

## 11. FUNÇÕES *FRIEND*

Uma das principais razões para se usar técnicas de programação orientada a objetos é o isolamento de dados dentro de classes. Fazendo assim, apenas funções membro podem obter acesso a valores críticos. Na maioria das vezes, esconder dados dentro de classes dá aos programas uma boa medida de controle, impedindo a modificação indiscriminada de valores críticos pelos processos em andamento.

Como muitas regras em programação, entretanto, aquelas do isolamento de dados são feitas para serem quebradas. Em C++, pode-se romper a proteção aos membros de uma classe usando os *friends*, ou amigos. As classes do C++ podem declarar dois tipos de *friends*. Uma classe inteira pode ser *friend* de uma outra classe, ou uma única função pode ser declarada como sendo um *friend*. Uma classe *friend* pode ser qualquer outra classe no sistema, geralmente fora da hierarquia da classe a que pertence *friend*. Uma função *friend* pode ser qualquer uma do programa, incluindo funções membros de outra classe [4, 9]. Esta seção descreve as funções *friend*.

Uma função declarada como *friend* em uma classe, recebe acesso aos membros privados e protegidos desta classe. Se essa função for membro de uma outra classe, como é o normal, apenas essa função, e nenhum outro membro da classe, receberá permissão de acesso aos membros privados da classe que faz a declaração.

Declarando uma função específica como amiga de duas classes, dá-se a essa função acesso aos campos privados e protegidos de instâncias de ambas as classes. A função *friend* pode ser uma função global do C++ ou um membro de uma outra classe. Em um projeto típico, a função *friend* declara parâmetros das duas classes para as quais ela deve a sua amizade. Dentro da função *friend*, as instruções podem então acessar normalmente membros escondidos nos parâmetros passados como argumentos para a função. O exemplo a seguir demonstra como declarar e usar esse gênero típico de função *friend*.

```
#include <iostream>
using namespace std;
class dois;
class um {
    friend void exhibe (um &c1, dois &c2);
private:
    char *s1;
public:
    um() { s1 = "Testando..."; }
};
class dois {
    friend void exhibe (um &c1, dois &c2);
private:
    char *s2;
public:
    dois() { s2 = "Um, Dois, Tres!"; }
};
void main () {
    um c1;
    dois c2;
    exhibe (c1, c2);
}
void exhibe (um &c1, dois &c2) {
    cout << c1.s1 << c2.s2 << '\n';
}
```

O exemplo anterior declara duas classes, "um" e "dois". Como cada uma dessas classes irá se referir à outra e porque o C++ requer que sejam declarados os identificadores antes de usá-los, a classe "dois" é declarada antes da "um". Declarar uma classe antecipadamente diz ao compilador para aceitar referências ao nome da classe antes dela ser declarada formalmente. As declarações da função "exibe" listam os parâmetros de referência "c1" e "c2" dos dois tipos de classe. Como "exibe" é amigas dessas duas classes, suas instruções podem acessar membros privados e protegidos de argumentos passados para "exibe".

O tipo de acesso a uma função *friend* é qualquer um, ou seja, a declaração desta função pode estar entre os membros privados, públicos ou protegidos de uma classe. Isto não deve causar surpresa, já que uma função *friend* não pertence à classe e, portanto, não tem qualquer declaração de acesso. Isto justifica a posição da declaração da função *friend* nas classes do exemplo anterior.

Torna-se importante salientar que uma função amiga não precisa ser uma função comum em C++ como demonstrado no exemplo anterior. Funções *friend* podem também ser membro de uma classe. Tipicamente, uma classe irá declarar uma função de uma outra classe como amiga. A função *friend* irá então ganhar acesso aos membros privados e protegidos da classe original. O próximo exemplo, semelhante ao anterior, delinea a estratégia básica de funções membro amigas. A comparação dos dois programas revela várias diferenças importantes entre funções *friend* globais e aquelas que são membros de uma classe.

```
#include <iostream>
using namespace std;
class um;
class dois {
    private:
        char *s2;
    public:
        dois() { s2 = "Um, Dois, Tres!"; }
        void exibe (um &c1);
};
class um {
    friend void dois :: exibe (um &c1);
    private:
        char *s1;
    public:
        um() { s1 = "Testando..."; }
};
void dois :: exibe (um &c1) {
    cout << c1.s1 << s2 << "\n";
}
void main () {
    um c1;
    dois c2;
    c2.exibe (c1);
}
```

A primeira regra a ser lembrada sobre funções membro *friend* está relacionada com a ordem das declarações da classe. A classe que prototipa a função membro deve vir antes da classe que declara essa função como sendo um *friend*. Usar uma declaração antecipada não é suficiente.

Uma outra diferença em relação ao primeiro exemplo desta seção é o modo como a função "exibe" faz referências aos dados privados das duas classes. A função agora só possui um parâmetro, "one &c1". Como a função é um membro da classe "dois" não se faz necessário listar um argumento do tipo "dois". De fato, fazê-lo seria um erro. A função está encapsulada dentro de "dois"; portanto, a instrução da função "exibe" pode se referir ao campo privado "s2" de "dois" diretamente. A referência a "c1.s1" é permitida, pois "exibe" é *friend* da classe "um".

Como a função "exibe" é um membro de "dois", ela agora possui um ponteiro *this* que endereça a instância para a qual a função foi chamada. Consequentemente, o programa não pode mais chamar a função *friend* diretamente. Ele agora precisa definir uma variável do tipo "dois" e chamar "exibe" para essa variável.

É importante notar que *friends* são interessantes para dar a objetos acesso aos campos privados de outros objetos. Usadas cuidadosamente, *friends* podem melhorar o desempenho, eliminando o desperdício associado com a chamada a funções membro. Todavia, *friends* também rompem as barreiras que protegem os dados dentro das classes. Usar *friends* reduz as vantagens da programação orientada a objetos advindas do encapsulamento de funções e dados, e do isolamento de instruções que acessam valores críticos. Por isso, deve-se ter muito cuidado para usar funções *friend* [4].