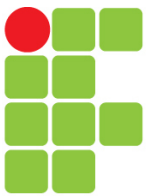
Entrada e saída com arquivos

Programação de computadores II

Prof. Renan Augusto Starke

Instituto Federal de Santa Catarina – IFSC
Campus Florianópolis
renan.starke@ifsc.edu.br

28 de fevereiro de 2020



INSTITUTO FEDERAL
SANTA CATARINA

Ministério da Educação
Secretaria de Educação Profissional e Tecnológica
INSTITUTO FEDERAL DE SANTA CATARINA

Tópicos da aula

Entrada e saída

- A linguagem C não contém nenhum comando específico para entrada e saída dados.
- A linguagem somente define sintaxe e semântica para:
  - utilização de dados.
  - controle de fluxo.
- Todas as operações de Entrada e Saída ocorrem mediante chamadas de funções de bibliotecas.
- Isto torna o sistema do C poderoso, flexível e multiplataforma.

Streams e arquivos

- A interface de E/S do C é independente do dispositivo.
- A abstração fornecida é chamada de *stream*.

Stream

Um **stream** é um fluxo contínuo de dados ordenados que pode vir de qualquer dispositivo: teclado, arquivos do disco, tela, ...

- Esta abstração permite que a mesma função escreva em um arquivo de disco ou tela.
- *Streams* podem ser:
  - textos.
  - binários.
- **Quando trabalhamos com streams nos arquivos do disco do computador, eles são lidos e escritos através de um buffer, ou seja, uma alteração será feita primeiro em memória e somente depois será transferido para o sistema de arquivos.**

Streams e arquivos

Stream de texto

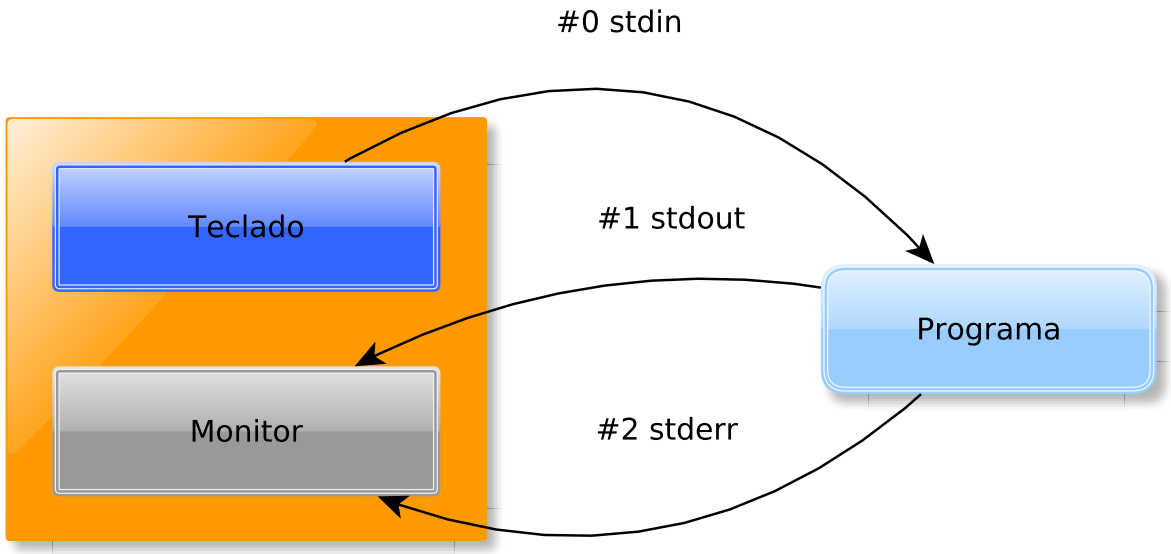
Um **stream** de texto é uma sequência de caracteres. O padrão C ANSI permite (mas não exige) que uma stream de texto seja organizada em linhas terminadas pelo caractere *'\n'*. Nos **streams** de texto, os *bytes* lidos são automaticamente traduzidos e interpretados como **texto**.

