Estruturas (Structs)

Baseado em slides do Prof. Bruno Travençolo.
Adaptado pela Profa. Gina Oliveira

Variáveis

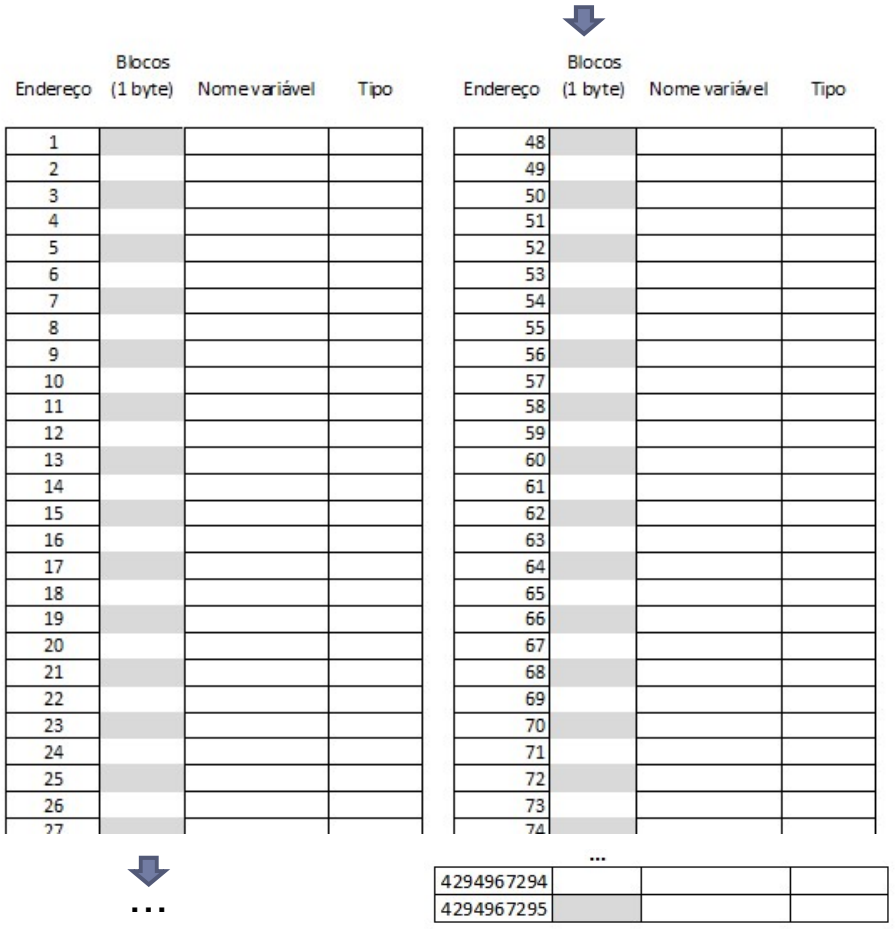
- As variáveis vistas até agora eram:
  - simples: definidas por tipos **int**, **float**, **double** e **char**;
  - compostas homogêneas (ou seja, do mesmo tipo): definidas por **array**.

- No entanto, a linguagem C permite que se criem novas estruturas a partir dos tipos básicos.
  - Struct

- Mas antes, vamos rever alguns conceitos sobre a memória alocada para as variáveis simples e arrays

Memória

- Podemos pensar na memória como uma sequência linear de bytes, sendo que cada byte possui um endereço.
- A memória é limitada
- O Sistema operacional (SO) gerencia a memória
- Vale observar que esse esquema é usado para entender alocação. A que ocorre de fato depende de vários fatores: so, compiliador, otimização, etc.



