

Estruturas

Prof. Paulo Henrique Pisani

março/2022

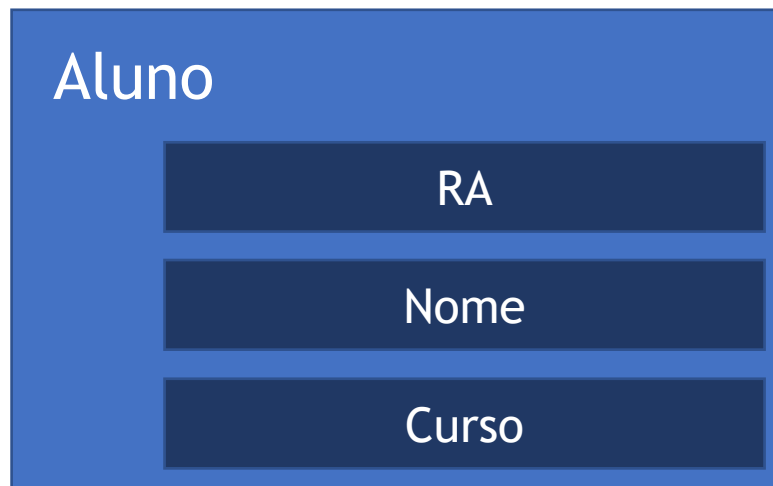
Tópicos

- Estruturas
- Alocação dinâmica de estruturas

Estruturas (struct)

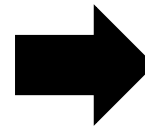
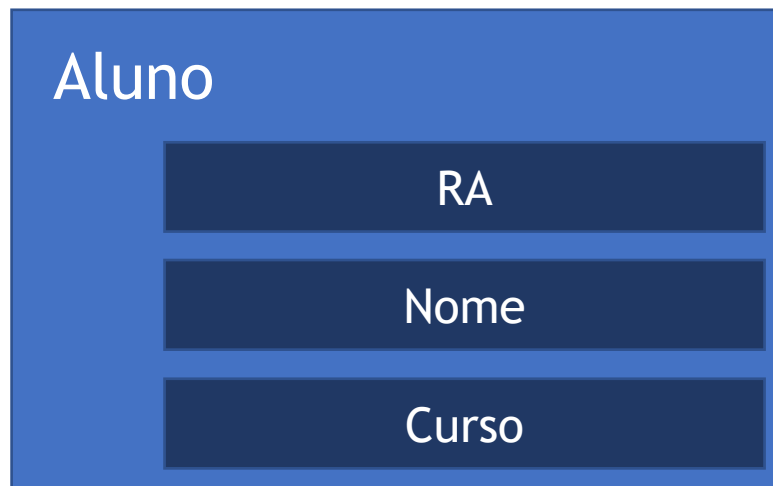
Estruturas

- 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**:

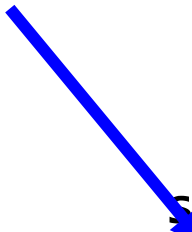
```
struct aluno aluno1;  
struct aluno aluno2, aluno3;
```



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.curso);
```

```
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 novo nome para o tipo de dados, e assim facilitar a declaração;

