

# Unidade IV - Entrada e Saída

Disciplina Linguagens de Programação I  
Bacharelado em Ciência da Computação da Uerj  
Professores Guilherme Abelha e Gilson Costa

# ANSI C

```
#include <stdio.h>
int main () {
    printf("Hello, World!");
    return 0;
}
```

# Que assuntos serão abordados nesta unidade?

- E/S padrão
  - `getchar()`
  - `putchar()`
  - `fgets()`
  - `puts()`
  - `<, | e >`
- E/S formatada
  - `printf()`
  - `sprintf()`
  - `scanf()`
  - `sscanf()`
- Canais de E/S
  - `stdin`
  - `stdout`
  - `stderr`
- 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>  
  
int main() {  
    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* fgets(char* str, int num, FILE* stream)` → Lê uma string e coloca em `str`
  - O caractere `\n` é incluído
  - `stream = stdin` → teclado (por padrão)
- `int puts(const char* str)` → Imprime `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

```
/* fgets example */  
#include <stdio.h>  
  
int main() {  
    char string [256];  
    printf ("Insert your full address: ");  
    fgets (string, 255, stdin);  
    puts ("Your address is:");  
    puts(string);  
    return 0;  
}
```



# Exemplo

```
/* puts example : hello world! */  
#include <stdio.h>  
  
int main () {  
    char string [] = "Hello world!";  
    puts (string);  
    return 0;  
}
```

# **E/S Formatada**

# 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)
%e	double; notação científica
%g	double; menor representação entre %f e %e
%p	void *; apontador (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 apontadores para posições de memória previamente alocadas
- Em caso de sucesso, retorna o número de argumentos lidos
- Em caso de sucesso parcial, 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 */  
int main(void){  
    double sum, v;  
    sum=0;  
    while(scanf("%lf", &v) == 1)  
        printf("\t%.2f\n", sum += v);  
    return 0;  
}
```

## scanf() e sscanf()

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

```
int month, day, year;  
char monthname[16], line[256];  
  
while (fgets(line, sizeof(line), stdin) > 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