Exemplo

- Indique, usando um mapa de memória similar ao slide anterior, um possível estado da memória ao fim da execução do seguinte programa.
  - Indique quantos bytes as variáveis declaradas ocupam
  - Continue o programa para mostrar a soma da quantidade de memória ocupada (use a variável tamanho_total)

```
int  idade;
char nome[10] = "Maria";
double peso, altura;
int casada;
float grau_miopia[2];
unsigned int tamanho_total;

altura = 1.65;
peso = 70;
casada = 0; // false
grau_miopia[0] = 2.75; // olho esquerdo
grau_miopia[1] = 3; // olho direito
```

Exemplos de alocação

- `char nome[10] = "Maria"`

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1	'M'	nome[0]	char
2	'a'	nome[1]	char
3	'r'	nome[2]	char
4	'i'	nome[3]	char
5	'a'	nome[4]	char
6	'\0'	nome[5]	char
7	lx	nome[6]	char
8	lx	nome[7]	char
9	lx	nome[8]	char
10	lx	nome[9]	char
11			

São alocados 10 bytes de memória do tipo char
(o tipo char ocupa 1 byte)

*** obs: na verdade as posições de 7 a 10 são inicializadas com \0, mas esse comportamento não é padrão em comandos como gets e strcpy

Exemplos de alocação

- `double peso = 70;`

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1			
2	70	peso	double
3			
4			
5			
6			
7			
8			
9			
10			

Um tipo double ocupa 8 bytes. Assim, independente do valor atribuído a variável peso (10, 20, 1milhão), serão 8 bytes ocupados.

Exemplos de alocação

- `double altura = 1.65;`

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1			
2	1.65	altura	double
3			
4			
5			
6			
7			
8			
9			
10			

Um tipo double ocupa 8 bytes. Assim, independente do valor atribuído a variável altura, serão 8 bytes ocupados.

Exemplos de alocação

- `int casada = 0;`

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1			
2	0	casada	int
3			
4			
5			
6			
7			
8			
9			
10			

Um tipo int ocupa 4 bytes. Assim, independente do valor atribuído a variável casada, serão 4 bytes ocupados.

Exemplos de alocação

- `unsigned int tamanho_total;`

Endereço	Blocos (1 byte)	Nome variável	Tipo
0 / NULL	indefinido	----	----
1			
2	lx	altura	double
3			
4			
5			
6			
7			
8			
9			
10			

Um tipo unsigned int também ocupa 4 bytes. Como a variável não foi inicializada os 4 bytes no início estão com lixo.

Exemplos de alocação

- `float grau_miopia[2];`
- `grau_miopia[0] = 3; grau_miopia[1]=2.5;`

Endereço	Blocos (1 byte)	Nome variável	Tipo
47	3	grau_miopia[0]	float
48			
49			
50			
51	2.5	grau_miopia[1]	float
52			
53			
54			
55			
56			

Cada elemento do vetor é do tipo Float e ocupa 4 bytes. O endereço de grau_miopia[0] é 47 e o de grau_miopia[1] é 51.

Por se tratar de um vetor, as posições dos 2 elementos são contíguas na memória.

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

int main()
{
    int  idade;
    char nome[10] = "Maria";
    double peso, altura;
    int casada;
    float grau_miopia[2];
    unsigned int tamanho_total;

    altura = 1.65;
    peso = 70;
    casada = 0; // false
    grau_miopia[0] = 2.75; // olho esquerdo
    grau_miopia[1] = 3; // olho direito

    // obs: o símbolo \ serve para continuar um comando em
    // uma outra linha.
    tamanho_total = sizeof(nome) + sizeof(altura) + sizeof(peso)+  \
                    sizeof(casada)+sizeof(grau_miopia)+sizeof(idade) + \
                    sizeof(tamanho_total);
    printf("\n Tamanho em bytes ocupado: %u", tamanho_total);


    return 0;
}
```

Estruturas

- A linguagem C permite que se criem novas estruturas a partir dos tipos básicos.
  - Struct

- Ao contrário dos vetores, que são estruturas de dados homogêneas, a **struct** permite que sejam criadas estruturas heterogêneas (diferentes tipos de dados).

Estruturas

- Uma estruturas pode ser vista como um novo tipo de dado, que é formado por composição de outros tipos.
  - Pode ser declarada em qualquer escopo (local ou global)
  - Ela é declarada da seguinte forma:

```
struct nomestruct {
 tipo1 campo1;
 tipo2 campo2;
 ...
 tipoN campoN;
};
```

Estruturas

- Uma estrutura pode ser vista como um agrupamento de dados.
  - Ex.: cadastro de pessoas.

