Fundamentos da Programação

Enumerações e Tipos estruturados

P.PORTO
ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

Conteúdo

- Enumerações
- Definição de tipos
- Registos

Enumerações

- As enumerações permite criar variáveis que apenas podem tomar um conjunto finito de valores.
  - Exemplo: dias da semana, estado civil, …
- Uma enumeração é na realidade um conjunto de valores inteiros representados por identificadores que têm que ser únicos.

```
…
enum dia { DOM, SEG, TER, QUA, QUI, SEX, SAB };
enum estadoCivil { SOL, CAS, VIU, DIV };
…
```

Enumerações

- O primeiro identificador assume o valor **0**.
  - Os restantes assumem valores consecutivos.

```c
#include <stdio.h>

enum dia { DOM, SEG, TER, QUA, QUI, SEX, SAB };

int main() {
    enum dia hoje = DOM;

    printf("\n%d", hoje++);

    if (hoje == SEG) {
        printf("\nHoje é segunda.");
    }

    return 0;
}
```

Enumerações

- Os valores da enumeração podem ser definidos explicitamente com o **=**.
  - Os restantes assumem valores consecutivos.
- É possível indicar valores específicos para cada identificador.

Cria uma nova enumeração **mes** em que os identificadores (**JAN**, **FEV**, etc.) correspondem aos inteiros de **1** a **12**.

```
…
enum mes {JAN = 1, FEV, MAR, ABR, MAI, JUN,
          JUL, AGO, SET, OUT, NOV, DEZ
};
enum curso { LEI = 123, LSIRC = 456 };
…
```

Enumerações

```c
#include <stdio.h>

#define MENSAGEM_ESTADO_CIVIL "O seu estado civil: %s"

int main() {
    unsigned int estado;

    printf("Indique o seu estado civil [0 a 3]: ");
    scanf("%u", &estado);

    switch (estado) {
        case 0:
            printf(MENSAGEM_ESTADO_CIVIL, "Solteiro");
            break;
        case 1:
            printf(MENSAGEM_ESTADO_CIVIL, "Casado");
            break;
        case 2:
            printf(MENSAGEM_ESTADO_CIVIL, "Viuvo");
            break;
        case 3:
            printf(MENSAGEM_ESTADO_CIVIL, "Divorciado");
            break;
    }

    return 0;
}
```

```c
#include <stdio.h>

#define MENSAGEM_ESTADO_CIVIL "O seu estado civil: %s"

enum EstadoCivil { SOLTEIRO, CASADO, VIUVO, DIVORCIADO };

int main() {
    enum EstadoCivil estado;

    printf("Indique o seu estado civil [0 a 3]: ");
    scanf("%u", &estado);

    switch (estado) {
        case SOLTEIRO:
            printf(MENSAGEM_ESTADO_CIVIL, "Solteiro");
            break;
        case CASADO:
            printf(MENSAGEM_ESTADO_CIVIL, "Casado");
            break;
        case VIUVO:
            printf(MENSAGEM_ESTADO_CIVIL, "Viuvo");
            break;
        case DIVORCIADO:
            printf(MENSAGEM_ESTADO_CIVIL, "Divorciado");
            break;
    }

    return 0;
}
```

typedef

- Pode ser usado para dar um novo nome a um tipo.

```c
#include <stdio.h>

typedef enum { false, true } bool;
typedef float nota;

int main() {
    nota teste1 = 10, teste2 = 10;
    bool passou = false;

    if (teste1 >= 9.5 && teste2 >= 9.5) {
        passou = true;
    }

    return 0;
}
```

Registos

- Permite a criação de registos criando variáveis que combinam dados de diferentes tipos (dados estruturados).

```c
#include <stdio.h>
int main() {
    int diaHoje, mesHoje, anoHoje;
    char diaExtensoHoje[10];
    int diaAmanha, mesAmanha, anoAmanha;
    char diaExtensoAmanha[10];
    return 0;
}
```

```c
#include <stdio.h>

struct Data {
    int dia, mes, ano;
    char diaExtenso[10];
};

int main() {
    struct Data hoje, amanha;
    return 0;
}
```

Definição

```
struct nome_registo {
        tipo nome1, nome2;
        ...
        tipo nomeN;
};
```

- **nome_registo**: nome a dar ao registo.
- **tipo**: tipo de dados do elemento
  - Exemplo.: **int**, **float**, **char**, registo, enumeração, …
- **nomeX**: nome do elemento (também conhecido por campo ou membro do registo).

```
…
struct Data {
    int dia, mes, ano;
};
…
```

Definição

- A definição de uma estrutura não corresponde à declaração de uma nova variável, mas sim à definição de um novo tipo de dados que pode ser usado para declarar variáveis.
  - Depois de definirmos uma estrutura podemos criar variáveis desse novo tipo de dados.

