Capítulo 6

Tipos enumerados

Além dos tipos de dados pré-definidos no C++, os chamados tipos básicos, podem-se criar tipos de dados adicionais. Essa é, aliás, uma das tarefas fundamentais da programação centrada nos dados. Para já, abordar-se-ão extensões mais simples aos tipos básicos: os tipos enumerados. No próximo capítulo falar-se-á de tipos de primeira categoria, usando classes C++

Uma variável de um tipo enumerado pode conter um número limitado de valores, que se enumeram na definição do tipo[1]. Por exemplo,

```
enum DiaDaSemana {
    segunda_feira,
    terça_feira,
    quarta_feira,
    quinta_feira,
    sexta_feira,
    sábado,
    domingo
};
```

define um tipo enumerado com sete valores possíveis, um para cada dia da semana. Convencionalmente no nome dos novos tipos todas as palavras começam por uma letra maiúscula e não se usa qualquer caractere para as separar.

O novo tipo utiliza-se como habitualmente. Pode-se, por exemplo, definir variáveis do novo tipo[2]:

```
DiaDaSemana dia = quarta_feira;
```

Pode-se atribuir atribuir à nova variável `dia` qualquer dos valores listados na definição do tipo enumerado `DiaDaSemana`:

[1]Esta afirmação é uma pequena mentira "piedosa". Na realidade os enumerados podem conter valores que não correspondem aos especificados na sua definição [12, página 77].

[2]Ao contrário do que se passa com as classes, os tipos enumerados sofrem do mesmo problema que os tipos básicos: variáveis automáticas de tipos enumerados sem inicialização explícita contêm lixo!

```
dia = terça_feira;
```

Cada um dos valores associados ao tipo `DiaDaSemana` (viz. `segunda_feira`, ..., `domingo`) é utilizado como se fosse um valor literal para esse tipo, tal como `10` é um valor literal do tipo `int` ou `'a'` é um valor literal do tipo `char`. Convencionalmente nestes valores literais as palavras estão em minúsculas e usa-se um sublinhado (_) para as separar[3].

Como se trata de um tipo definido pelo programador, não é possível, sem mais esforço, ler valores desse tipo do teclado ou escrevê-los no ecrã usando os métodos habituais (viz. os operadores de extracção e inserção em canais: >> e <<). Mais tarde ver-se-á como se pode "ensinar" o computador a extrair e inserir valores de tipos definidos pelo utilizador de, e em, canais.

Na maioria dos casos os tipos enumerados são usados para tornar mais claro o significado dos valores atribuídos a uma variável. Por exemplo, `segunda_feira` tem claramente mais significado que `0`. Na realidade, os valores de tipos enumerados são representados como inteiros atribuídos sucessivamente a partir de zero. Assim, `segunda_feira` tem representação interna 0, `terça_feira` tem representação 1, etc. De facto, se se tentar imprimir `segunda_feira` o resultado será surgir `0` no ecrã, que é a sua representação na forma de um inteiro. É possível associar inteiros arbitrários a cada um dos valores de uma enumeração, pelo que podem existir representações idênticas para valores com nome diferentes:

```
enum DiaDaSemana { // agora com nomes alternativos...
    primeiro = 0,  // inicialização redundante: o primeiro valor é sempre 0.
    segunda = primeiro,
    segunda_feira = segunda,
    terça,
    terça_feira = terça,
    quarta,
    quarta_feira = quarta,
    quinta,
    quinta_feira = quinta,
    sexta,
    sexta_feira = sexta,
    sábado,
    domingo,
    último = domingo,
};
```

Se um operando de um tipo enumerado ocorrer numa expressão, será geralmente convertido num inteiro. Essa conversão também se pode explicitar, escrevendo `int(segunda_feira)`, por exemplo. As conversões opostas também são possíveis, usando-se `DiaDaSemana(2)`, por exemplo, para obter `quarta_feira`. Na próxima secção ver-se-á como redefinir os operadores existentes na linguagem de modo a operarem sobre tipos enumerados sem surpresas

[3]Existe uma convenção alternativa em que os valores literais de tipos enumerados e os nomes das constantes se escrevem usando apenas maiúsculas com as palavras separadas por um sublinhado (e.g., `SEGUNDA_FEIRA`). Desaconselha-se o uso dessa convenção, pois confunde-se com a convenção de dar esse tipo de nomes a macros, que serão vistas no Capítulo 9.

desagradáveis para o programador (pense-se no que deve acontecer quando se incrementa uma variável do tipo `DiaDaSemana` que contém o valor `domingo`).

6.1 Sobrecarga de operadores

Da mesma forma que se podem sobrecarregar nomes de funções, i.e., dar o mesmo nome a funções que, tendo semanticamente o mesmo significado, operam com argumentos de tipos diferentes (ou em diferente número), também é possível sobrecarregar o significado dos operadores usuais do C++ de modo a que tenham um significado especial quando aplicados a tipos definidos pelo programador. Se se pretender, por exemplo, sobrecarregar o operador ++ prefixo (incrementação prefixa) para funcionar com o tipo `DiaDaSemana` definido acima, pode-se definir uma rotina[4] com uma sintaxe especial:

```
DiaDaSemana operator ++ (DiaDaSemana& dia)
{
    if(dia == último)
        dia = primeiro;
    else
        dia = DiaDaSemana(int(dia) + 1);

    return dia;
}
```

ou simplesmente

```
DiaDaSemana operator ++ (DiaDaSemana& dia)
{
    if(dia == último)
        return dia = primeiro;
    else
        return dia = DiaDaSemana(int(dia) + 1);
}
```

A única diferença relativamente à sintaxe habitual da definição de funções e procedimentos é que se substitui o habitual nome do procedimento pela palavra-chave `operator` seguida do operador a sobrecarregar[5]. Não é possível sobrecarregar todos os operadores, ver Tabela 6.1.

O operador foi construído de modo a que a incrementação de uma variável do tipo `DiaDaSemana` conduza sempre ao dia da semana subsequente. Utilizou-se `primeiro` e `último` e

[4]Este operador não é, em rigor, nem um procedimento nem uma função. Não é um procedimento porque não se limita a incrementar: devolve o valor do argumento depois de incrementado. Não é uma função porque não se limita a devolver um valor: altera, incrementando, o seu argumento. É por esta razão que o operador ++ tem efeitos laterais, podendo a sua utilização descuidada conduzir a expressões mal comportadas, com os perigos que daí advêm (Secção 2.7.8).

[5]Em todo o rigor o operador de incrementação prefixa deveria devolver uma referência para um `DiaDaSemana`. Veja-se a Secção 7.7.1.

Tabela 6.1: Operadores que é possível sobrecarregar.

+	-	*	/	%	xor	bitand
bitor	compl	not	=	<	>	+=
-=	*=	/=	%=	^=	&=	|=
<<	>>	<<=	>>=	==	!=	<=
>=	and	or	++	--	->*	,
->	[]	()	new	new[]	delete	delete[]

não `segunda_feira` e `domingo`, pois dessa forma pode-se mais tarde decidir que a semana começa ao `Domingo` sem ter de alterar o procedimento acima, alterando apenas a definição da enumeração.

Este tipo de sobrecarga, como é óbvio, só pode ser feito para novos tipos definidos pelo programador. Esta restrição evita redefinições abusivas do significado do operador + quando aplicado a tipos básicos como o `int`, por exemplo, que poderiam ter resultados trágicos.