

Programação Estruturada

Prof. Paulo Henrique Pisani

<http://professor.ufabc.edu.br/~paulo.pisani/>

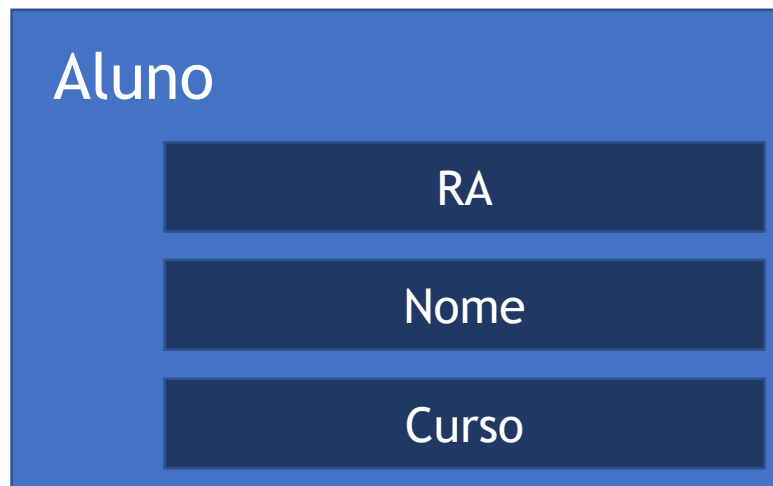
Tópicos

- Estruturas
- Alocação dinâmica de estruturas
- Arquivos

Estruturas (struct)

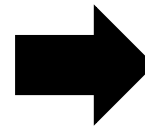
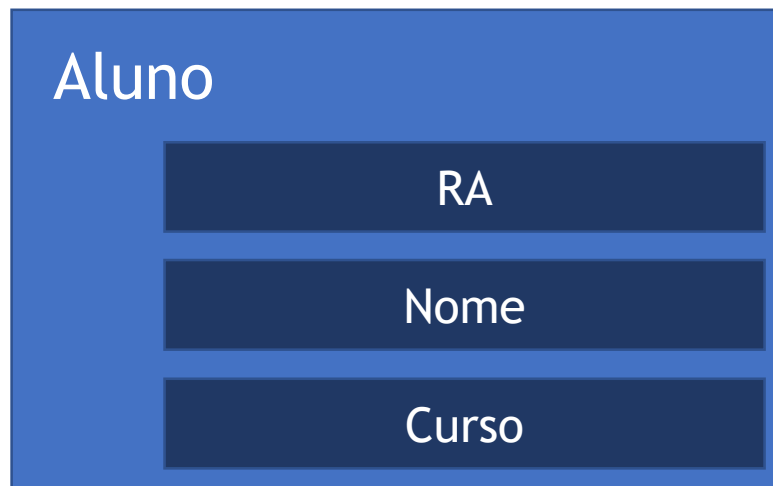
Estruturas

- Com o struct, definimos um novo tipo de dados;
- Esse tipo é uma estrutura que permite a combinação de itens de diferentes tipos de dados.



Estruturas

- Com o struct, definimos um novo tipo de dados;
- Esse tipo é uma estrutura que permite a combinação de itens de diferentes tipos de dados.




```
struct aluno {  
    int ra;  
    char nome[100];  
    char curso[20];  
};
```

Declarar variável do tipo estrutura

- Para declarar uma variável do tipo **struct aluno**:

```
struct aluno aluno1;  
struct aluno fulano1, fulano2;
```



Tipo da variável

Acesso a membros da estrutura

- Para acessar membros da estrutura, usamos o **ponto:**

```
struct aluno a1;  
a1.ra = 123;
```

```
struct aluno a2, a3;  
a2.ra = 100;  
a3.ra = 200;
```

```
scanf("%d", &a2.ra);  
scanf("%s", a2.nome);
```

```
struct aluno {  
    int ra;  
    char nome[100];  
    char curso[20];  
};
```

```
#include <stdio.h>
```

```
struct aluno {  
    int ra;  
    char nome[100];  
    char curso[20];  
};
```

} Declaração do tipo de dados (estrutura);

```
int main() {
```

```
    struct aluno p; ← Variável do tipo struct.
```

```
    scanf("%d", &p.ra);  
    scanf("%s", p.nome);  
    scanf("%s", p.curso);
```

```
    printf("RA=%d Nome=%s Curso=%s\n", p.ra, p.nome, p.curso);
```

```
    return 0;
```

```
}
```


Declarar variável do tipo estrutura

- Podemos criar um sinônimo para o tipo de dados, e assim facilitar a declaração;

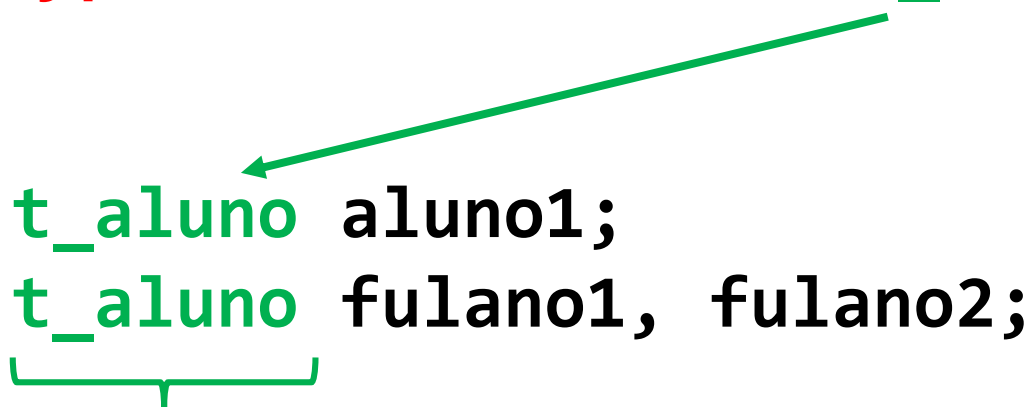
```
typedef struct aluno novo_nome;
```

Declarar variável do tipo estrutura

- Podemos criar um sinônimo para o tipo de dados, e assim facilitar a declaração;

```
typedef struct aluno t_aluno;
```

```
t_aluno aluno1;  
t_aluno fulano1, fulano2;
```



Tipo da variável

```
#include <stdio.h>
```

```
struct aluno {  
    int ra;  
    char nome[100];  
    char curso[20];  
};
```

Declaração do tipo de dados (estrutura);

```
typedef struct aluno t_aluno;
```

```
int main() {
```

```
    t_aluno p; ← Variável do tipo struct.
```

```
    scanf("%d", &p.ra);  
    scanf("%s", p.nome);  
    scanf("%s", p.curso);
```

```
    printf("RA=%d Nome=%s Curso=%s\n", p.ra, p.nome, p.curso);
```

```
    return 0;
```

```
}
```

