

# Unidade V - Entrada e Saída

Disciplina Linguagens de Programação I  
Bacharelado em Ciência da Computação da Uerj  
Professores Guilherme Mota e Leandro Marzulo

## ANSI C

```
#include <stdio.h>
main ()
{
    printf("Hello, World!");
}
```

# Que assuntos serão abordados nesta unidade?

- E/S padrão
  - `getchar()`
  - `putchar()`
  - `gets()`
  - `puts()`
  - `<, | e >`
- E/S formatada
  - `printf()`
  - `sprintf()`
  - `scanf()`
  - `sscanf()`
- Canais de E/S
  - `stdin`
  - `stdout`
  - `stderr`
  - `fflush()`
- Manipulação de arquivos
  - Arquitetura geral de E/S
  - *Buffer*
  - Arquivos texto
  - Arquivos binários

# E/S Padrão

# O básico

- Lidando com caracteres
- `int getchar(void)` → Lê e retorna um caractere
  - `EOF` = término da leitura (End of File)
  - `\n` = fim da linha
- `int putchar(int)` → Imprime um caractere
  - Retorna o caractere impresso ou `EOF` em caso de erro.

# Exemplo

```
/* progPag125.c */
#include <stdio.h>
#include <ctype.h>
main() /* lower: convert input to lower case */
{
    int c;
    while ((c = getchar()) != EOF)
        putchar(tolower(c));
    return 0;
}
```

# Redirecionamento de E/S

- O sistema operacional gerencia o redirecionamento de E/S de um programa
- O redirecionamento pode ser definido na linha de comando na entrada em execução do(s) programa(s)
  - ♦ > redireciona a saída para um arquivo (apaga o arquivo se ele existir)
  - ♦ >> redireciona saída que é anexada ao fim de um arquivo (*append*)
  - ♦ < redireciona o arquivo para a entrada do programa
  - ♦ | redireciona a saída de um programa para a entrada de outro

# Lendo e escrevendo uma linha por vez

- `char* gets(char* str)` → Lê uma string e coloca em `str`
  - O caractere `\n` não é incluído
- `int puts(const char* str)` → Imprime a string `str`
  - Segue até encontrar o caractere `\0`
  - Inclui o caractere `\n`
  - Retorna inteiro maior que zero em caso de sucesso
  - Retorna `EOF` em caso de erro

# Exemplo

```
/* gets example */  
#include <stdio.h>  
  
int main()  
{  
    char string [256];  
    printf ("Insert your full address: ");  
    gets (string);  
    printf ("Your address is: %s\n", string);  
    return 0;  
}
```



# Exemplo

```
/* puts example : hello world! */  
#include <stdio.h>  
  
int main ()  
{  
    char string [] = "Hello world!";  
    puts (string);  
}
```

# **E/S Formata**

# printf() e sprintf()

```
int printf(char* format, arg1, arg2, ...);
```

```
int sprintf(char* string, char* format, arg1, arg2, ...);
```

- Imprime a string `format`
- Caso `format` contenha especificadores de formato estes são substituídos pelo respectivo argumento coerentemente formatado
- Retorna o número de caracteres impressos
- Caso ocorra algum erro retorna um número negativo

# printf() e sprintf()

```
int printf(char* format, arg1, arg2, ...);
```

```
int sprintf(char* string, char* format, arg1, arg2, ...);
```

Caractere	Tipo do argumento; impresso como
d,i	int; decimal
o	int; octal sem sinal
x, X	int; hexa sem sinal (sem o 0X ou 0x na frente)
u	int; decimal sem sinal
c	int; caractere
s	char *; imprime até encontrar o \0 ou o número de caracteres indicado pela precisão
f	double; [-]m.dddddd, onde o número de d's é dado pela precisão (padrão 6)
p	void *; ponteiro (representação dependente de implementação)
%	Nenhum argumento é convertido; imprime um %

