FILE que especifica uma stream de entrada

Retorno:

O número total de elementos lidos com sucesso é retornado.

fread - Exemplo

```
int main (){
   FILE *infile;
   struct person input;

   // Open person.dat for reading
   infile = fopen ("person.dat", "r");
   if (infile == NULL)    {
       fprintf(stderr, "\nError opening
file\n");
       exit (1);
   }
```

```
// read file contents till end of file
   while(fread(&input, sizeof(struct person),
1, infile))
       printf("id = %d name = %s %s\n",
input.id, input.fname, input.lname);
    // close file
   fclose (infile);
   return 0;
 }
```

Exercício

- Escreva um programa em C para contar o número de palavras e caracteres em um arquivo

Exercício

- Escreva um programa em C para deletar uma linha específica de um arquivo

Suponha que o conteúdo do arquivo test.txt seja:

linha de teste 1
linha de teste 2
linha de teste 3
linha de teste 4

Insira o nome do arquivo a ser aberto: test.txt
Insira a linha que deseja remover: 2

Resultado esperado:

O conteúdo do arquivo test.txt é:
linha de teste 1
linha de teste 3
linha de teste 4

- Você pode precisar das seguintes funções
  - `int remove ( const char * filename );`
    - retorna 0, caso sucesso e diferente de 0 caso haja falha
  - `int rename ( const char * oldname, const char * newname );`
    - retorna 0, caso sucesso e diferente de 0 caso haja falha

Exercício

- Escreva um programa em C para mesclar dois arquivos e gravá-lo em um novo arquivo

  Ex.:

Arquivo1.txt:
1
2
3

Arquivo2.txt:
4
5
6

Mesclado.txt:
1
4
2
5
3
6