Outro exemplo

- Estrutura para armazenar um ponto de duas dimensões:

```
typedef struct ponto t_ponto;  
struct ponto {  
    int x, y;  
};
```

```
#include <stdio.h>
#include <math.h>

struct ponto {
    int x, y;
};
typedef struct ponto t_ponto;

double distancia(t_ponto p1, t_ponto p2) {
    return sqrt((p1.x - p2.x) * (p1.x - p2.x)
        + (p1.y - p2.y) * (p1.y - p2.y));
}

int main() {
    t_ponto p1, p2;
    p1.x = 3;
    p1.y = 4;

    p2.x = 1;
    p2.y = 2;

    printf("%.2lf\n", distancia(p1, p2));
    return 0;
}
```

O que será
impresso?

```
#include <stdio.h>
#include <math.h>

struct ponto {
    int x, y;
};
typedef struct ponto t_ponto;

double distancia(t_ponto p1, t_ponto p2) {
    return sqrt((p1.x - p2.x) * (p1.x - p2.x)
        + (p1.y - p2.y) * (p1.y - p2.y));
}

int main() {
    t_ponto p1, p2;
    p1.x = 3;
    p1.y = 4;

    p2.x = 1;
    p2.y = 2;

    printf("%.2lf\n", distancia(p1, p2));
    return 0;
}
```

O que será
impresso?

2.83

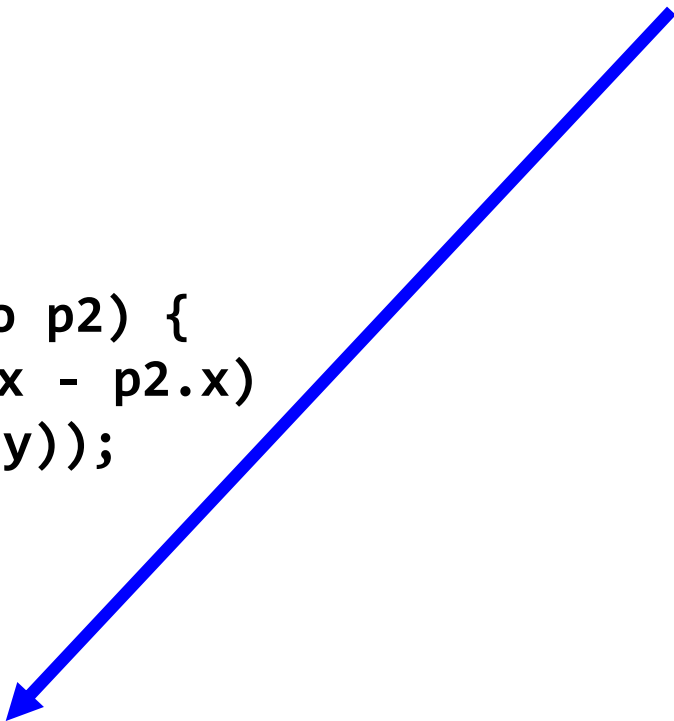
```
#include <stdio.h>
#include <math.h>
```

```
struct ponto {
    int x, y;
};
typedef struct ponto t_ponto;
```

```
double distancia(t_ponto p1, t_ponto p2) {
    return sqrt((p1.x - p2.x) * (p1.x - p2.x)
        + (p1.y - p2.y) * (p1.y - p2.y));
}
```

```
int main() {
    t_ponto p1 = {3, 4}, p2 = {1, 2};
    printf("%.2lf\n", distancia(p1, p2));
    return 0;
}
```

Podemos inicializar os membros de uma estrutura dessa forma também (similar ao modo como inicializamos um vetor).



```
#include <stdio.h>

struct aluno {
    int *ra;
    char *nome;
    double *nota;
};
typedef struct aluno t_aluno;

int main() {
    t_aluno a1;
    scanf("%d", &a1.ra);
    scanf("%s", a1.nome);
    scanf("%lf", &a1.nota);

    printf("RA=%d Nome=%s Nota=%.2lf\n",
        a1.ra, a1.nome, a1.nota);

    return 0;
}
```

O usuário entrará com
os valores 123 Teste 8

O que será
impresso?


```
#include <stdio.h>
```

```
struct aluno {  
    int *ra;  
    char *nome;  
    double *nota;  
};  
typedef struct aluno t_aluno;
```

```
int main() {  
    t_aluno a1;  
    scanf("%d", &a1.ra);  
    scanf("%s", a1.nome);  
    scanf("%lf", &a1.nota);  
  
    printf("RA=%d Nome=%s Nota=%.2lf\n",  
        a1.ra, a1.nome, a1.nota);  
  
    return 0;  
}
```

ra é um ponteiro, não precisa do & para ler o inteiro! O mesmo vale para nota, que é um ponteiro para double.

ra é um ponteiro, então temos que resolver o endereço. O mesmo vale para nota, que é um ponteiro para double.

O usuário entrará com os valores 123 Teste 8

O que será impresso?

Erro!

```
#include <stdio.h>

struct aluno {
    int *ra;
    char *nome;
    double *nota;
};
typedef struct aluno t_aluno;

int main() {

    t_aluno a1;
    scanf("%d", a1.ra);
    scanf("%s", a1.nome);
    scanf("%lf", a1.nota);

    printf("RA=%d Nome=%s Nota=%.2lf\n",
        *a1.ra, a1.nome, *a1.nota);

    return 0;
}
```

O usuário entrará com
os valores 123 Teste 8

O que será
impresso?

```
#include <stdio.h>

struct aluno {
    int *ra;
    char *nome;
    double *nota;
};
typedef struct aluno t_aluno;

int main() {

    t_aluno a1;
    scanf("%d", a1.ra);
    scanf("%s", a1.nome);
    scanf("%lf", a1.nota);

    printf("RA=%d Nome=%s Nota=%.2lf\n",
        *a1.ra, a1.nome, *a1.nota);

    return 0;
}
```

A memória para o ra, nome e nota não foi alocada! Apenas temos ponteiros (e que estão com valor indefinido!)

O usuário entrará com os valores 123 Teste 8

O que será impresso?

Erro!

```
#include <stdio.h>
#include <stdlib.h>
```

```
struct aluno {
    int *ra;
    char *nome;
    double *nota;
};
typedef struct aluno t_aluno;
```

```
int main() {
    t_aluno a1;
    a1.ra = malloc(sizeof(int));
    a1.nome = malloc(sizeof(char) * 100);
    a1.nota = malloc(sizeof(double));
    scanf("%d", a1.ra);
    scanf("%s", a1.nome);
    scanf("%lf", a1.nota);
    printf("RA=%d Nome=%s Nota=%.2lf\n",
        *a1.ra, a1.nome, *a1.nota);
    return 0;
}
```

O usuário entrará com os
valores 123 Teste 8

O que será
impresso?

```
#include <stdio.h>
#include <stdlib.h>

struct aluno {
    int *ra;
    char *nome;
    double *nota;
};
typedef struct aluno t_aluno;

int main() {
    t_aluno a1;
    a1.ra = malloc(sizeof(int));
    a1.nome = malloc(sizeof(char) * 100);
    a1.nota = malloc(sizeof(double));
    scanf("%d", a1.ra);
    scanf("%s", a1.nome);
    scanf("%lf", a1.nota);
    printf("RA=%d Nome=%s Nota=%.2lf\n",
        *a1.ra, a1.nome, *a1.nota);
    return 0;
}
```

O usuário entrará com os
valores 123 Teste 8

O que será
impresso?