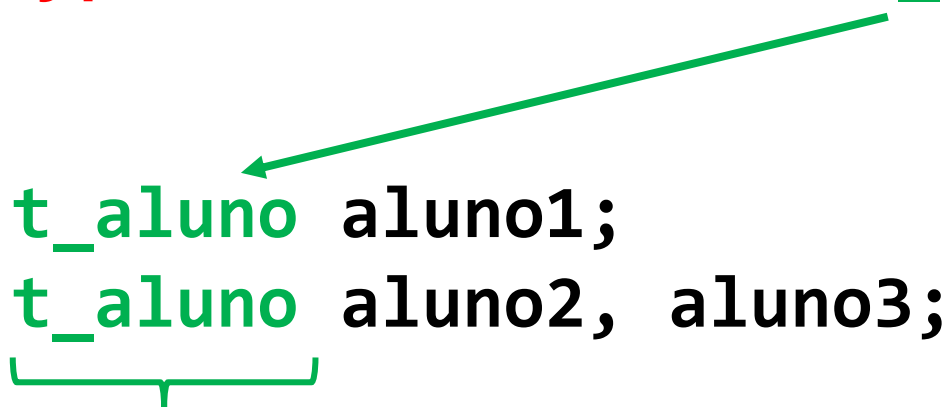
```
typedef struct aluno novo_nome;
```

Declarar variável do tipo estrutura

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

```
typedef struct aluno t_aluno;
```

```
t_aluno aluno1;  
t_aluno aluno2, aluno3;
```



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;  
};
```

Vamos escrever um programa que utilize essa estrutura t_ponto...

```
#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() {
```

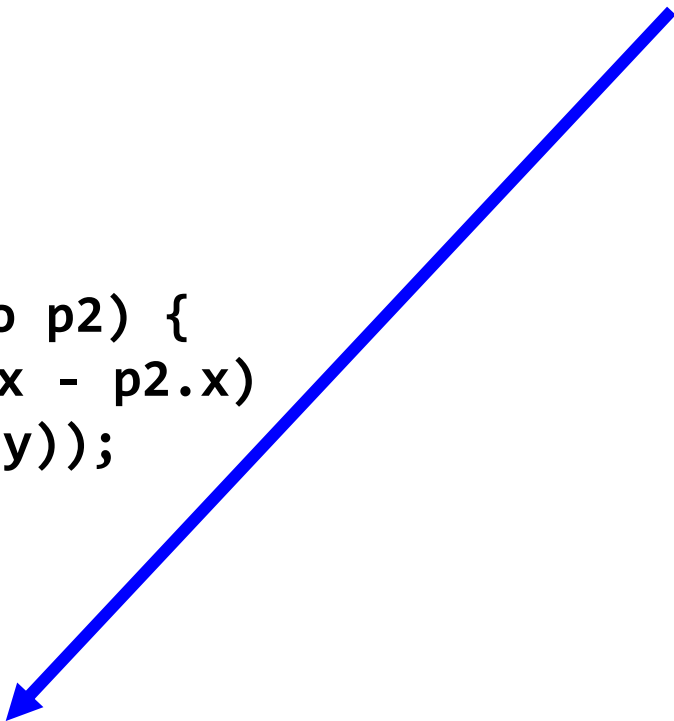
```
    t_ponto p1 = {3, 4}, p2 = {1, 2};
```

```
    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.

O usuário entrará com os valores 123 Teste 8

O que será impresso?

Erro!

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

```
#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);
    free(a1.ra);
    free(a1.nome);
    free(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);
    free(a1.ra);
    free(a1.nome);
    free(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 a partir de um ponteiro para estrutura, temos duas alternativas:

1) Dereferenciar 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 a partir de um ponteiro para estrutura, temos duas alternativas:

1) Dereferenciar 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 credits;
};
```

O que será impresso?

```
int main() {
    t_disciplina *pe = malloc(sizeof(t_disciplina));
    pe->cod = 555;
    pe->nome = "Programacao";
    pe->credits = 4;

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

    return 0;
}
```

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

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

O que será impresso?

cod=555 nome=Programacao credits=4

```
int main() {
    t_disciplina *pe = malloc(sizeof(t_disciplina));
    pe->cod = 555;
    pe->nome = "Programacao";
    pe->credits = 4;

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

    return 0;
}
```

Uma estrutura pode referenciar a si mesma!

- Será que podemos fazer isso então?

