C++ Multiplataforma e Orientação a Objetos

Exercícios da parte 4:
C++ e orientação a objetos (adendo)

Por: Sergio Barbosa Villas-Boas (sbVB) sbVB

http://www.sbVB.com.br/c++.html
Versão 8.0, de 06 de Outubro de 2003
Copyright © 1992~2003 by sbVB
Email do autor: sbvb@sbvb.com.br

Exercício 1)

Deseja-se fazer 2 componentes de programação, muito parecidos. Um dos componentes é a função global extendedToUpper, e o outro é a função global extendedToLower, com os protótipos abaixo. A finalidade de desenvolver esses componentes é permitir escrever as funções convertToUpper e convertToLower como mostrado no programa. Essas funções convertem uma string, letra por letra, para letra maiúscula ou minúscula respectivamente. Mas as funções originais convertToUpper e convertToLower são baseadas em toupper e tolower (da biblioteca padrão), que convertem apenas letras não acentuadas. Mas na nova versão, as funções convertToUpper e convertToLower serão baseadas nas novas funções extendedToUpper e extendedToLower a serem desenvolvidas.

Esses componentes serão usados em na plataforma Windows. E em Windows, há uma codificação para letras acentuadas. As variáveis globais chgHiCaps (chgLoCaps) contém as letras maiúsculas (minúsculas) acentuadas que deverão ser usadas nas funções a serem desenvolvidas.

```
#include <iostream>
#include <fstream>
#include "vblib.h"  // VBString
using namespace std;

// prototypes of standard library
// char toupper(char c);
// char tolower(char c);

// exteded chars to change (global variables)
char chgHiCaps[] = "ÁÉÍÓÚÃÕÑÀÈÌÒÙÜÇÂÊÎÔÛ";
char chgLoCaps[] = "áéíóúãõñàèìòùüçâêîôû";

char extendedToUpper(char in)
{
  // add code here
}

char extendedToLower(char in)
{
  // add code here
}

void convertToUpper(char* buf)
{
  int len = strlen(buf);
  for (int i=0 ; i < len ; i++)
//    buf[i] = toupper(buf[i]); // commented out
    buf[i] = extendedToUpper(buf[i]);
}

void convertToLower(char* buf)
{
  int len = strlen(buf);
  for (int i=0 ; i < len ; i++)
//    buf[i] = tolower(buf[i]); // commented out
```

```
    buf[i] = extendedToLower(buf[i]);
}

int main()
{
  VBString s;
  ifstream originalFile("original.txt");
  ofstream toUpperFile("toUpper.txt");
  ofstream toLowerFile("toLower.txt");
  if (!originalFile || !toUpperFile || ! toLowerFile)
  {
    cout << "File not open" << endl;
    exit(1);
  }

  char *buffer;
  while (true)
  {
    originalFile >> s;
    if (originalFile.eof()) break;
    buffer = new char [strlen(s) + 1];  // +1 for the terminator char
    strcpy(buffer,s);
    convertToUpper(buffer);
    toUpperFile << buffer << endl;
    convertToLower(buffer);
    toLowerFile << buffer << endl;
    delete [] buffer;
  }
  cout << "done" << endl;
  return 0;
}
```

Exercício 2)

Faça um programa que leia um arquivo texto "names.dat" com *n* linhas, no formato abaixo.

nome_1
nome_2
...
nome_*n*

O programa deve gerar *n* arquivos texto, no mesmo diretório, chamados "user-<nome>.dat", sendo que <nome> é a string lida de cada linha do arquivo "names.dat". O conteúdo de cada um dos arquivos "user-<nome>.dat" é uma linha com <nome>.