# Imprimindo strings

```
printf("%. *s", max, s); /*string s é  
impressa com até max caracteres */
```

formatação	Impressão (“Hello, World” → 12 caracteres)
:%s:	:Hello, World:
:%10s:	:Hello, World:
:%.10s:	:Hello, Wor:
:%-10s:	:Hello, World:
:%.15s:	:Hello, World:
:%-15s:	:Hello, World :
:%15.10s:	: Hello, Wor:
:%-15.10s:	:Hello, Wor :

## scanf() e sscanf()

```
int scanf(char* format, arg1, arg2, ...);  
int sscanf(char* string, char* format, arg1, arg2, ...);
```

- Lê dado(s) do canal de entrada default e armazena na variável correspondente, considerando a string format (formatação)
- Os argumentos adicionais são ponteiro para posições de memória previamente alocadas
- Em caso de sucesso retorna o número de argumentos lidos
- Em caso de sucesso parcial um as funções ferror e feof são setadas convenientemente
- Em caso de insucesso total esta função retorna EOF

## scanf() **e** sscanf()

```
int scanf(char* format, arg1, arg2, ...);  
int sscanf(char* string, char* format, arg1, arg2, ...);
```

```
#include <stdio.h> /* ProgPag129.c */  
main() /* rudimentary calculator */  
{  
    double sum, v;  
    sum=0;  
    while(scanf("%lf", &v) == 1)  
        fprintf("\t%.2f\n", sum += v);  
}
```

## scanf() e sscanf()

```
int scanf(char* format, arg1, arg2, ...);  
int sscanf(char* string, char* format, arg1, arg2, ...);
```

```
while (getline(line, sizeof(line)) > 0) {  
    if (sscanf(line, "%d %s %d", &day, monthname, &year) == 3)  
        printf("valid: %s\n", line); /* 25 Dec 1988 form */  
    else if (sscanf(line, "%d/%d/%d", &month, &day, &year) == 3)  
        printf("valid: %s\n", line); /* mm/dd/yy form */  
    else  
        printf("invalid: %s\n", line); /* invalid form */  
}
```



