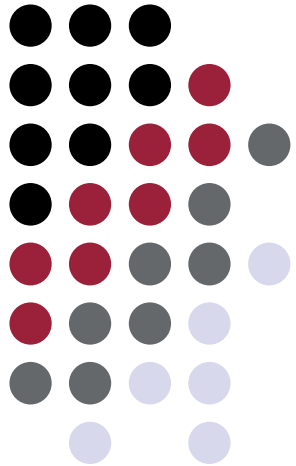
A12 Registros

CSI030 – Programação de Computadores I

Prof. Dr. George H. G. Fonseca
Universidade Federal de Ouro Preto



Ponteiros

- Um registro (struct) é a forma de agrupar variáveis em C, sejam elas do mesmo tipo ou de tipos diferentes

- As variáveis unidas em um registro se relacionam de forma a criar um contexto maior

- Ex.:
  - Registro de alunos: nome, matrícula, período, curso, coeficiente etc

  - Registro de funcionários: nome, cpf, endereço, salário, função etc

Declaração do tipo Registro

- Um registro é declarado com a palavra-chave struct

```
struct nome_tipo_registro {
    tipo_1 variavel_1;
    tipo_2 variavel_2;
    ...
    tipo_n variavel_n;
};
```

- Devemos declarar variável

```
struct nome_tipo_registro variavel_registro;
```

Declaração do tipo Registro

- Um registro define um novo tipo de dados com nome `nome_tipo_registro` que possui os campos `variavel_i`
- O campo `variavel_i` é uma variável do tipo `tipo_i` e será acessível a partir de uma variável do novo tipo `nome_tipo_registro`.
- `variavel_registro` é uma variável do tipo `nome_tipo_registro` e possui, internamente, os campos `variavel_i` do tipo declarado.
- A declaração de um registro pode ser feita dentro de uma sub-rotina ou fora dela.

Declaração do tipo Registro

- Exemplo de registro:

```
struct regAluno {
     int matricula;
     char nome[50];
     char curso[30];
     char sexo;
     float coeficiente,
};

int main(void){
     struct regAluno aluno1, aluno2;
}
```

Acesso aos campos do Registro

- Os campos de um registro podem ser acessados individualmente a partir de variáveis do tipo do registro da seguinte forma:

```
variavel_registro.variavel_i
```

- Cada campo, `variavel_registro.variavel_i` se comporta como uma variável do tipo do campo, ou seja, `tipo_i`

- Isto significa que todas as operações válidas para variáveis do tipo `tipo_i` são válidas para o campo acessado por `variavel_registro.variavel_i`

Inicialização de Registros

- Na inicialização de registros os campos obecedem a ordem de declaração

```
struct contaBancaria {
    int numero;
    char idCorrentista[15];
    float saldo;
};

struct contaBancaria conta = {1, "MG1234567", 100.0};
```

Atribuição de Registros

- Pode-se copiar o conteúdo de uma estrutura para outro utilizando uma atribuição simples

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

int main() {
   struct ponto p = {1, 5}, q;
   q.x = 3;
   q.y = 9;
   q = p;
   q.x = 15;
   return 0;
}
```

Variável	Conteúdo
p.x	1
p.y	5
q.x	~~3~~ ~~1~~ 15
q.y	~~9~~ 5

Vetores de Registros

- Pode-se criar vetores de registros da mesma maneira que se criam vetores de tipos primitivos (**`int`**, **`float`**, **`char`**, **`double`**)

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

int main() {
   struct ponto v[2];
   v[0].x = 1;
   v[0].y = 2;
   v[1].x = 3;
   v[1].y = 4;
   return 0;
}
```

Registros como parâmetro de sub-rotinas

- Pode-se passar um registro como parâmetro de uma sub-rotina

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

void imprimePonto(struct ponto p) {
    printf("Coordenadas: (%d, %d)", p.x, p.y);
}

int main() {
   struct ponto p = {1, 2};
   imprimePonto(p);
   return 0;
}
```

Sinônimos de tipos com typedef

- A palavra-chave **typedef** permite dar novos nomes (sinônimos) a tipos de dados existentes

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

Ponto p, q;
```

Sinônimos de tipos com typedef

- Pode-se associar uma definição de **`struct`** a um novo tipo de dados evitando a repetição da palavra **`struct`**

