Notas de aula – Algoritmos e Estrutura de Dados 1 (AE42CP) – structs
Prof. Jefferson T. Oliva

Vocês devem ter visto vários tipos de dados escalares (cujas variáveis podem conter apenas um valor): int, char, float, double, long, bool…

Também vocês já viram estruturas de dados homogêneas (tipos compostos, conjuntos de dados), como vetores, matrizes e strings. Para essas estruturas existem várias formas para declaração e atribuição de valores. Exemplo para vetores:

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

ou

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

int vec[]; // está errado
```

ou

```
int vec[5];
vec[0] = 1;
vec[1] = 2;
vec[2] = 3;
vec[3] = 4;
vec[4] = 5;
```

Como é feita a declaração de matrizes? Um exemplo: int mat[2][2] = {{1, 2},{3, 4}};

ou

```
int mat[2][2];
mat[0][0] = 1;
mat[0][1] = 2;
mat[1][0] = 3;
mat[1][1] = 4;
```

Como o é feita a declaração de strings?

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

ou

```
char str[30] = {'h', 'e', 'r', 'm', 'e', 's'};
```

ou

```
char str[30];
strcpy(str, “hermes”);
```

Os conjuntos que vimos até aqui são para dados heterogêneos.

Problema: Como organizar um conjunto de informações heterogêneas? Exemplo: cadastro de pokemons. Que informações podemos utilizar? Nome, nível, tipo, ataque 1, ataque 2, ataque 3, ataque 4. Possível solução: criar vetores e matrizes para cada dado

```
int main(void){
    int qtd = 0, i;
    char resp = 's';
    int pokemons = 6;
    int caracteres = 15;
    char nome[pokemons][caracteres + 1];
    int level[pokemons];
    char item[pokemons][caracteres + 1];
    char atk1[pokemons][caracteres + 1];
    char atk2[pokemons][caracteres + 1];
    char atk3[pokemons][caracteres + 1];
    char atk4[pokemons][caracteres + 1];

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

        if ((resp == 's') && (qtd < pokemons)){
            printf("\nNome: ");
            scanf(" %[^\n]s", nome[qtd]);
            printf("\nNivel: ");
            scanf("%d", &level[qtd]);
            printf("\nitem: ");
            scanf(" %[^\n]s", item[qtd]);
            printf("\nataque 1: ");
            scanf(" %[^\n]s", atk1[qtd]);
            printf("\nataque 2: ");
            scanf(" %[^\n]s", atk2[qtd]);
            printf("\nataque 3: ");
            scanf(" %[^\n]s", atk3[qtd]);
            printf("\nataque 4: ");
            scanf(" %[^\n]s", atk4[qtd]);
            qtd++;
        }
    }while(resp == 's');

    return 0;
}
```

Quais os problemas com essa implementação?
- Difícil gerenciar os dados: organizar e alterar
- Dificuldade de manter integridade entre os dados e seus índices

Pergunta: como agrupar diferentes tipos de dados em uma estrutura?
A solução é a definição de uma estrutura de dados heterogênea. Para isso, na linguagem C há um comando para isso: struct (registro)

Um registro é um conjunto de variáveis de diversos tipos agrupadas em uma única estrutura. Cada campo de um struct é uma variável. O comando struct permite criar tipos de dados (estruturas) personalizados.

Cada campo do registro possui o seu próprio identificador.

Sintaxe para definição de uma struct

```
struct nome{
    tipo1 nome1;
    …
    tipoN nomeN;
};
```

Exemplo:

