

MC-102 — Aula 22

Registros

Alexandre M. Ferreira

IC – Unicamp

24/05/2017

Roteiro

1 Registros

- Declarando um novo tipo de Registro
- Acessando os campos de um Registro
- Lendo e Escrevendo Registros
- Atribuição e Registros
- Vetores e Registros
- Funções e Registros
- Ponteiros e Registros

2 Exercícios

3 Informações Extras: Redefinição de tipos

Registros

Um registro é um mecanismo da linguagem C para agrupar várias variáveis, que inclusive podem ser de tipos diferentes, mas que dentro de um contexto, fazem sentido estarem juntas.

- Exemplos de uso de registros:
 - ▶ Registro de alunos para guardar os dados: nome, RA, médias de provas, médias de labs, etc...
 - ▶ Registro de pacientes para guardar os dados: nome, endereço, histórico de doenças, etc...

Declarando um novo tipo de registro

- Para criarmos um novo tipo de registro usamos a palavra chave **struct** da seguinte forma:

```
struct nome_registro {
    tipo_1 nome_campo_1;
    tipo_2 nome_campo_2;
    tipo_3 nome_campo_3;
    ...
    tipo_n nome_campo_n;
};
```

- Cada **nome_campo_i**, é um identificador que será do tipo **tipo_i** (são declarações de variáveis simples).

Exemplo:

```
struct Aluno{
    char nome[80];
    float nota;
}; //estamos criando um novo tipo "struct Aluno"
```

Declarando um novo tipo de registro

- A declaração do registro pode ser feita dentro de uma função ou fora dela. Usualmente, ela é feita fora de qualquer função, para que qualquer função possa usar dados do tipo de registro criado.

```
#include <stdio.h>

    /* Declare tipos registro aqui */

int main () {
    /* Construa seu programa aqui */
}
```

Declarando um registro

A próxima etapa é declarar uma variável do tipo **struct nome_registro**, que será usada dentro de seu programa, como no exemplo abaixo:

```
#include <stdio.h>
struct Aluno{
    char nome[80];
    float nota;
};

int main(){
    struct Aluno a, b; //variáveis a, b são do tipo "struct Aluno"
    .....
}
```

Utilizando os campos de um registro

- Podemos acessar individualmente os campos de uma determinada variável registro como se fossem variáveis normais. A sintaxe é:

```
variável_registro.nome_do_campo
```

- Os campos individuais de um variável registro tem o mesmo comportamento de qualquer variável do tipo do campo.

```
struct Aluno{
    char nome[45];
    float nota;
};

int main(){
    struct Aluno a, b; //variáveis do tipo "struct Aluno"
    a.nota = 4.7;
    b.nota = 2*a.nota;
}
```

Utilizando os campos de um registro

```
#include <stdio.h>
#include <string.h>

struct Aluno{
    char nome[45];
    float nota;
};

int main(){
    struct Aluno a, b;

    strcpy(a.nome, "Helen");
    a.nota = 8.6;

    strcpy(b.nome, "Dilbert");
    b.nota = 8.2;

    printf("a.nome = %s, a.nota = %f\n", a.nome, a.nota);
    printf("b.nome = %s, b.nota = %f\n", b.nome, b.nota);
}
```


Lendo e Escrevendo Registros

- A leitura dos campos de um registro a partir do teclado deve ser feita campo a campo, como se fossem variáveis independentes.
- A mesma coisa vale para a escrita, que deve ser feita campo a campo.

```
#include <stdio.h>
#include <string.h>

struct Aluno{
    char nome[81];
    float nota;
};

int main(){
    struct Aluno a, b;

    printf(" Digite o nome:");
    fgets(a.nome, 80, stdin);
    a.nome[strlen(a.nome) -1] = '\0'; //remove '\n'
    printf(" Digite a nota:");
    scanf("%f", &a.nota);

    printf(" Digite o nome:");
    fgets(b.nome, 80, stdin);
    b.nome[strlen(b.nome) -1] = '\0'; //remove '\n'
    printf(" Digite a nota:");
    scanf("%f", &b.nota);

    printf("a.nome = %s, a.nota = %.2f\n", a.nome, a.nota);
    printf("b.nome = %s, b.nota = %.2f\n", b.nome, b.nota);
}
```

Atribuição de registros

- Podemos atribuir um registro a outro diretamente:

```
var1_registro = var2_registro;
```

- Automaticamente é feita uma cópia de cada campo de **var2** para **var1**.

Exemplo:

```
#include <stdio.h>
#include <string.h>

struct Aluno{
    char nome[80];
    float nota;
};
int main(){
    struct Aluno a, b;

    printf(" Digite o nome:");
    fgets(a.nome, 80, stdin);
    a.nome[strlen(a.nome) -1] = '\0'; //remove '\n'
    printf(" Digite a nota:");
    scanf("%f", &a.nota); getchar();

    b = a; //Atribuição de registros

    printf("b.nome = %s, b.nota = %.2f\n", b.nome, b.nota);
}
```

Vetores e Registros

- A declaração e uso de vetores de registros se dá da mesma forma que vetores dos tipos básicos vistos anteriormente.

