UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Structs em C

Aula 12

Marcos Silvano Almeida
marcossilvano@professores.utfpr.edu.br
Departamento de Computação
UTFPR Campo Mourão

BCC31A

Registro ou Tipo Estruturado

- Um **registro** permite definir um tipo composto de vários campos
  - Permite agrupar dados de uma mesma entidade.

- Exemplos:

```
Registro Pessoa:        Registro Aluno:        Registro Ponto:
    Nome                    RA                     X
    CPF                     Nome                   Y
    E-mail                  E-mail
    Telefone                Curso
    Endereço                Coeficiente
                            Período
```

- 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
    CPF
    E-mail
    Telefone
    Endereço
```

```
struct Person {
    char name[51];
    char cpf[12]; // 000.000.000-00
    char email[51];
    char phone[12]; // 4491234-5678
    char address[101];
};
```

```
Registro Ponto:
    X
    Y
```

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

```
Tamanho do struct:
------------------
Soma dos tamanhos
dos campos.
```

Structs: acesso aos campos

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

```c
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()

```
struct Person {
    char name[51];
    char cpf[15]; // 000.000.000-00
    char phone[14]; // 44 91234-5678
    char address[101];
};
int main() {
    struct Person p1;
    strcpy(p1.name, "John Doe");
    strcpy(p1.cpf, "876.456.001-45");
    strcpy(p1.phone, "44 98760-4567");
    strcpy(p1.address, "Rua das Rudas, 647, Pequenópolis");
    return 0;
}
```

Structs: inicialização e atribuição

- Variáveis struct são tratadas da mesma forma que variáveis primitivas
  - Possível inicializar e atribuir (copiar)
  - Quando passadas como parâmetros de funções, são copiadas
    - Exceto se passarmos seu endereço
    - Podemos retornar a cópia de uma struct de uma função
  - Desta forma, strucs são diferentes de vetores

```c
struct Person {
    int id;
    char name[51];
    char email[51];
};
int main() {
    struct Person p1 = {5, "John Doe", "john@email.com"};// inicialização
    struct Person p2 = p1; // cópia de p1
    return 0;
}
```

Primitivos vs Structs vs Vetores

Primitivos: int, float, char	Struct	Vetor/String
✅Inicializar	✅Inicializar	✅Inicializar
✅Atribuir (copiar)	✅Atribuir (copiar)	⛔ Atribuir (copiar)
✅Comparar	✅Comparar campo	✅Comparar elemento
✅Passar cópia para função	✅Passar cópia para função	⛔ Passar cópia para função
✅Passar endereço para função	✅Passar endereço para função	✅Passar endereço para função

OBS: Vetor/String: para atribuir (copiar) e comparar, somente utilizando função que realizar a operação elemento a elemento.

Parte 2

Passando e retornando **structs** em funções

Structs para Funções: passagem por valor

```c
// Passagem por valor: p recebe o valor da variável passada à função
void printPerson(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"};
    printPerson(p1);

    return 0;
}
```

Retornando struct de função

```c
// Retornando uma cópia de struct Point pela função
struct Point createPoint(float x, float y) {
    struct Point p;
    p.x = (int)(x + 0.5); // arredondado
    p.y = (int)(y + 0.5);
    return p;
}
void printPoint(struct Point p) {
    printf("point: {x:%.2f, y:%0.2f}\n", p.x, p.y);
}
int main() {
    struct Point p1;
    p1 = createPoint(2.4, 5.8);
    printPoint(p1);
    return 0;
}
```

Retornando struct de função

```
// Retornando uma cópia de struct Person pela função
struct Person createPerson(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++)        // substitui ' ' por '_'
        if (p.email[i] == ' ')
            p.email[i] = '_';
    strcat(p.email, "@email.com");
    return p;
}
int main() {
    struct Person p2 = createPerson(3, "Jane");
    printPerson(p2);
    ...
```

Structs para funções: passagem por endereço

```
// Passagem por endereço: p recebe o endereço da variável passada à função.
// Podemos alterar a variável externa com p.
void setZero(struct Point* p) {
    p->x = 0;
    p->y = 0;
}

int main() {
    struct Point p1;
    p1 = createPoint(2.4, 5.8);
    printPoint(p1);

    return 0;
}
```

Structs para funções: passagem por endereço

```
// Passagem por endereço: p recebe o endereço da variável passada à função.
void capitalizeWords(struct Person* p) {
    int space = 1; // garante que o primeiro nome possa ser modificado
    for (int i = 0; p->name[i] != '\0'; i++) {
        if (p->name[i] == ' ')
            space = 1;
        else {
            // havia espaço antes da letra atual: início de um nome
            if (space == 1) {
                if (p->name[i] >= 'a' && p->name[i] <= 'z')
                    p->name[i] -= 32; // diferença entre 'a' e 'A'
            }
            space = 0;
        }
    }
}
```

Structs para funções: passagem por endereço

```
// Utilizando a função do slide anterior.
// A função coloca as iniciais dos nomes em maiúsculo.

int main() {
    struct Person pe1 = {1, "joe doe samson", "joe.sam@email.com"};

    // A função espera receber um endereço de struct Person.
    // Logo, devemos usar o operador & antes da variável.
    fixName(&pe1);

    printPerson(pe1);

    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:

```
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"} };
```

Acessanto/Modificando elementos

```c
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);
}
```

Um exemplo interessante

Gerando struct Person randomicamente

Passando vetores de structs para funções

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

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

Gerando “pessoas” randomicamente

```
void generatePeople(int n, struct Person v[n]) {
    for (int i = 0; i < n; i++)
        v[i] = generateRandomPerson();
}
void printPeople(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];
    generatePeople(50, people);
    printPeople(50, people);
    return 0;
}
```

Gerando struct Person randomicamente

```
struct Person generateRandomPerson() {
    const char names[7][20]= {"John", "Kena", "Jake", "Joe", "Ada", "Joanna"};
    int id = rand() % 50 + 1; // 1 à 50

    char name[51] = "";
    int numberOfNames = rand()% 3 + 1; // sorteia quantidade de nomes
    for (int i = 0; i < numberOfNames; i++) {
        strcat(name, names[rand()%7]);
        strcat(name, " ");
    }
    name[strlen(name)-1] = '\0';        // remove o último espaço

    struct Person p = createPerson(id, name);
    stringToLower(p.email);
    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