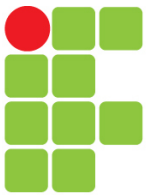
Entrada e saída com arquivos

Programação de computadores II

Prof. Renan Augusto Starke

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

23 de fevereiro de 2018



INSTITUTO FEDERAL
SANTA CATARINA

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

Tópicos da aula

Tópico

Objetivos

- Aprender a utilizar dados que estejam em arquivos
- E/S com Arquivos Textos
- E/S com Arquivos Binários

Tópico

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

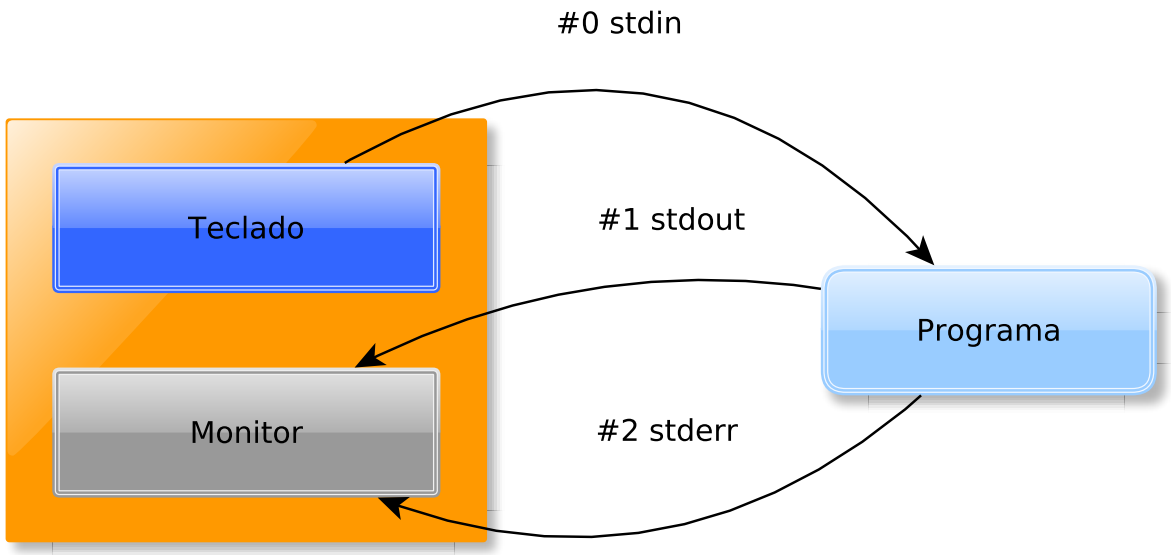
Um **stream** de texto é uma sequência de caracteres. O padrão C ANSI permite (mas não exige) que uma stream de texto seja organizada em linhas terminadas pelo caractere *'\n'*. Nos **streams** de texto, os *bytes* lidos são automaticamente traduzidos e interpretados como **texto**.

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



Tópico

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
```

Tópico

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);
```

Tópico

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: ' '        (...)
```

“Deslendo” um caractere

ungetc

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

Efeito:

- “Virtualmente” coloca um caractere de volta no arquivo
- Não modifica o arquivo
- Pode ser um caractere diferente do lido anteriormente

retorno da função:

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

Exemplo

```c
char key;

while (key != 'q'){
  /* Lê um caracter da entrada padrão */
  key = getchar();

  /* Se 'a', coloca '4' no lugar */
  if (key == 'a')
      ungetc('4', stdin);

  printf("%c\n", key);
}
```

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 80

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...
```

Tópico

Verificar se chegou-se ao final do arquivo

feof

```
int feof ( FILE * stream )
```

retorno da função:

- se chegou-se ao final do arquivo (EOF), retorna um valor diferente de 0
- senão, retorna 0

Obs: EOF é setado por fgets, fgetc, etc.

Exemplo

```
Este e' teste.

Saindo...
```

```c
#define TAM_BUFFER 80

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);
    }

    while (1){
        fgets(c, TAM_BUFFER, fp);

        if (feof(fp))
            break;

        printf("linha: %s \n",c); }
```

```
linha: Este e' teste.

linha:

linha: Saindo...
```

Exemplo – duas iterações

```
UW\n
CSE\n
\n
```

```c
while ( !feof(fp)){
   fgets(buf,BUFFER_SIZE,fp);
   printf("Read line: %s\n",buf);
}
```

```
Read line: UW
Read line: CSE
Read line: CSE
```

Tópico

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[TAM_BUFFER];
    int idade;
    char sexo;

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

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

    /* Manipulação de dados */
    while (1){
        fscanf(fp, "%80[^;];%d;%c\n", nome, &idade, &sexo);

        printf("%s  ---  %d  ---- %c\n",nome, idade, sexo);

        if (feof(fp))
            break;
    }
```

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

Exemplo

Há algum problema com este código?

