

Structs em C

Aula 12

Marcos Silvano Almeida

marcossilvano@professores.utfpr.edu.br

Departamento de Computação

UTFPR Campo Mourão

Registro ou Tipo Estruturado

- Um **registro** permite definir um tipo composto de vários campos
 - Permite agrupar dados de uma mesma “entidade” ou “objeto”.
- Exemplos:

Registro Pessoa:

Nome

CPF

E-mail

Telefone

Endereço

Registro Aluno:

RA

Nome

E-mail

Curso

Coeficiente

Período

Registro Ponto:

X

Y

- Essencialmente, usamos registros para agrupar dados relacionados

Registros em C: struct

- Na linguagem C, o tipo registro é definido por **struct** (*structure*)

Registro Pessoa:

Nome	<code>char name[51];</code>
CPF	<code>char cpf[12]; // 000.000.000-00</code>
E-mail	<code>char email[51];</code>
Telefone	<code>char phone[12]; // 4491234-5678</code>
Endereço	<code>char address[101];</code>
	<code>};</code>

Registro Ponto:

X	<code>float x;</code>
Y	<code>float y;</code>
	<code>};</code>

Tamanho do struct:

≈ tamanhos dos
campos +
alinhamento

Structs: acesso aos campos

- A variável declarada como um tipo struct contém todos os campos:
 - Acesso ocorre via: `variável.campo`

```
struct Point {  
    float x;  
    float y;  
};
```

```
int main() {  
    struct Point point;  
    point.x = 5.6f;  
    point.y = 2.9f;  
    printf("ponto: %.2f, %.2f\n", point.x, point.y);  
    return 0;  
}
```

Structs: campos como strings

- As strings precisam ser copiadas para a struct usando strcpy()

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

```
struct person {
    int id;
    char name[51];
};
```

```
int main() {
    struct Person p1;
    p1.id = 5;
    strcpy(p1.name, "John Doe");
    return 0;
}
```

Typedef

- Podemos utilizar typedef para apelidar uma struct e encurtar sua declaração

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

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

```
typedef struct person {
```

```
    int id;
```

```
    char name[51];
```

```
} Person;
```

```
int main() {
```

```
    Person p1;
```

```
    p1.id = 5;
```

```
    strcpy(p1.name, "John Doe");
```

```
    return 0;
```

```
}
```

Structs: inicialização e atribuição

- Variáveis struct são tratadas como primitivas (**diferente de vetores**)
 - Possível inicializar, atribuir (copiar) e retornar de função
 - Como parâmetros de funções, são passadas por valor
 - Exceto se definirmos parâmetros por endereço

```
int main() {  
    struct Person p1 = {5, "John Doe"};           // inicialização  
  
    struct Person p2 = p1;                         // atribuição de variável: p2 tem cópia de p1  
    p1 = (struct Person){7, "Joanna Dark"};       // atribuição de literal  
  
    struct Person p3 = {name: "Finn, The Human", id: 4};  
  
    printf("id: %d, name: %s\n", p1.id, p1.name);  
    printf("id: %d, name: %s\n", p2.id, p2.name);  
    printf("id: %d, name: %s\n", p3.id, p3.name);  
    return 0;  
}
```

```
struct person {  
    int id;  
    char name[51];  
};
```

Primitivos vs Structs vs Vetores

Primitivo/escalar: int, float, char	Struct	Vetor/String
✓ Inicializar	✓ Inicializar	✓ Inicializar
✓ Atribuir	✓ Atribuir	✗ Deve atribuir elemento(s)
✓ Comparar	✗ Deve comparar campo(s)	✗ Deve comparar elemento(s)
✓ Passar valor para função	✓ Passar valor para função	✗ Passar valor para função
✓ Passar endereço para função	✓ Passar endereço para função	✓ Passar endereço para função

OBS: Vetor/String ⇒ para atribuir (copiar) ou comparar é necessária função que realize a operação, elemento a elemento.