```
struct cliente{
    char nome[101];
    char sexo;
    int registro;
    double renda;
};
```

Por convenção, structs são declaradas fora das funções e próximas do topo do arquivo

Também, ao definir uma estrutura, a mesma não é alocada na memória, mas sim introduzida como um novo tipo. Mas, ao declarar uma variável struct, a mesma é alocada na memória, onde é alocado espaço suficiente.

Assim, em todo o código podem ser utilizadas variáveis dos novos tipos de dados

Declaração de uma variável do tipo struct: struct nome_estrutura = nome_variável;

Exercício em sala de aula: definir uma estrutura para representar um livro.

Para evitar o uso da palavra struct em cada declaração, pode ser utilizada a palavra reservada **typedef**, que é utilizada para renomear variáveis (exempo: typedef int integer;). Em uma struct, o comando pode ser aplicado na seguinte forma: typedef struct nome_estrutura novo_nome;

Exemplos:

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

typedef struct aluno Aluno;
```

ou

```
typedef struct aluno Aluno;

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

Ou você pode ser mais direto:

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

ou

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

Exemplo de declaração para o novo tipo:

```
Aluno a;
```

Operações com structs

Após a definição de novas estruturas, diversas operações podem ser realizadas, como a inicialização.

Por exemplo, aproveitando a estrutura Aluno:

```
Aluno a = {“Renato”, 123456, 0.986};
```

É importante ressaltar que os elementos do registro devem ser declarados na ordem em a struct foi definida, caso a atribuição seja no formato apresentado no parágrafo anterior.

O acesso aos elementos da struct é feita da seguinte forma: estrutura.campo

Exemplo:

```
Aluno a = {“Renato”, 123456, 0.986};
printf(“Nome: %s\n”, a.nome);
printf(“RA: %d\n”, a.RA);
```

```
printf("Coeficiente: %d\n", a.coef);
```

Em structs podem ser feitas atribuições. Exemplo:

```
Aluno a, b;
strcpy(a.nome, "Renato");
a.RA = 123456;
a.coef =  0.986;

b = a;
```

Obs.: para operações entre variáveis structs, as mesmas devem ser instâncias da mesma estrutura. Caso existam dois tipos de estruturas (A e B) com os mesmos campos, nas variáveis do tipo A não podem ser atribuídas as do tipo B.

Exemplo:

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


typedef struct{
    char nome[101];
    int RA;
    float coef;
}Estudante;

Aluno a;
Estudante e;

a = e; // erro de compilação, pois, por mais que Estudante e Aluno contenham os mesmos tipos de dados, ambas estruturas são consideradas diferentes
```

As atribuições abaixo são validas?

```
typedef struct{
  int a;
}A;

typedef A B;

int main() {
   A s1;
   B s2;

   s1.a = 10;

   s2 = s1; // é válido, pois foram definidos dois nomes para um mesmo tipo
}
```

Também é possível definir vetores de structs para agrupá-los em um conjunto. Cada registro terá o seu conjunto. Exemplo:

```
Aluno alunos[10];

for (i = 0; i < 10; i++){
    printf("\n\nNome: ");
    scanf("%[^\n]s", a[i].nome);
    printf(" \nRA: ");
    scanf(" %d", &a[i].RA);
    printf("\nCoeficiente: ");
    scanf(" %f", &a[i].coef);
}
```

Structs podem ser usados como argumento de funções. Exemplos

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

Funções podem retornar structs. Exemplo:

```
Aluno inicializar_aluno(char nome[], int RA, float coef){
     Aluno a;

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

     return a;
}
```

Por fim, structs podem ser aninhados. Exemplo:

```
typedef struct{
    int dia, mes, ano;
}Nasc;

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

O acesso ao conteúdo da struct interna pode ser feito na seguinte forma:

```
printf("Dia: %d\n", a.data_nasc.dia);
printf("Mes: %d\n", a.data_nasc.mes);
printf("Ano: %d\n", a.data_nasc.ano);
```

Structs com aninhamento anônimo. Exemplo:

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

O acesso ao conteúdo da struct anônima pode ser feito na seguinte forma:

```
printf("Dia: %d\n", a.dia);
printf("Mes: %d\n", a.mes);
printf("Ano: %d\n", a.ano);
```

Referências

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.