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



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 *prerequisito;  
};
```

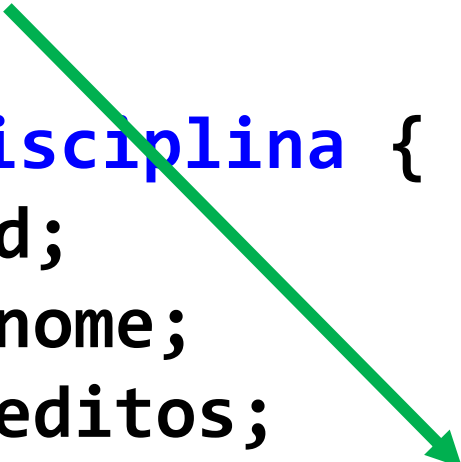


Qual seria o tamanho de um *struct disciplina* em memória com a definição anterior? Seria infinito!

Uma estrutura pode referenciar a si mesma!

- Podemos referenciar usando ponteiros:

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



O que será impresso?

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

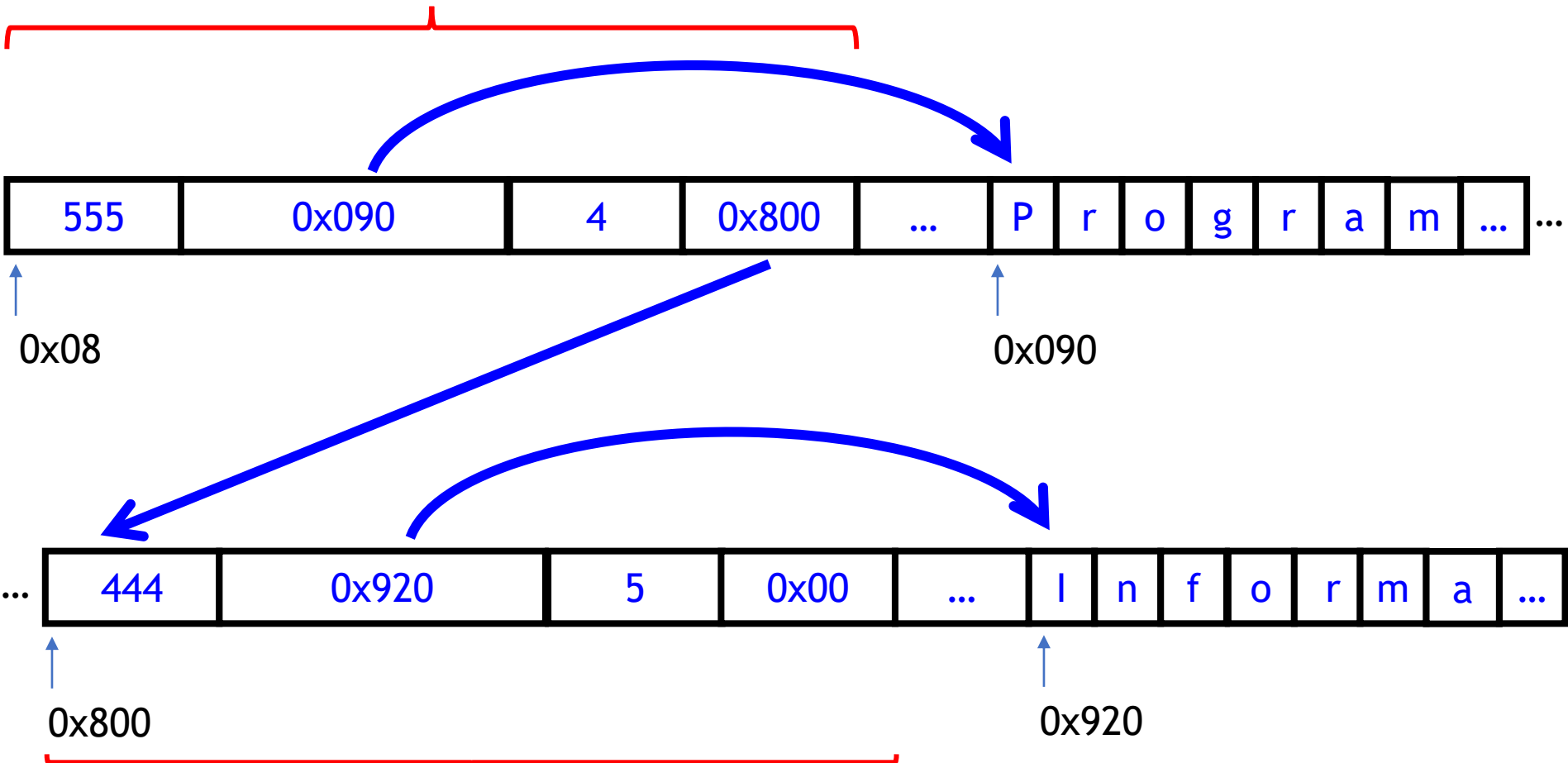
O que será impresso?

Req: cod=444 nome=Informacao creditos=5

