Aula 8

Objectivos: Redefinição de operadores e herança.

Problema A1

Altere o problema A1 da aula anterior, criando uma classe abstracta (sem representação interna para a DATA) com a seguinte definição:

```
typedef enum {Dom,Seg,Ter,Qua,Qui,Sex,Sab};

class DATA
{
public:
  virtual int dia(void) = 0;
  // mês
  // ano
  // dia da semana

  virtual const char *nome_do_mes(void); // Pode ter já uma implementação!
  // nome do dia da semana (pode ter também uma implementação)

  virtual void escreve(void);

  virtual void operator ++(int) = 0; // postfix
  // postfix operator --

  virtual void operator +=(int num_dias) = 0;
  // operator -=

  virtual int operator -(DATA &outra);
  virtual DATA &operator =(DATA &outra) = 0;
  // operator ==

  static int e_valida(int dia,int mes,int ano,int dia_da_semana);
  static int dias_do_mes(int mes,int ano);
  static int dias_do_ano(int ano);
  static int ano_bissexto(int ano);
};
```

A1.1. Altere o nome da classe definida e implementada na aula anterior

para `DATA_DIA_MES_ANO`, e faça com que ela seja descendente (especialização) da classe abstracta `DATA`.

(**Nota:** Uma vez que o mecanismo de sobrecarga de métodos (*overloading*) e o de redefinição geram ambiguidades (não facilmente resolúveis em *C++*), mantenha, na classe descendente, exactamente a mesma assinatura dos operadores que tomam como parâmetros `DATAS`.)

A1.2. Faça com que os programas do exercício da aula anterior continuem a funcionar com estas novas classes (minimizando as modificações).

A1.3. Faça uma nova implementação para a classe abstracta `DATA`, usando como representação interna para a data o número de dias desde 1 de Janeiro de 2000 (naturalmente que o valor pode ser negativo). Chame a esta nova classe: `DATA_NUMERO_DIAS`.

A1.4. Verifique se os programas continuam a funcionar se se passar objectos do tipo `DATA_NUMERO_DIAS` onde se esperam ponteiros para `DATA`.

Problema B1

B1.1. Crie uma classe `CData` que inclui três variáveis-membro: dia, mês e ano. Implemente um construtor por defeito que inicialize o objecto com a data de hoje (para tal pode utilizar as funções `time` e `localtime` definidas no `<ctime>`) e um construtor com três argumentos (dia, mês e ano).

B1.2. Implemente as funções `AnoBissexto` (diz se o ano do objecto é bissexto) e `DiasNoMes` (devolve o número de dias no mês do objecto). Redefina os operadores `==`, `!=`, `>`, `<`, `>=`, `<=`, `<<` (visualização), `>>` (introdução), `-` (este operador deve devolver a diferença em dias (pode ser positiva ou negativa) entre duas datas), `++` e `--` (em versões `postfix` e `prefix`). Na implementação do operador `!=` aproveite o operador `==` chamando-o de forma explicita. Na implementação de operadores `<=` e `>=` aproveite também outros operadores disponíveis.

B1.3. Implemente a função `DiaSemana` que imprime no écran o dia de semana do objecto. Redefina os operadores necessários para poder somar uma data e um inteiro `n` (estes devem produzir uma data nova, avançada `n` dias para frente ou para trás, no caso de o número ser negativo).

B1.4. Teste a classe desenvolvida com a função `main` seguinte:

```
int main(int argc, char* argv[])
{
  CData hoje;  cout << "Hoje: " << hoje << endl;  hoje.DiaSemana();
  CData t1 (25, 3, 2003);  t1.DiaSemana();
  CData t2 (1, 03, 2000);  t2.DiaSemana();

  CData d0(20, 3, 2100);  CData d1(20, 3, 2000);
  cout << "d0: "<< d0 << "E um ano bissexto? "<< d0.AnoBissexto() << endl;
  cout << "d1: "<< d1 << "E um ano bissexto? "<< d1.AnoBissexto() << endl;

  cout << "Dif. em dias entre d0 e d1: " << d0-d1 << endl;
  cout << "Dif. em dias entre d1 e d0: " << d1-d0 << endl;

  CData d2(31, 12, 1955);
  cout << "d2: " << d2 << endl;
  cout << "d2++ " << d2++ << endl;
  cout << "d2: " << d2 << endl;

  CData d3(1, 3, 1960);
  cout << "d3: " << d3 << endl;
  cout << "--d3 " << --d3 << endl;
```

```
  CData d4;  cout <<"Especifique uma data (dia, mes, ano)" << endl;
  cin >> d4; cout << "d4: " << d4 << endl;

  cout << "d4 + 14: " << d4 + 14 << endl;
  cout << "2 + d4: " << 2 + d4 << endl;
  cout << "-9 + d4: " << -9 + d4 << endl;

  return 0;
}
```

Problema B2

Responda as perguntas seguintes:

1. A declaração da função seguinte está correcta?

   ```
   class CInt
   {
     public:
       CTest operator+ (const CTest& t1, const CTest& t2);
     ...
   }
   ```

2. No código seguinte indique os números das linhas em quais será chamado o construtor de cópia da classe `CTest` e os números das linhas em quais será chamado o operador `=`. Considere que a implementação da classe `CTest` é correcta.

   ```
   void function (CTest t)       //1
   {                             //2
     CTest t1(/*argumentos*/);   //3
     CTest t2 = t1;              //4
     CTest t3(t);                //5
     CTest t4, t5;               //6
     t4 = t1;                    //7
     t5 = t3;                    //8
   }                             //9
   ```