

Structs

Prof. Jefferson T. Oliva

Algoritmos e Estrutura de Dados I (AE22CP)
Engenharia de Computação
Departamento Acadêmico de Informática (Dainf)
Universidade Tecnológica Federal do Paraná (UTFPR)
Campus Pato Branco

- Problema
- *Struct*
- Operações com *Struct*
- Erros Comuns em *Structs*

- Tipos básicos de dados escalares

- int

```
int i = 2020;
```

- char

```
char c = 'm';
```

- float

```
float r = 101.5;
```

- double

```
double d = 8002.01;
```

- Estruturas de dados (tipos compostos) homogêneas
 - Vetores
 - Matrizes
 - Strings

10
15
32
8
23
1

1	4	6	0
4	1	4	5
3	9	1	1

'a'	'b'	'c'
-----	-----	-----

- Vetores

```
int vec[6] = {2, 4, 8, 7, 9, 1};
```

ou

```
int vec[6];
```

```
vec[0] = 2;
```

```
vec[1] = 4;
```

```
vec[2] = 8;
```

```
vec[3] = 7;
```

```
vec[4] = 9;
```

```
vec[5] = 1;
```

- Matrizes

```
int mat[2][2] = {{1, 0},{0, 1}};
```

ou

```
int mat[2][2];
```

```
mat[0][0] = 1;
```

```
mat[0][1] = 0;
```

```
mat[1][0] = 0;
```

```
mat[1][1] = 1;
```

- String

```
char str[30] = "hermes e renato";
```

ou

```
char str[30] = {'h', 'e', 'r', 'm', 'e', 's', ' ', ' ', 'e', ' ',  
' ', 'r', 'e', 'n', 'a', 't', 'o'};
```

ou

```
char str[30];
```

```
strcpy(str, "hermes e renato"); // função da biblioteca  
string.h
```

Problema

- Como organizar um conjunto informações heterogêneas?
 - Exemplo: cadastro de pessoa física
 - Nome, data de nascimento, rg, level no D&D, etc
- Possível solução: criando vetores e matrizes para cada dado
 - Localizar pelo índice
 - Manter estruturas avulsas dentro do mesmo código

Problema

```
int main(void){
    int qtd = 0;
    char resp = 's';
    int dbz = 4;
    int caracteres = 15;
    char nome[dbz][caracteres + 1];
    int level[dbz];
    char atk1[dbz][caracteres + 1];
    char atk2[dbz][caracteres + 1];

    do{
        printf("deseja incluir um novo lutador (s/n)?");
        scanf(" %c", &resp);

        if (resp == 's'){
            printf("Informe o nome do seu lutador: ");
            scanf(" %s", nome[qtd]);
            printf("Nivel: ");
            scanf(" %d", &level[qtd]);
            printf("ataque 1: ");
            scanf(" %[^\n]s", atk1[qtd]);
            printf("ataque 2: ");
            scanf(" %[^\n]s", atk2[qtd]);
            qtd++;
        }
    }while((resp == 's') && (qtd < dbz));

    return 0;
}
```

Problema

- Problemas com a implementação anterior
 - Difícil organizar e alterar os dados
 - Dificuldade em manter integridade entre os dados e seus índices

Problema

- Como agrupar diferentes tipos de dados em uma única estrutura?

Problema

- Como agrupar diferentes tipos de dados em uma única estrutura?
- Solução: definir uma estrutura heterogênea
 - Na linguagem C, estruturas heterogêneas podem ser definidas pelo comando *struct*

Struct

- Um registro (*struct*) é um conjunto de variáveis (provavelmente de tipos diferentes)
- Cada variável é um campo do registro
- São estruturas de dados heterogêneas
- Permite criar tipos de dados personalizados

- Cada campo do registro possui o seu próprio identificador

```
struct nome_registro{  
    tipo1 campo1;  
    tipo2 campo2;  
    ...  
    tipoN campoN;  
};
```


- Exemplo

```
struct cdz{
    char nome[101];
    char cavaleiro_de[15];
    double cosmo;
    int nivel;
};

int main(void){
    struct cdz seiya;

    return 0;
}
```

- A estrutura não é alocada na memória
 - É apenas introduzida como um novo tipo de dados
- Por convenção, as estruturas são declaradas próximas ao topo do arquivo
- Todo o código poderá declarar e utilizar variáveis dos novos tipos de dados
- Declaração de uma variável do tipo *struct*

```
struct nome_estrutura nome_variavel;
```

- Ao declarar uma variável *struct*, na memória é alocada uma quantidade suficiente de espaço