# Arquivos

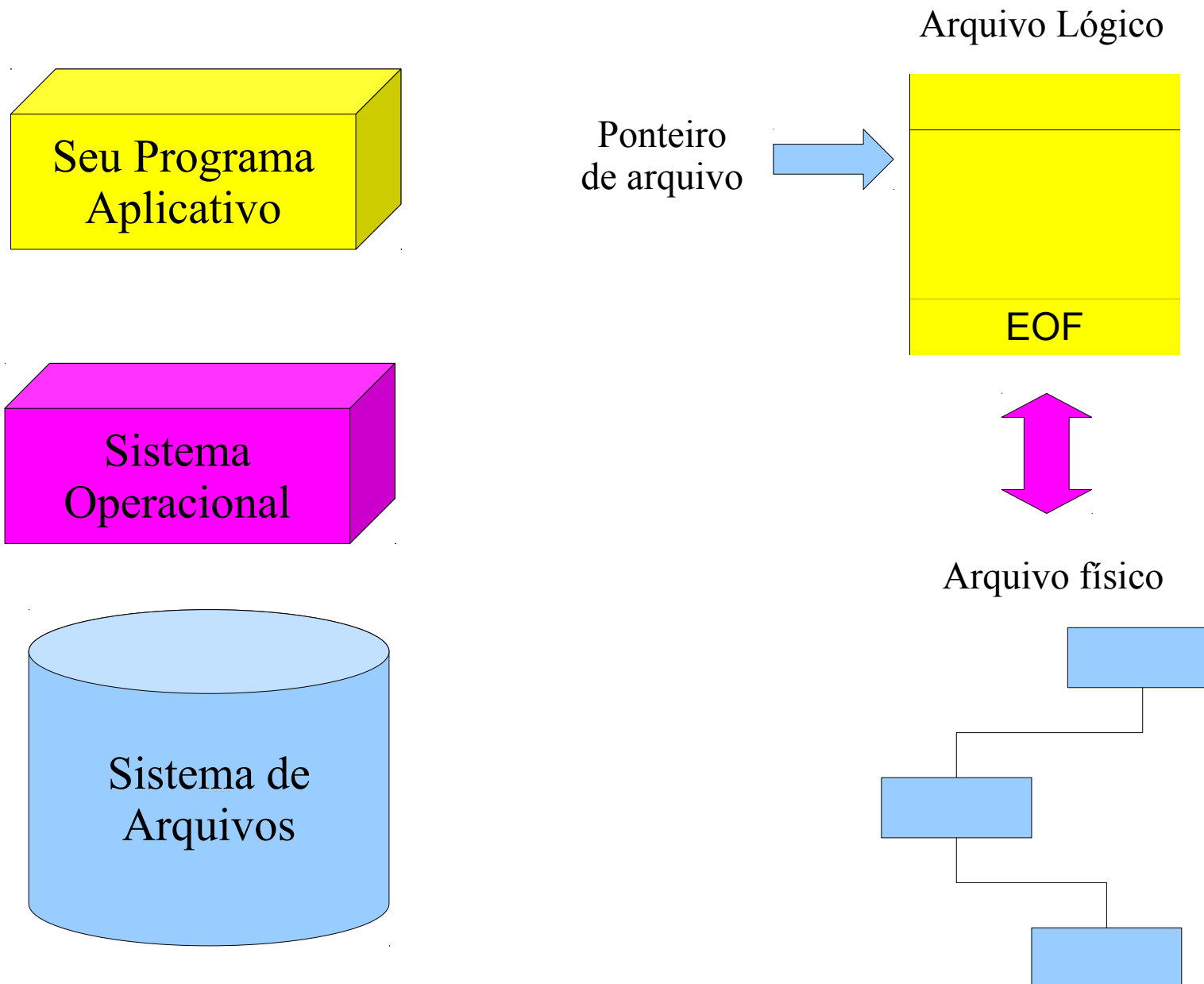
# Streams - arquivos padrão

- `stdin` – Standard Input
  - Conectada ao teclado
- `stdout` – Standard Output
  - Conectada ao terminal
- `stderr` – Standard Erro
  - Conectada ao terminal

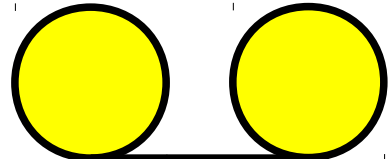
# Arquivos Físico e Lógico

- Arquivo Físico:
  - Conjunto de bytes armazenada no disco
  - Geralmente agrupados em setores de dados.
  - Gerenciado pelo sistema operacional
- Arquivo Lógico:
  - Modo como a linguagem de programação enxerga os dados.
  - Uma sequência de bytes, eventualmente organizados em registros ou outra estrutura lógica.
- Associação arquivo físico – arquivo lógico: iniciada pelo aplicativo, gerenciada pelo S.O.

