Universidade Tecnológica Federal do Paraná
UTFPR – Campus Curitiba

Orientação a Objetos
-
Programação em C++

Grupo de Slides 11 – Parte A: Herança et al.

- Funções Virtuais, Polimorfismo, Herança Múltipla, Funções Virtuais Puras e Classes Abstratas.

Prof. Dr. Jean Marcelo SIMÃO

Aluno Monitor (em 2011): Vagner Vengue (alguns slides de suporte)

Exercício 1

- Criar uma lista no sistema com todas as pessoas criadas independentemente de suas especializações (Professor, Aluno).

- Na classe *Professor* criar um atributo *Salario,* criar um atributo *Bolsa Projeto* e um métoda *InformaProventos.*

- Criar uma classe *Estagiario* derivado de *Aluno*, com um atributo *BolsaEstudo* e um método *InformaProventos*.

Exercício 1

- **Criar uma lista no sistema com todas as pessoas criadas independentemente de suas especializações (Professor, Aluno).**

- Na classe *Professor* criar um atributo *Salario* e um método *InformaProventos.*

- Criar uma classe *Estagiario* derivada de *Aluno*, com um atributo *BolsaEstudo* e um método *InformaProventos*.

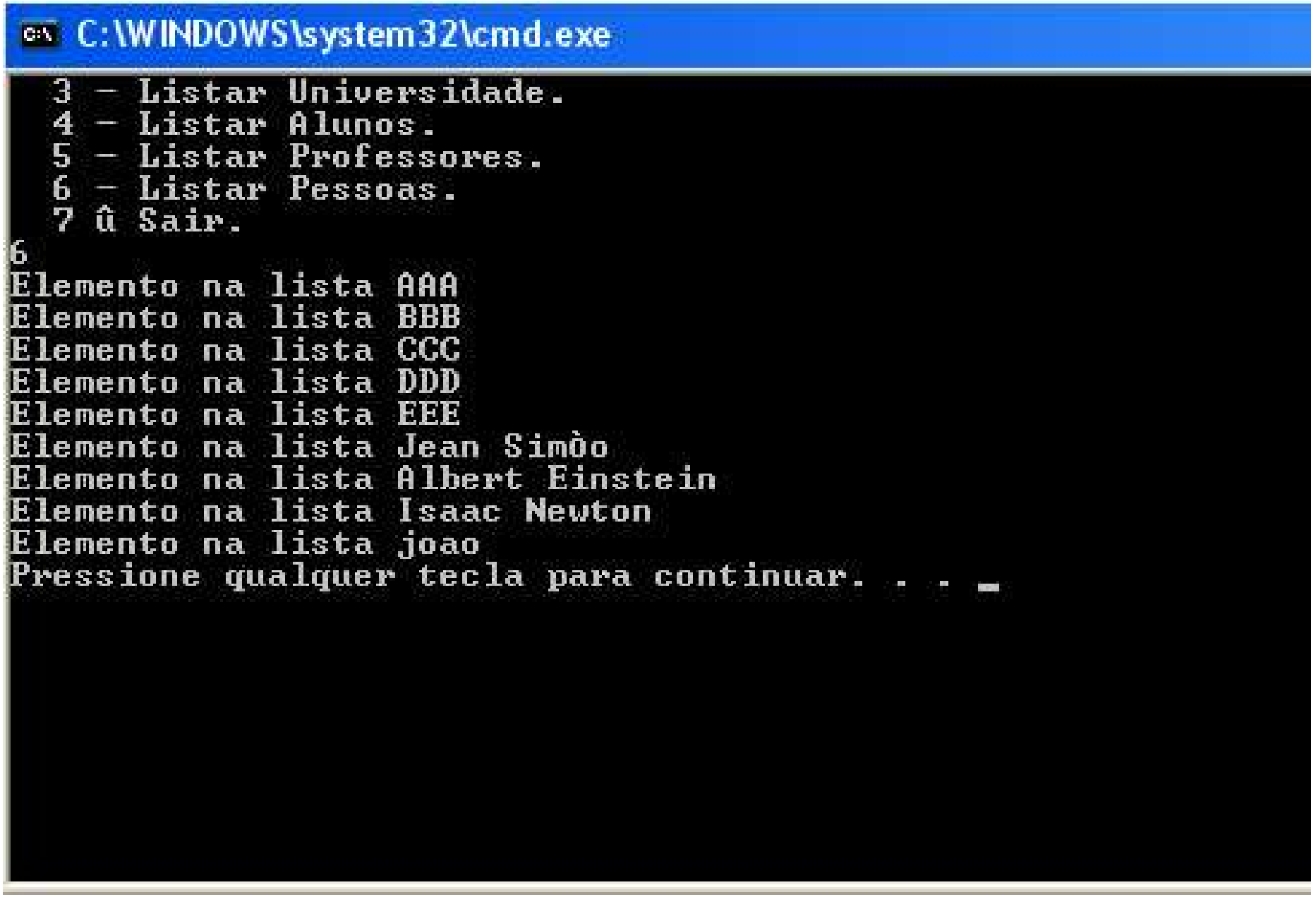
```
#ifndef _PRINCIPAL_H_
#define _PRINCIPAL_H_

#include "Lista.h"
#include "Professor.h"
#include "Universidade.h"
#include "Departamento.h"
#include "Disciplina.h"
#include "Aluno.h"

class Principal
{
 private:
   int cont_idDisc;
   int cont_idDepart;
   int cont_idAluno;

   Universidade UTFPR;
   Universidade Princeton;
   Universidade Cambridge;

   Departamento EletronicaUTFPR;
   Departamento MatematicaUTFPR;
   Departamento FisicaUTFPR;
   Departamento MatematicaPrinceton;
   Departamento FisicaPrinceton;
   Departamento MatematicaCambridge;
   Departamento FisicaCambridge;

   Professor    Simao;
   Professor    Einstein;
   Professor    Newton;

   Disciplina   Computacao1_2006;
   Disciplina   Introd_Alg_2007;
   Disciplina   Computacao2_2007;
   Disciplina   Metodos2_2007;

   Aluno        AAA;
   Aluno        BBB;
   Aluno        CCC;
   Aluno        DDD;
   Aluno        EEE;

   int                  diaAtual;
   int                  mesAtual;
   int                  anoAtual;

   Lista <Universidade> LUniversidades;
   Lista <Departamento> LDepartamentos;
   Lista <Disciplina>   LDisciplinas;
   Lista <Aluno>        LAlunos;

   Lista <Pessoa>       LPessoas;

 public:
   Principal ( );
   void Inicializa();
   void InicializaUnivesidades();
   void InicializaDepartamentos();
   void InicializaProfessores();
   void InicializaAlunos();
   void InicializaDisciplinas();
   void Executar();

   void CalcIdadeProfs();
   void UnivOndeProfsTrabalham();
   void DepOndeProfsTrabalham();
   void ListeDiscDeptos ( );
   void ListeAlunosDisc();

   void CadDisciplina();
   void CadDepartamento();
   void CadUniversidade();
   void CadAluno();

   void GravarTudo();
   void GravarUniversidades();
   void GravarDepartamentos();
   void GravarDisciplinas();
   void GravarAlunos();
   void GravarProfessores();

   void MenuCad();
   void MenuExe();
   void Menu();
};
#endif
```

```
void Principal::InicializaAlunos()
{
