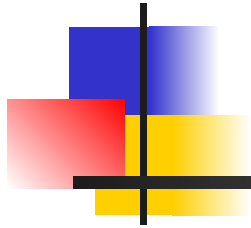


ESTRUTURAS COMPOSTAS



REGISTROS (Struct)

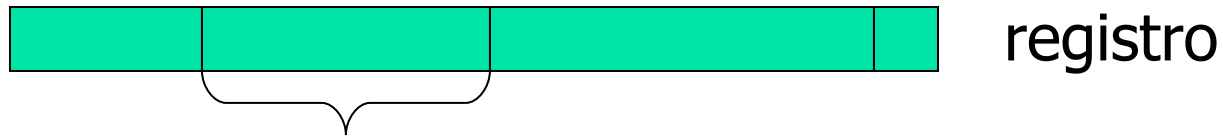
Introdução à Ciência da Computação I
Simone Senger Souza

Estruturas

- **Vetor e Matriz**
 - estruturas compostas homogêneas
- **Registro**
 - estruturas compostas heterogêneas

Registro

- Um registro (struct) é uma coleção de dados que podem ser de tipos diferentes.
-



campo

Cada campo pode ser de qualquer tipo (menos do tipo arquivo)

Registro - Exemplo

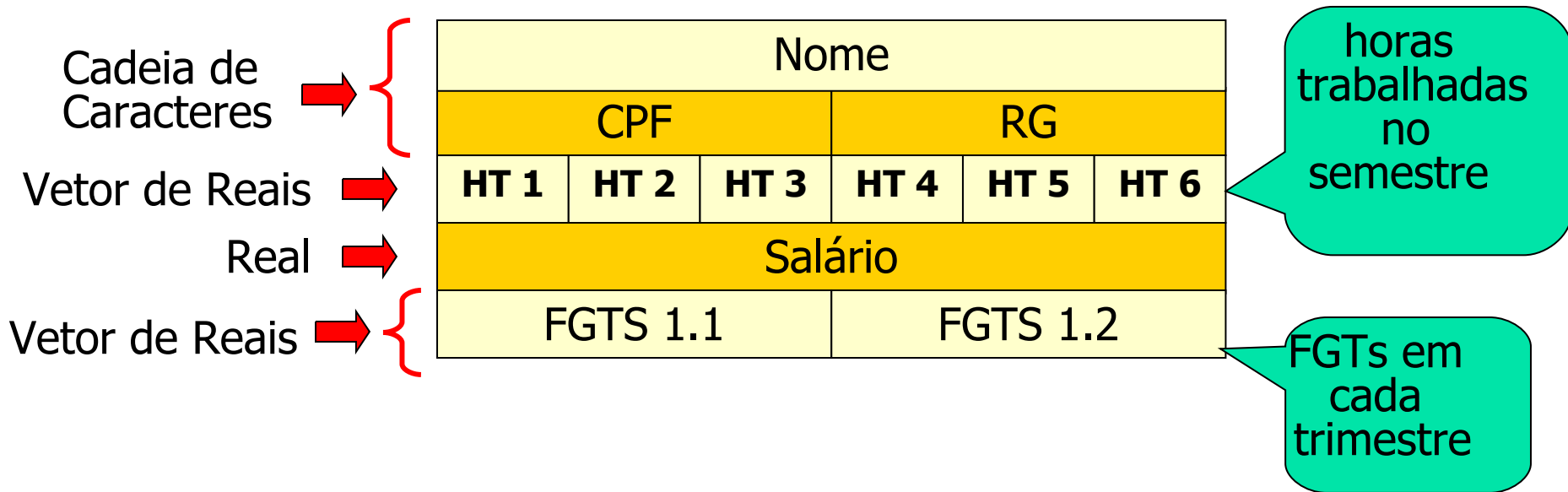
- Registro Aluno:
 - Nome
 - Número USP
 - Data de nascimento
 - Sexo
 - Endereço
 - Telefone
 - Email
 - ...

Registro - Exemplo

- Disciplina
 - Código da disciplina
 - Nome da disciplina
 - Ementa
 - Sistema de avaliação
 - Bibliografia
 - Cursos em que é oferecida
 - ...