```
WA
MO
```

```
...
FILE *fp = ...
char state[3];

while(fscanf(fp,"%s", state) != EOF)
   printf("Eu li: %s\n",state);
}
...
```

Exemplo

Há algum problema com este código?

```
WA
MO
Florianopolis
```

```
...
FILE *fp = ...
char state[3];

while(fscanf(fp,"%s", state) != EOF)
   printf("Eu li: %s\n",state);
}
...
```

Estouro de buffers

- Conhecido como *Buffer overruns*
- Dados são escritos na memória após o tamanho do buffer
- Explorado para executar código malicioso
- Usuário do programa **sempre** pode inserir uma entrada maior do que o tamanho do buffer
- Melhor **não** usar: scanf, fscanf, gets
- Utilizar funções que limitam o buffer explicitamente, **fgets**, e depois formatar com **sscanf**

Exercício

- Elaborar um programa em C com a seguinte especificação:
  - Implementar a leitura do arquivo "winterGames.csv"
  - Salvar os dados lidos em um *array* de estruturas com a seguinte especificação:

```
struct jogosInver {
  unsigned char pos;
  char nome[32];
  (...)
};
```

  - Use alocação estática do *array*
  - Faça o controle adequado de erros e *buffers overflows*

Tópico

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 */
        if (fputc(str[i], fp) < 0) {
            perror("fputc");
            exit(EXIT_FAILURE);
        }
    }

    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" ;
    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 */
    if (fputs(str, fp) < 0) {
        perror("fputs");
        exit(EXIT_FAILURE);
    }

    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
```

Tópico

Ocorreu um erro?

ferror

```
int ferror ( FILE * stream )
```

Saída:

- Se o indicador de erro estiver setado, retorna um número diferente de 0
- Senão retorna 0

Exemplo

```c
int main(){

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

    perror("fopen");
    printf("Em errro?  %d\n", ferror(fp));
}
```

```
fopen: Success
Em errro?  0
```

Imprimindo descrição do erro

perror

```
void perror ( const char * str )
```

Efeito:

- Imprime a descrição do erro. Pode fornecer detalhes através de **str**
- **str** pode ser NULL

Exemplo

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

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

    /* Manipulação de dados */

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

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

Tópico

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
```

Tópico

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 **arquivo_entrada.mif** (Moodle)
- Gerar um arquivo **binário** contendo apenas os valores hexadecimais

Texto

```
CONTENT BEGIN
0 : 080F4200;
1 : 95000000;
2 : A0000208;

(...)

60  : 00000000;
61  : 40002000;
62  : 80000000;
63  : 00000100;
[64..255] : FFFFFFFF;
```


Binário

```
000000 4200 080f 0000 9500 0208 a000 c008 8000
0000010 0003 b000 001f 3180 0000 0800 0000 8800
0000020 830c 881e 47cc a501 0005 b000 002c b000
0000030 fffc 317f 0000 0800 0000 0800 0000 8800
0000040 830c 881e 4280 0800 82c0 080f 0000 9500
0000050 c240 080f 0000 1500 0200 080f 0000 9500
0000060 428c a501 028b a000 028c a501 0249 a000
0000070 824c a500 0208 a000 2208 8880 420c a500
0000080 000c 3100 000c a500 020c a000 824c a000
0000090 028c 2001 2208 8840 9248 0000 0000 8800
00000a0 0249 2000 8288 8000 0248 a500 020c a000
00000b0 1208 8800 020c a500 420c a000 024c a000
00000c0 0209 8660 ffee 3c7f 0000 0800 0000 8800
00000d0 001f 3180 830c 0801 0000 0800 0000 8800
00000e0 001f 3180 0400 0800 0000 0800 0000 8800
00000f0 0000 0000 2000 4000 0000 8000 0100 0000
0000100 0000 0000 ffff ffff ffff ffff ffff ffff
0000110 ffff ffff ffff ffff ffff ffff ffff ffff
*
0000400
```

Mais informações

- Dentro do Code::Blocks, aparece detalhes das funções
- Se Linux, manpages: man fprintf
- http://www.cplusplus.com/reference/cstdio/