Stream binários

Um *stream* binária é uma sequência de bytes com uma correspondência de um para um com aqueles encontrados no dispositivo. **Não ocorre nenhuma tradução de caracteres**.

Arquivos em C

- #include <stdio.h>
- **Arquivo** é um objeto que contém informações em sequência
- Saídas especiais definidos em stdio.h
  - **stdin**: entrada padrão, ex: teclado
  - **stdout**: saída padrão, ex tela
  - **stderr**: saída de erros padrão
- **EOF**: *end of file* – final de aquivo. É uma constante especial utilizada para detectar quando chega-se no final de um arquivo



Abrindo arquivos

fopen

```
FILE* fopen (char* filename, char* mode)
```

mode:

- "r": (reading) abre somente para leitura. Arquivo deve existir
- "w": (writing) cria um arquivo vazio para escrita. Se houver arquivo com mesmo nome, ele é sobre-escrito
- "a": (append) acrescenta dados no final do arquivo. Se arquivo não existir, cria um novo

retorno da função:

- se função executa com sucesso, retorna um ponteiro para o arquivo em FILE
- se falha, retorna NULL em FILE

Exemplo

```c
FILE *fp = fopen("meu_arquivo.txt", "r");

if (fp == NULL){
    /* Imprime erro */
    perror("Erro em main: fopen");
    /* Aborta programa */
    exit(EXIT_FAILURE);
}
```

- Saída em caso de erro:

```
Erro em main: fopen: No such file or directory
```

Fechando arquivos

fclose

```
int fclose ( FILE * stream )
```

retorno da função:

- se função executa com sucesso, retorna 0
- se falha, retorna EOF

Exemplo

```c
/* Abre arquivo */
FILE *fp = fopen("meu_arquivo.txt", "r");

if (fp == NULL){
    /* Imprime erro e aborta */
    perror("Erro em main: fopen");
    exit(EXIT_FAILURE);
}

/* Manipulação de dados */
(...)

/* Fecha arquivo*/
fclose(fp);
```

Lendo um caractere

fgetc

```
int fgetc ( FILE * stream )
```

retorno da função:

- se função executa com sucesso, retorna um caractere
- se falha, retorna EOF e seta FILE no estado de final de aquivo

Exemplo

```
Este e' teste.

Saindo...
```

```c
FILE *fp = fopen("arquivo.txt", "r");

if (fp == NULL){
    /* Imprime erro e aborta */
    perror("Erro em main: fopen");
    exit(EXIT_FAILURE);
}

/* Manipulação de dados */
while ( (c = fgetc(fp)) != EOF){
    printf("car: '%c' \n",c);
}

/* Fecha arquivo*/
fclose(fp);
```

```
car: 'E'          car: ' '
car: 's'          car: 'e'
car: 't'          car: '''
car: 'e'          car: ' '        (...)
```

Lendo uma string

fgets

```
char * fgets ( char * str, int num, FILE * stream )
```

Comportamento:

- Lê até (**num**-1) caracteres de **FILE** em **str**
- Leitura das strings é terminada com NULL ('0')
- Pára quando uma nova linha é encontrada
- Pára quando final de aquivo é encontrado
- **str** não é modificado quando não se consegue ler nada

retorno da função:

- se função executa com sucesso, retorna str
- se falha, retorna NULL

Exemplo

```
Este e' teste.

Saindo...
```

```c
#define TAM_BUFFER 64

int main(){
    /* Abre arquivo */
    char c[TAM_BUFFER];
    FILE *fp = fopen("arquivo.txt", "r");

    if (fp == NULL){
        /* Imprime erro e aborta */
        perror("Erro em main: fopen");
        exit(EXIT_FAILURE);
    }

    /* Manipulação de dados */
    while ( fgets(c, TAM_BUFFER, fp) != NULL){
        printf("linha: %s \n",c);
    }
```

```
linha: Este e' teste.

linha:

linha: Saindo...
```