Registro - Exemplo

- Registro de Pagamento



Registro

- Cada campo deve ter um nome e deve ser referenciado por este nome
- Não confundir com matriz e vetor onde todos os elementos são do mesmo tipo e são referenciados por um índice

Como definir uma variável tipo registro em C

```
typedef struct {  
    tipo1 campo1;  
    tipo2 campo2;  
    ...  
    tipon campon;  
} <nome do registro>;
```

Descrição
dos
campos

Não é uma variável,
mas um novo tipo de
dado!!

Como definir uma variável tipo registro em C

- **Outra forma de fazer (sem typedef):**

```
struct REGALUNO {  
    char numUSP[10];  
    char nome[30];  
    char sexo;  
    float renda_familiar;  
};
```

- Na declaração de variáveis coloca-se:

```
struct REGALUNO ALUNO;
```

Como definir uma variável tipo registro em C

- **Uma estrutura não pode conter uma instância de si mesma, exceto que seja um ponteiro:**

```
struct REGALUNO{
    char numUSP[10];
    char nome[30];
    char sexo;
    float renda_familiar;
    struct REGALUNO aluno; //ERRO!!!
    struct REGALUNO *aluno; //OK!!
};
```

Como definir uma variável tipo registro em C

- **Mais outra forma de fazer (sem typedef):**

```
struct {  
    char numUSP[10];  
    char nome[30];  
    char sexo;  
    float renda_familiar;  
} ALUNO1, ALUNO2;
```

- **Declaração de variáveis já foi realizada criando apenas **ALUNO1** e **ALUNO2****

Como definir uma variável tipo registro em C

- **No início do programa (em geral, fora de todas as funções):**

```
typedef struct {  
    char numUSP[10];  
    char nome[30];  
    char sexo;  
    float renda_familiar;  
} REGALUNO;
```

- **Na declaração de variáveis coloca-se:**

```
REGALUNO ALUNO;
```

Como Iniciar um Elemento de uma Variável do Tipo Struct

<nome-da-variável-tipo-struct> . <nome do campo>

REGALUNO aluno;

- **No Caso do Exemplo**

uma atribuição de valores poderia ser:

```
strcpy(ALUNO.numUSP, "1842655");  
strcpy(ALUNO.nome, "Pedro Henrique");  
ALUNO.sexo = 'M';  
ALUNO.Renda_familiar = 8.500,00 ;
```

Como Iniciar um Elemento de uma Variável do Tipo Struct

- **No Caso do Exemplo**
uma leitura de valores poderia ser:

```
printf("\n\nEntre com o numero USP:");  
scanf("%o",ALUNO.numUSP);  
printf("\n\nEntre com o nome:");  
scanf("%o[a-z A-Z]s",ALUNO.nome);  
//scanf("[^\n]s",ALUNO.nome);  
printf("\n\nEntre com o sexo(M/F:");  
scanf("%oc", &ALUNO.sexo);  
printf("\n\nEntre com a renda familiar:");  
scanf("%of", &ALUNO.renda_familiar);
```

Como Iniciar um Elemento de uma Variável do Tipo Struct

```
typedef struct{          struct fruta{
float re;                char nome[10];
float im;                char caloria;
} complexo;              };
```

```
complexo a[2][3] = {
    {{1.0, -0.1} , {2.0, 0.2}, {3.0,0.3}},
    {{4.0, -0.4} , {5.0, 0.5}, {6.0,0.6}}
};
```

```
struct fruta banana = {"banana", 100}, maca={0};
```

Manipulação da Variável Tipo Struct

- As variáveis do tipo struct podem ser manipuladas do mesmo modo que outros dados na memória
- Exemplo:

```
Renda=0.10*ALUNO.renda_familiar;
```


Exibição da Variável Tipo Struct

- As variáveis do tipo struct podem ser exibidas.
- Exemplo:

```
printf ("nome do aluno: %s", ALUNO.nome);
```

Estrutura dentro de estrutura

```
struct tipo_endereco
```

```
{  
    char rua [50];  
    int numero;  
    char bairro [20];  
    char cidade [30];  
    char sigla_estado [3];  
    char CEP [10];  
};
```

```
struct ficha_pessoal
```

```
{  
    char nome [50];  
    long int telefone;  
    struct tipo_endereco  
        endereco;  
};
```

