Aula 7

Objectivos: Redefinição de operadores.

Problema A1

Construa uma classe para manipulação de datas – `DATA` – descritas por: dia, mês, ano e dia da semana. A utilização desta classe deve permitir: a definição e obtenção dos valores do dia/mês/ano, e a sua escrita na consola de saída (com um formato adequado).

A1.1. Implemente os operadores `++` e `--` de forma a que o valor da data seja, respectivamente, incrementado ou decrementado num dia.

A1.2. Implemente um operador `-` relativamente a outra data, que devolva o número de dias entre as mesmas.

A1.3. Implemente o operador `=` relativamente a outra data, que faça a respectiva atribuição do valor (pode fazer com que esse operador tenha como resultado o valor [referência] dessa atribuição, para permitir a atribuição de valores em cadeia).

A1.4. Implemente os operadores `+=` e `-=` relativamente a um número inteiro de dias.

A1.5. Implemente o operador `==` relativamente a outra data, cujo resultado deve ser verdadeiro ou falso consoante as datas são, ou não, iguais.

A1.6. Faça um programa que permita a manipulação de datas (definição de uma valor, incremento, decremento, comparação com outra data, cálculo do número de dias relativamente a outra data, etc.).

A1.7. Faça um programa que escreva na consola de saída o calendário de um qualquer ano (numa única coluna escrever correctamente os 12 meses).

Problema A2

Prosseguindo o problema das figuras das aulas anteriores, acrescente (e experimente) os seguintes operadores às figuras:

```
virtual void operator >>(SIMPLE_GRAPHICAL_OUTPUT &sgout);
virtual void operator <<(SIMPLE_GRAPHICAL_INPUT &sgin);
```

O operador `>>` deve escrever em `sgout` a informação da figura respectiva, e o operador `<<` deve ler de `sgin` a informação da figura a criar (ou seja deve chamar o método `leitura`).

A2.1. Acrescente o seguinte operador às figuras:

```
int operator -(FIGURA &other);
```

Este operador deve devolver a distância entre os centros das duas figuras.

Problema A3

Construa uma classe para manipular matrizes (dinâmicas): `MATRIZ`. Defina e implemente os operadores de atribuição de valor, igualdade, soma, subtracção e multiplicação de matrizes (com o cuidado de só permitir estas operações quando elas forem possíveis). Quando ache adequado, faça com que as operações criem novas matrizes (por forma a que expressões aritméticas com matrizes tenham o comportamento matematicamente esperado).

A3.1. Implemente os operadores `+=`, `-=` e `*=` relativamente a outras matrizes.

A3.2. Implemente os operadores `+=`, `-=` e `*=` relativamente valores escalares (`double`).

A3.3. Implemente o operador `()` aplicado a dois índices (linha e coluna), de forma a que este devolva a referência do elemento da matriz respectivo.

A3.4. Estude, discuta e resolva o problema da libertação da memória associada a matrizes criadas temporariamente. Por exemplo a seguinte expressão:

```
result = m1 + m2;
```

,gera uma matriz temporária resultante da operação de soma (atenção que a atribuição de valor deve ela própria criar uma nova matriz).

A3.5. Faça um programa para cálculos aritméticos com matrizes (tipo calculadora). Para simplificar considere que este programa tem o mesmo comportamento que as calculadoras simples, ou seja, todas as operações fazem-se relativamente ao resultado da operação anterior (ou do valor definido inicialmente).

Problema B1

B1.1. Construa uma classe `CString` que inclui um membro privado que é um `array` de caracteres. Desenvolva as funções `int find_first_of (char ch)` e `int find_first_of(const char* str)` que devolvem a posição do caracter `ch` caso este exista no `array` de caracteres ou a primeira posição de um dos caracteres da `str` caso este exista no `array`. Redefina os operadores `+`, `+=` (para concatenar duas `CString`), `=`, `[]`, `<<` (para visualizar o `array` de caracteres no écran), `>>` (para especificar uma `CString` com a ajuda do teclado), `<`, `>`, `<=`, `>=` , `==`, `!=` (para comparar duas `CStrings`)

B1.2. Implemente a seguinte função `main`:

```
int main(int argc, char* argv[])
{
  CString s1 = "abc";
  CString s2 = "def";
  CString s3 = s1 + s2;
  CString ss1, ss2, ss3;
  ss1 = ss2 = ss3 = s1;

  assert (s1 < s2);
  assert (s1 <= s2);
  assert (s1 != s2);

  CString s4(s1+s2);
  s4 = s1 + s2 + s3;

  cout << s4 << endl;

  s1 = s2 = s3;
  cout << s1 << endl;
  cout << s2 << endl;
  cout << s3 << endl;

  CString s;
  s + s1;
  cin >> s;
  cout << s << endl << endl;
  cout << s + s4 << endl;

  s[s.size()/2] = 'a';
  cout << s[s.size()/2] << endl;

  cout << s1.find_first_of("xyfd") << endl;
```

```
  return 0;
}
```

Problema B2

Analise o código seguinte. Responda as perguntas:

B2.1. Quantas vezes serão chamados o construtor e o destrutor da classe `CTest`?

B2.2. O código tem algum erro?

```
class CTest
{
  char* m_sTest;
public:
  CTest(char* t);
  virtual ~CTest();
};

CTest::CTest(char* t)
{
  m_sTest = new char[ strlen(t) + 1];
  strcpy (m_sTest, t);
}

CTest::~CTest()
{
  delete [] m_sTest;
}

int main(int argc, char* argv[])
{
  CTest t1("aaa");
  CTest t2(t1);

  return 0;
}
```