Lendo dados formatados

fscanf

```
int fscanf ( FILE * stream, const char * format, ... )
```

Entrada:

- **format** é análogo ao **printf**
  - %d para inteiro
  - %c para caractere
  - %s para string
- deve-se ter um argumento (variável) para cada especificador de formato

retorno da função:

- se sucesso, retorna o números de itens lidos. 0 se o padrão não foi encontrado
- se falhar, retorna EOF

Exemplo

```
Juca;45;M
Zeca;33;M
Maria;12;F
```

```c
int main(){
    char nome[64];      /* Nunca usar %s no fscanf */
    int idade;
    char sexo;

    FILE *fp = fopen("arquivo.csv", "r");

    if (fp == NULL){  (...)   }

    /* Manipulação de dados */
    while (fscanf(fp, "%63[^;];%d;%c\n", nome, &idade, &sexo) == 3)  {
        printf("%s  --  %d  --- %c\n",nome, idade, sexo);

    }
```

```
Juca  --  45  --- M
Zeca  --  33  --- M
Maria  --  12  --- F
```

Escrevendo um caractere

fputc

```
int fputc ( int character, FILE * stream )
```

Saída/efeito:

- no sucesso, escreve o caractere no arquivo e retorna o caractere escrito
- se falhar, retorna EOF e seta indicação de erro

Obs: verificação de erro pelo retorno menor que 0

Exemplo

```c
int main(){
    char str[] = "Teste  de string 12345566" ;
    int i;

    FILE *fp = fopen("texto.txt", "w");

    if (fp == NULL){
        /* Imprime erro e aborta */
        perror("Erro em main: fopen");
        exit(EXIT_FAILURE);
    }

    /* Manipulação de dados */
    for (i=0; i < strlen(str); i++){

        /* Escreve caractere por caractere no arquivo */
        fputc(str[i], fp);
    }

    fclose(fp);
```

```
texto.txt:
----------
Teste  de string 12345566
```

Escrevendo uma string

fputs

```
int fputs ( const char * str, FILE * stream )
```

Saída/efeito:

- no sucesso, escreve a string no arquivo e retorna um valor não negativo
- se falhar, retorna EOF e seta indicação de erro

Obs: verificação de erro pelo retorno menor que 0

Exemplo

```c
int main(){
    char str[] = "Teste  de string 12345566" ;

    FILE *fp = fopen("texto.txt", "w");

    if (fp == NULL){
        /* Imprime erro e aborta */
        perror("Erro em main: fopen");
        exit(EXIT_FAILURE);
    }

    /* Manipulação de dados */
    fputs(str, fp);

    fclose(fp);
```

```
texto.txt:
----------
Teste  de string 12345566
```

Escrevendo dados formatados

fprintf

```
int fprintf ( FILE * stream, const char * format, ... )
```

Entrada:

- **format** igualmente ao **printf**
- Argumento para cada formatador

Saída/efeito:

- no sucesso, retorna o número de caracteres escritos
- se falhar, retorna um número negativo

Obs: verificação de erro pelo retorno menor que 0

Exemplo

```c
int main(){
    char str[ ] = "Time 1 > Time 2" ;
    int h = 16;
    int t = 13;

    FILE *fp = fopen("texto.txt", "w");

    if (fp == NULL){
        /* Imprime erro e aborta */
        perror("Erro em main: fopen");
        exit(EXIT_FAILURE);
    }

    /* Escreve dados formatados no arquivo */
    fprintf(fp, "%s | Pontos: %d para %d\n", str, h ,t);

    /* Fecha arquivo*/
    fclose(fp);
}
```

```
texto.txt:
__________
Time 1 > Time 2 | Pontos: 16 para 13
```

Indo para o início

rewind

```
void rewind ( FILE * stream )
```

Efeito:

