

Registro (*struct*)

Registros

- Registros: criação de novos tipos de dados a partir dos tipos básicos da linguagem
- Registros são caracterizados pelos seus campos
- Em linguagens orientadas a objeto, temos as classes: tipo de dados + operações associadas

Registros

- Um registro possui:
 - Campos, cada um com seu nome
 - Nome do tipo do registro: usado para declaração de variáveis deste tipo

Registros - etapas

1. Definição dos campos de um registro e nome do registro
2. Declaração de variáveis do tipo registro
3. Leitura/escrita de dados em campos do registro

Definição de novos tipos de dados

- Se cada fração compreende dois inteiros, como é possível fazer uma função para somar duas frações passando apenas dois parâmetros?
- Isto é possível porque a linguagem C permite a definição de **novos tipos de dados** com base nos **tipos primitivos**: `char`, `int`, `float` e `double`.
- Estes novos tipos de dados, formados a partir dos tipos primitivos são chamados de **tipos estruturados**.

Definição de novos tipos de dados

- Uma variável de um determinado tipo estruturado definido pelo usuário é comumente chamada de uma estrutura.
- Uma estrutura agrupa várias variáveis de diversos tipos em uma só variável.
- Para criar uma estrutura usa-se o comando **struct**:

```
struct nome_da_estrutura
{
    tipo_1 variavel_1;
    ...
    tipo_n variavel_n;
};
```

As variáveis que compõem a estrutura são chamadas de **campos** da estrutura.

Definição de novos tipos de dados

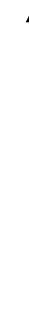
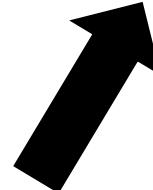
- Exemplos:

```
struct ponto
{
    float coord_x;
    float coord_y;
};
```

```
struct cilindro
{
    float altura;
    struct circulo base;
};
```

```
struct circulo
{
    float raio;
    struct ponto centro;
};
```

A declaração de variáveis de um tipo estruturado (estruturas) é feita da mesma forma que para um tipo simples.



Definição de novos tipos de dados

- Para se acessar os campos de uma estrutura, basta separar o nome da variável pelo símbolo **ponto** (**.**).
- Para os exemplos anteriores:

```
struct ponto
{
    float coord_x;
    float coord_y;
};
```

```
struct cilindro
{
    float altura;
    struct circulo base;
};
```

```
struct circulo
{
    float raio;
    struct ponto centro;
};
```

```
struct cilindro d;
d.altura = 3.0;
d.base.raio = 5.5;
d.base.centro.coord_x = 1.2;
d.base.centro.coord_y = 3.8;
```

O comando `typedef`

- O Comando `typedef` permite ao programador definir um novo nome para um determinado tipo.
- Sua forma geral é:

```
typedef nome_antigo nome_novo;
```

- Exemplo:

Dando o nome `inteiro` para o tipo `int`:

```
typedef int inteiro;
```

```
inteiro num;
```

O comando `typedef`

- O comando `typedef` também pode ser utilizado para dar nome a tipos complexos como estruturas.
- Exemplos:

```
typedef struct tipo_endereco
{
    char rua[50];
    int numero;
    char bairro[20];
    char cidade[30];
    char sigla_estado[3];
    long int CEP;
} TEndereco;
```

```
typedef struct frac
{
    int num;
    int den;
} frac;
```



Exemplo do programa [p22.c](#)

O comando `typedef`

■ Observação:

Utilizando-se o comando `struct` juntamente com o comando `typedef`, pode-se dispensar o uso da palavra `struct` na declaração da variável.

■ Exemplos:

```
typedef struct ponto
{
    float coord_x;
    float coord_y;
} ponto;
```

```
typedef struct circulo
{
    float raio;
    ponto centro;
} circulo;
```

```
typedef struct cilindro
{
    float altura;
    circulo base;
} cilindro;
```

Registro – C – protótipo

- Criação de um registro: usamos a palavra-chave (*keyword*) struct
 1. struct aluno {
 2. int matricula;
 3. char nome[100];
 4. int disciplinas[10];
 5. double nota[10];
 6. };

Registro – C – protótipo

- Nome do registro: aluno
- Campos: numeromatricula, nome, disciplinas, nota

```
1. struct aluno {  
2.     int matricula;  
3.     char nome[100];  
4.     int disciplinas[10];  
5.     double nota[10];  
6. };
```

Registro – C – declaração

- Para declarar uma variável, usamos o seu nome, junto com a keyword struct;
 1. struct aluno aluno1;
 2. struct aluno al_aeds1[55];

Registro – C – declaração

- Podemos encurtar a declaração se usarmos `typedef` para definir um “apelido”
 1. `typedef struct aluno pessoa;`
 - Ou
 1. `typedef struct {`
 2. `...`
 3. `...`
 4. `} pessoa;`
- Declaração: `pessoa aluno1;`

Registro – C – declaração

- Com typedef, podemos declarar estruturas sem nome, que depois irão receber um apelido:
 1. `typedef struct ?nome deveria vir aqui? {`
 2. `...`
 3. `...`
 4. `} pessoa;`

Registro – C – declaração

- Assim, podemos declarar registros de duas formas:
 1. `struct registro var; //Somente com registro nomeado`
 2. `registro var; //Somente com typedef`

Registro – C – Acesso aos campos

- O acesso aos campos é dado pela forma variavel.campo ou variavel->campo.
- variavel.campo: acesso pela variável
- variavel->campo: acesso por ponteiro
 - Igual a (*variavel).campo

Registro – C – Acesso aos campos

- Acesso via variável
 - 1. struct aluno novo;
 - 2. novo.matricula=2012113491;
 - 3. scanf("%s",novo.nome);
 - 4. printf("Nome do novo aluno: %s",novo.nome);

Registro – C – Acesso aos campos

- Acesso via ponteiro
 1. struct aluno novo;
 2. struct aluno *p = &novo;
 3. p->matricula=2012113491;
 4. scanf("%s",p->nome);
 5. printf("Nome do novo aluno: %s",p->nome);

Registro – C – funções

- Registros são passados por valor: o programa abaixo imprime 10.

```
1. #include<stdio.h>          8. int main() {  
2. struct teste {             9. struct teste t;  
3.     int inteiro;          10. t.inteiro = 10;  
4. };                         11. funcao(t);  
5. void funcao(struct teste 12. printf("%d\n",t.inteiro);  
6. a) {                      13. return 0;  
7.     a.inteiro *= 10;        14. }
```

Registro – C – funções

- Registro – passo como parâmetro por valor ou por referência?
 - Depende do custo da cópia do registro de uma função para outra
 - Passagem por valor: melhor se o(s) registro(s) for(em) pequeno(s)
 - Passagem por referência: melhor se o(s) registro(s) for(em) grande(s)