ECOP13A-Lab10

Templates

Guia de Laboratório
Prof. André Bernardi
andrebernardi@unifei.edu.br

10º Laboratório ECOP13A
18 de junho 2025

1ª Questão

Crie uma classe vetor que utilize ***template*** e processamento de ***exceções***.

Implementar uma função membro Ordena().

Na função membro ordena que será implementada quais restrições devem ser observadas para utilizar um objeto como tipo do template?

1ª questão
Exemplo de solução

```
#ifndef CVETOR_H
#define CVETOR_H

#include <iostream>
using namespace std;

template <typename T>
class CVetor
{
    private:
        T *m_v;
        int m_tamanho;

    public:
        CVetor() ;
        CVetor (int);
        virtual ~CVetor() {delete[] m_v;}
        CVetor(const CVetor<T>& other);
        CVetor& operator=(const CVetor& other) ;

        T& operator[](int i);
        void ordena();
```

```
        friend ostream& operator <<(ostream& output, const CVetor<T>& vet){
            for(int i=0;i<vet.m_tamanho;i++){
                output << vet.m_v[i]<< " ";
            }
            return output;
        }

        friend istream& operator >>(istream& input, CVetor<T>& vet){
             cout << "Digite " << vet.m_tamanho << " valores: ";
            for(int i=0;i<vet.m_tamanho;i++){
                input >> vet.m_v[i];
            }
            cout << endl;
            return input;
        }
};

template <typename T>
void CVetor<T>::ordena(){
    T aux;
    for(int i=0;i<m_tamanho;i++){
        for(int j=0;j<m_tamanho-1-i;j++){
             if( m_v[j] > m_v[j+1])      // precisa do operador >
             {
                 aux = m_v[j];           // precisa do operador =
                 m_v[j]=m_v[j+1];
                 m_v[j+1]=aux;
             }
        }
    }
}
```

```cpp
template <typename T>
T& CVetor<T>::operator[](int i){
    if(i>=0 && i<m_tamanho){
        return m_v[i];
    }else{
        throw out_of_range("Indice fora da faixa!");
    }
}

template <typename T>
CVetor<T>::CVetor(int tamanho){
    m_v = new T[tamanho];
    m_tamanho = tamanho;
    for(int i=0;i<tamanho;i++){
        m_v[i] = 0;
    }
}

template <typename T>
CVetor<T>::CVetor(const CVetor& vet){
    m_tamanho = vet.m_tamanho;
    m_v = new T[m_tamanho];
    for(int i=0;i<m_tamanho;i++){
        m_v[i] = vet.m_v[i];
    }
}
```

```
template <typename T>
CVetor<T>::CVetor(){
    m_tamanho = 10;
    m_v = new T[10];
    for(int i=0;i<10;i++){
        m_v[i] = 0;
    }
}

//operador de atribuiçao
template <typename T>
CVetor<T>& CVetor<T>:: operator = (const CVetor<T> & p)
{
    delete [] m_v; // remover ponteiro antigo
    m_tamanho = p.m_tamanho;
    m_v = new T[m_tamanho];
    for(int i = 0; i < p.m_tamanho; i++)
    {
        m_v [i] = p. m_v [i];
    }
    return *this;
}

#endif // CVETOR_H
```

main

```cpp
#include <iostream>
#include "CVetor.h"

using namespace std;

int main()
{
    CVetor<int> a(5);
    CVetor<double> b(3);

     cin >> a;
     cout << "A desordenado: " << a << endl;
     a.ordena();
     cout << "A ordenado: " << a << endl;

     cin >> b;
     cout << "B desordenado: " << b << endl;
     b.ordena();
     cout << "B ordenado: " << b << endl;

     try{
          cout << b[5] << endl;
     } catch(out_of_range &ex) {
          cout << "out_of_range: " << ex.what() << endl;
     }

    return 0;
}
```

2ª Questão

Altere a classe CPilha do laboratório anterior para que ela possa utilizar Templates.

