PUC-Minas - Ciência da Computação
AED1 – Estudo Dirigido 14

Tema: Introdução à programação VI
Atividade: Classes

INSTRUÇÕES:

- Desenvolver classes/métodos em C++ para atender às especificações abaixo.
- Providenciar a documentação essencial:
  nome e matrícula,
  identificação, objetivo, parâmetros e condições especiais,
  se houver, e relatório de testes (exemplos de valores usados e condições testadas).

SUGESTÃO: Montar um menu para a escolha do método a ser testado
(ver modelo em Lista00.cpp).
Testes deverão ser realizados e os valores usados deverão
ser guardados no final do programa como comentários (/* e */).
O uso de recursão é opcional; se desejar utilizá-lo,
fazer também a implementação da forma não-recursiva.

O tratamento de erros deverá feito mediante derivação da classe Erro,
conforme a especificação abaixo, a ser revista, completada e guardada
em arquivo distinto (Erro.hpp), e que deverá tratar
uma variável inteira privada (erro), bem como
a lista de mensagens indicativas de cada situação tratada.

0.) Rever e completar as definições para tratamento de erros.

```
/**
  Sugestão para a classe para tratamento de erros.
 */

// dependencias
#include <string>

/**
 * Classe para tratar erro.
 */

#ifndef _ERRO_H_
#define _ERRO_H_

class Erro
{
 /*
  * tratamento de erro.
    Codigos de erro:
    0. Nao ha' erro.
  */

 /**
  * atributos privados.
  */
  private:
   int erro;

 /**
  * definicoes publicas.
  */
  public:
  /**
   * Destrutor.
   */
   ~Erro ( );

  /**
   * Construtor padrao.
   */
   Erro ( ) { };

  /**
   * Constante da classe.
   */
   static const std::string NO_ERROR;
```

```
  // ------------------------------------------- metodos para acesso

  /**
    * Funcao para obter o codigo de erro.
    * @return codigo de erro guardado
    */
    int getErro ( )
    {
      return ( 0 ); // valor provisorio, precisara' ser definido futuramente
    } // end getErro ( )

  /**
    * Funcao para testar se ha' erro.
    *  @return true, se houver;
    *          false, caso contrario
    */
    bool hasError ( );

  /**
    * Funcao para obter mensagem
    * relativa ao código de erro.
    * @return mensagem sobre o erro
    */
    virtual std::string getErroMsg ( )
    {
      return ( "" );
    } // end getErroMsg ( )

  /**
    * definicoes com acesso restrito.
    */
     protected:

  // ------------------------------------------- metodos para acesso restrito
  /**
    * Metodo para estabelecer novo codigo de erro.
    * @param codigo a ser guardado
    */
    void setErro ( int codigo );

};  // end class Erro

const std::string Erro::NO_ERROR = "[ERRO] Nao ha' erro."; // definir o valor da constante

#endif
```

Editar outro programa em C++, na mesma pasta, cujo nome será Exemplo1400.cpp, para testar definições da classe MyString a serem desenvolvidas:

```
/*
  Exemplo1400 - v0.0. - __ / __ / _____
  Author: _____

 */

// ------------------------------------------------ preparacao

// dependencias

#include <iostream>
using std::cin  ;          // para entrada
using std::cout;           // para saida
using std::endl;           // para mudar de linha

#include <iomanip>
using std::setw;           // para definir espacamento

#include <string>
using std::string;         // para cadeia de caracteres

#include <fstream>
using std::ofstream;     // para gravar arquivo
using std::ifstream;     // para ler    arquivo

// outras dependencias

void pause ( std::string text )
{
   std::string dummy;
   std::cin.clear ( );
   std::cout << std::endl << text;
   std::cin.ignore( );
   std::getline(std::cin, dummy);
   std::cout << std::endl << std::endl;
} // end pause ( )
```

```
// ------------------------------------------------ classes

#include "Erro.hpp"  // classe para tratar erros

class MyString : public Erro
{
   public:

  /**
   * Funcao para obter mensagem
   * relativa ao código de erro.
   * @return mensagem sobre o erro
   */
   std::string getErroMsg ( )
   {

      return ( NO_ERROR );     // COMPLETAR A DEFINICAO

   } // end getErroMsg ( )

}; // end classe MyString

// ------------------------------------------------ definicoes globais

using namespace std;

// ------------------------------------------------ metodos

/**
  Method_00 - nao faz nada.
 */
void method_00 ( )
{
 // nao faz nada
} // end method_00 ( )

/**
  Method_01 - Testar definicoes da classe.
 */
void method_01 ( )
{
 // definir dados
   MyString *s = new MyString ( );

 // identificar
   cout << "\nMethod_01 - v0.0\n" << endl;

 // encerrar
   pause ( "Apertar ENTER para continuar" );
} // end method_01 ( )
```

```
// ------------------------------------------------- acao principal

/*
  Funcao principal.
  @return codigo de encerramento
*/
int main ( int argc, char** argv )
{
 // definir dado
    int x = 0;                  // definir variavel com valor inicial

 // repetir até desejar parar
    do
    {
     // identificar
        cout << "EXEMPLO1401 - Programa - v0.0\n     " << endl;

     // mostrar opcoes
        cout << "Opcoes                                  " << endl;
        cout << " 0 - parar                              " << endl;
        cout << " 1 - testar definicoes                  " << endl;

     // ler do teclado
        cout << endl << "Entrar com uma opcao: ";
        cin  >> x;

     // escolher acao
        switch ( x )
        {
          case 0:
            method_00 ( );
           break;
          case 1:
            method_01 ( );
           break;
          default:
            cout << endl << "ERRO: Valor invalido." << endl;
        } // end switch
    }
    while ( x != 0 );

 // encerrar
    pause  ( "Apertar ENTER para terminar" );
    return ( 0 );
} // end main ( )
```