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

Ponto p, q;
```

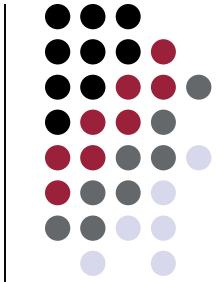
Exemplo completo de Registros

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3  typedef struct aluno {
 4      int matricula;
 5      char nome[30];
 6      float coeficiente;
 7  } Aluno;
 8
 9  void leVetorAlunos(Aluno alunos[], int n) {
10      int i;
11      for(i = 0; i < n; ++i) {
12          printf("\nAluno %d\n", i + 1);
13          printf("Matricula: ");
14          scanf("%d", &alunos[i].matricula);
15          printf("Nome: ");
16          scanf(" %[^\n]s", &alunos[i].nome);
17          printf("Coeficiente: ");
18          scanf("%f", &alunos[i].coeficiente);
19      }
20  }
    ...
```

Exemplo completo de Registros

```
21
22  void imprimeVetorAlunos(Aluno alunos[], int n) {
23      int i;
24      printf("              ALUNOS                \n");
25      printf("____________________________________\n");
26      for(i = 0; i < n; ++i) {
27          printf("\nAluno %d\n", i + 1);
28          printf("   Matricula: %d\n", alunos[i].matricula);
29          printf("   Nome: %s\n", alunos[i].nome);
30          printf("   Coeficiente: %.2f\n", alunos[i].coeficiente);
31          printf("____________________________________\n");
32      }
33  }
34
35  int main() {
36      int n;
37      printf("Informe o numero de alunos: ");
38      scanf("%d", &n);
39
40      Aluno alunos[n];
41      leVetorAlunos(alunos, n);
42      system("cls");
43      imprimeVetorAlunos(alunos, n);
44
45      return 0;
46  }
```

Alocação Dinâmica de Registros



- Para acessar os elementos de um registro, deve-se acessar o registro e, depois, os elementos;

- Os parênteses são obrigatórios porque a precedência do operador ***** é menor que o do operador

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

Ponto* ponto_ptr ;
ponto_ptr = (Ponto*) malloc (sizeof(Ponto));
(*ponto_ptr).x = 3;
(*ponto_ptr).y = 4;
free(ponto_ptr);
```

Alocação Dinâmica de Registros

- O operador -> facilita o acesso aos registros, permitindo acessar diretamente o conteúdo de um campo do registro

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

Ponto* ponto_ptr ;
ponto_ptr = (ponto*) malloc (sizeof(Ponto));
ponto_ptr->x = 3;
ponto_ptr->y = 4;
free(ponto_ptr);
```

Arquivos de Cabeçalho

- A diretiva #define é usada para criar definições e constantes em C

- Em projetos mais complexos, é uma boa prática de programação separar a definição de um registro e a assinatura de funções relacionadas a ele em um arquivo .h (header)

- A implementação das funções relacionadas a esse registro deve estar em um arquivo fonte .c separado

```
#include "aluno.h"
```

- Deve-se incluir o arquivo .h no código fonte que for acessar o registro e/ou usar as funções definidas

Exercício

- Crie uma struct para representar as partes reais e imaginarias de um numero complexo. Seu programa deve solicitar dois numeros complexos e apresentar o resultado da soma de ambos

- O cálculo da soma deve ser feito em uma função separada

Exercício

- Crie uma struct para representar funcionários, com as informações cpf, nome, sexo e salário. Seu programa deve perguntar a quantia de funcionários a ser informada, ler e armazenar os dados dos funcionários em um vetor.

- Escreva uma função que aumenta o salário de todos os funcionários em 15%

Exercício

- Crie uma struct para representar produtos, com as informações código, descrição, estoque e preço. Seu programa deve perguntar a quantia de produtos a ser informada, ler e armazenar os dados dos produtos em um vetor.

- Escreva uma função que processa a venda de um produto – receba o código do produto e o número de unidades vendidas como parâmetro e atualize a quantia de estoque disponível
  - A venda só deve ser autorizada se o estoque for suficiente para o número de unidades vendidas

Bibliografia

- Anido, R., Notas de aula. UNICAMP, disponível em http://www.ic.unicamp.br/~ranido/mc102/, acessado em Maio de 2015.

- Mota, V. F., Notas de aula. UFMG. Disponível em https://sites.google.com/site/virginiaferm/home/disciplinas acessado em Maio de 2015.

- Deitel, P, Deitel, H. C How to Program. 6a Ed. Pearson, 2010.