```
int  idade;
char nome[10] = "Maria";
double peso, altura;
int estado_civil;
float grau_miopia[2];
```

Todas essas informações são da mesma pessoa – podemos agrupá-las. Isso facilita também lidar com dados de outras pessoas no mesmo programa

Estruturas

- O uso de estruturas facilita na manipulação dos dados do programa. Imagine declarar 4 cadastros, para 4 pacientes diferentes:

```
char nome1[10], nome2[10], nome3[10], nome4[10];
int idade1, idade2, idade3, idade4;
double grau_miopia1[2],grau_miopia2[2],grau_miopia3[2],grau_miopia4[2];
```

Ou

```
char nome1[4][10];
int idade[4];
double grau_miopia1[4][2];
```

Estruturas

- Uma estrutura pode ser vista como um agrupamento de dados.
  - Ex.: cadastro de pessoas.

```
int  idade;
char nome[10] = "Maria";
double peso;
double altura;
int estado_civil;
float grau_miopia[2];
```

```
struct dados_pacientes {
    int  idade;
    char nome[10];
    double peso;
    double altura;
    int estado_civil;
    float grau_miopia[2];
};
```

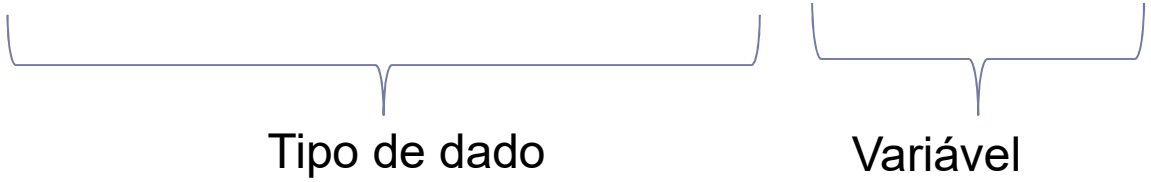
Estruturas – declaração de variáveis

- Uma vez definida a estrutura, uma variável pode ser declarada de modo similar aos tipos já existente:
  - `struct dados_pacientes paciente1;`

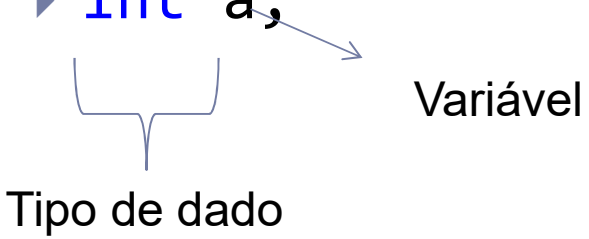
- Obs: por ser um tipo definido pelo programador, usa-se a palavra **struct** antes do tipo da nova variável

Estruturas – declaração de variáveis

- Obs: por ser um tipo definido pelo programador, usa-se a palavra **struct** antes do tipo da nova variável
  - `struct dados_pacientes paciente1;`



- Compare com a declaração de uma variável inteira:
  - `int a;`



Estruturas

- Utilizando uma estrutura, o mesmo pode ser feito da seguinte maneira:
  - Declarando variáveis para 3 pacientes

```
struct dados_pacientes {
   int  idade;
   char nome[10];
   double peso;
   double altura;
   int estado_civil;
   float grau_miopia[2];
};

struct dados_pacientes paciente1, paciente2,  \
                       paciente3, paciente4;

struct dados_pacientes cliente_especial;
```

Exercício

- Declare uma estrutura capaz de armazenar o número inteiro e 3 notas (float) para um dado aluno.

Exercício - Solução

