C

Guilherme P. Telles

IC

30 de março de 2023

struct, union, enum

Registros

Registro

- Um registro é um conjunto de dados que não precisam ser do mesmo tipo.
- Cada elemento de um registro é chamado de *campo*.

- A definição típica de registros em C tem a forma

```
struct identificador {
  defs-de-campos;
  ...
  defs-de-campos;
};
```

- Os campos são definidos da mesma forma que variáveis.

- Por exemplo:

```
struct item {
  char tipo;
  int valor, situacao;
};
```

- Esta definição cria o tipo `struct item`.

Operadores

- O operador de acesso aos campos de um registro é `.`
- O operador de acesso aos campos de um registro apontado é `->`.
- O operador de atribuição = pode ser usado com registros e causa a atribuição dos valores de todos os campos.
- Outros operadores devem ser aplicados a cada campo, um a um.

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

struct tarefa {
  char descricao[30];
  int duracao;
};

struct pessoa {
  char nome[50];
  int idade;
  struct tarefa tarefas[5];
};


int main(void) {

  struct pessoa p1;

  strcpy(p1.nome,"J. Pinto Fernandes");
  p1.idade = 51;
  strcpy(p1.tarefas[0].descricao,"fazer lab 6");
  p1.tarefas[0].duracao = 1;

  struct pessoa p2;
  p2 = p1;

  strcpy(p2.nome,"Neo Anderson");
  p2.idade = 31;

  printf("%s, %d, %s, %d\n", p1.nome, p1.idade, p1.tarefas[0].descricao, p1.tarefas[0].
      duracao);
  printf("%s, %d, %s, %d\n", p2.nome, p2.idade, p2.tarefas[0].descricao, p2.tarefas[0].
      duracao);
```

Outras formas de definição

```
struct carta {
  char naipe;
  int valor;
} c1, c2, baralho[52];
```

```
struct carta {
  char naipe;
  int valor;
};

typedef struct carta carta;

carta c3, c4;
```

```c
struct {
  char naipe;
  int valor;
} c1, c2;
```

```c
typedef struct {
  char naipe;
  int valor;
} carta;
```

Composição

- Um registro pode agrupar um número arbitrário de dados de tipos diferentes.
- Vetores, registros e apontadores também podem ser membros de registros.
- É possível definir um apontador dentro de um registro que é do seu próprio tipo:

```
struct qualquer {
  ...
  struct qualquer *outro;
}
```

- Dessa forma podemos organizar dados conectando-os por apontadores.

Escopo

- Os nomes dos membros de uma registro devem ser distintos.
- Registros distintos podem ter membros com nomes iguais.
- Os nomes de registros estão em um espaço de nomes próprio.

```
struct aux {
int i;
};

float aux; /* funciona */
```

Inicialização de registros

- Registros podem ser inicializados de forma similar aos vetores:

```
struct ponto{
  int x;
  int y;
};

struct ponto p1 = {220,110};
struct ponto p2 = {110};   // y = 0
```

Registros e funções

- Uma registro é sempre passado por valor (todos os membros, incluindo vetores e registros são copiados).
- Registros podem ser retornados por funções. Eles são retornados por valor.

```
ponto equidistante(ponto p1, ponto p2);
```

Uniões

União

- Uma união define um conjunto de campos que serão armazenados numa porção compartilhada da memória, isto é, apenas um campo será armazenado de cada vez.
- É responsabilidade do programador interpretar corretamente o dado armazenado em uma união.
- O espaço alocado é suficiente para armazenar o maior dos seus campos.
- A sintaxe é similar à de `struct`.

exm-union-1.c

```c
#include <stdio.h>


int main(void) {

  union int_or_float {
    int i;
    float f;
  };

  union int_or_float n;

  n.i = -123456789;
  printf("%d %e\n", n.i, n.f);

  n.f = 1234.0;
  printf("%d %f\n", n.i, n.f);

  return 0;
}
```

- Tem dois usos principais:
  - economizar espaço de memória em situações onde queremos quardar apenas um dado e o tipo dele pode variar.
  - permitir a construção de estruturas de dados flexíveis em relação ao tipo de dados que armazenam (não é a única forma de fazer isso em C).

```c
typedef union {
  char c;
  int i;
  double d;
} multiple;

typedef struct {
  char type;
  multiple data;
} item;

item V[100];

V[0].type = 'D';
V[0].data.d = 3.14;
```

Enumerações

Enumeração

- Uma enumeração é um tipo de dados que pode assumir um conjunto restrito de valores.
- Em C, são valores inteiros que têm um nome.

- A definição típica de uma enumeração tem a forma

```
enum id {lista-de-identificadores};
```

- Por exemplo

```
enum dias {dom,seg,ter,qua,qui,sex,sab};
```

- Esta definição cria o tipo `enum dias`.
- Os enumeradores são os identificadores `dom,...,sab`.
- Os enumeradores são constantes `int` com valores 0, 1, ...

exm-enum-1.c

```c
int main() {

  enum dias {dom,seg,ter,qua,qui,sex,sab};
  enum dias d1, d2;

  d1 = dom;
  d2 = qui;

  if (d1 == d2)
    ;

  while (d1 < sab) {
    ...
    d1++;
  }

  return 0;
}
```

Outras formas de definição

```
enum cor {azul,verde} c1,c2;
```

```
enum cor {azul=4,verde,roxo=3,rosa};
```

o que implica verde=5, rosa=4 e azul==rosa.

```
enum {azul,verde} c1,c2;
```

```
typedef enum cor {azul,verde} cor;
```

```
typedef enum {azul,verde} cor;
```

Escopo

- Os enumeradores estão no mesmo espaço de nomes das variáveis.

```
enum cor {azul,verde};
float azul; /* nao funciona */
```

- Os nomes de enumerações estão em um espaço de nomes próprio.

```
enum cor {azul,verde};
float cor; /* funciona */
```

Enumerações e funções

- Uma enumeração pode ser passada como parâmetro e retornada por uma função.

```
typedef enum {cavalo,boi,pato,burro} bicho;

bicho funcao(int dia, bicho um_bicho);
```