Exercício 3)

Faça um programa que leia um arquivo de texto (abaixo)

```
Título da tabela
Nome do cliente,Telefone do cliente, código do cliente
Caetano Veloso,222-3344,29837
Jô Soares,234-5678,31232
...
```

```
Gilberto Gil,256-9876,28763
```

e gere uma saída como mostrado abaixo.

```
<font color="#001122">
<h1>Título da tabela</h1>
<table border="1">
<tr>
<td>Nome do cliente</td>
<td>Telefone do cliente</td>
<td>código do cliente</td>
</tr><tr>
<td>Caetano Veloso</td><td>2222-3344</td><td>29837</td>
</tr><tr>
<td>Jô Soares</td><td>2234-5678</td><td>31232</td>
</tr>
<!-- ... -->
<tr>
<td>Gilberto Gil</td><td>2256-9876</td><td>28763</td>
</tr>
</table>
</font>
```

(Para sua referência, a saída do programa, se vista no browser é como mostrado abaixo)

Título da Tabela

Nome do cliente	Telefone do cliente	Código do cliente
Caetano Veloso	2222-3344	29837
Jô Soares	2234-5678	31232
Gilberto Gil	2256-9876	28763

Exercício 4)

O programa abaixo apresenta um esqueleto de classe de conexão com base de dados. A finalidade dessa classe é fazer o programa do usuário conectar-se a uma base de dados através de algum tipo de conexão (por exemplo ODBC). Para que seja feita a conexão, é preciso que exista um nome para a base de dados, e também que exista uma autenticação com login, password. O método connect da classe myDataBase verifica se foi conectada, e joga (throw) dois tipos de exceção no caso de não conseguir a conecção. Uma exceção é para o caso de não existir o nome da base de dados (noDataBaseName), e a outra é pela falha na autenticação de login, password (badPassword).

1) No programa abaixo, acrescente código no local indicado, de modo a dar suporte para as duas exceções da classe myDataBase.

2) Conforme pode-se notar pelo programa, a confirmação de conexão (a string "Connection to database ok" copiada para o console) ocorre incondicionalmente (mesmo que exista algum erro na conexão). Altere o programa de forma que essa string somente seja copiada para o console no caso de conexão bem sucedida.

```
#include "vblib.h"

// add code here for exception handling of noDataBaseName and badPassword

class MyDataBase
{
  VBString m_data;
  bool m_connected;
public:
  MyDataBase()
  {
    // default constructor
    m_connected = false;
  }
  void connect(VBString name, VBString login, VBString password)
  {
    // connect to dataBase
    bool nameOK = (name == "dataBaseName");
    if (!nameOK)
      throw MyDataBaseException_NoDataBaseName();

    bool passwordOK = ((login == "dataBaseLogin") &&
      (password == "dataBasePassword")) ;
    if (!passwordOK)
    {
      throw MyDataBaseException_BadPassword();
    }
    m_connected = true;
  }
};

int main()
{
  MyDataBase obj;
  VBString name     = "dataBaseName";
  VBString login    = "dataBaseLogin";
  VBString password = "dataBasePassword";
  try
  {
    obj.connect(name,login,password);
  }
  catch (const MyDataBaseException_BadPassword & E)
  {
    E.ShowExplanation();
  }
  catch (const MyDataBaseException_NoDataBaseName & E)
  {
    E.ShowExplanation();
  }
  cout << "Connection to database ok" << endl;
}
```

Exercício 5)

Seja um arquivo de dados desordenados, cujo formato é “nome;peso;altura;anoNascimento”, linha a linha, conforme o exemplo abaixo.

```
name1;80.1;1.77;1933
name2;66.6;1.85;1930
name3;88.4;1.71;1944
```

O programa abaixo lê o arquivo de dados, colocando-o numa lista encadeada. Em seguida, copia os dados da lista encadeada num array alocado no tamanho exato do número de dados (desconhecido a priori). Os dados são mostrados no console de forma desordenada, ordenados pela função global bubble_sort, e em seguida exibidos novamente.