```
struct aluno {
  int num_aluno;
  float nota1;
  float nota2;
  float nota3;
};
```

ou

```
struct aluno {
  int num_aluno;
  float nota1, nota2, nota3;
};
```

ou

```
struct aluno {
  int num_aluno;
  float nota[3];
};
```

Acesso às variáveis

- Como nos arrays, uma estrutura pode ser previamente inicializada:

```
struct ponto {
    int x;
    int y;
};

struct ponto p1 = { 220, 110 };
```

Acesso às variáveis

- Alternativa: definir a estrutura e declara a variável (além de inicializar):

```
struct ponto {
    int x;
    int y;
} p1 = { 220, 110 };
```

Acesso às variáveis

- Como é feito o acesso às variáveis da estrutura?
  - Cada variável da estrutura pode ser acessada com o operador **ponto** "**.**"
  - Ex.:

```
 // definindo a estrutura aluno
struct aluno {
    int num_aluno;
    float nota1, nota2, nota3;
 };

struct aluno Aluno1; // declarando a variável da struct aluno

float Media;
// acessando os elementos da struct
Aluno1.num_aluno = 72345;
Aluno1.nota1 = 23.5;
Aluno1.nota2 = 25.0;
Aluno1.nota3 = 27.5;
Media = (Aluno1.nota1 + Aluno1.nota2 + Aluno1.nota3)/3.0;
```

Acesso às variáveis

- E se quiséssemos ler os valores das variáveis a partir do teclado?
  - Resposta: basta ler cada variável independentemente, respeitando seus tipos.

```
gets(cliente_especial.nome); //string
scanf("%d",&cliente_especial.idade); //int
scanf("%f",&cliente_especial.grau_miopia[0]); //float
scanf("%f",&cliente_especial.grau_miopia[1]); //float
```

Acesso às variáveis

- Note que cada variável dentro da estrutura pode ser acessada como se apenas ela existisse, não sofrendo nenhuma interferência das outras.
  - Uma estrutura pode ser vista como um simples agrupamento de dados.
  - Se faço um scanf para estrutura.idade não me obriga a fazer um scanf para estrutura.peso

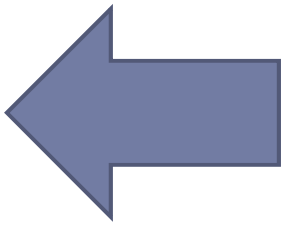
Memória

- Os elementos de um struct também são alocados sequencialmente na memória (como nos vetores).

- Veja o código do exemplo a seguir e faça o mapa de memória.

```c
int main()
{
    struct dados_pacientes {
       int  idade;
       char nome[10];
       double peso;
       double altura;
       int estado_civil;
       float grau_miopia[2];
   };

   unsigned int tamanho_da_struct;
   struct dados_pacientes paciente1, paciente2 ;

   // lembre que string é um vetor, não posso fazer
   // paciente1.nome = “José"
   // o correto é usar strcpy –string copy
    strcpy(paciente1.nome,"Jose");
    paciente1.altura = 1.25;
    paciente1.peso = 73;
    paciente1.estado_civil = 1; // 0 para solteiro
    paciente1.grau_miopia[0] = 1.75; // olho esquerdo
    paciente1.grau_miopia[1] = 0; // olho direito
}
```

```
struct dados_pacientes {
        int  idade;
        char nome[10];
        double peso;
        double altura;
        int estado_civil;
        float grau_miopia[2];
};

int main()
{
   unsigned int tamanho_da_struct;
   struct dados_pacientes paciente1, paciente2 ;

