Programação e Desenvolvimento de Software I

Registros e enumeradores

Aula de hoje

- **Criar um novo tipo de dado!**
- Tipos definidos pelo programador
  - Estruturas: **struct**
    - Variáveis heterogêneas
  - Uniões: **union**
    - Lista de variáveis, cada uma delas pode ter qualquer tipo
  - Enumerações: **enum**
    - Lista de constantes, em que cada constante possui um nome significativo
  - Comandos **typedef**
    - Renomear um tipo existente

Estruturas

Estruturas - struct

Até agora

- Dois tipos de variáveis:
  - Tipos básicos: **int, float, double, char**
  - Tipos homogêneos: **vetores**

- Porém, a linguagem C permite que se criem novas estruturas a partir dos tipos básicos
  - **struct**

Estruturas: tipos definidos pelo programador

- **Estruturas** são tipos de dados compostos heterogêneos definidos pelo programador

- A principal **vantagem** do uso de estruturas é poder **agrupar** de forma organizada **vários tipos** de dados diferentes **dentro** de uma **única variável**

- Exemplo: estrutura do tipo **cadastro**
  - Campos: nome, idade, sexo, rua, número, ….

Estruturas: tipos definidos pelo programador

- **Declaração** de uma estrutura
  - Corresponde à definição de um tipo
  - Cada componente é denominado campo ou membro da estrutura

```
struct nome_struct{
   tipo1 campo1;
   tipo2 campo2;
   ...
   tipon campoN;
};
```

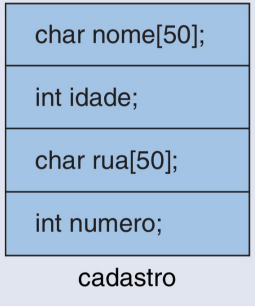
Estruturas: tipos definidos pelo programador

- **Declaração** de uma estrutura
  - Corresponde à definição de um tipo
  - Cada componente é denominado campo ou membro da estrutura

```
struct nome_struct{
   tipo1 campo1;
   tipo2 campo2;
   ...
   tipon campoN;
};
```

Exemplo: estrutura

```
01  struct cadastro{
02        char nome[50];
03        int idade;
04        char rua[50];
05        int numero;
06  };
```



Estruturas: tipos definidos pelo programador

- **Declaração** de uma estrutura
  - Corresponde à definição de um tipo
  - Cada componente é denominado campo ou membro da estrutura

```
struct nome_struct{
    tipo1 campo1;
    tipo2 campo2;
    ...
    tipon campoN;
};
```

Exemplo: estrutura

```
01  struct cadastro{
02        char nome[50];
03        int idade;
04        char rua[50];
05        int numero;
06  };
```

```
struct aluno{
      char nome[50];
      int matricula;
      float nota1,nota2,nota3;

};
```

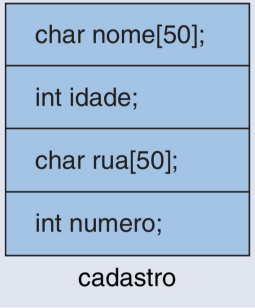
Estruturas: tipos definidos pelo programador

- Uma estrutura pode ser vista como um agrupamento de dados

- Ex: cadastro de pessoas
  - Todas as informações são da mesma pessoa, logo podemos agrupá-las
  - Isso facilita também lidar com dados de outras pessoas no mesmo programa

Exemplo: estrutura

```
01  struct cadastro{
02        char nome[50];
03        int idade;
04        char rua[50];
05        int numero;
06  };
```



Estruturas - declaração de variáveis

- Uma vez definida a estrutura, ou seja, definido o tipo
  - Podemos declarar variáveis do tipo da estrutura

```
struct cadastro c;
```

- Por ser um tipo definido pelo programador, usa-se a palavra **struct** antes do tipo da nova variável declarada

Estruturas - resumo definição e declaração

- **Definição**
  - Especifica como a estrutura é composta, quais são seus campos e de que tipo
  - A definição da estrutura **não** define variáveis

```
struct cadastro{
    char nome[50];
    int idade;
    char rua[50];
    int numero;
} cad1, cad2;
```

- **Declaração**
  - A declaração de variáveis deve ser explícita

```
struct cadastro c;
```