1) Complete o programa abaixo, acrescentando código nos locais indicados.
2) Mostre o que o programa faz aparecer na tela.

```
#include <iostream>
#include <fstream>
#include "vblib.h"
using namespace std;

template <class T>
void bubble_sort(T* array, int n, bool (*compare)(T,T))
{
  for (int i=0 ; i < n ; i++)
    for (int j=i+1 ; j < n ; j++)
      if (compare(array[i],array[j]))
      {  // swap (array[i], array[j])
        T temp = array[i];
        array[i] = array[j];
        array[j] = temp;
      }
}

struct Person
{
  VBString m_name;
  float m_weight;
  float m_height;
  int m_yearOfBitrh;
  friend ostream & operator<<(ostream & stream, Person &obj);

};

// if necessary, add code here (aux)

// overload insersor
ostream & operator << (ostream & stream, Person & obj) {
// add code here (1)
}

Person decodeStringForPerson(VBString s)
{
// add code here (2)
}

int main()
{
  const char *fileName = "t.txt";
  ifstream myFile(fileName);
  if (!myFile)
  {
    cout << "File not open" << endl;
    exit(1);
  }
  VBList<Person> personList;
```

```
  VBString s;
  Person p;
  Person *pp;
  int n=0;
  while (true)
  {
    myFile >> s;
    if (myFile.eof()) break;
    p = decodeStringForPerson(s);
    personList.add(p);
    n++;
  }

  // create an array of person of the same dimension of data in the linked list
  Person *personArray = new person [n];

  // copy data from linked list to array, to let data be sorted using bubble sort
  int i=0;
  // add code here (4)

  cout << "Unsorted data" << endl;
  for (i=0 ; i < n ; i++)
    cout << personArray[i] << endl;

  bubble_sort(personArray,n,ComparePersonPerAge);

  cout << "Sorted data, by age" << endl;
  for (i=0 ; i < n ; i++)
    cout << personArray[i] << endl;

  delete [] personArray; // not to leak memory
}
```

Exercício 6)

Seja o arquivo de dados em formato texto com nome “data.txt”, como mostrado abaixo. Trata-se de um arquivo de texto, em que a primeira linha é um número inteiro, que contém a quantidade de linhas subsequentes. Cada linha do arquivo de dados corresponde aos atributos (nome, anoDeNascimento, peso) de uma classe Person, que já está pronta e com métodos de apoio já desenvolvidos, como pode ser observado no programa abaixo.

```
5
Giberto Gil,1954,1.72
Chico Buarque,1951,1.74
João Gilberto,1960,1.68
Tom Jobim,1959,1.79
Caetano Veloso,1955,1.71
```

O programa abaixo possui uma classe chamada PersonArray, que possui um ponteiro para Person. Essa classe deve alocar um array de Person com dimensão apropriada e armazenar todos os dados do arquivo de dados.

Preencha com código no método readFromFile (lê o arquivo completo e o armazena no array alocado).

Preencha com código no método show (exibe o conteúdo do array no console).

```
#include <fstream>
#include "vblib.h"

struct Person
{
  VBString m_name;
  unsigned m_yearOfBirth;
  float m_weight;
  char m_tok;
  Person()
  {
    m_tok = ',';
  }
  // overload insersor
  friend ostream & operator <<(ostream & stream, Person &obj)
  {
    stream << obj.m_name << obj.m_tok <<
      obj.m_yearOfBirth << obj.m_tok << obj.m_weight;
    return stream;
  }
  // overload extrator
  friend istream & operator>>(istream & stream, Person &obj)
  {
    VBString aux;
    stream >> aux; // read all attributes of object in one string
    obj.m_name = aux.strtok(obj.m_tok,0);
    obj.m_yearOfBirth = atoi(aux.strtok(obj.m_tok,1));
    obj.m_weight = atof(aux.strtok(obj.m_tok,3));
    return stream;
  }
};

class PersonArray
{
  Person *m_p_Person;
  unsigned m_size;
  void myDelete()
  {
    if (m_p_Person)
    {
      delete[] m_p_Person;
      m_p_Person = 0;
    }
  }
  void allocArray(unsigned size)
  {
    m_size = size;
    m_p_Person = new Person[size];
  }
public:
  PersonArray()
  {
    m_p_Person = 0;
  }
  ~PersonArray()
  {
    myDelete();
  }
  void readFromFile(const char *fileName)
  {
     // add code here
  }
```

```
  void show()
  {
     // add code here
  }
};

int main()
{
  const char *fileName = "data.txt";
  PersonArray a;
  a.readFromFile(fileName);
  a.show();
  return 0;
}
```