Parte 2

Passando e retornando **structs** de funções

Structs para Funções: passagem por valor

```
// Passagem por valor:
//   p recebe o valor da variável passada à função
//   (parâmetro p contém uma cópia do argumento p1)
void print_person(struct Person p) {
    printf("Person: %d, %s, %s\n", p.id, p.name, p.email);
}

int main() {
    struct Person p1 = {5, "John Doe", "john@email.com"};

    // valor do argumento p1 será copiado para o parâmetro p
    print_person(p1);

    return 0;
}
```

```
struct person {
    int id;
    char name[51];
    char email[51];
};
```

Retornando struct de função

// Retorna uma cópia de struct Point pela função

```
struct Point create_point(float x, float y) {
```

```
    struct Point p;
```

```
    p.x = (int)(x + 0.5); // arredondado
```

```
    p.y = (int)(y + 0.5);
```

```
    return p;
```

```
}
```

```
void print_point(struct Point p) {
```

```
    printf("point: {x:%.2f, y:%.2f}\n", p.x, p.y);
```

```
}
```

```
int main() {
```

```
    struct Point p1;
```

```
    p1 = create_point(2.4, 5.8); // p1 recebe uma cópia do struct retornado
```

```
    print_point(p1);
```

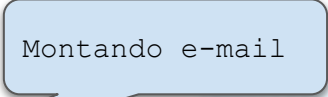
```
    return 0;
```

```
}
```

```
struct Point {  
    float x;  
    float y;  
};
```

Retornando struct de função

```
struct Person create_person(int id, char name[]) {  
    struct Person p;  
    p.id = id;  
    strcpy(p.name, name);  
  
    strcpy(p.email, name);  
    for (int i = 0; p.email[i] != '\0'; i++)  
        if (p.email[i] >= 'A' && p.email[i] <= 'Z') // maiúscula -> minúscula  
            p.email[i] += 32;  
        else if (p.email[i] == ' ') p.email[i] = '_'; // substitui ' ' por '_'  
    strcat(p.email, "@email.com"); // concatena @email.com  
    return p; // retorna cópia  
}  
  
int main() {  
    struct Person p2 = create_person(3, "Jane");  
    print_person(p2);  
}
```



Structs para funções: passagem por endereço

// Passagem por endereço: p recebe o endereço da variável passada como
// argumento à função. Podemos alterar a variável externa com p.

```
void set_zero(struct Point* p) {  
    p->x = 0; // Observe o emprego de "->" ao invés de "."  
    p->y = 0;  
}
```

```
int main() {  
    struct Point p1 = {2.4f, 5.8f};  
    set_zero(&p1); // << endereço da variável p1  
    print_point(p1);  
  
    return 0;  
}
```

Structs para funções: passagem por endereço

// Passagem por endereço: p recebe o endereço da variável passada à função.

```
void capitalize(char name[]) {  
    int space = 1; // garante que o primeiro nome possa ser modificado  
    for (int i = 0; name[i] != '\0'; i++) {  
        if (name[i] == ' '){  
            space = 1;  
        }  
        else {  
            // havia espaço antes da letra atual: início de um nome  
            if (space == 1) {  
                if (name[i] >= 'a' && name[i] <= 'z')  
                    name[i] -= 32; // diferença entre 'a' e 'A'  
            }  
            space = 0;  
        }  
    }  
}
```

Structs para funções: passagem por endereço

```
// Utilizando a função do slide anterior, que define todas as iniciais dos  
// nomes para maiúsculas.
```

```
int main() {  
    struct Person p1 = {1, "joe doe samson", "joe.sam@email.com"};  
  
    // Como o campo Person.name é um vetor (variável do tipo endereço),  
    // basta o passamos à função  
    capitalize(p1.name);  
  
    printf("name: %s\n", p1.name);  
  
    return 0;  
}
```



Parte 3

Vetores de structs

Vetores de structs

- Um vetor armazena uma sequência de dados de um mesmo tipo.

