Linguagem C

Enumerações, registros e uniões

g.p. telles

Enumerações

- A definição 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, ...
- Exemplo: `exm-enum-1.c`

Variações na 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 {boi,camelo,pato} E;

E funcao(int i, E e);
```


Registros

- A definição de registros tem a forma

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

- Os campos são definidos da mesma forma que variáveis.
- Por exemplo:

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

Operadores

- O operador de acesso aos campos é o ponto `.` para registros e a seta `->` para apontadores para registro.
- Exemplo: `exm-struct-1.c`
- O operador de atribuição `=` pode ser usado com registros e causa a atribuição de valores campo a campo.
- Outros operadores devem ser aplicados campo a campo.

Variações na definição

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

typedef struct carta carta;

carta c3, c4;
```

Variações na definição

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

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


Composição

- A 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 {
  definições-de-campos;
  ...
  struct qualquer *proximo;
}
```

Escopo

- Os nomes dos membros de uma registro devem ser distintos.
- Registros distintas 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 inicializadas 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 passada por valor e todos os membros, incluindo vetores e registros são copiados.
- Registros podem ser retornadas por funções. Elas são retornadas por valor.

Uniões

- Uma união define um conjunto de membros que serão armazenados numa porção compartilhada da memória, isto é, apenas um membro 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 membros.
- A sintaxe é similar à do `struct`.
- Exemplo: `exm-union-1.c`