   // lembre que string é um vetor, não posso fazer
   // paciente1.nome = “José"
   // o correto é usar strcpy –string copy
    strcpy(paciente1.nome,"Jose");
    paciente1.altura = 1.25;
    paciente1.peso = 73;
    paciente1.estado_civil = 1; // 0 para solteiro
    paciente1.grau_miopia[0] = 1.75; // olho esquerdo
    paciente1.grau_miopia[1] = 0; // olho direito
}
```



Podemos declarar a struct fora do main()

Isso na verdade é o mais comum, e será importante quando a struct for usada por outras funções do programa

Estruturas

- Voltando ao exemplo anterior, se, ao invés de 2 cadastros, quisermos fazer 100 cadastros de pacientes?

Array de estruturas

- SOLUÇÃO: criar um **array de estruturas**.
- Sua declaração é similar a declaração de uma array de um tipo básico
  - `struct dados_pacientes pacientes[100];`
  - Desse modo, declara-se um array de 100 posições, onde cada posição é do tipo `struct dados_pacientes`
  - `struct dados_pacientes pacientes[100];`

Array de estruturas

- SOLUÇÃO: criar um **array de estruturas**.
- Sua declaração é similar a declaração de uma array de um tipo básico
  - `struct dados_pacientes pacientes[100];`

  - Desse modo, declara-se um array de 100 posições, onde cada posição é do tipo `struct dados_pacientes`

  - `struct dados_pacientes pacientes[100];`

Quantos bytes ocupa a variável pacientes?

Array de estruturas

- Lembrando:
  - array: é uma "lista" de elementos de mesmo tipo (HOMOGÊNEO).
  - struct: define um "conjunto" de variáveis que podem ser de tipos diferentes (HETEROGÊNEO);

Array de estruturas

- Num array de estruturas, o operador de ponto (.) vem depois dos colchetes ([ ]) do índice do array.

```
int main(){
   struct cadastro c[4];
   int i;
   for(i=0; i<4; i++){
      gets(c[i].nome);
      scanf("%d",&c[i].idade);
      gets(c[i].rua);
      scanf("%d",&c[i].numero);
   }
   system("pause");
   return 0;
}
```

Exercício

- Utilizando a estrutura abaixo, faça um programa para ler o número e as 3 notas de 10 alunos. Calcule a média para cada aluno e armazene na estrutura.

```
struct aluno {
  int num_aluno;
  float nota1, nota2, nota3;
  float media;
};
```

Exercício – Solução (sem printfs)

```c
struct aluno {
    int num_aluno;
    float nota1, nota2, nota3;
    float media;
};

int main(){

  struct aluno a[10];
  int i;
  for (i=0;i<10;i++){
    scanf("%d",&a[i].num_aluno);
    scanf("%f",&a[i].nota1);
    scanf("%f",&a[i].nota2);
    scanf("%f",&a[i].nota3);
    a[i].media = (a[i].nota1 + a[i].nota2 + a[i].nota3)/3.0
  }
}
```

Exercício

- Utilizando a estrutura abaixo, faça um programa para ler o número e as 3 notas de 10 alunos. Calcule a média para cada aluno e armazene na estrutura.
  - Note que temos um vetor dentro da estrutura

```
struct aluno {
  int num_aluno;
  float nota[3];
  float media;
};
```

Exercício – Solução (sem printfs)

```
int main(){
  struct aluno a[10];
  int i;
  for (i=0;i<10;i++){
    scanf("%d",&a[i].num_aluno);
    scanf("%f",&a[i].nota[0]);
    scanf("%f",&a[i].nota[1]);
    scanf("%f",&a[i].nota[2]);
    a[i].media = (a[i].nota[0] + a[i].nota2[1] + a[i].nota[2])/3.0
  }
}
```

Atribuição entre estruturas

- Atribuições entre estruturas só podem ser feitas quando os campos são IGUAIS!
  - Ex:

```
struct cadastro c1,c2;
c1 = c2; //CORRETO
```

  - Ex:

```
struct cadastro c1;
struct ficha c2;
c1 = c2; //ERRADO!! TIPOS DIFERENTES
```

Atribuição entre estruturas

- No caso de estarmos trabalhando com arrays, a atribuição entre diferentes elementos do array é válida
  - Ex:

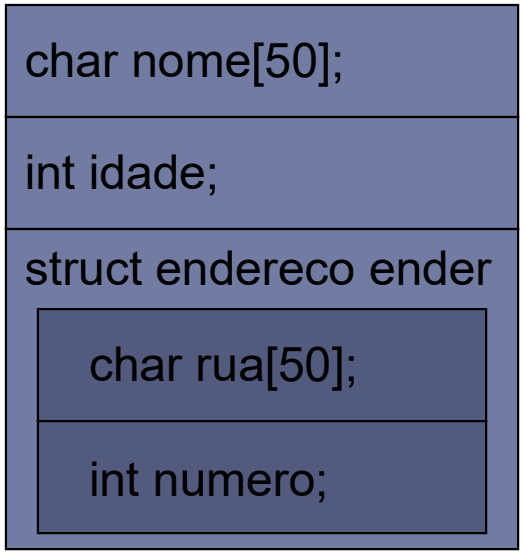
    ```
    struct cadastro c[10];
    c[1] = c[2]; //CORRETO
    ```

- Note que nesse caso, os tipos dos diferentes elementos do array são sempre IGUAIS.

Estruturas de estruturas (EXTRA)

- Sendo uma estrutura um tipo de dado, podemos declarar uma estrutura que utilize outra estrutura previamente definida:

```
struct endereco{
   char rua[50]
   int numero;
};
struct cadastro{
   char nome[50];
   int idade;
   struct endereco ender;
};
```



cadastro

Estruturas de estruturas

- Nesse caso, o acesso aos dados do **endereço** do cadastro é feito utilizando novamente o operador "**.**".

```
struct cadastro c;

gets(c.nome);
scanf("%d",&c.idade);
gets(c.ender.rua);
scanf("%d",&c.ender.numero);
```

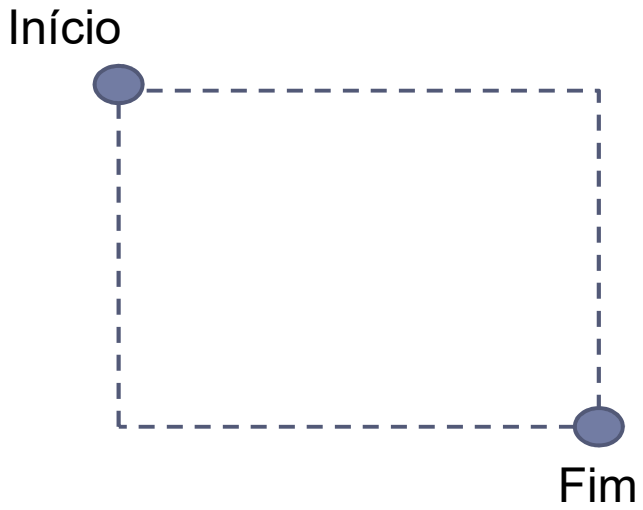
Estruturas de estruturas

- Outros exemplos

```
struct cadastro c;

strcpy(c.nome,"João");
c.idade  = 30;
strcpy(c.ender.rua,"Avenida 1");
c.ender.numero = 45;
```

Lendo um retângulo

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

struct retangulo {
   struct ponto inicio, fim;
};

struct retangulo r;

scanf(“%d”,&r.inicio.x);
scanf(“%d”,&r.inicio.y);
scanf(“%d”,&r.fim.x);
scanf(“%d”,&r.fim.y);
```



Estruturas de estruturas

- Inicialização de uma estrutura de estruturas:

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

struct retangulo {
   struct ponto inicio, fim;
};

struct retangulo r = {{10,20},{30,40}};
```

inicio fim