# Arquivo Físico e Lógico



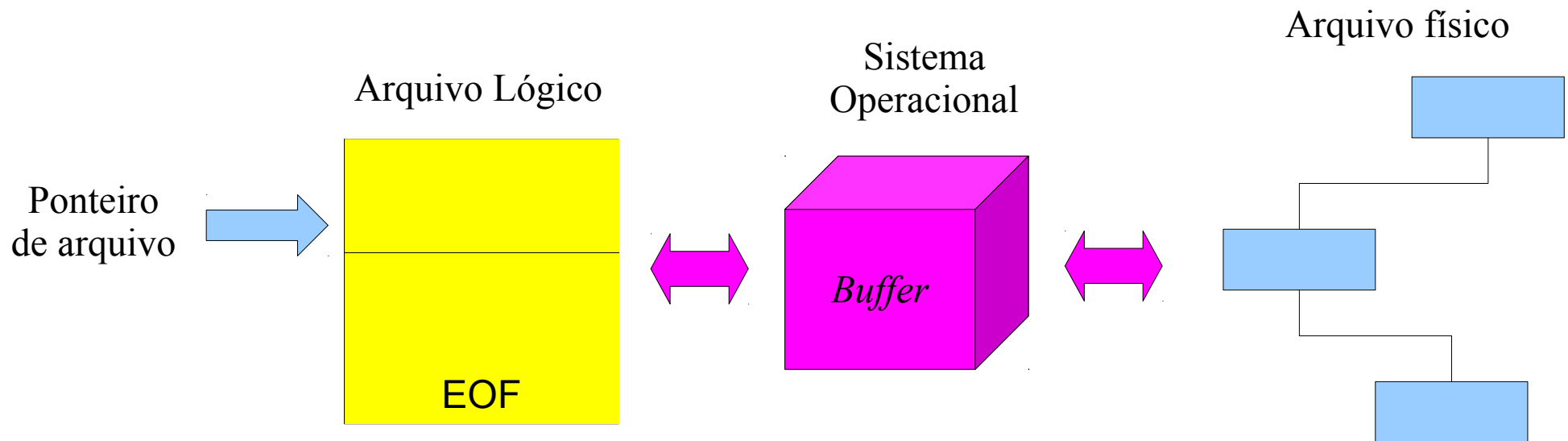
# Manipulando arquivos

- Arquivo lógico é uma estrutura sequencial representada por um ponteiro de arquivo `FILE *`
- O ponteiro de arquivo aponta para a posição do “cabeçote de leitura” no arquivo lógico 
- Abertura faz a conexão entre arquivo físico e lógico
  - `int fopen(char* name, char* mode)`
- Fechamento desfaz a conexão
  - `int fclose(FILE* fp)`

# `fopen()` – modos

Modo	Significado
r	Abre o arquivo somente para leitura
w	Abre o arquivo para escrita (cria ou sobrescreve o arquivo existente)
a	Abre o arquivo para escrita (cria ou anexa no arquivo existente)
t	O arquivo é tratado na forma de texto
b	O arquivo é tratado na forma de binário

# Buffer



```
FILE* FP;  
FP = fopen("arquivo");  
fflush(FP);  
fclose(FP);
```

## Forçando a descarga com `fflush()`

- Arquivos do usuário, bem como `stdout` e `stderr` são associados a *buffers*
- Os dados provenientes da entrada padrão `stdin` são tratados de forma imediata
- A escrita só acontece quando o *buffer* enche
- A função `int fflush(FILE * stream)` força a escrita do conteúdo do *buffer* no arquivo físico.



# Manipulando arquivos texto

- Abertura com as opções "r" "w" "a" etc
- Leitura e escrita com as funções
  - `int getc(FILE* fp)`
  - `int putc(int c, FILE* fp)`
  - `char * fgets ( char * str, int num, FILE * stream )`
  - `int fputs ( const char * str, FILE * stream );`
  - `int fscanf(FILE* fp, char* format, ...)`
    - `int fprintf(FILE* fp, char* format, ...)`

# Manipulando arquivos binários

- Abertura com as opções "rb" "wb" "ab" etc
- Leitura e escrita com as funções
  - `size_t fread(void *ptr, size_t sizeofelements, size_t num_of_elements, FILE *a_file);`
  - `size_t fwrite(const void *ptr, size_t sizeofelements, size_t num_of_elements, FILE *a_file);`