Estrutura dentro de estrutura

```
typedef struct {
    int dia, mes, ano;
} REGDATA;
typedef struct{
    char nome[20];
    REGDATA nasc, contrato;
    float salario;
} REGF;
REGF dados[100];
```

```
dados[i].nasc.dia = 1;
dados[i].nasc.mes = 12;
dados[i].nasc.ano = 1980;
...
dados[i].contrato.dia = 5;
dados[i].contrato.mes = 3;
dados[i].contrato.ano = 2007;
```

Vetor de Registros

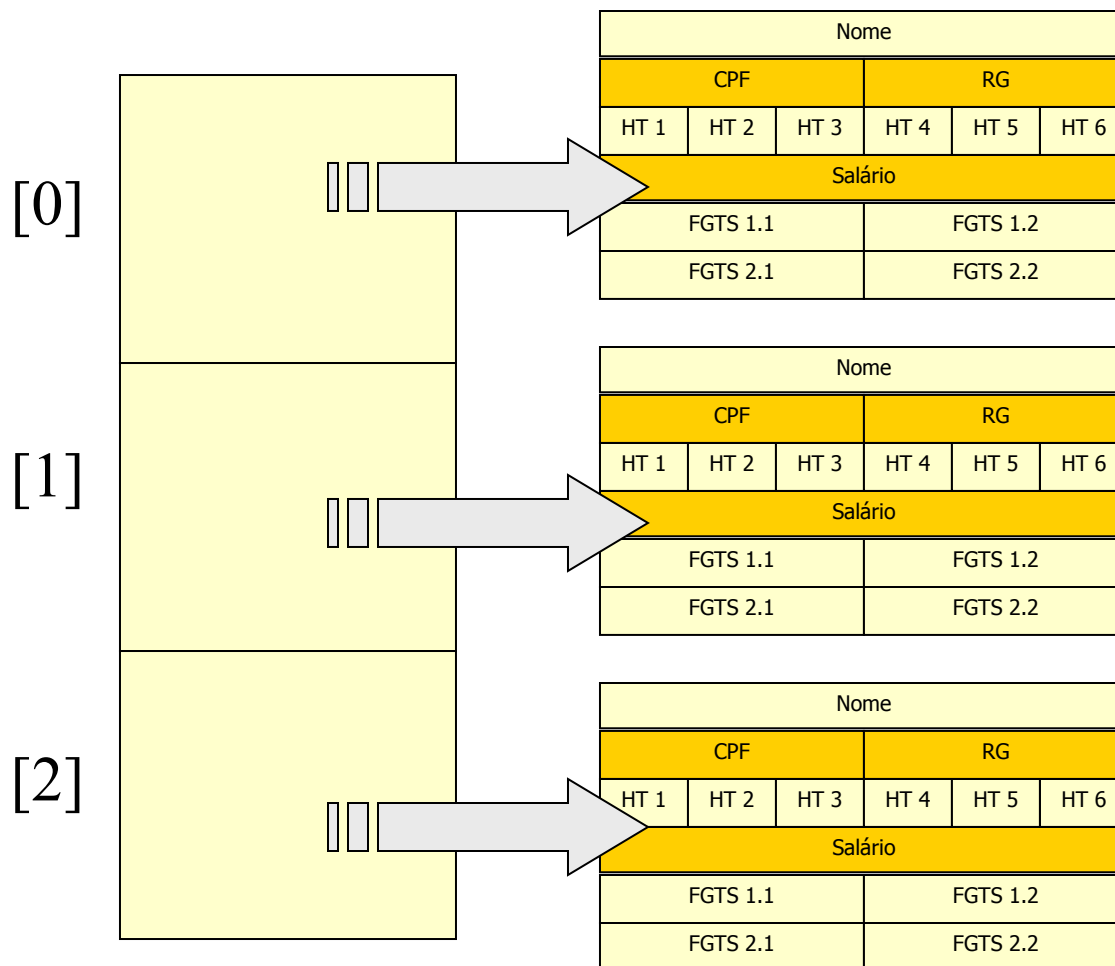


- Se, ao invés de uma única ficha do empregado, quisermos cadastrar várias fichas?