RA=123 Nome=Teste Nota=8.00

Podemos ter vetores de estruturas também!

- Por exemplo: `#include <stdio.h>`

```
typedef struct aluno t_aluno;
struct aluno {
    int ra;
    double nota;
};

int main() {

    t_aluno alunos[3];

    return 0;
}
```

```
#include <stdio.h>
```

```
typedef struct aluno t_aluno;  
struct aluno {  
    int ra;  
    double nota;  
};
```

```
int main() {  
    t_aluno alunos[3];  
    int i;  
    for (i = 0; i < 3; i++) {  
        alunos[i].ra = i+1;  
        alunos[i].nota = i*i;  
    }  
  
    for (i = 0; i < 3; i++)  
        printf("RA=%d Nota=%.11f\n",  
            alunos[i].ra, alunos[i].nota);  
  
    return 0;  
}
```

O que será
impresso?

```
#include <stdio.h>
```

```
typedef struct aluno t_aluno;  
struct aluno {  
    int ra;  
    double nota;  
};
```

```
int main() {  
    t_aluno alunos[3];  
    int i;  
    for (i = 0; i < 3; i++) {  
        alunos[i].ra = i+1;  
        alunos[i].nota = i*i;  
    }  
  
    for (i = 0; i < 3; i++)  
        printf("RA=%d Nota=%.11f\n",  
            alunos[i].ra, alunos[i].nota);  
  
    return 0;  
}
```

O que será
impresso?

```
RA=1 Nota=0.0  
RA=2 Nota=1.0  
RA=3 Nota=4.0
```


Alocação dinâmica de estruturas

Podemos alocar estruturas dinamicamente também!

- Por exemplo:

```
typedef struct aluno t_aluno;  
struct aluno {  
    int ra;  
    char *nome;  
    double nota;  
};
```



```
t_aluno *a1;  
a1 = malloc(sizeof(t_aluno));
```

```
t_aluno *a2 = malloc(sizeof(t_aluno));
```

Acesso a membros de um ponteiro para estrutura

- Para acessar membros de um ponteiro para estrutura, temos duas alternativas:

1) Resolver ponteiro e acessar com o **ponto**:

```
t_aluno *a1 = malloc(sizeof(t_aluno));  
(*a1).ra = 123;
```

Acesso a membros de um ponteiro para estrutura

- Para acessar membros de um ponteiro para estrutura, temos duas alternativas:

1) Resolver ponteiro e acessar com o **ponto**:

```
t_aluno *a1 = malloc(sizeof(t_aluno));  
(*a1).ra = 123;
```

2) Utilizar o operador “->”:

```
t_aluno *a1 = malloc(sizeof(t_aluno));  
a1->ra = 123;
```

```
#include <stdio.h>
#include <stdlib.h>
```

```
typedef struct disciplina t_disciplina;
struct disciplina {
    int cod;
    char *nome;
    int creditos;
};
```

O que será impresso?

```
int main() {
    t_disciplina *pe = malloc(sizeof(t_disciplina));

    pe->cod = 555;
    pe->nome = "Prog Estruturada";
    pe->creditos = 4;

    printf("cod=%d nome=%s creditos=%.d\n",
        pe->cod, pe->nome, pe->creditos);

    return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>
```

```
typedef struct disciplina t_disciplina;
struct disciplina {
    int cod;
    char *nome;
    int credits;
};
```

```
int main() {
    t_disciplina *pe = malloc(sizeof(t_disciplina));

    pe->cod = 555;
    pe->nome = "Prog Estruturada";
    pe->credits = 4;

    printf("cod=%d nome=%s credits=%.d\n",
        pe->cod, pe->nome, pe->credits);

    return 0;
}
```

O que será impresso?

cod=555 nome=Prog Estruturada credits=4

Uma estrutura pode referenciar a si mesma!

- Será que podemos fazer isso então?

```
struct disciplina {  
    int cod;  
    char *nome;  
    int creditos;  
    struct disciplina requisito;  
};
```

Uma estrutura pode referenciar a si mesma!

- Será que podemos fazer isso então?

```
struct disciplina {
    int cod;
    char *nome;
    int creditos;
    struct disciplina *requisito;
};
```

NÃO !!! Isso torna a declaração recursiva!

Uma estrutura pode referenciar a si mesma!

- Será que podemos fazer isso então?

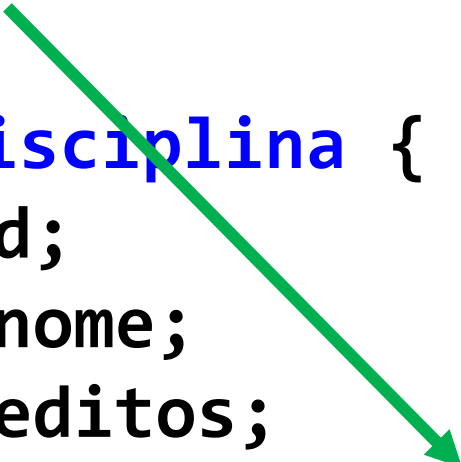
```
struct disciplina {
    int cod;
    char *nome;
    int creditos;
    struct disciplina *prerequisite;
};
```

Qual seria o tamanho de um *struct disciplina* em memória com a definição anterior? Seria infinito!

Uma estrutura pode referenciar a si mesma!

- Mas podemos referenciar usando ponteiros:

```
struct disciplina {  
    int cod;  
    char *nome;  
    int creditos;  
    struct disciplina *requisito;  
};
```



O que será impresso?

```
#include <stdio.h>
#include <stdlib.h>
struct disciplina {
    int cod;
    char *nome;
    int creditos;
    struct disciplina *requisito;
};
int main() {
    struct disciplina pe;
    pe.cod = 555;
    pe.nome = "Prog Estruturada";
    pe.creditos = 4;
    pe.requisito = malloc(sizeof(struct disciplina));
    pe.requisito->cod = 444;
    pe.requisito->nome = "Proc Informacao";
    pe.requisito->creditos = 4;
    pe.requisito->requisito = NULL;
    printf("Req: cod=%d nome=%s cred=%.d\n", pe.requisito->cod,
        pe.requisito->nome, pe.requisito->creditos);
    return 0;
}
```

O que será impresso?

