Estruturas definidas pelo programador

Variáveis

- As variáveis vistas até agora eram:
  - simples: definidas por tipos **int**, **float**, **double** e **char**;
  - compostas homogêneas: definidas por **array**.
- No entanto, a linguagem C permite que se criem novas estruturas a partir dos tipos básicos.

Estruturas

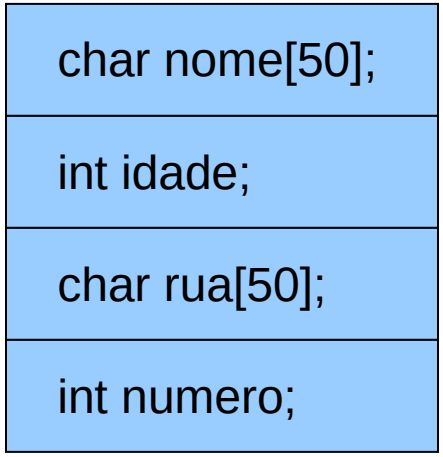
- Uma estruturas pode ser vista como um novo tipo de dados, que é formado por variáveis de outros tipos.
  - Pode ser declarada em qualquer escopo.
  - Ela é declarada da seguinte forma:

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

Estruturas

- Uma estrutura pode ser vista como um agrupamento de dados.

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



cadastro

Estruturas - declaração

- Uma variável pode ser declarada de modo similar aos tipos já existente:
  - **struct** cadastro c;
- Por ser um tipo definido pelo programador, usa-se a palavra **struct** antes do tipo da nova variável

Exercício

- Declare uma estrutura capaz de armazenar o número de matrícula e 3 notas para um dado aluno.

Exercício - Solução

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

Estruturas

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

```
char nome1[50], nome2[50], nome3[50], nome4[50];
int idade1, idade2, idade3, idade4;
char rua1[50], rua2[50], rua3[50], rua4[50]
int numero1, numero2, numero3, numero4;
```

Estruturas

- Utilizando uma estrutura, o mesmo pode ser feito da seguinte maneira:

```
struct cadastro{
    char nome[50];
    int idade;
    char rua[50]
    int numero;
};
struct cadastro c1, c2, c3, c4;
```

Acesso às variáveis

- Cada variável da estrutura pode ser acessada com o operador ponto “**.**”

```
struct cadastro c;
strcpy(c.nome,"João");
scanf(“%d”,&c.idade);
strcpy(c.rua,"Avenida 1");
c.numero = 1082;
```

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

- E se quiséssemos ler os valores das variáveis da estrutura do teclado?

```
gets(c.nome);//string
scanf(“%d”,&c.idade);//int
gets(c.rua);//string
scanf(“%d”,&c.numero);//int
```

Estruturas

- Voltando ao exemplo anterior, se, ao invés de 5 cadastros, quisermos fazer 100 cadastros?

Array de estruturas

- Declaração:
  - struct cadastro c[100];
  - Desse modo, declara-se um array de 100 posições, onde cada posição é do tipo **struct cadastro**.

Array de estruturas

- Lembrando:
  - struct: define um “conjunto” de variáveis que podem ser de tipos diferentes;
  - array: é uma “lista” de elementos de mesmo tipo.

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 do exercício anterior, faça um programa para ler o número de matrícula e as 3 notas de 10 alunos.

Exercício - Solução

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

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:

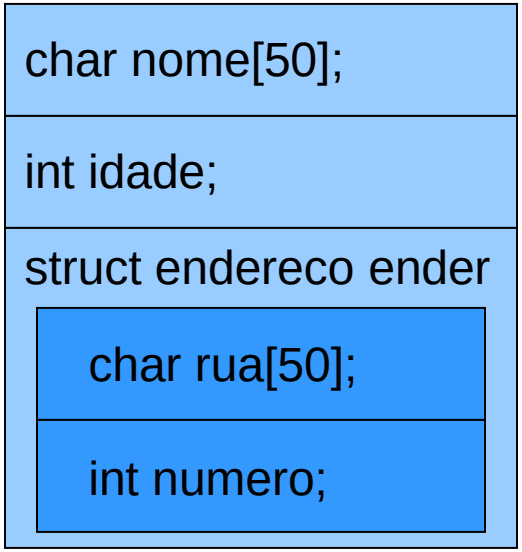
    struct cadastro c[10];

    c[1] = c[2]; //CORRETO

Estruturas de estruturas

```
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;
strcpy(c.nome,"João");
scanf(“%d”,&c.idade);
strcpy(c.ender.rua,"Avenida 1");
scanf(“%d”, &c.ender.numero);
```

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}};
```

Enumerações

- Definem um tipo com valores determinados em um certo domínio:
- Trabalhador pode ser: empregado, desempregado, aposentado
- Conta de banco pode ser: ativa, inativa, bloqueada

Enumerações

- Primeiro, definimos os valores possíveis da enumeração:
  - `enum colors {black, blue, green, cyan, red, purple, yellow, white};`
- Em seguida instanciamos um tipo enumerado:
  - `enum colors cordaparede = white;`

Enumerações - C

- Em muitas linguagens de programação, variáveis baseadas em enumeração podem receber SOMENTE os valores enumerados
- Esse não é o caso no C!

```
1. enum colors {black, blue, green,
   cyan, red, purple, yellow, white};
2. enum colors cordaparede = 23;
```

Enumerações - C

- Enumerações no C são um valor inteiro
- Cada elemento é associado a um valor inteiro, por exemplo:
  - `enum colors {black, blue, green, cyan, red, purple, yellow, white};`
    - Black – 0
    - Blue – 1
    - Green – 2
    - Cyan – 3
    - Purple – 4
    - Yellow – 5
    - White – 6

Enumerações - C

- Podemos definir os nossos próprios valores, que podem ser repetidos:

```
1. enum diadasemana {
2.  domingo = 0,
3.  segunda = 1,
4.  terca   = 2,
5.  quarta  = 3,
6.  quinta  = 4,
7.  sexta   = 5,
8.  sabado  = 0,
9. };
```

Enumerações - C

- Quando não definimos um valor explícito, o primeiro elemento recebe o valor zero
- Elementos subsequentes recebem 1, 2, 3, ...

C - Typedef

- O C ainda permite a criação “apelidos” para tipos básicos e seus modificadores:

- Exemplos:
  - `typedef bool char;`
  - `typedef inteiropos unsigned int;`