- **SOLUÇÃO**
- criar um
- vetor de
- registros !

Nome							
Nome							
Nome							
CPF			RG			HT 6	T 6
HT 1	HT 2	HT 3	HT 4	HT 5	HT 6		
Salário							
FGTS 1.1			FGTS 1.2				
FGTS 2.1			FGTS 2.2				

Vetor de Registros



Como definir um vetor de registros

- Na seção de declaração de tipo coloca-se:

```
typedef struct {  
    tipo1 campo1;  
    tipo2 campo2;  
    ...  
    tipon campon;  
} <nome do registro>;
```

```
<nome-do-registro> <nome-da-variavel [dimensao]>
```

No Exemplo

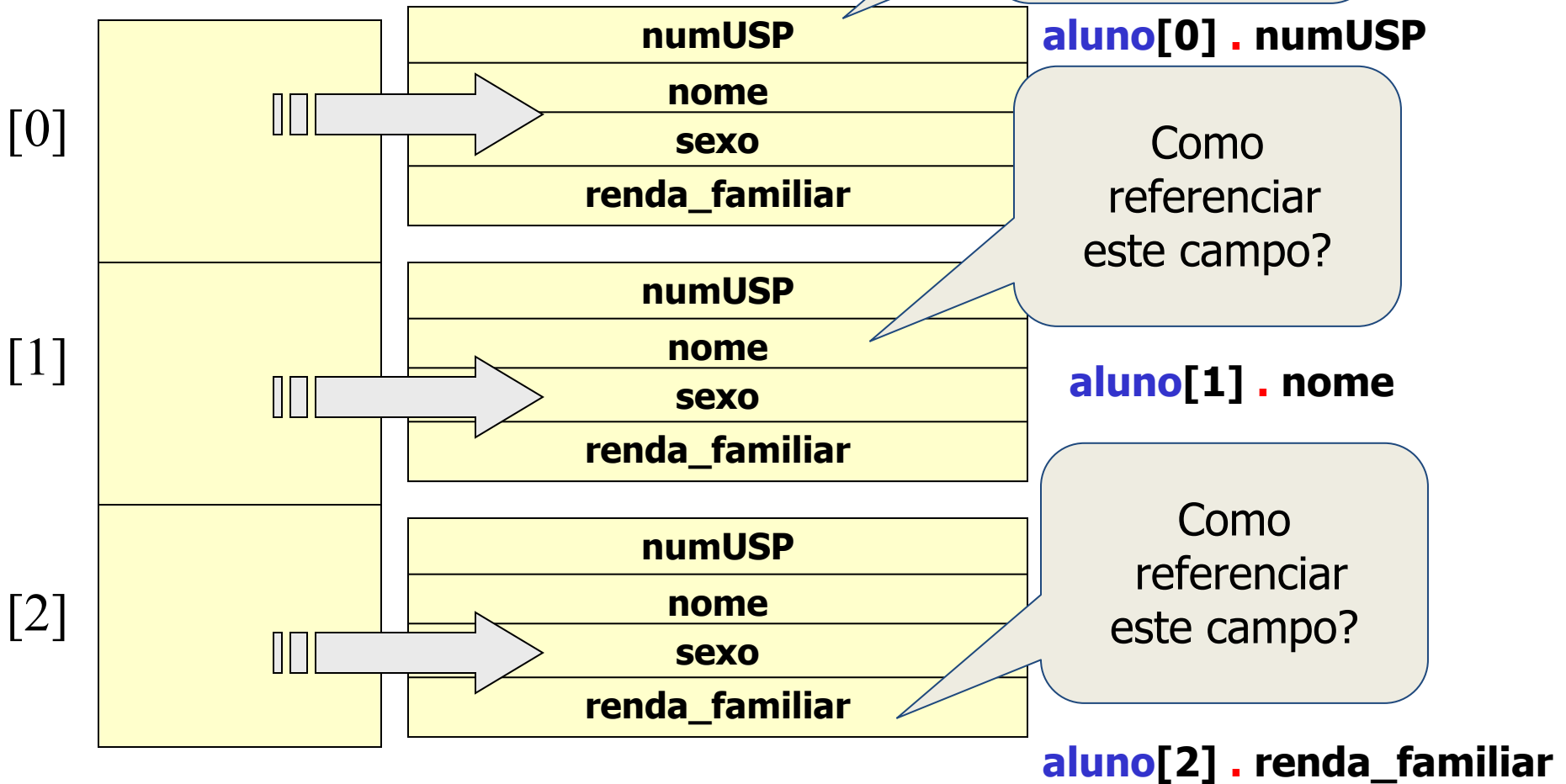
```
typedef struct {  
    char numUSP[10];  
    char nome[30];  
    char sexo;  
    float renda_familiar;  
} REGALUNO;
```