Exercício - definição de estruturas

- Defina uma estrutura capaz de armazenar o número de matrícula e 3 notas de um aluno

Exercício - definição de estruturas

- Defina uma estrutura capaz de armazenar o número de matrícula e 3 notas de um aluno

Possíveis soluções

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

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

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

Estruturas - acesso aos campos

```
struct cadastro{
    char nome[50];
    int idade;
    char rua[50];
    int numero;
} cad1, cad2;
```

Uma vez que a estrutura foi definida e uma variável do tipo dessa estrutura foi declarada

- Precisamos acessar os campos da estrutura
- Para isso, utilizamos o operador ponto "."
- Por exemplo:

```
//declarando a variável
struct cadastro c;

//acessando os seus campos
strcpy(c.nome,"João");
scanf("%d",&c.idade);
strcpy(c.rua,"Avenida 1");
c.numero = 1082;
```

Estruturas - acesso aos campos

Exemplo de inicialização de campos

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

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

Exemplo de inicialização por meio do teclado

```
struct cadastro c;

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

Estruturas - acesso aos campos

Exemplo de inserção de **um cadastro apenas**

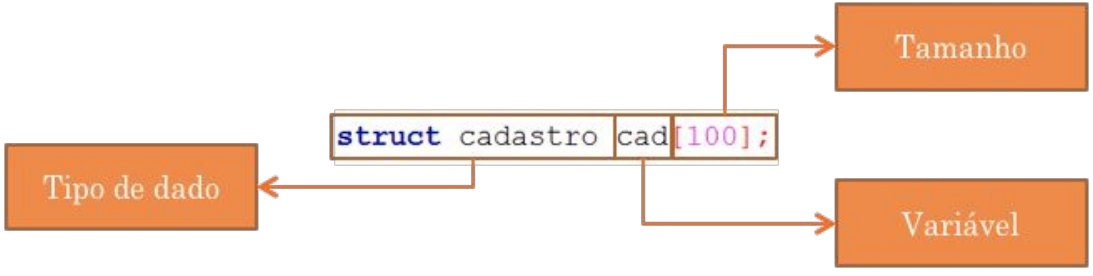
```
struct cadastro c;

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

E se precisássemos cadastrar 100 pessoas?

- Solução: criar um **array/vetor de estruturas**

Estruturas - vetores

A declaração de um **array/vetor de estruturas** é similar à declaração de um array de um tipo básico



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

Estruturas - vetores

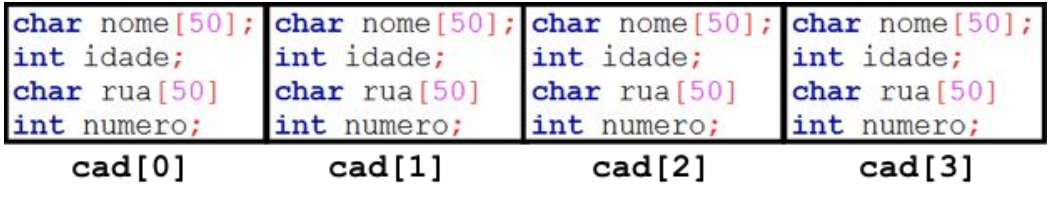
Lembrando

- **struct**: define um "conjunto" de variáveis que podem ser de tipos diferentes

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

- **array**: é uma "lista" de elementos de mesmo tipo

struct cadastro cad[4];



Estruturas - vetores

Num array de estruturas, primeiro acessamos o elemento do vetor e depois o campo da estrutura

- Considerando a variável **cad** que é um vetor de quatro posições do tipo **struct cadastro**

```
struct cadastro cad[4];
```

- Por exemplo, para atribuir o valor 81 ao campo idade da primeira posição do vetor, devemos:

```
cad[0].idade = 81;
```

Estruturas - vetores

Exemplo para ler todos os valores a partir do teclado:

```
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;
}
```

Estruturas - vetores - exercício

Utilizando a estrutura abaixo, faça um programa para ler o número e as 3 notas de 10 alunos

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

Estruturas - vetores - exercício - solução

Utilizando a estrutura abaixo, faça um programa para ler o número e as 3 notas de 10 alunos

```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
    }
}
```

