OO – Engenharia Eletrônica

Orientação a Objetos
-
Programação em C++

Slides 3 - C:
Listas em C++ - Introdução

Prof. Jean Marcelo SIMÃO – DAELN/UTFPR

Listas

Introdução em a Listas em C++

Qual o problema do código abaixo?

```
#ifndef _UNIVERSIDADE_H_
#define _UNIVERSIDADE_H_

#include "Departamento.h"

class Universidade {
  private:
     char nome[130];
     Departamento* LpDptos[50];
     int ctd;

  public:
     Universidade();
     ~Universidade();

     void setNome(char* n);
     char* getNome();
     void setDepartamento(Departamento* pdep, int ctd);
      . . .
};
#endif
```

Número limitado de departamentos relacionados a cada Universidade.

```
#include "stdafx.h"
#include "Universidade.h"
. . .

void Universidade::setDepartamento(Departamento* pdep, int ctd)
{
   LpDptos[ctd] = pdep;
}
```

Solução: Usar o *Vector* do C++ (*STL*)

```
#ifndef _UNIVERSIDADE_H_
#define _UNIVERSIDADE_H_

#include "Departamento.h"
#include <vector>
using namespace std;

class Universidade
{
  private:
     char nome[130];
      vector< Departamento* > LpDptos;
      int ctd;

  public:
     Universidade();
     ~Universidade();

     void setNome(char* n);
     char* getNome();
     void setDepartamento(Departamento* pdep);
};

#endif
```

Há *slides* subsequentes que detalham os componentes (do tipo) *List* e *Vector* da *Standard Template Library* (*STL*) do C++.

A *STL* traz inclusive um conjunto de estruturas de dados como lista, filas e pilhas. Estas estruturas são implementadas genericamente, na forma de gabaritos ou *templates*.

```
#include "stdafx.h"
#include "Universidade.h"

void Universidade::setDepartamento(Departamento* pdep)
{
    LpDptos.push_back(pdep);
}
```

Solução: Usar o *Vector* do C++ (*STL*)

```
#ifndef _UNIVERSIDADE_H_
#define _UNIVERSIDADE_H_

#include "Departamento.h"
#include <vector>
using namespace std;

class Universidade
{
  private:
     char nome[130];
      vector< Departamento* > LpDptos;
      int ctd;

  public:
     Universidade();
     ~Universidade();

     void setNome(char* n);
     char* getNome();
     void setDepartamento(Departamento* pdep);
     void imprimeDepartamentos();
};
#endif
```

```
#include "stdafx.h"
#include "Universidade.h"
...
void Universidade::setDepartamento(Departamento* pdep)
{
    LpDptos.push_back(pdep);
}

void Universidade::imprimeDepartamentos()
{
    Departamento* pDep = NULL;
    int tam = (int) LpDptos.size();
    for ( int i = 0; i < tam; i++)
    {
       pDep = LpDptos[ i ];
       cout << pDep->getNome() << end;;
    }
}
```

O *Vector* da *Standard Template Library* (*STL*) seria um certo tipo de lista (similar ao *List* da *STL*), com a vantagem de usar o operador de colchetes tal qual fosse um vetor comum (uma vez que tenha dados no *vector* em questão...).

Solução: Usar o *Vector* do C++ (*STL*)

```
#ifndef _UNIVERSIDADE_H_
#define _UNIVERSIDADE_H_

#include "Departamento.h"
#include <vector>
using namespace std;

class Universidade
{
  private:
     char nome[130];
      vector< Departamento* > LpDptos;
      int ctd;

  public:
     Universidade();
     ~Universidade();

     void setNome(char* n);
     char* getNome();
     void setDepartamento(Departamento* pdep);
     void imprimeDepartamentos();
};
#endif
```

```
#include "stdafx.h"
#include "Universidade.h"
...
void Universidade::setDepartamento(Departamento* pdep)
{
    LpDptos.push_back(pdep);
}

void Universidade::imprimeDepartamentos()
{
    int tam = (int) LpDptos.size();
    for ( int i = 0; i < tam; i++)
    {
       cout << (LpDptos[ i ])->getNome() << end;
    }
}
```

O *Vector* da *Standard Template Library* (*STL*) seria um certo tipo de lista (similar ao *List* da *STL*), com a vantagem de usar o operador de colchetes tal qual fosse um vetor comum (uma vez que tenha dados no *vector* em questão...).