A saída do programa deve ser como mostrado abaixo.

```
=========================
Array of Person
Giberto Gil,1954,1.72
Chico Buarque,1951,1.74
João Gilberto,1960,1.68
Tom Jobim,1959,1.79
Caetano Veloso,1955,1.71
```

Explique com palavras (não precisa escrever código) como você faria para escrever um código para um método chamado showOrdered (exibirOrdenado), da classe PersonArray, de tal forma que a exibição no console mostre os dados por ordem crescente de idade.

Exercício 7)

Abaixo há um programa, com uma classe Couple (casal), que contém 2 atributos: o nome dele e o nome dela. Esses atributos são tipo VBString. Complete a classe Couple, acrescentando código no local indicado. Um objeto da classe Couple deve poder efetuar a operação + com um objeto VBString de duas formas diferentes, como mostrado no exemplo abaixo. A classe Couple também deve ter o operator << sobrecarregado.

```
#include <iostream>
#define VBLIB_USE_NAMESPACE_STD
#include "vblib.h"
using namespace std;

class Couple
{
  VBString m_nameHim, m_nameHer;
public:
  void setHim(VBString him) {m_nameHim = him;}
  void setHer(VBString her) {m_nameHer = her;}

  // add code here
};

int main()
{
  Couple c1,c2;
  VBString extra = " NOME EXTRA ";
```

```
  c1.setHim("Nome dele");
  c1.setHer("Nome dela");
  c2 = c1 + extra;
  cout << c2 << endl;
  c2 = extra + c1;
  cout << c2 << endl;
  return 0;
}
```

Saída esperada para o programa

```
Nome dele NOME EXTRA :Nome dela NOME EXTRA
 NOME EXTRA Nome dele: NOME EXTRA Nome dela
```

Exercício 8)

Exercício 9)

O programa abaixo é o esqueleto de uma biblioteca de classes "XYZ", que tem a finalidade de conectar o programa em questão com um sistema remoto. Como se pode ver, há 3 esqueletos de classes da biblioteca XYZ. O programa principal tenta a conexão, e em seguida confere a senha. No caso de haver erro na conexão, ocorre uma exceção. No caso de a senha não conferir, também ocorre uma exceção.

- O código principal, não se diferencia uma exceção da outra. Faça uma adaptação no programa para no caso de uma exceção ou de outra exceção apareça uma mensagem diferente no programa principal.
- Considerando o programa modificado, faça aparecer uma exceção e mostre o que o programa faz aparecer no console.

```
#include <iostream>
using namespace std;

class XYZ_ExceptionHandler
{
public:
};

class XYZ_Connect
{
  bool actualConnectCode()
  {
    // real code will be inserted here later
    // return code true means an error happened
    return false;
  }
public:
  void connect()
  {
    bool failConnect = actualConnectCode();
    if (failConnect)
    {
      XYZ_ExceptionHandler e;
      throw e;
```

```
    }
  }
};

class XYZ_CheckPassword
{
  bool actualCode()
  {
    // real code will be inserted here later
    // return code true means an error happened
    return false;
  }
public:
  void CheckPassword()
  {
    bool fail = actualCode();
    if (fail)
    {
      XYZ_ExceptionHandler e;
      throw e;
    }
  }
};

int main()
{
  try
  {
    XYZ_Connect con;
    con.connect();
    cout << "connected !" << endl;
    XYZ_CheckPassword passwd;
    passwd.CheckPassword();
    cout << "password ok !" << endl;
  }
  catch (XYZ_ExceptionHandler e)
  {
    cout << "XYZ exception." << endl;
  }
  catch (...)
  {
    cout << "Unknown exception" << endl;
  }
  return 0;
}
```