numUSP
nome
sexo
renda_familiar

- **REGALUNO** aluno[42];

Vetor de Registros

aluno



Estruturas e Funções

- Estruturas são passadas por valor

- Exemplo:

```
typedef struct{  
    char nome[20];  
    int idade;  
} REG;  
REG dados;
```

```
void ler(REG dados)
```

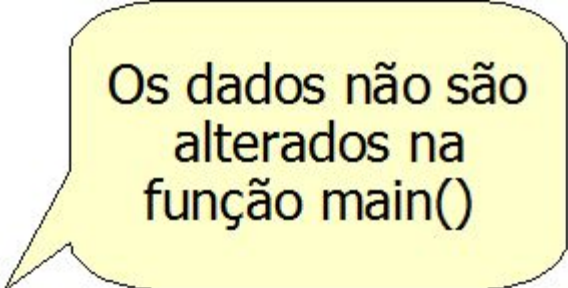
```
int main()
```

```
{
```

```
...
```

```
ler(dados);
```

```
}
```



Os dados não são alterados na função main()

Estruturas e Funções

- Ponteiros para estruturas: possibilitam a **passagem por referência**

- Exemplo:

```
typedef struct{  
    char nome[20];  
    int idade;  
} REG;  
REG dados;
```

```
void ler(REG *dados)
```

```
int main()
```

```
{
```

```
...
```

```
ler(&dados);
```

```
}
```

Estruturas e Funções

- Ponteiros para estruturas: possibilitam a **passagem por referência**

- Exemplo:

```
typedef struct{
    char nome[20];
    int idade;
} REG;
REG dados;
```

```
void ler(REG *dados)
{
    dados->nome = "Felipe";
    dados->idade = 16;
}
int main()
{
    ler (&dados);
}
```

Exemplo 1: Números Complexos

```
operationsCplx.c x complexo.h x main.c x ioCplx.c x
1 | #include "complexo.h"
2
3  int main()
4  {
5      int op;
6      srand(time(NULL));
7      op=menu();
8      complexo x, y, z;
9      x=iniciaCplx();
10     y=iniciaCplx();
11     imprimeCplx(x);
12     imprimeCplx(y);
13     switch (op){
14         case 1: z=somaCplx(x,y);
15                 printf("x+y=");
16                 imprimeCplx(z);
17                 break;
18
19         case 2: produtoCplx(&z,&x,&y);
20                 printf("x*y=");
21                 imprimeCplx(z);
22                 break;
23         default: printf("\nERRO: Opção Invalida");
24     }
25
26     return 0;
27
28 }
29
```

Exemplo 1: Números Complexos

operationsCplx.c x complexo.h x main.c x ioCplx.c x

```
1 | #include<stdio.h>
2 | #include<stdlib.h>
3 | #include<string.h>
4 | #include<time.h>
5 |
6 | #define MIN -1000
7 | #define MAX 1000
8 | #define precisao 100.0
9 | #define Aleatorio(MIN,MAX) ( ( MIN+rand()%(MAX-MIN+1) )/precisao)
10 |
11 | typedef struct {
12 |     float re;
13 |     float im;
14 | } complexo;
15 |
16 | int menu();
17 | complexo iniciaCplx();
18 | void imprimeCplx(complexo );
19 | complexo somaCplx(complexo, complexo );
20 | void produtoCplx(complexo *, complexo*, complexo*);
21 |
```