Estruturas - atribuição entre estruturas

Atribuições entre estruturas só podem ser feitas quando as estruturas são as mesmas, ou seja, **possuem o mesmo tipo**

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

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

Estruturas - atribuição entre estruturas

No caso de estarmos trabalhando com vetores, a atribuição entre diferentes elementos dos vetores é válida

```
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 - estruturas aninhadas

Uma estrutura pode agrupar um número arbitrário de variáveis de tipos diferentes

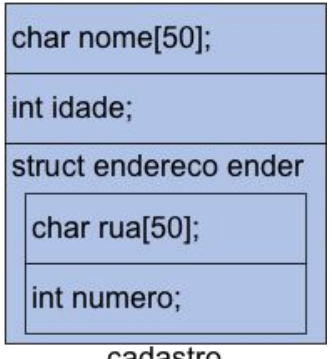
Assim, sendo uma estrutura um tipo de dado, podemos declarar uma estrutura que utilize outra estrutura previamente definida

Estruturas aninhadas são estruturas que utiliza outra estrutura em sua definição

Estruturas de estruturas - estruturas aninhadas

Nesse exemplo, a **estrutura cadastro** é **aninhada** pois utiliza a **estrutura endereço** em sua definição

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



cadastro

Como acessar os campos?

Estruturas de estruturas - estruturas aninhadas

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

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

```
struct cadastro c;

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

//atribuição
strcpy(c.nome,"João");
c.idade = 34;
strcpy(c.ender.rua,"Avenida 1");
c.ender.numero = 131;
```

Estruturas aninhadas - exemplo

Inicialização de uma estrutura de estruturas

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

struct retangulo {
    struct ponto inicio, fim;
};

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

Union (uniões)

Uniões - union

Uma união pode ser vista como uma lista de variáveis, e cada uma delas pode ser de qualquer tipo

A ideia básica é similar à da estrutura:

- criar apenas um tipo de dado que contenha vários membros
- que são outras variáveis

Tanto a declaração quanto o acesso aos elementos de uma união são similares aos de uma estrutura.

União - declaração

Para declarar uma união, utilizamos a palavra reservada **union** da seguinte forma:

```
union nome_union{
    tipo1 campo1;
    tipo2 campo2;
    ...
    tipon campoN;
};
```

União - diferença para estrutura

Estrutura

- reserva espaço de memória para todos os seus elementos
- é alocado espaço de memória suficiente para armazenar todos os seus elementos

União

- reserva espaço de memória para o seu maior elemento e compartilha essa memória com os demais
- é alocado espaço de memória para armazenar o maior dos elementos que a compõe

União - diferença para estrutura

Por exemplo, dada a seguinte declaração de uma união:

```
union tipo{
   short int x;
   unsigned char c;
};
```

Essa união possui o nome **tipo** e duas variáveis: *x*, do tipo **short int** (dois bytes) e *c* do tipo **unsigned char** (um byte). Assim, uma variável desse tipo

```
union tipo t;
```

Ocupará dois bytes na memória, que é o tamanho do maior elemento da união (**short int**).

União

Em uma união, apenas um membro pode ser armazenado de cada vez

Isso acontece porque o espaço de memória é compartilhado

Logo, é de **responsabilidade do programador** saber qual dado foi mais recentemente armazenado em uma união

Quando usar uma união?

União - uso

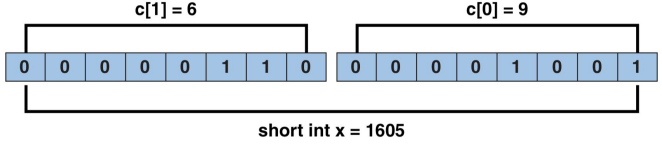
Um dos usos mais comuns de uma união é unir um tipo básico a um vetor de tipos menores

Assumindo a definição da **union** tipo

```
union tipo{
   short int x;
   unsigned char c[2];
};
```

Temos que a variável *x* ocupa dois bytes na memória. Como cada posição da variável *c* ocupa apenas um byte

- podemos acessar cada parte da variável *x* sem precisar de operações de manipulação de bits (operações lógicas e de deslocamento de bits)



Enum (enumerações)

Enumeração - enum

Uma enumeração pode ser vista como uma lista de constantes

- Cada constante possui um nome significativo

A ideia básica é criar apenas um tipo de dado que contenha várias constantes

- E uma variável desse tipo só poderá receber como valor uma dessas constantes

Enumeração - declaração

A sintaxe para definir uma enumeração é:

