Curso de Ciência da Computação

Algoritmos e Programação de Computadores 2per

Programação Orientada a Objetos POO

Profa. Fernanda dos Santos Cunha

Classes e Funções Amigas

- Em C++, há 2 formas de romper o encapsulamento de uma classe:
  - com herança (classes derivadas podem acessar membros protegidos).
  - **com funções/classes amigas**.

Funções Amigas

- Funções amigas podem acessar os dados privados de uma classe mesmo não sendo membro desta classe (embora a permissão para tal acesso seja dada dentro da classe).
- As funções amigas têm os mesmos privilégios de uma função-membro, mas não estão associadas a um objeto da classe.

Funções Amigas

- Podemos ter como funções amigas:
  - funções independentes/genéricas (não-membro)
  - funções-membro de outra classe
- Funções amigas são especialmente usadas em **sobrecarga de operadores.**

Funções Amigas

- Para criar uma função amiga basta colocar a palavra-chave **friend** no início da declaração da função e passar um objeto da outra classe como argumento da função.
  - O argumento é necessário pois, mesmo quando se dá a uma função amiga o privilégio de acesso aos dados privados da classe, ela não é parte daquela classe, devendo ser informada sobre qual objeto agirá.

Funções Amigas

- Para funções amigas independentes

```
class nome-classe
{
    ...
    friend protótipo-função-amiga
};
```

- Para funções amigas membro de outra classe

```
class nome-classe-A
{
    ...
    friend nome-classe-B :: protótipo-função-amiga
};
```

Funções Amigas

Saída: 19:05:34

```
class Tempo {
  int h, m, s;
public:
  Tempo(int hh, int mm, int ss) { h=hh, m=mm; s=ss; }
  friend char* prnTM(Tempo); // declaração fcao genérica amiga
};
char* prnTM(Tempo tm) {   // converte tm para texto no formato hh:mm:ss
  char* buff = new char[9];
  buff[0]= tm.h/10 + '0';      buff[1]= tm.h%10 + '0';      buff[2]=':';
  buff[3]= tm.m/10 + '0';      buff[4]= tm.m%10 + '0';      buff[5]=':';
  buff[6]= tm.s/10 + '0';      buff[7]= tm.s%10 + '0';      buff[8]='\0';
  return buff;
}
int main(){
   Tempo tm(19,5,34); cout << prnTM(tm) << endl; return 1; }
```

Funções Amigas Como Interface

- Uma função **friend** pode agir como elo entre duas ou mais classes diferentes.

```
class Tempo {
    int h, m, s;
 public:
    Tempo(int hh, int mm, int ss) { h=hh, m=mm; s=ss; }
    friend char* prnTD(Tempo, Data);  // declaração fcao genérica amiga
};

class Data {
    int d, m, a;
 public:
    Data(int dd, int mm, int aa) { d=dd; m=mm; a=aa%100; }
    friend char* prnTD(Tempo, Data);  // declaração fcao genérica amiga
};
```

Funções Amigas Como Interface

Saída: 20:05:20
27/04/2017

```
char* prnTD(Tempo tm, Data dt) {
  char* buff = new char[18];
  buff[0]= tm.h/10 + '0';     buff[1]= tm.h%10 + '0';     buff[2]=':';
  buff[3]= tm.m/10 + '0';     buff[4]= tm.m%10 + '0';     buff[5]=':';
  buff[6]= tm.s/10 + '0';     buff[7]= tm.s%10 + '0';
  buff[8]='\n'; //pular linha
  buff[9]= dt.d/10 + '0';     buff[10]= dt.d%10 + '0';    buff[11]='/';
  buff[12]= dt.m/10 + '0';    buff[13]= dt.m%10 + '0';    buff[14]='/';
  buff[15]= dt.a/10 + '0';    buff[16]= dt.a%10 + '0';    buff[17]='\0';
  return buff;
}
int main(){
   Tempo tm(20,5,20); Data dt(27,4,2017); cout<<prnTD(tm, dt)<<endl;
    return 1;
}
```

Sobrecarga de operadores e funções amigas

- Qdo sobrecarga for por meio de função **friend,** ela deve receber um argumento a mais do que se fosse por meio de método da classe.

```
class Ponto {
   ...
public:
   friend Ponto& operator++ (Ponto&);
   friend Ponto& operator++ (Ponto&, int);
   friend Ponto& operator-- (Ponto&);
   friend Ponto& operator-- (Ponto&, int);
   };
```

ATENÇÃO: as sobrecargas dos operadores binários ++ e -- tb podem ser implementadas via função-membro

Sobrecarga de operadores e funções amigas

```
class Data {
   int d, m, a;
 public:
   Data(int dd, int mm, int aa)
        { d=dd; m=mm; a=aa%100; }
   friend istream& operator >> (istream& is, Data& d);
   friend ostream& operator << (ostream& os, Data& d);
};
```

ATENÇÃO: as sobrecargas dos operadores binários >> e << não podem ser implementadas via função-membro, pq o primeiro operando (cout) não é um objeto da classe construída pelo programador.

Classes Amigas

- Além de ser possível declarar funções independentes como amigas, pode-se declarar uma classe toda **friend** de outra classe.
- **TODAS** as funções-membro da classes serão amigas da outra classe.

Classes Amigas

- A classe X define-se como **friend** da classe Y significa que X libera o acesso aos seus atributos para Y.
  - Quem tem o **friend** sofre o acesso !!!!
- **ATENÇÃO**: mas isso não significa que o inverso é verdadeiro. Somente será se a classe Y declarar-se também **friend** de X.
  - Esta forma de uso garante a **INTEGRIDADE DA CLASSE !!!**

Classes Amigas

Data: 27/04/2017
Hora: 20:05:20

```
class Tempo {
    int h, m, s;
  public:
    Tempo(int hh, int mm, int ss) { h=hh, m=mm; s=ss; }
    friend class Data; // declara-se amiga de Data, libera acesso
};                     // com a classe Data já pronta seria : friend Data;
class Data {
    int d, m, a;
  public:
    Data(int dd, int mm, int aa) { d=dd; m=mm; a=aa%100; }
    void prnDT(Tempo tm) {
          cout << "Data: "<<d<<'/'<<m<<'/'<<a<<endl;
          cout << "Hora: "<<tm.h<<':'<<tm.m<<':'<<tm.s<<endl; }
};
int main(){
  Tempo tm(20,5,20); Data dt(27,4,2017); dt.prnDT(tm); return 1; }
```