Exemplo 1: Números Complexos

```
operationsCplx.c ✕ complexo.h ✕ main.c ✕ ioCplx.c ✕
1  #include "complexo.h"
2
3  int menu()
4  {
5      int opcao;
6      printf("1-Soma complexos x e y"
7            "\n2-Produto complexos x e y\n0pção:");
8      scanf("%d",&opcao);
9      return opcao;
10 }
11
12 void imprimeCplx(complexo a)
13 {
14     printf("%.2f+%.2f\n", a.re,a.im);
15 }
16
```

Exemplo 1: Números Complexos

```
operationsCplx.c x complexo.h x main.c x ioCplx.c x
1 | #include "complexo.h"
2
3   complexo iniciaCplx()
4   {
5       complexo a;
6       a.re = Aleatorio(MIN,MAX);
7       a.im = Aleatorio(MIN,MAX);
8       imprimeCplx(a);
9       return a;
10  }
11
12  complexo somaCplx(complexo a, complexo b)
13  {
14      complexo c;
15      c.re = a.re + b.re;
16      c.im = a.im + b.im;
17      return c;
18  }
19
20  void produtoCplx(complexo *z, complexo *a, complexo *b)
21  {
22      z->re = (a->re)*(b->re) - (a->im)*(b->im);
23      z->im = (b->re)*(a->im) + (a->re)*(b->im);
24  }
25
```

Exemplo 2: Cadastro

```
main.c x empresa.h x cadastro.c x
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "empresa.h"
4
5  int main()
6  {
7      dados *funcionarios, maior;
8      int nFunc=0;
9
10     //Cadastra funcionarios
11     funcionarios = cadastro(&nFunc);
12     //Encontra funcionario com maior salario
13     maior = maiorSal(funcionarios, nFunc);
14     printf("Maior salário:");
15     imprime(maior);
16     //Encontra funcionario com menor salario
17     maior = maiorIdade(funcionarios, nFunc);
18     printf("Maior idade:");
19     imprime(maior);
20     //Imprime a media salarial
21     printf("Média salarios: %.2f", mediaSal(funcionarios, nFunc));
22
23     return 0;
24 }
25
```


Exemplo 2: Cadastro

```
main.c x empresa.h x cadastro.c x
1  #include<stdio.h>
2  #include<stdlib.h>
3  #include<assert.h>
4
5  typedef struct{
6      char rua[30];
7      int numero;
8  }endereco;
9
10 typedef struct{
11     char nome[30];
12     int idade;
13     float salario;
14     endereco |ender;
15 } dados;
16
17
18 dados *cadastro(int *);
19 dados maiorSal(dados *, int);
20 dados maiorIdade(dados *, int);
21 float mediaSal(dados *, int);
22 void imprime(dados);
23
```

Exemplo 2: Cadastro

```
main.c ✕ cadastro.c ✕ empresa.h ✕
1  #include "empresa.h"
2
3  dados *cadastro(int *qtd)
4  {
5      printf("Cadastro de dados");
6      dados *vetFunc;
7      vetFunc = NULL;
8      int op;
9      do{
10         *qtd = *qtd+1;
11         vetFunc = (dados *) realloc(vetFunc, *qtd*sizeof(dados));
12         assert(vetFunc!=NULL);
13         printf("\nNome:"); scanf("%[^\n]s", vetFunc[*qtd-1].nome);
14         printf("Idade:"); scanf("%d",&vetFunc[*qtd-1].idade);
15         printf("Salario:"); scanf("%f",&vetFunc[*qtd-1].salario);
16         printf("Rua:"); scanf("%s", vetFunc[*qtd-1].endFunc.nomeRua);
17         printf("Numero:"); scanf("%d",&vetFunc[*qtd-1].endFunc.numero);
18         printf("Digite 0 para sair");
19         scanf("%d",&op);
20     }while(op!=0);
21
22     return vetFunc;
23 }
```

main.c ✕

cadastro.c ✕

empresa.h ✕

```
25 dados maiorSal(dados *vetFunc, int qtd)
26 {
27     int i;
28     float maiorSal=-1;
29     dados maior;
30     for(i=0; i<qtd; i++){
31         if(maiorSal<vetFunc[i].salario){
32             maiorSal = vetFunc[i].salario;
33             maior = vetFunc[i];
34         }
35     }
36     return maior;
37 }
38
39
40 dados maiorIdade(dados *vetFunc, int qtd)
41 {
42     int i;
43     float maiorIdade=-1;
44     dados maior;
45     for(i=0; i<qtd; i++){
46         if(maiorIdade<vetFunc[i].idade){
47             maiorIdade=vetFunc[i].idade;
48             maior = vetFunc[i];
49         }
50     }
51     return maior;
52 }
```