/*

---------- documentacao complementar

---------- notas / observacoes / comentarios

---------- previsao de testes

---------- historico

Versao	Data	Modificacao
0.1	__/__	esboco

---------- testes

Versao	Teste	
0.1	01. (OK)	identificacao de programa

***/**

Exercícios:

DICAS GERAIS: Consultar o Anexo CPP 02 na apostila para outros exemplos.
Não usar métodos ou funções já prontos em bibliotecas nativas da linguagem.
Prever, realizar e registrar todos os testes efetuados.
- Desenvolver e testar cada um dos protótipos de métodos sugeridos abaixo.
Não usar métodos ou funções já prontos em bibliotecas nativas da linguagem.
Integrar as chamadas de todos os testes em um só programa.

01.)

```
/**
    Funcao para converter conteudo do objeto para valor inteiro, se possivel.
    @return valor inteiro equivalente, se valido;
            (-1), caso contrario
 */
   int getInt ( )
```

02.)

```
/**
    Funcao para converter conteudo do objeto para valor real, se possivel.
    @return valor real equivalente, se valido;
            (0.0), caso contrario
 */
   double getDouble ( )
```

03.)

```
/**
    Funcao para converter conteudo do objeto para valor lógico, se possivel.
    Nota: Considerar válidos: { true, false, T, F, 0, 1 }
    @return valor logico equivalente, se valido;
            false, caso contrario
 */
   bool getBoolean( )
```

04.)

```
/**
    Funcao para verificar se o parametro esta' contido no conteudo.
    @return true , se contiver (a partir de qualquer posicao);
            false, caso contrario
 */
   bool contains ( std::string texto )
```

05.)

```
/**
    Funcao para converter letras para maiusculas.
    @return valor equivalente em maiusculas, se houver;
            o proprio valor, caso contrario
 */
   std::string toUpperCase( )
```

06.)

```
/**
    Funcao para converter letras para minusculas.
    @return valor equivalente em minusculas, se houver;
                o proprio valor, caso contrario
 */
   std::string toLowerCase( )
```

07.)

```
/**
    Funcao para trocar todas as ocorrencias de certo caractere por outro novo.
    @return valor com substituicoes, se houver;
                o proprio valor, caso contrario
 */
   std::string replace ( char original, char novo )
```

08.)

```
/**
    Funcao para codificar o conteudo segundo a cifra de César (pesquisar).
    @return valor equivalente codificado, se houver;
                o proprio valor, caso contrario
 */
   std::string encrypt ( )
```

09.)

```
/**
    Funcao para decodificar o conteudo previamente cifrado pela funcao acima.
    @return valor equivalente decodificado, se houver;
                o proprio valor, caso contrario
 */
   std::string decrypt ( )
```

10.)

```
/**
    Funcao para separar todas as sequencias de caracteres separadas por espaços em branco.
    @param sequencia - arranjo para armazenar possiveis cadeias de caracteres identificadas
    @return quantidade de sequencias de caracteres identificadas, se houver;
                zero, caso contrario
 */
   int split ( std::string sequencia [ ] )
```

Tarefas extras

E1.) Acrescentar um método (e testes) para fragmentar o conteúdo usando um delimitador à escolha, diferente de espaço em branco.

E2.) Acrescentar um método (e testes) para inverter a ordem dos símbolos na cadeia de caracteres.