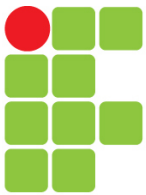
# Entrada e saída com arquivos

## Programação de computadores II

Prof. Renan Augusto Starke

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

22 de agosto de 2016



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

# 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

## fopen

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

if (fp == NULL){
   //informa erro e tenta recuperar-se
}else{
   //faz algo com o arquivo aberto
}
```

# Tópico

# Fechando arquivos

## fclose

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

retorno da função:

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

# 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

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

## fgetc

```
FILE *fp = ...
...
while ( (c = fgetc(fp)) != EOF){
  printf("car:'%c'\n",c);
}
```

```
car:'U'
car:'W'
car:'
'
car:'C'
car:'S'
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

## ungetc

```
...
FILE *fp = ...
int c;
...
while ((c = fgetc(fp)) != EOF){
  if (c == 'a'){
    ungetc('4',fp);
  }else{
    printf("read char %c\n",c);
  }
}
...
```

# 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

## fgets

```
#define BUFFER_SIZE 80
...
FILE *fp = ...
...
char buf[BUFFER_SIZE];
fgets(buf, BUFFER_SIZE, fp);
```

# 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

## feof

```
FILE *fp = ...
...
while (!feof(fp)){
  //leia algo
}
```

# Exemplo

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

## feof

```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

```
1 string1
42 string2
54 string3
...
```

## fscanf

```
FILE *fp = ...

char buf[TAMANHO_BUFFER];
int num;

while (!feof(fp)){
   fscanf(fp, "%d %s", &d, buf)
   //faca algo
}
```

# Exemplo

Há algum problema com este código?

```
WA
MO
...
```

## fscanf

```
...
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
```

## fscanf

```
...
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**

# 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

## fputc

```
...
FILE *fp = fopen("myfile.txt","w");
char str[] = "Teste de string 12345566";
int i;

if (fp != NULL){
  for (i = 0; i < strlen(str); i++){

    if (fputc(str[i], fp) < 0){

        // algo errado aconteceu
    }
  }
  fclose(fp);
}
...
```

# 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

## fputs

```
...

FILE *fp = fopen("myfile.txt","w");
char str[] = "Mais um teste de string";

if (fp != NULL){
  if (fputs(str, fp) < 0){
    // Algo malefico aconteceu
  }
  fclose(fp);
}
...
```

# 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

## fprintf

```
...
FILE *fp = fopen("myfile.txt","w");
int h = 16;
int t = 13;
char str[] = "Time 1 > Time 2";

if (fp != NULL){
   fprintf(stdout,"%s | Pontos: %d para %d\n",str,h,t);
   fclose(fp);
}
...
```

```
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

## ferror

```
...
FILE *fp = ...
...
fputs("Eu gosto de C!",fp);

if (ferror(fp)){
  //Reporta o erro e recupera-se
}
...
```

# 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

## perror

```
...
FILE *fp = ...
...

fputs("Eu gosto de C!",fp);

if (ferror(fp)){
   perror("Nao consegui escrever no arquivo");
   //Reporta o erro e recupera-se
}
```

# Limpando indicação de erro

## clearerr

```
void clearerr ( FILE * stream )
```

Efeito:

- Limpa indicação do error
- Limpa indicação EOF

# 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

## fseek

```
...
FILE * fp = fopen("myfile.txt" , "w" );
fputs ( "This is an apple." , fp );
fseek ( fp, 9, SEEK_SET );
fputs ( " sam", fp );
fclose ( fp );
...
```

```
This is a sample
```

# Tópico

# Removendo arquivos

## remove

```
int remove ( const char * filename )
```

Saída:

- Sucesso, retorna 0
- Se falha, retorna valor diferente de 0

# Renomeando arquivos

## rename

```
int rename ( const char * oldname, const char * newname );
```

Saída:

- Sucesso, retorna 0
- Se falha, retorna valor diferente de 0

# 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

## fwrite

```
...
int ret = 0;
FILE *fp = fopen("myfile.bin","wb");
...
int nums[] = {1,2,3};
ret = fwrite(nums, sizeof(int), 3, fp);
//Verificacao de erros

double dub = 3.1;
ret = fwrite(&dub, sizeof(double), 1, fp);
//Verificacao de erros
...
```

# 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

## fread

```
...
FILE *fp = fopen("myfile.bin","rb");
...
int nr;
int nums[3];
nr = fread(nums, sizeof(int), 3, fp);

//Verificacao de erros
double dub;
nr = fread(&dub, sizeof(double), 1, fp);

//Verificacao de erros
```

# Mais informações

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