Matrícula: 17-50-BP					**Ano:** 2000				
Combustível: Diesel					**Cilindrada:** 1.9 L				
Nº velocidades: 5					**Potência:** 80 cv				
Últimos abastecimentos (litros)									
40,2	27,5	12,0	38,1	21,7					

```
struct automovel {
    char matricula[9];
    int ano, potencia, numeroVelocidades;
    enum combustivel comb;
    float cilindrada, ultimosAbast[ABAST_TAM];
};
…
struct automovel meu_carro, carro_empresa;
struct automovel frota[25];
```

Criação de tipos

- Podemos criar tipos através da definição de registos.

```c
#include <stdio.h>

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

int main() {
    Data data[5];
    return 0;
}
```

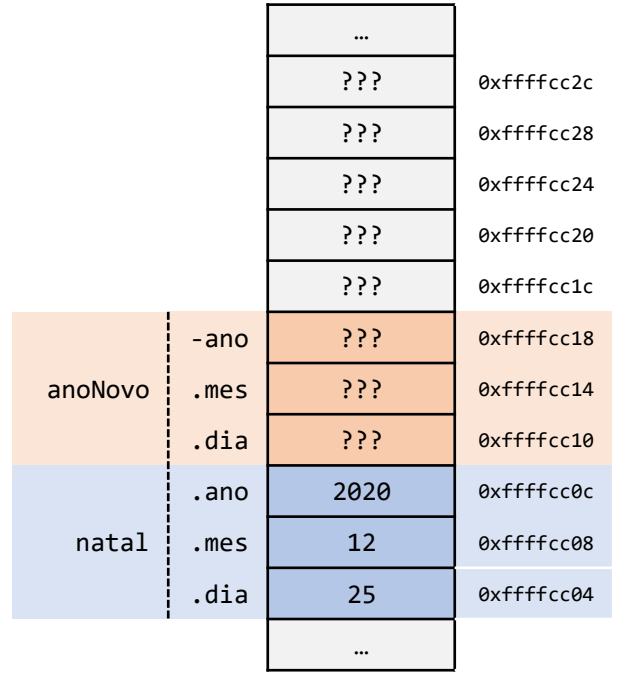
```c
#include <stdio.h>

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

int main() {
    Data data[5];
    return 0;
}
```

```c
#include <stdio.h>

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

int main() {
    Data data[5];
    return 0;
}
```

Acesso

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

- O acesso aos campos é realizado com o operador **.** (ponto).




```
…
Data natal, anoNovo;
natal.dia = 25;
natal.mes = 12;
natal.ano = 2020;
…
```

Inicialização

- Um registo pode ser inicializado quando é declarado.
  - Pela ordem em que foram definidos.
  - Utilizando o operador de acesso (`.`).

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

```
…
Data natal = {25, 12, 2020};
Data anoNovo = {.ano = 2020, .mes = 1, .dia = 1};
…
```

Copiar

- Podemos copiar todo o conteúdo dos elementos de um registo para outro registo como se fosse uma qualquer variável usando a atribuição (**=**).

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

```
…
Data vespera = {24, 12, 2020}, natal;
natal = vespera;
natal.dia++;
…
```

Comparar

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

- Não é possível fazer comparações diretas entre registos.
  - Apenas se podem comparar os seus elementos.

```
…
Data hoje = {25, 12, 2020}, natal = {25, 12, 2020};
if (hoje == natal) {
    puts("Hoje é natal.");
}
…
```

```
…
Data hoje = {25, 12, 2020}, natal = {25, 12, 2020};
if (hoje.dia == natal.dia && hoje.mes == natal.mes
     && hoje.ano == natal.ano) {
    puts("Hoje é natal.");
}
…
```

Registos de registos

- Podemos ter registos com registos.

```
#include <stdio.h>

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

typedef struct {
    int numDias;
    Data dia[31];
} Mes;

int main() {
    int i;
    Mes janeiro = { .numDias = 31 };

    for (i = 0; i < janeiro.numDias; i++) {
        janeiro.dia[i].dia = i + 1;
        janeiro.dia[i].mes = 1;
        janeiro.dia[i].ano = 2021;
    }

    for (i = 0; i < janeiro.numDias; i++) {
        printf("\n%d-%d-%d", janeiro.dia[i].dia,
                             janeiro.dia[i].mes,
                             janeiro.dia[i].ano);
    }

    return 0;
}
```

Apontadores

- Um apontador também pode ser do tipo **`struct`**.
- Existem duas formas distintas mas equivalentes de aceder aos membros:

  `(*apontador_struct).elemento`

- ou

  `apontador_struct->elemento`

```c
#include <stdio.h>

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

Data obtemData(int dia, int mes, int ano) {
    Data data = {.dia = dia, .mes = mes, .ano = ano};
    return data;
}

void incrementaData(Data *data) {
    data->dia++;
}

void decrementaData(Data *data) {
    (*data).dia--;
}

void imprimeData(Data data) {
    printf("\n%d-%d-%d", data.dia, data.mes, data.ano);
}

int main() {
    Data anoNovo, natal = {.dia=25, .mes=12, .ano=2020};
    anoNovo = obtemData(1, 1, 2021);

    incrementaData(&natal);

    imprimeData(natal);
    imprimeData(anoNovo);

    return 0;
}
```

Leitura recomendada

- (**Capítulo 11, 12**) Damas, L. Linguagem C; FCA – Editora de Informática, Lda, 1999; ISBN 9789727221561.