2ª questão


```
//header file para classe pilha
#ifndef PILHA_H
#define PILHA_H

#include <iostream>
using namespace std;

template < typename T >
class CPilha
{
    private:
        T *m_dados;              // ponteiro para dados da pilha
        int m_ponteirodaPilha;   // apontador do topo da pilha
        int m_tamanho;           // espaço de memória reservado para o objeto

    public:
        CPilha ( int memoria ); // construtor com parâmetros
        CPilha ( void );        // construtor sem parâmetros
        ~CPilha ( void );       // destrutor

        bool pop ( T &pop_to );         // puxar dados da pilha
        bool push ( T push_this );      // empurrar dados para a pilha

        CPilha ( const CPilha& );               // construtor de cópia
        CPilha& operator = (const CPilha&);   // operador de atribuição

     template < typename T1>
    friend ostream& operator << ( ostream& , const CPilha<T1>& );
    template < typename T1>
    friend istream& operator >> ( istream& , CPilha<T1>& );
};
#endif
```

```
//implementação de templates direto com a definição
template < typename T >
ostream& operator<<(ostream& out, const CPilha<T>& p)
{
    for (int i = 0; i <= p.m_ponteirodaPilha ; i++)
        out << p.m_dados[i] << " ";
    out << endl;
    return out;
}

template < typename T >
istream& operator >>(istream& in, CPilha<T>& p)
{
    T a;
    p.m_ponteirodaPilha = -1; // esvaziar a pilha
    cout << "Entre com os " << p.m_tamanho << " dados da pilha: ";
    for (int i = 0; i < p.m_tamanho ; i++)
    {
        in >> a;
        if (!p.push(a)) break; // se não conseguiu inserir aborta.
    }
    return in;
}

// construtor com parâmetros
template < typename T >
CPilha<T>::CPilha ( int memoria )
{
    m_ponteirodaPilha  = -1;   // sinaliza pilha vazia
    m_tamanho = (memoria > 0)? memoria : 10;
    m_dados = new T [ m_tamanho ];
}
```


```
// construtor sem parâmetros
template < typename T >
CPilha<T>::CPilha ( void )
{
    m_ponteirodaPilha  = -1;    // sinaliza pilha vazia
    m_tamanho =  10;
    m_dados = new T [ m_tamanho ];
}

// destrutor
template < typename T >
CPilha<T>::~CPilha ( void )
{
    delete [] m_dados;
}
// puxar dados da pilha
template < typename T >
bool CPilha<T>::pop ( T &pop_to )
{
    if (m_ponteirodaPilha == -1) return false;
    pop_to = m_dados[m_ponteirodaPilha--];
    return true;
}

// empurrar dados para a pilha
template < typename T >
bool CPilha<T>::push ( T push_this )    // melhor seria const T&
{
    if (m_ponteirodaPilha == m_tamanho-1) return false;
    m_dados[++m_ponteirodaPilha] = push_this;
    return true;
}
```

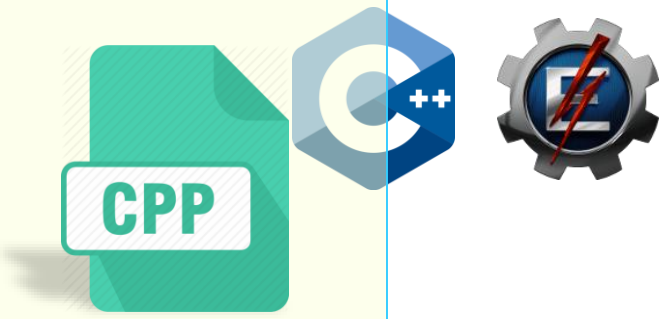

```
// construtor de cópia
template < typename T >
CPilha<T>::CPilha ( const CPilha<T>& p )
{
    m_ponteirodaPilha = p.m_ponteirodaPilha;
    m_tamanho =  p.m_tamanho;
    m_dados = new T [ m_tamanho ];
    for (int i = 0; i <= m_ponteirodaPilha ; i++)
        m_dados[i] = p.m_dados[i];
}

// operador de atribuição
template < typename T >
CPilha<T>& CPilha<T>::operator = (const CPilha<T>& p)
{
    delete [] m_dados;
    m_ponteirodaPilha = p.m_ponteirodaPilha;
    m_tamanho =  p.m_tamanho;
    m_dados = new T [ m_tamanho ];
    for (int i = 0; i <= m_ponteirodaPilha ; i++)
        m_dados[i] = p.m_dados[i];

    return *this;
}
```

3ª Questão

Utilize a classe CPilha criada para empilhar dados do tipo:

- int
- float
- char
- CPilha<int>

main




```cpp
#include <iostream>
#include "cpilha.h"

using namespace std;

int main()
{
    CPilha <int> pInt;
    CPilha <char> pChar;
    CPilha <float> pFloat;
    CPilha <CPilha <int> > ppInt;

   // usando a pilha de inteiros
    cin >> pInt;
    cout << pInt;

   // usando a pilha de char
    cin >> pChar;
    cout << pChar;

   // usando a pilha de float
    cin >> pInt;
    cout << pInt;


   // usando a pilha de pilha de inteiros
    cin >> ppInt;
    cout << ppInt;

    return 0;
}
```

Main
outro exemplo


```
#include <iostream>
#include " cpilha.h"

using namespace std;

int main()
{
    CPilha<char> ch(10);
    CPilha<int> in(10);
    CPilha<float> fl(10);
    CPilha<Pilha<int> > pi(10);

    for(int i=0;i<10;i++){
        in.push(i);
        fl.push(i*1.5);
        ch.push(i+'A');
        pi.push(in);
    }

    for(int i=0;i<10;i++){
        in.pop();
        fl.pop();
        ch.pop();
        pi.pop();
    }

    return 0;
}
```