- Move FILE para o início do arquivo
- Limpa indicação do EOF
- Limpa indicação de erros
- Esquece qualquer caractere virtual fornecido por **ungetc**

Movendo-se para uma localização

fseek

```
int fseek ( FILE * stream, long int offset, int origin )
```

Entrada:

- Offset em bytes
- Origem:
  - SEEK_SET: início do arquivo
  - SEEK_CUR: localização atual
  - SEEK_END: final do arquivo

Saída/efeito:

- Se sucesso:
  - retorna 0
  - limpa indicador EOF
  - esquece qualquer caractere virtual fornecido por **ungetc**
- Se falhar, retorna um valor diferente de 0

Exemplo

```c
int main(){
    /* Abre arquivo */
    FILE *fp = fopen("meu_arquivo.txt", "w");

    if (fp == NULL){
        /* Imprime erro e aborta */
        perror("Erro em main: fopen");
        exit(EXIT_FAILURE);
    }

    /* Manipulação de dados */
    fputs("This is an apple", fp);
    fseek(fp, 9, SEEK_SET);
    fputs(" sam", fp);

    /* Fecha arquivo*/
    fclose(fp);
}
```

```
meu_arquivo.txt:
----------------
This is a sample
```

Abrindo arquivos binários

fopen

```
FILE* fopen (char* filename, char* mode)
```

Adiciona-se "b" em **mode** na função **fopen**

- "rb": ler arquivo binário
- "wb": escrever arquivo binário
- "ab": acrescentar para um arquivo binário

Escrevendo arquivos binários

fwrite

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

Entrada:

- **ptr**: um array de elementos ou somente um
- **size:** tamanho de cada elemento em bytes
- **count**: número de elementos

Saída:

- No sucesso, retorna o número de elementos escritos
- Se o retorno for diferente de **count**, houve um erro

Exemplo

```c
int main(){
    int ret, nums[] = {1,2,3};
    double d = 3.556887;

    /* Abre arquivo */
    FILE *fp = fopen("meu_arquivo.bin", "wb");

    if (fp == NULL){
        /* Imprime erro e aborta */
        perror("Erro em main: fopen");
        exit(EXIT_FAILURE);
    }

    ret = fwrite(nums, sizeof(int), 3, fp);
    printf("Escritos: %d elementos\n", ret);

    ret = fwrite(&d, sizeof(double), 1, fp);
    printf("Escritos: %d elementos\n", ret);

    /* Fecha arquivo*/
    fclose(fp);
}
```

- Valor binário em big endian:

```
meu_arquivo.bin:
----------------
0001 0000 0002 0000 0003 0000 8a59 2be4 7481 400c
```

Lendo arquivos binários

fread

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

Entrada:

- **ptr**: um ponteiro alocado com tamanho de no mínimo (*size* * *count*)
- **size:** tamanho de cada elemento em bytes
- **count**: número de elementos

Saída:

- No sucesso, retorna o número de elementos lidos
- Se o retorno for diferente de **count**, houve um erro ou atingiu-se o final do arquivo

Exemplo

```c
int main(){
    int ret,i, nums[5] = {0,0,0,0,0};

    /* Abre arquivo */
    FILE *fp = fopen("meu_arquivo.bin", "rb");

    if (fp == NULL){
        /* Imprime erro e aborta */
        perror("Erro em main: fopen");
        exit(EXIT_FAILURE);
    }

    ret = fread(nums, sizeof(int), 5, fp);
    printf("Lido: %d elementos\n", ret);

    for (i=0; i < 5; i++)
        printf("0x%x\n", nums[i]);

    /* Fecha arquivo*/
    fclose(fp);
}
```

```
Lido: 5 elementos
0x1   —   0x2    —   0x3    —   0x2be48a59   — 0x400c7481
```

Exercício

- Processar o arquivo **csv** (Moodle)
- Exibir os valores das colunas de forma tabulada na tela.