- Para evitar do uso da palavra *struct* em cada declaração de variável, pode ser utilizada a palavra reservada *typedef*

```
typedef tipo novo_nome;  
  
typedef struct nome_estrutura nome_simplificado;
```

- Exemplo:

```
struct aluno{  
    char nome[101];  
    int RA;  
    float coef;  
};  
  
typedef struct aluno Aluno;  
  
int main(void){  
    Aluno a; // em vez de usar struct aluno a;  
  
    return 0;  
}
```

- O typedef pode ser usado de forma mais "direta"

```
typedef struct aluno{
    char nome[101];
    int RA;
    float coef;
}Aluno;

int main(void){
    Aluno a;

    return 0;
}
```

- O typedef pode ser usado de forma mais "direta" (2)
 - Essa forma de definição é conhecida como "*struct* anônima"

```
typedef struct {  
    char nome[101];  
    int RA;  
    float coef;  
}Aluno;  
  
int main(void){  
    Aluno a;  
  
    return 0;  
}
```

Operações com *Struct*

- Inicialização de *struct*

```
typedef struct aluno{
    char nome[101];
    int RA;
    float coef;
}Aluno;

int main(void){
    Aluno a = {"Renato", 1234567, 0.986};

    return 0;
}
```

- Para inicializar, os elementos do registro devem ser declarados na ordem em que a *struct* foi definida

- Acesso aos elementos membros de uma *structs*

```
typedef struct aluno{
    char nome[101];
    int RA;
    float coef;
}Aluno;

int main(void){
    Aluno a = {"Renato", 1234567, 0.986};
    printf("%s - %d - %f\n", a.nome, a.RA, a.coef);
    return 0;
}
```


- Atribuição entre *structs*

```
typedef struct aluno{
    char nome[101];
    int RA;
    float coef;
}Aluno;

int main(void){
    Aluno a = {"Renato", 1234567, 0.986};
    printf("%s - %d - %f\n", a.nome, a.RA, a.coef);
    Aluno b = a;
    printf("%s - %d - %f\n", b.nome, b.RA, b.coef);
    return 0;
}
```

- Para operações com variáveis *struct*, as mesmas devem ser instâncias da mesma estrutura

- Vetores de *struct*
 - É possível agrupar um conjunto de *structs*
 - Cada registro em sua respectiva posição terá o seu conjunto de variáveis

- Vetor de *struct*

```
typedef struct aluno{
    char nome[101];
    int RA;
    float coef;
}Aluno;

int main(void){
    int i;
    Aluno a[10];
    for (i = 0; i < 10; i++){
        printf("Nome:  ");
        scanf(" %[^\\n]s", a[i].nome);
        printf("RA: ");
        scanf(" %d", &a[i].RA);
        printf("Coeficiente:  ");
        scanf(" %f", &a[i].coef);
    }
    return 0;
}
```

- Argumento de função *struct*

```
typedef struct aluno{
    char nome[101];
    int RA;
    float coef;
}Aluno;

void imprime_dados_aluno(Aluno a){
    printf("\n Nome:  %s", a.nome);
    printf("\n RA:  %d", a.RA);
    printf("\n Coeficiente:  %f\n", a.coef);
}

int main(void){
    int i;
    Aluno a[10];
    for (i = 0; i < 10; i++){
        printf("Nome:  ");
        scanf("%[^\n]s", a[i].nome);
        printf("RA:  ");
        scanf("%i", &a[i].RA);
        printf("Coeficiente:  ");
        scanf("%f", &a[i].coef);
    }

    for (i = 0; i < 10; i++)
        imprime_dados_aluno(a[i]);

    return 0;
}
```

- Definindo funções que retornam *Structs*

```
typedef struct aluno{
    char nome[101];
    int RA;
    float coef;
}Aluno;

Aluno cadastra_aluno(char nome[101], int RA, float coef){
    Aluno a;

    strcpy(a.nome, nome);
    a.RA = RA;
    a.coef = coef;

    return a;
}

int main(void){
    int i;
    Aluno a = cadastra_aluno("Dona Maxima", 006671, 0.87);

    return 0;
}
```

- Aninhamento de structs
 - Uma estrutura pode conter outra estrutura como um dos seus campos
 - Exemplo: cadastro de uma pessoa
 - Nome
 - RG
 - CPF
 - Data de nascimento
 - Endereço
 - Contato