- Vetor de tipos primitivos:

```
int v1[5] = {1,2,3,4,5};
```

- Vetor de vetor (multidimensional):

```
int v[3][2] = { {11,2}, {-5,7}, {0,9} };
```

```
int v[2][3][2] = { { {11,2},{-5,7},{0,9} }, { {4,-2},{0,77},{12,8} } };
```

- Vetor de struct:

```
struct Person {  
    int id;  
    char name[51];  
};
```

```
struct Person v[3]= { {id:1, name:"John"}, {2, "Joanna"}, {3, "Finn"} };
```

Acessando/Modificando elementos

```
struct Person v[3] = { {1, "John"}, {2, "Joanna"}, {3, "Finn"} };
```

```
printf("Elemento 1: {%d, %s}\n", v[0].id, v[0].name); // acesso à elemento
```

```
v[1].id = 5; // modificando elemento do vetor (opção 1)  
strcpy(v[1].name, "Jane");
```

```
struct Person p = v[2]; // modificando elemento do vetor (opção 2)  
p.id = 9;  
strcpy(p.name, "Jake");  
v[2] = p;
```

```
for (int i = 0; i < 3; i++) { // percorrendo vetor  
    printf("{%d, %s}\n", v[i].id, v[i].name);  
}
```

Passando vetores de structs para funções

```
void generate_points(int n, struct Point v[n]) {  
    for (int i = 0; i < n; i++) {  
        struct Point p;  
        p.x = (float)rand()/RAND_MAX * 2 - 1; // -1.0 à 1.0  
        p.y = (float)rand()/RAND_MAX * 2 - 1;  
        v[i] = p;  
    }  
}
```

```
void print_points(int n, struct Point v[n]) {  
    for (int i = 0; i < n; i++) {  
        printf("{%.2f, %.2f} ", v[i].x, v[i].y);  
    }  
    printf("\n");  
}
```

```
// Início do programa  
int main() {  
    struct Point points[50];  
    generate_points(50,  
points);  
    print_points(50, points);  
    return 0;  
}
```

Um exemplo interessante
Gerando struct Person randomicamente

Gerando “pessoas” randomicamente

```
void generate_people(int n, struct Person v[n]) {
    for (int i = 0; i < n; i++)
        v[i] = generate_random_person(); // No próximo slide >>
}

void print_people(int n, struct Person v[n]) {
    for (int i = 0; i < n; i++) {
        printf("{%d, %s, %s}\n", v[i].id, v[i].name, v[i].email);
    }
    printf("\n");
}

int main() {
    struct Person people[50];
    generate_people(50, people);
    print_people(50, people);
    return 0;
}
```

Gerando struct Person randomicamente

```
struct Person generate_random_person() {  
    const char names[6][20] = {"John", "Kena", "Jake", "Joe", "Ada", "Joanna"};  
    int id = rand() % 50 + 1; // 1 à 50  
  
    char name[51] = "";  
    int number_of_names = rand() % 3 + 1; // sorteia quantidade de nomes  
    for (int i = 0; i < number_of_names; i++) {  
        strcat(name, names[rand() % 6]);  
        strcat(name, " ");  
    }  
    name[strlen(name)-1] = '\\0'; // remove o último espaço  
  
    struct Person p = create_person(id, name); // OBS: função do slide 12  
    return p;  
}
```

Referências

- Algoritmos e Programação
 - Marcela Gonçalves dos Santos
 - Disponível pelo Moodle
- Estruturas de Dados, Waldemar Celes e José Lucas Rangel
 - PUC-RIO - Curso de Engenharia
 - Disponível pelo Moodle
- Linguagem C, Silvio do Lago Pereira
 - USP - Instituto de Matemática e Estatística
 - Disponível pelo Moodle
- Curso Interativo da Linguagem C
 - <https://www.tutorialspoint.com/cprogramming>