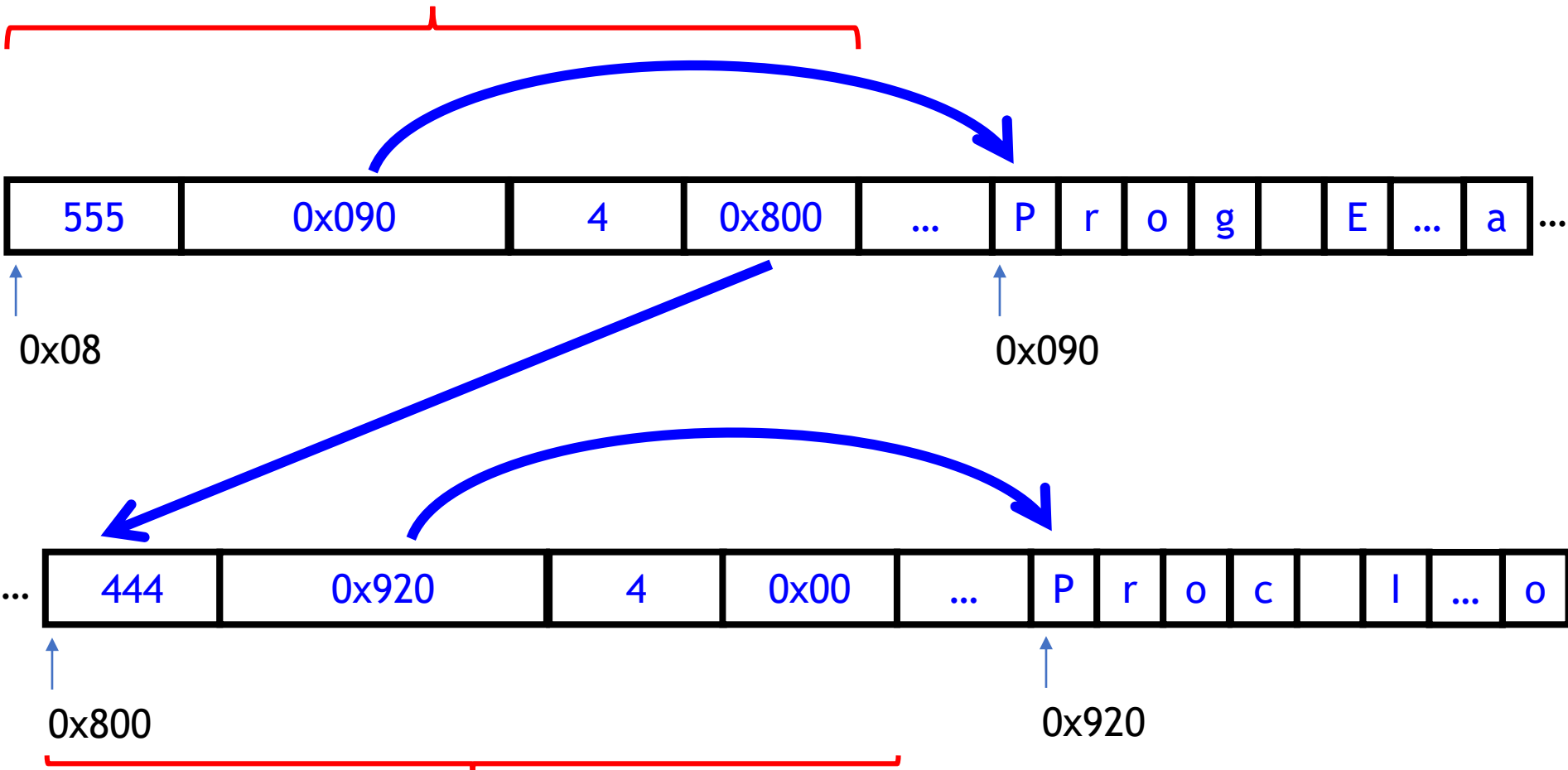
Req: cod=444 nome=Proc Informacao creditos=4

```
#include <stdio.h>
#include <stdlib.h>
struct disciplina {
    int cod;
    char *nome;
    int creditos;
    struct disciplina *requisito;
};

int main() {
    struct disciplina pe;
    pe.cod = 555;
    pe.nome = "Prog Estruturada";
    pe.creditos = 4;
    pe.requisito = malloc(sizeof(struct disciplina));
    pe.requisito->cod = 444;
    pe.requisito->nome = "Proc Informacao";
    pe.requisito->creditos = 4;
    pe.requisito->requisito = NULL;
    printf("Req: cod=%d nome=%s cred=%.d\n", pe.requisito->cod,
        pe.requisito->nome, pe.requisito->creditos);
    return 0;
}
```

Estrutura na memória

```
struct disciplina pe;
```



```
pe.requisito = malloc(sizeof(struct disciplina));
```

Exercício 1

- Se tivermos diversas disciplinas que tem o mesmo requisito, há alguma forma de evitar que esse mesmo requisito seja alocado repetidamente?
- Implemente um programa com essa estratégia: considere o caso da disciplina **Programação Básica**, que é requisito para **Programação A** e para **Programação B**.

Exercício 2 (a)

- Implemente a seguinte função, que cria e retorna uma disciplina com os valores passados nos parâmetros:

```
t_disciplina cria_disciplina(int cod, char *nome, int cred);
```

Exercício 2 (a)

- Implemente a seguinte função, que cria e retorna uma disciplina com os valores passados nos parâmetros:

```
t_disciplina cria_disciplina(int cod, char *nome, int cred);
```

```
t_disciplina cria_disciplina(int cod, char *nome, int cred) {  
    t_disciplina disc;  
    disc.cod = cod;  
    disc.nome = nome;  
    disc.creditos = cred;  
    return disc;  
}
```


Exercício 2 (b)

- Implemente a seguinte função, que cria e retorna **um ponteiro para disciplina** com os valores passados por parâmetro:

```
t_disciplina* cria_disciplina(int cod, char *nome, int cred);
```

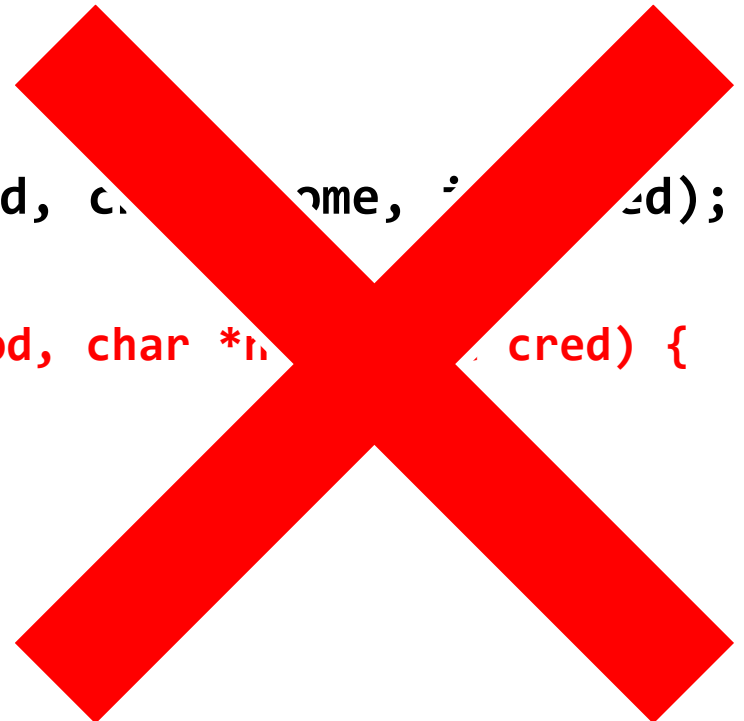
Exercício 2 (b)

Essa solução retorna
ponteiro para variável local!