```
enum nome_enum { lista_de_identificadores };
```

nessa definição, **lista_de_identificadores** é uma lista de palavras separadas por vírgula e delimitadas pelo operador de chaves {}

Por exemplo, o comando:

```
enum semana {Domingo, Segunda, Terca, Quarta, Quinta, Sexta, Sabado };
```

cria uma enumeração de nome **semana**, cujos valores constantes são os nomes dos dias da semana.

Enumeração - definição e declaração

Uma forma simplificada de definir uma enumeração é

```
enum semana {Domingo, Segunda, Terca, Quarta, Quinta, Sexta, Sabado} s1, s2;
```

Nesse exemplo, duas variáveis (*s1* e *s2*) são declaradas junto com a definição da enumeração.

Enumeração - declaração de variáveis

Dada a definição de uma enumeração

```
enum semana {Domingo, Segunda, Terca, Quarta, Quinta, Sexta, Sabado };
```

Podemos declarar uma variável desse tipo de modo similar aos tipos já existentes

```
enum semana s;
```

E podemos inicializar como qualquer outra variável, usando uma das constantes da enumeração

```
s = Segunda;
```

Enumerações e constantes

Para o programador

- uma enumeração é uma lista de constantes

Para o compilador

- cada constante é representada por um valor inteiro, começando de 0 (zero)

Exemplo

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  enum semana {Domingo, Segunda, Terca, Quarta, Quinta,
04   Sexta, Sabado};
05  int main(){
06    enum semana s1, s2, s3;
07    s1 = Segunda;
08    s2 = Terca;
09    s3 = s1 + s2;
10    printf("Domingo = %d\n",Domingo);
11    printf("s1 = %d\n",s1);
12    printf("s2 = %d\n",s2);
13    printf("s3 = %d\n",s3);
14    system("pause");
15    return 0;
16  }
```

Saída

```
Domingo = 0
s1 = 1
s2 = 2
s3 = 3
```

Typedef

Comando **typedef**

A linguagem C permite que o programador nomeie seus tipos com base em tipos já existentes

Para isso usamos o comando **typedef**, cuja forma geral é:

```
typedef tipo_existente novo_nome;
```

em que:

- **tipo_existente** é um tipo básico ou definido pelo programador (por exemplo, uma **struct**)
- **Novo_nome** é o nome para o tipo que estamos definindo

Comando **typedef** - exemplo

Comando **typedef**, cuja forma geral é:

```
typedef tipo_existente novo_nome;
```

Exemplo:

```
typedef int inteiro;
```

- O comando **typedef** NÃO cria um novo tipo chamado *inteiro*
- Ele apenas cria um sinônimo (*inteiro*) para o tipo *int*
- Ele novo nome se torna equivalente ao tipo já existente

Comando **typedef**

O comando **typedef** pode ser usado para simplificar a declaração de um tipo definido pelo programador

Por exemplo, dada a declaração da **struct**:

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

O seguinte comando é usado para declarar variáveis:

```
struct cadastro c;
```

Usando o **typedef** para criar um apelido:

```
typedef struct cadastro cad;
```

A declaração da variável passa a ser:

```
cad c;
```

Comando **typedef** - exemplo

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

typedef int inteiro;

int main(){
    int x = 10;
    inteiro y = 20;
    y = y + x;
    printf("Soma = %d\n",y);

    return 0;
}
```

Resumo da aula

Resumo da aula - Tipos definidos pelo programador

Além dos tipos básicos da linguagem C (**int, char, float, double**), podemos criar novos tipos

- Estruturas: **struct**
  - Variáveis heterogêneas
- Uniões: **union**
  - Lista de variáveis, cada uma delas pode ter qualquer tipo
- Enumerações: **enum**
  - Lista de constantes, em que cada constante possui um nome significativo
- Comandos **typedef**
  - Renomear um tipo existente

Capítulo 8 do livro texto.