Solução: Usar o *list* do C++ (*STL*)

```
#ifndef _UNIVERSIDADE_H_
#define _UNIVERSIDADE_H_

#include "Departamento.h"
#include <list>
using namespace std;

class Universidade {
  private:
       char nome[130];
       list< Departamento* > LpDptos;
       int ctd;
  public:
      Universidade();
      ~Universidade();
      void setNome(char* n);
      char* getNome();
      void setDepartamento(pDepartamento* pdep);
};
#endif
```

```
#include "stdafx.h"
#include "Universidade.h"
. . .
void Universidade::setDepartamento(Departamento* pdep)
{
   LpDptos.push_back(pdep);
}
```

Solução: Usar o *list* do C++ (*STL*)

```
#ifndef _UNIVERSIDADE_H_
#define _UNIVERSIDADE_H_

#include "Departamento.h"
#include <list>
using namespace std;

class Universidade {
  private:
       char nome[130];
       list< Departamento* > LpDptos;
       int ctd;
  public:
      Universidade();
      ~Universidade();
      void setNome(char* n);
      char* getNome();
      void setDepartamento(pDepartamento* pdep);
      void imprimeDepartamentos();
};
#endif
```

```
#include "stdafx.h"
#include "Universidade.h"
. . .
void Universidade::setDepartamento(Departamento* pdep)
{
   LpDptos.push_back(pdep);
}

void Universidade::imprimeDepartamentos()
{
     Departamento* pDep = NULL;

     list< Departamento* >::iterator  iterador;

    for ( iterador = LpDptos.begin();
          iterador != LpDptos.end();
          iterador++)
    {
        pDep =  *iterador;
        cout << pDep->getNome() << end;;
    }
}
```

Solução: Usar o *list* do C++ (*STL*)

```
#ifndef _UNIVERSIDADE_H_
#define _UNIVERSIDADE_H_

#include "Departamento.h"
#include <list>
using namespace std;

class Universidade {
  private:
      char nome[130];
      list< Departamento* > LpDptos;
      int ctd;
  public:
     Universidade();
     ~Universidade();
     void setNome(char* n);
     char* getNome();
     void setDepartamento(pDepartamento* pdep);
     void imprimeDepartamentos();
};
#endif
```

```
#include "stdafx.h"
#include "Universidade.h"
. . .
void Universidade::setDepartamento(Departamento* pdep)
{
   LpDptos.push_back(pdep);
}

void Universidade::imprimeDepartamentos()
{

    list< Departamento* >::iterator  iterador;

   for ( iterador = LpDptos.begin();
         iterador != LpDptos.end();
         iterador++)
   {
      cout << (*iterador)->getNome() << end;;
   }
}
```

Questões

- Como compor uma classe similar ao *List* ou mesmo ao *Vector*?
  - Isto será considerado nos slides seguintes.
- Para que saber isto?
  - Para poder elaborar sua própria lista caso necessite.
  - Porque é um bom exemplo para se aprender detalhes e nuances de orientação a objetos, dentre outros...
  - Ademais, saber implementar isto ajudará a ter uma introdução útil para a disciplina de Estruturas de Dados. Entretanto, muito que obviamente, é apenas um conhecimento (coincidentemente e felizmente) útil para tal disciplina.
  - Ao bem da verdade, a disciplina de Estrutura de Dados discorre sobre uma diversidade (muito) maior de estrutura de dados, além de listas (e mesmo pilhas e filas), e abordagens algorítmicas para tal conjunto de estruturas de dados.