# Tratamento de Erro

# stderr e tratamento de erro

```
#include <stdio.h> /* ProgPag133 cat: concatenate files, version 2*/
main(int argc, char *argv[])
{
    FILE *fp;
    char *prog = argv[0]; /* program name for errors */
    if (argc == 1) /* no args; copy standard input */
        filecopy(stdin, stdout);
    else
        while (--argc > 0)
            if ((fp = fopen(*++argv, "r")) == NULL) {
                fprintf(stderr, "%s: can't open %s\n", prog, *argv);
                exit(1);
            }
            else {
                filecopy(fp, stdout);
                fclose(fp);
            }
            if (ferror(stdout)) {
                fprintf(stderr, "%s: error writing stdout\n", prog);
                exit(2);
            }
        exit(0);
}
```

## Exercício - Arquivo Texto

Faça um programa que conte o número de caracteres imprimíveis, o número de linhas e o número total de caracteres de um arquivo. O nome do arquivo deve ser fornecido pelo teclado.

# Exemplos

# Exemplo: Programa Cat

```
#include <stdio.h> /* ProgPag132.c -> cat: concatenate files, version 1*/
void filecopy(FILE *, FILE *);
main(int argc, char *argv[])
{
    FILE *fp;
    if (argc == 1) /* no args; copy standard input */
        filecopy(stdin, stdout);
    else
        while(--argc > 0)
            if ((fp = fopen(*++argv, "r")) == NULL)
            {
                printf("cat: can't open %s\n", *argv);
                return 1;
            }
            else
            {
                filecopy(fp, stdout);
                fclose(fp);
            }

        return 0;
}

void filecopy(FILE *ifp, FILE *ofp)
{
    int c;
    while ((c = getc(ifp)) != EOF)
        putc(c, ofp);
}
```

# Exemplo: Implementação da função `fgets`

```
/* ProgPag134 */
/* fgets: get at most n chars from iop */
char *fgets(char *s, int n, FILE *iop)
{
    register int c;
    register char *cs;
    cs = s;
    while (--n > 0 && (c = getc(iop)) != EOF)
        if ((*cs++ = c) == '\n')
            break;
    *cs = '\0';
    return (c == EOF && cs == s) ? NULL : s;
}
```



# Exemplo: Implementação da função `fputs`

```
/* fputs: put string s on file iop */
int fputs(char *s, FILE *iop)
{
    int c;
    while (c = *s++)
        putc(c, iop);
    return ferror(iop) ? EOF : 0;
}
```

# Exemplo: Escrita em arquivo binário

```
#include <stdio.h>
main ()
{
    FILE * pFile;
    int bufferInt[]={11,21,31};
    pFile = fopen ("myfile.bin", "wb");
    fwrite (bufferInt , sizeof(int),
sizeof(bufferInt), pFile);
    fflush(pFile);
    fclose (pFile);
    return 0;
}
```

# Exemplo: Escrita em arquivo texto formatado

```
/* www.cplusplus.com fprintf example */
#include <stdio.h>
main ()
{
    FILE * pFile;
    int n;
    char name [100];

    pFile = fopen ("myfile.txt", "w");
    for (n=0 ; n<3 ; n++)
    {
        puts ("please, enter a name: ");
        gets (name);
        fprintf (pFile, "Name %d [%-10.10s]\n", n, name);
    }
    fclose (pFile);
}
```

# Exemplo: Leitura em arquivo texto formatado

```
/* www.cplusplus.com fscanf example */
#include <stdio.h>
main ()
{
    char str [80];
    float f;
    FILE * pFile;

    pFile = fopen ("myfile.txt", "w+"); /* "w+" -> write-
update-rewrite*/
    fprintf (pFile, "%f %s", 3.1416, "PI");
    rewind (pFile);
    fscanf (pFile, "%f", &f);
    fscanf (pFile, "%s", str);
    fclose (pFile);
    printf ("I have read: %f and %s \n", f, str);
}
```

# Exercício - Conversão Arquivo Texto para Binário

Faça um programa que leia um arquivo texto contendo números inteiros na notação octal e crie um arquivo binário correspondente.

**Fim**