ção, que cria e
disciplina com os
metro:

```
t_disciplina* cria_disciplina(int cod, char *nome, int cred);
```

```
t_disciplina* cria_disciplina(int cod, char *nome, int cred) {  
    t_disciplina disc;  
    disc.cod = cod;  
    disc.nome = nome;  
    disc.creditos = cred;  
    return &disc;  
}
```



Exercício 2 (b)

- Implemente a seguinte função, que cria e retorna **um ponteiro para disciplina** com os valores passados por parâmetro:

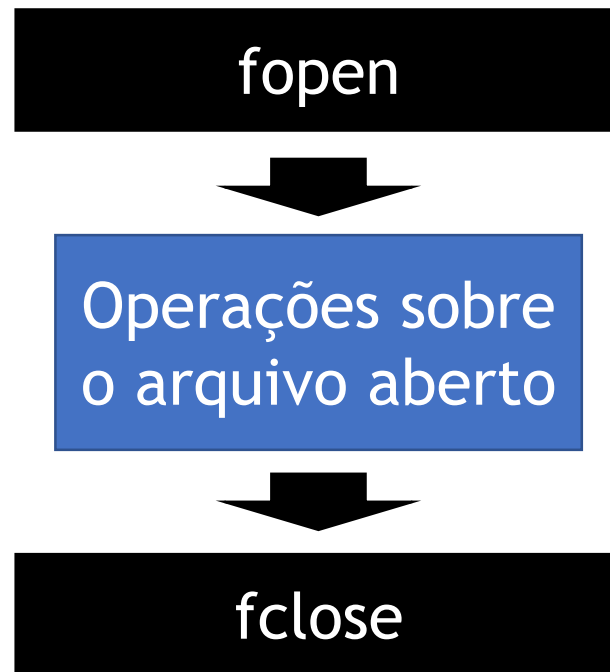
```
t_disciplina* cria_disciplina(int cod, char *nome, int cred);
```

```
t_disciplina* cria_disciplina(int cod, char *nome, int cred) {  
    t_disciplina *disc = malloc(sizeof(t_disciplina));  
    disc->cod = cod;  
    disc->nome = nome;  
    disc->creditos = cred;  
    return disc;  
}
```

Arquivos

Arquivos

- Para usar arquivos, precisamos primeiro abrir (**fopen**) e depois fechar os arquivos (**fclose**).



Abrir arquivo

- Usamos `fopen`, que recebe o modo de abertura do arquivo:

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



Modo	Descrição
<code>r</code>	Somente leitura.
<code>w</code>	Escrita. Primeiro zera o arquivo.
<code>a</code>	Escrita no final do arquivo.
<code>r+</code>	Leitura e escrita. Aponta para início do arquivo.
<code>w+</code>	Leitura e escrita. Primeiro zera o arquivo.
<code>a+</code>	Leitura e escrita. Leitura a partir do início e escrita no final do arquivo.

Abrir arquivo

- Usamos fopen, que recebe o modo de abertura do arquivo:

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



Para arquivos binários usamos os modos "rb", "wb", "ab", "rb+", "r+b", "wb+", "w+b", "ab+", "a+b"

Fechar arquivo

- Usamos fclose para fechar o arquivo:

```
int fclose(FILE *fp);
```


Abrindo e fechando o arquivo

- Portanto, para lidar com arquivos, usamos a seguinte estrutura básica:

```
#include <stdio.h>
```

```
int main() {
```

```
    FILE *arquivo;
```

```
    arquivo = fopen("teste.txt", "w");
```

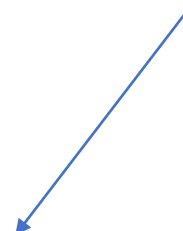
```
    // Uso do arquivo
```

```
    fclose(arquivo);
```

```
    return 0;
```

```
}
```

Abre para escrita



Escrita e leitura de
arquivos

Escrita e leitura de arquivos

- Para escrita/leitura com arquivos, usamos algumas funções semelhantes às aquelas que vimos para escrita/leitura no terminal:

`fputc/fgetc`

`fputs/fgets`

`fprintf/fscanf`

fprintf

```
#include <stdio.h>
```

```
int main() {
```

```
    FILE *arquivo;
```

```
    arquivo = fopen("teste.txt", "w");
```

```
    fprintf(arquivo, "Inicio do arquivo\n");
```

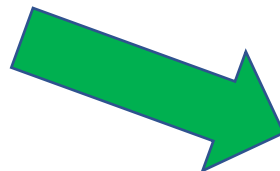
```
    int n = 507;
```

```
    fprintf(arquivo, "Valor de n = %d\n", n);
```

```
    fclose(arquivo);
```

```
    return 0;
```

```
}
```



Inicio do arquivo
Valor de n = 507

fprintf

E se o programa for executado
duas vezes?
Como ficará o arquivo teste.txt ?

```
#include <stdio.h>
```

```
int main() {
```

```
    FILE *arquivo;
```

```
    arquivo = fopen("teste.txt", "w");
```

```
    fprintf(arquivo, "Inicio do arquivo\n");
```

```
    int n = 507;
```

```
    fprintf(arquivo, "Valor de n = %d\n", n);
```

```
    fclose(arquivo);
```

```
    return 0;
```

```
}
```

fprintf

```
#include <stdio.h>
```

```
int main() {
```

```
FILE *arquivo;
```

```
arquivo = fopen("teste.txt", "w");
```

```
fprintf(arquivo, "Inicio do arquivo\n");
```

```
int n = 507;
```

```
fprintf(arquivo, "Valor de n = %d\n", n);
```

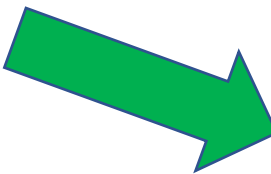
```
fclose(arquivo);
```

```
return 0;
```

```
}
```

E se o programa for executado
duas vezes?
Como ficará o arquivo teste.txt ?

O arquivo será sobrescrito, devido ao modo "w".