Exemplo 2: Cadastro

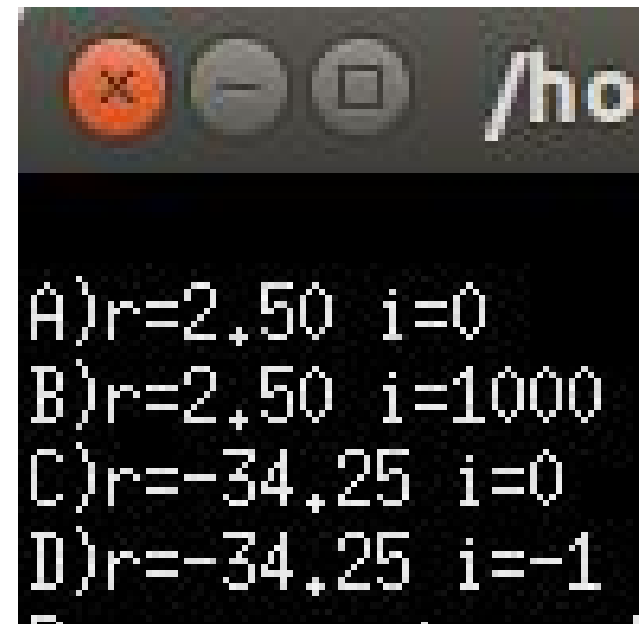
```
main.c ✕ cadastro.c ✕ empresa.h ✕
53
54 float mediaSal(dados *vetFunc, int qtd)
55 {
56     int i;
57     float soma = 0;
58
59     for(i=0; i<qtd; i++){
60         soma += vetFunc[i].salario;
61     }
62
63     return soma/qtd;
64
65 }
66
67 void imprime (dados func)
68 {
69     printf("%s %d %.2f %s %d\n", func.nome, func.idade, func.salario,
70         func.endFunc.nomeRua, func.endFunc.numero);
71 }
72
```

Union

- *Union* é um tipo de dados composto cujos membros compartilham o mesmo espaço de armazenamento.
- A quantidade de armazenamento ocupada será tão grande quanto o maior membro da *union*.
- Logo, *union* economiza espaço de armazenamento.

Union

```
Start here x unionExemplo1.c x
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  union real_ou_inteiro{
5      double r;
6      int i;
7  };
8
9  int main()
10 {
11
12     union real_ou_inteiro num;
13
14     num.r = 2.5;
15     printf("\nA)r=%.2f i=%d", num.r, num.i);
16     num.i = 1000;
17     printf("\nB)r=%.2f i=%d", num.r, num.i);
18     num.r = -34.25;
19     printf("\nC)r=%.2f i=%d", num.r, num.i);
20     num.i = -1;
21     printf("\nD)r=%.2f i=%d", num.r, num.i);
22 }
23
```



Union

Start here x unionExemplo1.c x

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  typedef struct{
5      char marca[30];
6      int ano;
7  }carro;
8
9  typedef struct{
10     char nome[10];
11     float preco;
12 }fruta;
13
14 typedef union{
15     carro c;
16     fruta f;
17 }fruta_ou_carro;
18
19 int main()
20 {
21
22     fruta_ou_carro x;
23     strcpy(x.c.marca, "FIAT");
24     x.c.ano = 2010;
25     printf("\nA)\nCarro: %s %d\nFruta:%s %.2f",x.c.marca, x.c.ano, x.f.nome, x.f.preco);
26
27     strcpy(x.f.nome, "banana");
28     x.f.preco = 4.99;
29     printf("\nB)\nCarro: %s %d\nFruta:%s %.2f",x.c.marca, x.c.ano, x.f.nome, x.f.preco);
30
31     return 0;
32 }
33
```

A)

Carro: FIAT 2010

Fruta:FIAT 0.00

B)

Carro: banana 2010

Fruta:banana 4.99