- Aninhamento de structs

```
typedef struct data{
    int dia, mes, ano;
}Data;

typedef struct endereco{
    char rua[151];    int nro;
    char bairro[41];
    char cep[9];
    char cidade[41];
    char estado[41];
}Endereco;

typedef struct contato{
    char tel[11];
    char cel[12];
    char email[101];
}Contato;

typedef struct pessoa{
    char nome[121];
    char rg[9];
    char cpf[12];
    Data data_nasc;
    Endereco endereco;
    Contato contato;
}Pessoa;
```

- Aninhamento de structs: o acesso às informações segue a ordem do aninhamento

```
int main(void){
    Pessoa p;

    printf("Nome:  ");
    scanf(" %[^\\n]s", p.nome);
    printf("RG: ");
    scanf(" %s", p.rg);
    printf("CPF: ");
    scanf(" %s", p.cpf);
    printf("Data de nascimento\\nDia:  ");
    scanf(" %d", p.data_nasc.dia);
    printf(":  ");
    scanf(" %d", p.data_nasc.mes);
    printf(":  ");
    scanf(" %d", p.data_nasc.ano);
    printf(":  ");

    ...

    return 0;
}
```


Erros Comuns em Structs

```
struct Algo{  
    int v1;  
    float v1;  
}
```

- Sintaxe:

```
struct nome_registro{  
    tipo1 campo1;  
    tipo2 campo2;  
    ...  
    tipoN campoN;  
};
```

- Correção: adicionar o ";" faltante

```
struct Algo{  
    int v1;  
    float v1;  
};
```

Erros Comuns em Structs

```
struct Algo{  
    int v1 = 1;  
    float v1;  
};
```

- *Struct* não permite inicialização seus respectivos campos
- Solução: criar uma variável e inicializar o respectivo campo

Erros Comuns em Structs

```
struct Algo{  
    int v1;  
    float v1;  
};  
int main(){  
    Algo a;  
    return 0;  
}
```

- Erro de sintaxe: o compilador não conhece "Algo"

- Soluções:

```
int main(){  
    struct Algo a;  
    return 0;  
}
```

ou use typedef, como no exemplo abaixo

```
typedef struct Algo{  
    int v1;  
    float v1;  
}Algo;
```

Erros Comuns em Structs

```
typedef struct {  
    int v1;  
    float v1;  
}Algo;  
int main(){  
    struct Algo a;  
    return 0;  
}
```

- Nesse caso, houve renomeação "direta" da estrutura, ou seja, foi usada uma definição anônima de struct

- Soluções:

```
int main(){  
    struct Algo a;  
    return 0;  
}
```

ou não deixe a definição da *struct* de forma anônima

```
typedef struct Algo{  
    int v1;  
    float v1;  
}Algo;
```

- Atribuição entre *structs* com os mesmos campos

```
struct Algo1{
    int v1;
    float v2;
};
struct Algo2{
    int v1;
    float v2;
};
int main(){
    struct Algo1 a;
    struct Algo2 b;
    a = b;
    return 0;
}
```

- Apesar de ambas *structs* possuírem os mesmos campos e na mesma ordem, o compilador "entende" que ambas definições são tipos diferentes

- Atribuição entre *structs* com os mesmos campos
 - Solução 1: atribuição entre variáveis do mesmo tipo

```
int main() {  
    struct Algo1 a, c;  
    struct Algo2 b;  
    a = c;  
    return 0;  
}
```

- Solução 2: atribuição por campo

```
int main() {  
    struct Algo1 a;  
    struct Algo2 b;  
    a.v1 = b.v1;  
    a.v2 = b.v2;  
    return 0;  
}
```

Erros Comuns em Structs

- Vetores de *struct*





```
typedef struct{
    int v1;
    float v1;
} Algo;

int main(){
    Algo v[10];
    v.v1[0] = 1;
    v.v2[0] = 1.0;
    v.v1[1] = 2;
    v.v1[1] = 2.0;
    ...
    return 0;
}
```

- v (*struct*) foi declarado como vetor, não seu s campos

- Solução:

```
int main(){
    Algo v[10];
    v[0].v1 = 1;
    v[0].v2 = 1.0;
    v[1].v1 = 2;
    v[1].v1 = 2.0;
    ...
    return 0;
}
```

-  Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; Clifford, S.
Algoritmos: teoria e prática.
Elsevier, 2012.
-  Pereira, S. L.
Estrutura de Dados e em C: uma abordagem didática.
Saraiva, 2016.
-  Tenenbaum, A.; Langsam, Y.
Estruturas de Dados usando C.
Pearson, 1995.
-  Ziviani, M.
Projetos de Algoritmos: com implementações em Pascal e C.
Thomson, 2004.