```
#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 = "Programacao";
    pe.creditos = 4;
    pe.requisito = malloc(sizeof(struct disciplina));
    pe.requisito->cod = 444;
    pe.requisito->nome = "Informacao";
    pe.requisito->creditos = 5;
    pe.requisito->requisito = NULL;
    printf("Req: cod=%d nome=%s cred=%.d\n", pe.requisito->cod,
    pe.requisito->nome, pe.requisito->creditos);
    free(pe.requisito);
    return 0;
}
```

Estrutura na memória

struct disciplina



struct disciplina

Exercício 1 (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 1 (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;  
    disc.requisite = NULL;  
    return disc;  
}
```

Exercício 1 (b)

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

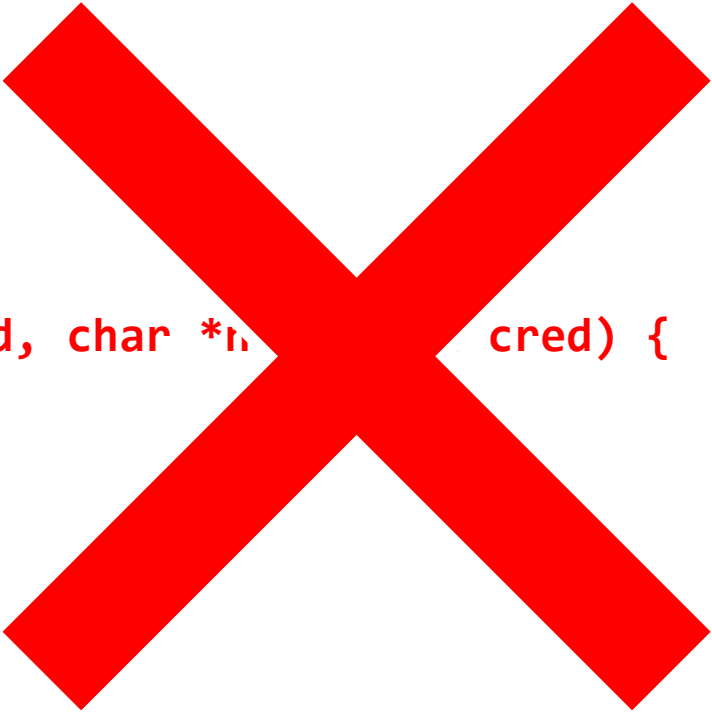
```
t_disciplina* cria_disciplina(int cod, char *nome, int cred);
```


Exercício 1 (b)

Essa solução retorna
ponteiro para variável local!

ção, que cria e
disciplina com os
netro:

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



Exercício 1 (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 *disc = malloc(sizeof(t_disciplina));  
    disc->cod = cod;  
    disc->nome = nome;  
    disc->creditos = cred;  
    return disc;  
}
```

Bibliografia básica

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

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. 3. ed. Rio de Janeiro, RJ: LTC, 1994.
- TENENBAUM, A. M.; LANGSAM, Y.; AUGENSTEIN, M. J. Estruturas de dados usando C. São Paulo, SP: Pearson Makron Books, 1995.