Tipos enumerados

Video desta aula

Enumerações, ou tipos enumerados, são tipos de dados definidos pelo programador. Uma variável de tipo enumerado pode assumir um valor dentre um conjunto fixo de valores possíveis previamente definido.

Um tipo enumerado é visto como um caso especial de tipo inteiro. Variáveis de tipos enumerados são implementadas pelo compilador usando valores inteiros, mas isso é transparente para o programador.

Declaração e uso

Um exemplo simples de declaração de tipo enumerado:

```
// definição do tipo enumerado
enum Color_t { BLACK, BLUE, GREEN, RED, YELLOW, WHITE } ;

// declaração de variável do tipo enum Color_t
enum Color_t c1;
```

A definição do tipo enumerado também pode ser feita usando `typedef`:

```
typedef enum { BLACK, BLUE, GREEN, RED, YELLOW, WHITE } Color_t ;

Color_t c1;
```

Variáveis enumeradas podem ser usadas como variáveis inteiras:

```
// atribuição
c1 = GREEN ;

// uso
if (c1 == RED) { ... }

// os valores enumerados têm uma ordem definida
if (c1 >= BLUE && c1 <= YELLOW)  { ... }

// variáveis enumeradas podem ser incrementadas
c1 = BLACK ;
while (c1 <= WHITE)
{
  ...
  c1++ ;
}

// um laço for enumerado
for (c1 = BLACK; c1 <= WHITE; c1++)
```

```
{
  ...
}
```

Por default, o primeiro valor de um tipo enumerado vale 0 (zero), o segundo valor vale 1 e assim por diante. Esses valores podem ser alterados se isso for conveniente:

```
typedef enum {
  BLACK = 1,
  BLUE,
  GREEN,
  RED = 10,
  YELLOW,
  WHITE
} Color_t ;
```

Os valores inteiros associados à enumeração são:

valor simbólico	valor original	valor com atribuição
BLACK	0	1
BLUE	1	2
GREEN	2	3
RED	3	10
YELLOW	4	11
WHITE	5	12

Tipos enumerados podem ser usados para tornar o código mais legível e também diminuir erros devidos a valores imprevistos em variáveis. Um exemplo clássico de uso de tipos enumerados é a definição de um tipo booleano em C:

```
typedef enum
{
  FALSE = 0,
  False = 0,
  false = 0,
  TRUE  = 1,
  True  = 1,
  true  = 1
} bool_t ;

bool_t flag ;
```

Imprimindo tipos enumerados

Internamente, um tipo enumerado é representado por um inteiro, os valores simbólicos (WHITE, RED, ...) são descartados durante a compilação. Por isso, **não é possível imprimir o valor simbólico** diretamente, apenas seus valores internos:

print_enum.c

```
#include <stdio.h>

// tipo "Cor"
```

```
typedef enum { BLACK, BLUE, GREEN, RED, YELLOW, WHITE } Color_t ;

int main ()
{
  Color_t c1;

  // ...
  c1 = BLUE ;

  printf ("Cor %s\n", c1) ;    // errado!
  printf ("Cor %d\n", c1) ;    // correto!
}
```

Uma forma simples de contornar esse problema consiste em usar um vetor de nomes:

color-array.c

```
#include <stdio.h>

// tipo "Cor"
typedef enum { BLACK, BLUE, GREEN, RED, YELLOW, WHITE } Color_t ;

// nome das cores
char *color_name[] = {"black", "blue", "green", "red", "yellow", "white" } ;

int main ()
{
  Color_t c1;

  // ...
  c1 = BLUE ;

  printf ("Cor %s\n", color_name[c1]) ;
}
```

A solução acima só funciona se a enumeração usar os valores numéricos *default* (0, 1, 2, 3, ...).

Uma solução mais robusta consiste em usar uma estrutura `switch`:

color-switch.c

```
#include <stdio.h>

// tipo "Cor"
typedef enum { BLACK, BLUE, GREEN, RED, YELLOW, WHITE } Color_t ;

// define o nome da cor
char* color_name (Color_t c)
{
  switch (c)
  {
    case BLACK   : return ("black")  ;
```

```
    case BLUE    : return ("blue")   ;
    case GREEN   : return ("green")  ;
    case RED     : return ("red")    ;
    case YELLOW  : return ("yellow") ;
    case WHITE   : return ("white")  ;
    default      : return ("") ;
  }
}

int main ()
{
  Color_t c1;

  // ...
  c1 = BLUE ;

  printf ("Cor %s\n", color_name(c1)) ;
}
```

Exercícios

1. Escreva um programa em C capaz de classificar os alunos de uma determinada matéria de acordo com sua situação. O programa deve ser capaz de:
    1. Ler os dados do aluno referentes a sua nota na disciplina e sua frequência.
    2. Classificar o aluno pelas enumerações `Aprovado`, `Exame Final` ou `Reprovado` de acordo com os dados obtidos.
2. Escreva um programa em C capaz de receber um número e classificá-lo, por meio de tipos enumerados, entre número par ou impar; positivo, negativo ou zero; palíndromo ou não-palíndromo; múltiplo de 10 ou não múltiplo;
3. Escreva um programa em C que leia uma determinada sequência de caracteres, que seja capaz de classificar cada um deles entre `letra`, `pontuação` e `caractere especial` e guarde os dados em uma estrutura. Para o caso de letras, o programa deve ainda classificá-las, por meio de enumerações, entre `maiúsculas` e `minúsculas`; `vogais` e `consoantes`.