Inicio do arquivo
Valor de n = 507

fprintf

E agora? Assumindo que o arquivo não existe, como ficará o arquivo após o programa ser executado três vezes?

```
#include <stdio.h>
```

```
int main() {
```

```
FILE *arquivo;
```

```
arquivo = fopen("teste.txt", "a");
```

```
fprintf(arquivo, "Inicio do arquivo\n");
```

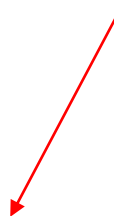
```
int n = 507;
```

```
fprintf(arquivo, "Valor de n = %d\n", n);
```

```
fclose(arquivo);
```

```
return 0;
```

```
}
```



fprintf

```
#include <stdio.h>
```

```
int main() {
```

```
FILE *arquivo;
```

```
arquivo = fopen("teste.txt", "a");
```

```
fprintf(arquivo, "Inicio do arquivo\n");
```

```
int n = 507;
```

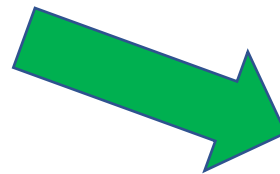
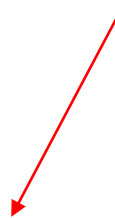
```
fprintf(arquivo, "Valor de n = %d\n", n);
```

```
fclose(arquivo);
```

```
return 0;
```

```
}
```

E agora? Assumindo que o arquivo não existe, como ficará o arquivo após o programa ser executado três vezes?



```
Inicio do arquivo
Valor de n = 507
Inicio do arquivo
Valor de n = 507
Inicio do arquivo
Valor de n = 507
```

Contudo novo é adicionado ao final do arquivo.

Leitura de arquivos

```
#include <stdio.h>

int main() {
    FILE *arquivo;
    arquivo = fopen("abc.txt", "r");

    char texto[100];

    fscanf(arquivo, "%s", texto);
    printf("%s\n", texto);

    fclose(arquivo);

    return 0;
}
```

```
#include <stdio.h>

int main() {
    FILE *arquivo;
    arquivo = fopen("abc.txt", "r");

    char texto[100];

    fgets(texto, 100, arquivo);
    printf("%s\n", texto);

    fclose(arquivo);

    return 0;
}
```

Há diferença na saída dos dois programas?

O `fscanf("%s", texto)` para de ler a string quando encontra um caractere espaço, mas o `fgets` não!!!

```
#include <stdio.h>

int main() {
    FILE *arquivo;
    arquivo = fopen("abc.txt", "r");

    char texto[100];

    fscanf(arquivo, "%s", texto);
    printf("%s\n", texto);

    fclose(arquivo);

    return 0;
}
```

Saída

Início

```
#include <stdio.h>

int main() {
    FILE *arquivo;
    arquivo = fopen("abc.txt", "r");

    char texto[100];

    fgets(texto, 100, arquivo);
    printf("%s\n", texto);

    fclose(arquivo);

    return 0;
}
```

Saída

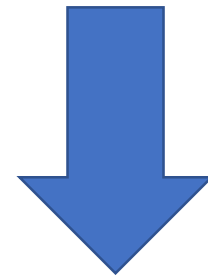
Início do arquivo

fscanf

```
#include <stdio.h>
```

```
int main() {  
    FILE *arquivo;  
    arquivo = fopen("teste.txt", "r");  
  
    char texto[100];  
  
    fscanf(arquivo, "%s", texto);  
    printf("%s\n", texto);  
  
    fscanf(arquivo, "%s", texto);  
    printf("%s\n", texto);  
  
    fclose(arquivo);  
  
    return 0;  
}
```

Início do arquivo
Valor de n = 507
Início do arquivo
Valor de n = 507
Início do arquivo
Valor de n = 507



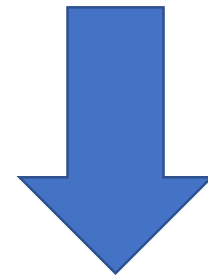
O que será impresso
no terminal?

fscanf

```
#include <stdio.h>
```

```
int main() {  
    FILE *arquivo;  
    arquivo = fopen("teste.txt", "r");  
  
    char texto[100];  
  
    fscanf(arquivo, "%s", texto);  
    printf("%s\n", texto);  
  
    fscanf(arquivo, "%s", texto);  
    printf("%s\n", texto);  
  
    fclose(arquivo);  
  
    return 0;  
}
```

Início do arquivo
Valor de n = 507
Início do arquivo
Valor de n = 507
Início do arquivo
Valor de n = 507



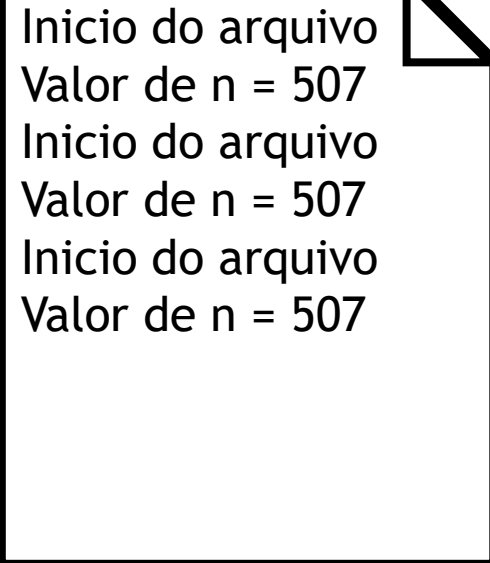
O que será impresso
no terminal?

Início
do

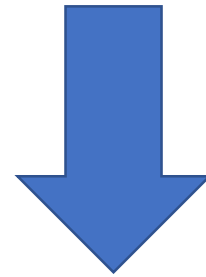
fgets

```
#include <stdio.h>
```

```
int main() {  
    FILE *arquivo;  
    arquivo = fopen("teste.txt", "r");  
  
    char texto[100];  
  
    fgets(texto, 100, arquivo);  
    printf("%s", texto);  
  
    fgets(texto, 100, arquivo);  
    printf("%s", texto);  
  
    fclose(arquivo);  
  
    return 0;  
}
```



Início do arquivo
Valor de n = 507
Início do arquivo
Valor de n = 507
Início do arquivo
Valor de n = 507