- ▶ Para declarar:

```
struct Aluno turma[60];
```

- ▶ Para usar:

```
turma[indice].campo;
```

Exemplo de vetor de registro:

```
#include <stdio.h>
#include <string.h>

struct Aluno{
    char nome[80];
    float nota;
};

int main(){
    struct Aluno turma[5];
    int i;

    for(i=0; i<5; i++){
        printf(" Digite o nome:");
        fgets(turma[i].nome, 80, stdin);
        turma[i].nome[strlen(turma[i].nome) -1] = '\0'; //remove '\n'
        printf(" Digite a nota:");
        scanf("%f", &turma[i].nota); getchar();
    }
    float media=0;
    for(i=0; i<5; i++){
        media = media + turma[i].nota;
    }

    printf(" Media da turma = %.2f\n", media/5.0);
}
```

Registros e Funções

- Registros podem ser usados tanto como parâmetros em funções bem como em retorno de funções.
- Neste caso o comportamento de registros é similar ao de tipos básicos.

Funções e Registros

Exemplo.

- Vamos criar as seguintes funções:

- ▶ `struct` Aluno leAluno();

Esta função faz a leitura dos dados de um registro **Aluno** e devolve o registro lido.

- ▶ `void` imprimeAluno(`struct` Aluno a);

Esta função recebe como parâmetro um registro **Aluno** e imprime os dados do registro.

- ▶ `void` listarTurma(`struct` Aluno turma[], `int` n);

Esta função recebe como parâmetros um vetor do tipo **Aluno** representando uma turma, e também um inteiro **n** indicando o tamanho do vetor. A função imprime os dados de todos os alunos.

Funções e Registros

Implementação das funções:

```
struct Aluno leAluno(){
    struct Aluno aux;

    printf(" Digite o Nome: ");
    fgets(aux.nome, 80, stdin);
    aux.nome[strlen(aux.nome) -1] = '\0'; //remove '\n'
    printf(" Digite a Nota: ");
    scanf("%f",&aux.nota); getchar();

    return aux;
}

void imprimeAluno(struct Aluno a){
    printf("Dados de um aluno — ");
    printf("Nome: %s. Nota: %.2f\n", a.nome, a.nota);
}

void listarTurma(struct Aluno turma[], int n){
    printf("Imprimindo a turma\n");
    int i;
    for(i=0; i<n; i++)
        imprimeAluno(turma[i]);
}
```

Funções e Registros

Com as funções implementadas podemos criar o seguinte exemplo de programa.

```
#include <stdio.h>
#include <string.h>

#define MAX 4

struct Aluno{
    char nome[80];
    float nota;
};

struct Aluno leAluno();
void imprimeAluno(struct Aluno a);
void listarTurma(struct Aluno turma[], int n);

int main(){
    int i;
    struct Aluno turma[MAX];
    for(i=0; i<MAX; i++)
        turma[i] = leAluno();

    listarTurma(turma, MAX);
}
```


Ponteiros e Registros

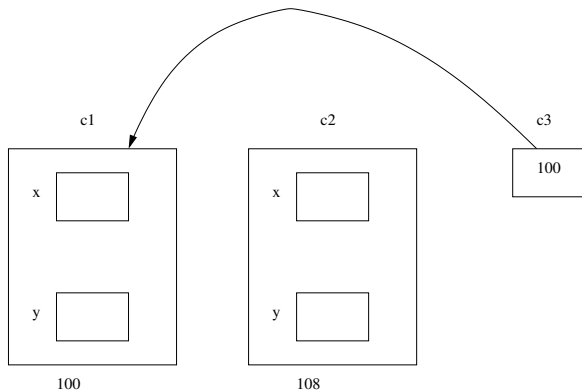
- Ao criarmos uma variável de um tipo **struct**, esta é armazenada na memória como qualquer outra variável, e portanto possui um endereço.
- É possível então criar um ponteiro para uma variável de um tipo **struct**!

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

```
struct Coordenada{  
    double x;  
    double y;  
};
```

```
int main(){  
    struct Coordenada c1, c2, *c3;  
    c3 = &c1;  
    .....
```

Ponteiros e Registros



Ponteiros e Registros

O que será impresso pelo programa abaixo??

```
#include <stdio.h>
struct Coordenada{
    double x;
    double y;
};

int main(){
    struct Coordenada c1, c2, *c3;

    c3 = &c1;
    c1.x = -1;
    c1.y = -1.5;

    c2.x = 2.5;
    c2.y = -5;

    *c3 = c2;

    printf("Coordenadas de c1: (%lf,%lf)\n",c1.x, c1.y);
}
```