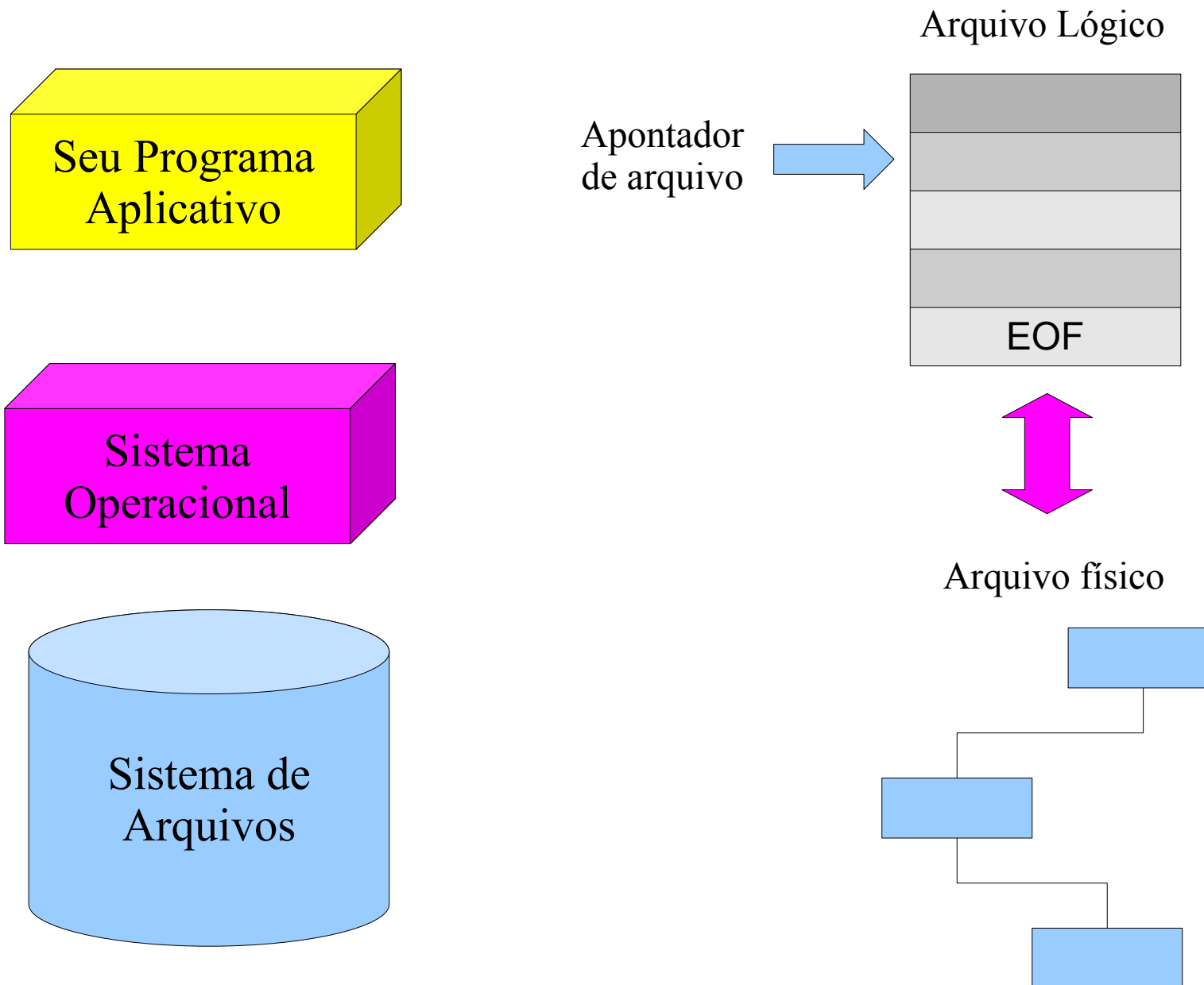
# Streams - arquivos padrão

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

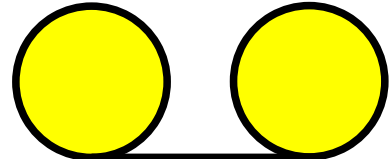
# Arquivos Físico e Lógico

- Arquivo Físico:
  - Conjunto de bytes armazenado no disco
  - Geralmente agrupado 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 apontador de arquivo `FILE *`
- O apontador de arquivo aponta para a posição do “cabeçote de leitura” no arquivo lógico 
- Abertura faz a conexão entre arquivo físico e lógico
  - `FILE* fopen(char* name, char* mode)`
- Fechamento desfaz a conexão
  - `int fclose(FILE* fp)`

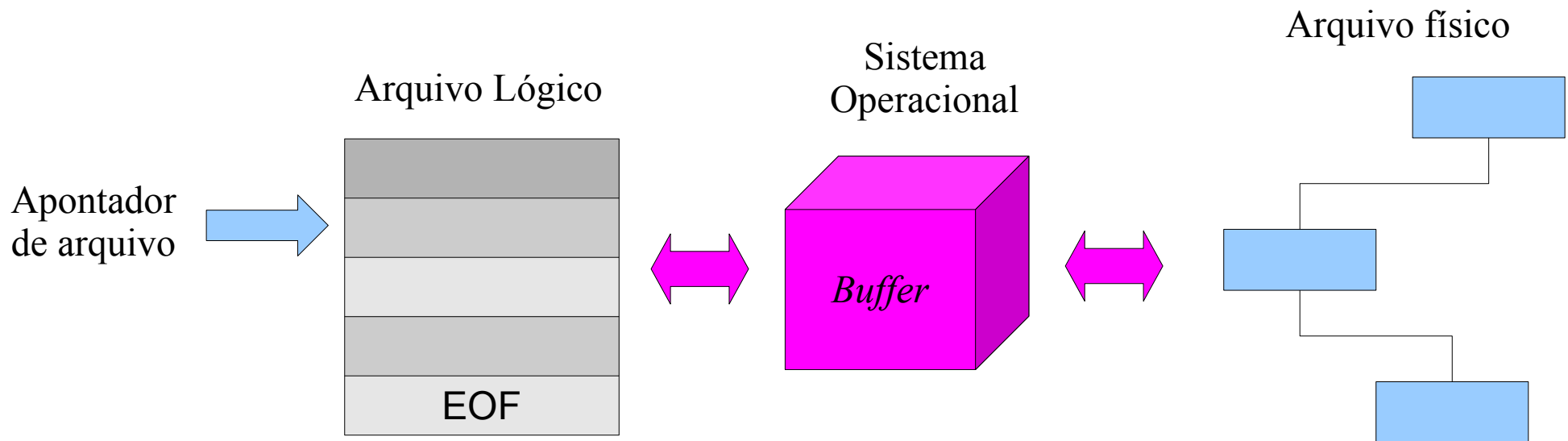
# `fopen()` – modos

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
w+ / r+	Abre o arquivo para leitura e escrita