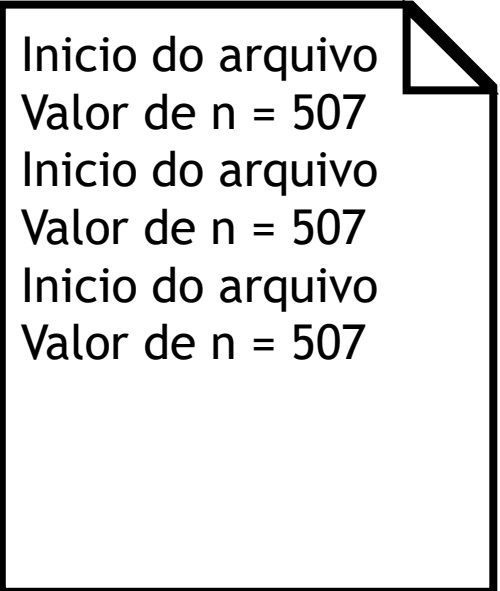


**O que será impresso
no terminal?**

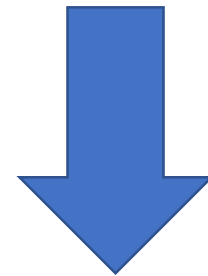
fgets

```
#include <stdio.h>
```

```
int main() {  
    FILE *arquivo;  
    arquivo = fopen("teste.txt", "r");  
  
    char texto[100];  
  
    fgets(texto, 100, arquivo);  
    printf("%s", texto);  
  
    fgets(texto, 100, arquivo);  
    printf("%s", texto);  
  
    fclose(arquivo);  
  
    return 0;  
}
```



Início do arquivo
Valor de n = 507
Início do arquivo
Valor de n = 507
Início do arquivo
Valor de n = 507



**O que será impresso
no terminal?**

Início do arquivo
Valor de n = 507

Misturando leitura e escrita

```
#include <stdio.h>

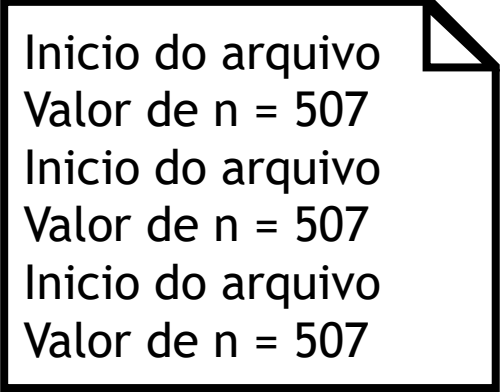
int main() {
    FILE *arquivo;
    arquivo = fopen("teste.txt", "a+");

    char texto[100];
    fscanf(arquivo, "%s", texto);
    printf("%s", texto);

    fprintf(arquivo, "01a 01a\n");

    fclose(arquivo);

    return 0;
}
```



Início do arquivo
Valor de n = 507
Início do arquivo
Valor de n = 507
Início do arquivo
Valor de n = 507



O que será impresso no terminal e escrito no arquivo?

Misturando leitura e escrita

```
#include <stdio.h>


int main() {
    FILE *arquivo;
    arquivo = fopen("teste.txt", "a+");

    char texto[100];
    fscanf(arquivo, "%s", texto);
    printf("%s", texto);

    fprintf(arquivo, "Ola Ola\n");

    fclose(arquivo);

    return 0;
}
```




Inicio do arquivo
Valor de n = 507
Inicio do arquivo
Valor de n = 507
Inicio do arquivo
Valor de n = 507



O que será impresso no terminal e escrito no arquivo?

Inicio

Arquivo não foi alterado!



Inicio do arquivo
Valor de n = 507
Inicio do arquivo
Valor de n = 507
Inicio do arquivo
Valor de n = 507

Misturando leitura e escrita

- Não execute operações de escrita e leitura intercaladas sem antes reposicionar o fluxo (stream);
- Uma forma de lidar com esse problema é fechando o arquivo e abrindo novamente. Nesse caso não há necessidade de usar um modo escrita+leitura como o “a+”.

Misturando leitura e escrita

```
#include <stdio.h>

int main() {
    FILE *arquivo;
    arquivo = fopen("teste.txt", "r");

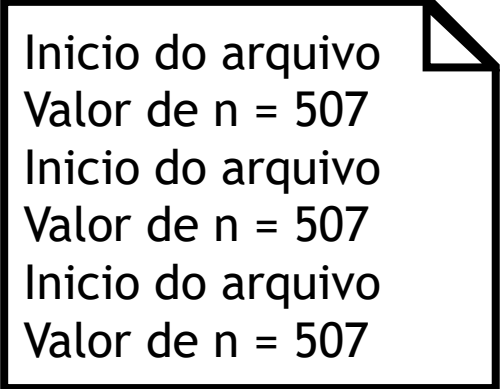
    char texto[100];
    fscanf(arquivo, "%s", texto);
    printf("%s", texto);

    fclose(arquivo);
    arquivo = fopen("teste.txt", "a");

    fprintf(arquivo, "Ola Ola\n");

    fclose(arquivo);

    return 0;
}
```



Inicio do arquivo
Valor de n = 507
Inicio do arquivo
Valor de n = 507
Inicio do arquivo
Valor de n = 507



O que será impresso no terminal e escrito no arquivo?

Misturando leitura e escrita

```
#include <stdio.h>

int main() {
    FILE *arquivo;
    arquivo = fopen("teste.txt", "r");


    char texto[100];
    fscanf(arquivo, "%s", texto);
    printf("%s", texto);

    fclose(arquivo);
    arquivo = fopen("teste.txt", "w");

    fprintf(arquivo, "Ola Ola\n");

    fclose(arquivo);

    return 0;
}
```




Inicio do arquivo
Valor de n = 507
Inicio do arquivo
Valor de n = 507
Inicio do arquivo
Valor de n = 507



O que será impresso no terminal e escrito no arquivo?

Inicio



Inicio do arquivo
Valor de n = 507
Inicio do arquivo
Valor de n = 507
Inicio do arquivo
Valor de n = 507
Ola Ola

Misturando leitura e escrita

- Outra maneira de lidar com esse problema é usar o **fseek** para reposicionar o fluxo.

Misturando leitura e escrita

```
#include <stdio.h>

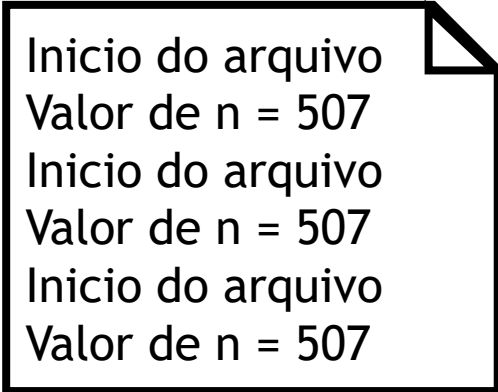
int main() {
    FILE *arquivo;
    arquivo = fopen("teste.txt", "a+");

    char texto[100];
    fscanf(arquivo, "%s", texto);
    printf("%s", texto);

    fseek(arquivo, 0, SEEK_SET);

    fprintf(arquivo, "Ola Ola\n");
    fclose(arquivo);

    return 0;
}
```



Inicio do arquivo
Valor de n = 507
Inicio do arquivo
Valor de n = 507
Inicio do arquivo
Valor de n = 507

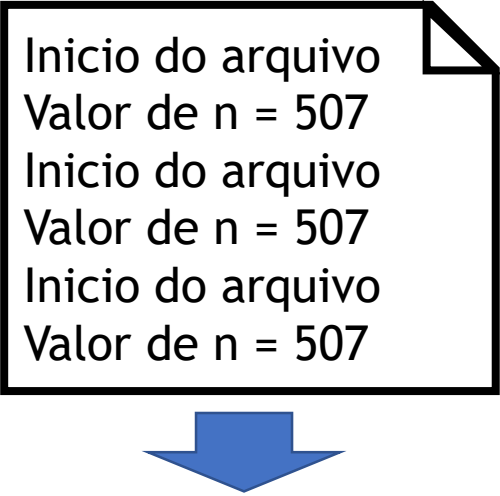


O que será impresso no terminal e escrito no arquivo?