Ponteiros e Registros

- Para acessarmos os campos de uma variável **struct** via um ponteiro, podemos utilizar o operador ***** juntamente com o operador **.** como de costume:

```
Coordenada c1, *c3;  
c3 = &c1;  
(*c3).x = 1.5;  
(*c3).y = 1.5;
```

- Em C também podemos usar o operador **->**, que também é usado para acessar campos de uma estrutura via um ponteiro.

```
Coordenada c1, *c3;  
c3 = &c1;  
c3->x = 1.5;  
c3->y = 1.5;
```

- Resumindo: Para acessar campos de estruturas via ponteiros use um dos dois:
 - ▶ `ponteiroEstrutura->campo`
 - ▶ `(*ponteiroEstrutura).campo`

Ponteiros e Registros

O que será impresso pelo programa abaixo??

```
#include <stdio.h>

struct Coordenada{
    double x;
    double y;
};

int main(){
    struct Coordenada c1, c2, *c3, *c4;
    c3 = &c1;
    c4 = &c2;

    c1.x = -1;
    c1.y = -1.5;

    c2.x = 2.5;
    c2.y = -5;

    (*c3).x = 1.5;
    (*c3).y = 1.5;

    c4->x = -1;
    c4->y = -1;

    printf("Coordenadas de c1: (%lf,%lf)\n",c1.x, c1.y);
    printf("Coordenadas de c2: (%lf,%lf)\n",c2.x, c2.y);
}
```

Ponteiros e Registros

- Podemos fazer alocação dinâmica de um vetor de registros da mesma forma que com tipos simples.

```
struct Aluno *vetAlu;  
vetAlu = malloc(10 * sizeof(struct Aluno));  
vetAlu[0].nota = 5.6;  
vetAlu[1].nota = 7.8;  
...
```

Ponteiros e Registros

Utilizando as funções criadas anteriormente podemos executar o exemplo:

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

struct Aluno{
    char nome[80];
    float nota;
};

struct Aluno leAluno();
void imprimeAluno(struct Aluno a);
void listarTurma(struct Aluno turma[], int n);

int main(){
    struct Aluno *vetAlu;
    int n, i;

    printf("Numero de alunos: ");
    scanf("%d", &n); getchar();

    vetAlu = malloc(n*sizeof(struct Aluno)); //Alocação dinâmica do vetor de registros

    for(i=0; i<n; i++){
        vetAlu[i] = leAluno();
    }
    listarTurma(vetAlu, n);

    free(vetAlu); //Liberação de memória alocada
}
```

Exercício

- Crie um novo tipo de registro para armazenar coordenadas no plano cartesiano.
- Crie uma função para imprimir um ponto do tipo criado.
- Crie uma função para cada uma destas operações: soma de dois pontos, subtração de dois pontos, multiplicação por um escalar.

Informações Extras: Redefinido um tipo

- Às vezes, por questão de organização, gostaríamos de criar um tipo próprio nosso, que faz exatamente a mesma coisa que um outro tipo já existente.
- Por exemplo, em um programa onde manipulamos médias de alunos, todas as variáveis que trabalhassem com nota tivessem o tipo **nota**, e não **double**.

Informações Extras: O comando typedef

- A forma de se fazer isso é utilizando o comando `typedef`, seguindo a sintaxe abaixo:

```
typedef <tipo_ja_existente> <tipo_novo>;
```

- Usualmente, fazemos essa declaração fora da função `main()`, embora seja permitido fazer dentro da função também.
- Ex: `typedef float nota;`
Cria um novo tipo, chamado `nota`, cujas variáveis desse tipo serão pontos flutuantes.

Informações Extras: Exemplo de uso do typedef

```
#include <stdio.h>

typedef double nota;

int main(){
    nota p1;
    printf(" Digite a nota:");
    scanf("%lf",&p1);
    printf("A nota digitada foi: %lf",p1);
}
```

Informações Extras: Exemplo de uso do typedef

- Mas o uso mais comum para o comando **typedef** é para a redefinição de tipos registro.
- No nosso exemplo de **struct Aluno**, poderíamos redefinir este tipo para algo mais simples como simplesmente **Aluno**:
 - ▶ **typedef struct Aluno Aluno;**

```

#include <stdio.h>

struct Aluno {
    int ra;
    double nota;
};
typedef struct Aluno Aluno; //redefinimos tipo struct Aluno como Aluno

int main (){
    Aluno turma[10];
    int i;    double media;

    for (i = 0; i < 10; i++) {
        printf ("Digite o RA do %dº aluno: ", i);
        scanf ("%d", &turma[i].ra);
        printf ("Digite a média do %dº aluno: ", i);
        scanf ("%lf", &turma[i].nota);
    }

    //calcula a media da turma
    media = 0.0;
    for (i = 0; i < 10; i++) {
        media = media + turma[i].nota;
    }
    media = media/10.0;
    printf("\nA media da turma é: %lf\n",media);
}

```