# fopen() - modos

	Modo de abertura					
	<b>r</b>	<b>r+</b>	<b>w</b>	<b>w+</b>	<b>a</b>	<b>a+</b>
<i>apontador no início</i>	X	X	X	X		
<i>apontador no fim</i>					X	X
<i>cria arquivo</i>			X	X	X	X
<i>sobrescreve arquivo</i>			X	X		
<i>inexistência causa erro</i>	X	X				
<i>permite leitura</i>	X	X		X		X
<i>permite escrita</i>		X	X	X	X	X

# Buffer



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



## Forçando a descarga com `fflush()`

- Arquivos do usuário, bem como `stdin`, `stdout` e `stderr` são associados a *buffers*
- Os dados provenientes da entrada padrão são imediatamente armazenados em `stdin`
- A escrita em `stdout`, `stderr` ou nos arquivos do usuário só é descarregada 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*/
int main(int argc, char *argv[]){
    FILE *fp;
    int i=1;
    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[i++], "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);
}
```

# Outras funções importantes

- `fseek` e `fsetpos` reposiciona stream
- `rewind` volta ao início do stream
- `fgetpos` e `ftell` obtém a posição atual
- `feof` verifica fim de arquivo
- `remove` remove arquivo
- `rename` renomeia arquivo

## Exercício U4.1- Análise de 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>

void filecopy(FILE *, FILE *);

int main(int argc, char *argv[]){
    FILE *fp;
    int ind;

    if (argc == 1) /* no args; copy standard input */
        filecopy(stdin, stdout);
    else
        for (ind=1; ind < argc ; ind++)
            if ((fp = fopen(argv[ind], "r")) == NULL){
                printf("cat: can't open %s\n", argv[ind]);
                return 1;
            }
            else {
                filecopy(fp, stdout);
                fflush(fp);
                fclose(fp);
            }
    return 0;
}
```

# Exemplo: Programa Cat

```
FILE *fp;
int ind;

if (argc == 1) /* no args; copy standard input */
    filecopy(stdin, stdout);
else
    for (ind=1; ind < argc ; ind++)
        if ((fp = fopen(argv[ind], "r")) == NULL){
            printf("cat: can't open %s\n", argv[ind]);
            return 1;
        }
        else{
            filecopy(fp, stdout);
            fflush(fp);
            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 int i=0;  
    while (--n > 0 && (c = getc(iop)) != EOF)  
        if ((s[i++] = c) == '\n')  
            break;  
    s[i] = '\0';  
    return (c == EOF && i==0) ? NULL : s;  
}
```

# Exemplo: Implementação da função `fputs`

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

# Exemplo: Escrita em arquivo binário

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

# Exemplo: Escrita em arquivo texto formatado

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

    pFile = fopen ("myfile.txt", "w");
    for (n=0 ; n<3 ; n++) {
        puts ("please, enter a name: ");
        fgets (name, 99, stdin);
        fprintf (pFile, "Name %d [%-10.10s]\n", n, name);
    }
    fflush(pFile);
    fclose (pFile);
    return 0;
}
```

# Exemplo: Leitura em arquivo texto formatado

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

    pFile = fopen ("myfile.txt", "w+");
    fprintf (pFile, "%f %s", 3.1416, "PI");
    fflush(pFile);
    rewind (pFile);
    fscanf (pFile, "%f", &f);
    fscanf (pFile, "%s", str);
    fclose (pFile);
    printf ("I have read: %f and %s \n", f, str);
    return 0;
}
```

## Trabalho 3- Leitura de Arquivo Binário

Faça um programa que leia um arquivo de imagem no formato PGM na variante P5 e apresente na tela do terminal o conteúdo correspondente ao formato PGM variante P2.



**Fim**