Misturando leitura e escrita

```
#include <stdio.h>
```

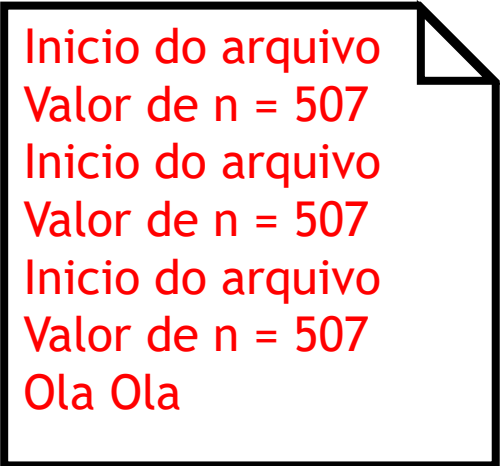
```
int main() {  
    FILE *arquivo;  
    arquivo = fopen("teste.txt", "a+");  
  
    char texto[100];  
    fscanf(arquivo, "%s", texto);  
    printf("%s", texto);  
  
    fseek(arquivo, 0, SEEK_SET);  
  
    fprintf(arquivo, "Ola Ola\n");  
    fclose(arquivo);  
  
    return 0;  
}
```



Inicio do arquivo
Valor de n = 507
Inicio do arquivo
Valor de n = 507
Inicio do arquivo
Valor de n = 507

O que será impresso no terminal e escrito no arquivo?

Inicio



Inicio do arquivo
Valor de n = 507
Inicio do arquivo
Valor de n = 507
Inicio do arquivo
Valor de n = 507
Ola Ola

Arquivos binários

Arquivos binários

- Podemos ler e escrever arquivos binários também;
- Dessa forma, é possível armazenar estruturas alocado em memória em um arquivo.

Arquivos binários

- Usaremos as funções **fwrite** e **fread** para escrita e leitura de arquivos binários:

Ponteiro para a região da memória que será escrita no arquivo.

Tamanho de cada elemento.

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

Quantidade de elementos.

Ponteiro para o arquivo aberto.

Arquivos binários

- Usaremos as funções `fwrite` e `fread` para escrita e leitura de arquivos binários:

Ponteiro para a região da memória
(já alocada!) que receberá os
dados do arquivo.

Tamanho de cada
elemento.

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

Quantidade
de elementos.

Ponteiro para o
arquivo aberto.

Exemplo

- Vamos armazenar uma lista de discos voadores (código e velocidade);
- Veremos um programa para armazenar o vetor com essa estrutura em um arquivo;
- Depois teremos um programa para ler o arquivo e mostrar a lista de discos voadores no terminal.

```
struct disco_voador {  
    int cod;  
    double velocidade;  
};
```

Escreve arquivo que
armazena a lista de
discos voadores.

```
#include <stdio.h>
#include <stdlib.h>
```

```
struct disco_voador {
    int cod;
    double velocidade;
};
```

```
int main() {
    struct disco_voador discos[3] = { {2, 4.5}, {6, 40.2},
        {300, 1750.25} };
```

```
FILE *arq = fopen("discos.bin", "wb");
```

```
fwrite(discos, sizeof(struct disco_voador), 3, arq);
```

```
fclose(arq);
```

```
return 0;
```

```
}
```

Lê arquivo e imprime lista de discos voadores.

```
#include <stdio.h>
#include <stdlib.h>

struct disco_voador {
    int cod;
    double velocidade;
};

void print_discos(struct disco_voador discos[], int n) {
    int i;
    for (i = 0; i < n; i++) {
        printf("-> [%d] [%.2lf]\n", discos[i].cod, discos[i].velocidade);
    }
}

int main() {
    struct disco_voador *discos = malloc(sizeof(struct disco_voador) * 3);

    FILE *arq = fopen("discos.bin", "rb");
    fread(discos, sizeof(struct disco_voador), 3, arq);
    fclose(arq);

    print_discos(discos, 3);
    return 0;
}
```

Curiosidade

- FILE é uma estrutura (trecho do stdio.h do MinGW):

```
typedef struct _iobuf
{
    char*  _ptr;
    int _cnt;
    char*  _base;
    int _flag;
    int _file;
    int _charbuf;
    int _bufsiz;
    char*  _tmpfname;
} FILE;
```

Comentário logo acima da
definição da estrutura:

“Some believe that nobody in their right mind should make use of the internals of this structure.”

Exercício 1 (a)

- Escreva um programa que leia o RA e a nota de n alunos. Armazene os dados em um vetor de struct;
- Após isso, guarde todos os dados em um arquivo.

Exercício 1 (b)

- Escreva um programa que leia o arquivo gerado no exercício anterior e mostre todos os dados dos alunos no terminal.

Exercício 1 (c)

- Altere o programa de modo que o vetor de alunos seja alocado dinamicamente com malloc. Compare essas duas alternativas:
 - Vetor de struct;
 - Vetor de ponteiros de struct.

Bibliografia básica

- PINHEIRO, F. A. C. Elementos de programação em C. Porto Alegre, RS: Bookman, 2012.
- FORBELLONE, A. L. V.; EBERSPACHER, H. F. Lógica de programação: a construção de algoritmos e estruturas de dados. 3ª edição. São Paulo, SP: Prentice Hall, 2005.
- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. Algoritmos: teoria e prática. 2ª edição. Rio de Janeiro, RJ: Campus, 2002.

Bibliografia complementar

- AGUILAR, L. J. Programação em C++: algoritmos, estruturas de dados e objetos. São Paulo, SP: McGraw-Hill, 2008.
- DROZDEK, A. Estrutura de dados e algoritmos em C++. São Paulo, SP: Cengage Learning, 2009.
- KNUTH D. E. The art of computer programming. Upper Saddle River, USA: Addison- Wesley, 2005.
- SEDGEWICK, R. Algorithms in C++: parts 1-4: fundamentals, data structures, sorting, searching. Reading, USA: Addison-Wesley, 1998.
- SZWARCFITER, J. L.; MARKENZON, L. Estruturas de dados e seus algoritmos. 3a edição. Rio de Janeiro, RJ: LTC, 1994.
- TEWNENBAUM, A. M.; LANGSAM, Y.; AUGENSTEIN, M. J. Estruturas de dados usando C. São Paulo, SP: Pearson Makron Books, 1995.