        Pessoa* ponteiroPessoa = NULL;
        Aluno*    ponteiroAluno = NULL;

        AAA.setNome ("AAA");
        LAlunos. incluaInfo (&AAA, AAA.getNome());
        ponteiroAluno = &AAA;
        // ponteiroPessoa = ( Pessoa * ) ( ponteiroAluno );
        ponteiroPessoa = static_cast < Pessoa * > ( ponteiroAluno );
        LPessoas.incluaInfo ( ponteiroPessoa, ponteiroPessoa->getNome() );

        BBB.setNome("BBB");
        LAlunos. incluaInfo (&BBB, BBB.getNome());
        ponteiroAluno = &BBB;
        ponteiroPessoa = static_cast<Pessoa*>(ponteiroAluno);
        LPessoas. incluaInfo (ponteiroPessoa, ponteiroPessoa->getNome());

        CCC.setNome("CCC");
        LAlunos. incluaInfo (&CCC, CCC.getNome());
        ponteiroAluno = &CCC;
        ponteiroPessoa = static_cast<Pessoa*>(ponteiroAluno);
        LPessoas. incluaInfo (ponteiroPessoa, ponteiroPessoa->getNome());

        DDD.setNome("DDD");
        LAlunos. incluaInfo (&DDD, DDD.getNome());
        ponteiroAluno = &DDD;
        ponteiroPessoa = static_cast<Pessoa*>(ponteiroAluno);
        LPessoas. incluaInfo (ponteiroPessoa, ponteiroPessoa->getNome());

        EEE.setNome("EEE");
        LAlunos. incluaInfo (&EEE, EEE.getNome());
        ponteiroAluno = &EEE;
        ponteiroPessoa = static_cast<Pessoa*>(ponteiroAluno);
        LPessoas. incluaInfo (ponteiroPessoa, ponteiroPessoa->getNome());
}
```

```
void Principal::CadAluno()
{
        char        nomeAluno [150];
        int         ra;
        Aluno*      ponteiroAluno;
        Pessoa*     ponteiroPessoa;

        cout << "Qual o nome do aluno. " << endl;
        cin  >> nomeAluno;

        cout << "Qual o RA do aluno."      << endl;
        cin  >> ra;

        ponteiroAluno = new Aluno ( cont_idAluno++ );
        ponteiroAluno->setNome ( nomeAluno );
        ponteiroAluno->setRA ( ra );

        LAlunos.incluaInfo (ponteiroAluno, ponteiroAluno->getNome());

        // Logo abaixo a forma moderna de fazer cast
        ponteiroPessoa = static_cast < Pessoa* > ( ponteiroAluno );

        // Logo abaixo a forma antiga e desaconselhavel de fazer cast
        // ponteiroPessoa = ( Pessoa* ) ponteiroAluno;

        LPessoas.incluaInfo ( ponteiroPessoa, ponteiroPessoa->getNome() );
}
```

```
void Principal::InicializaProfessores ( )
{
        Pessoa*         ponteiroPessoa;
        Professor*      ponteiroProfessor;

        // Inicialização do(s) ojeto(s) da classe Professor
        Simao.Inicializa        ( 3, 10, 1976, "Jean Simão" );
        Einstein.Inicializa     ( 14, 3, 1879, "Albert Einstein" );
        Newton.Inicializa       ( 4, 1, 1643, "Isaac Newton" );

        // "Filiação" ao departamento.
        Simao.setDepartamento           ( &EletronicaUTFPR );
        Einstein.setDepartamento        ( &FisicaPrinceton );
        Newton.setDepartamento          ( &MatematicaCambridge );

        ponteiroProfessor = &Simao;
        ponteiroPessoa = static_cast<Pessoa*>(ponteiroProfessor );
        LPessoas.incluaInfo (ponteiroPessoa, ponteiroPessoa->getNome());

        ponteiroProfessor = &Einstein;
        ponteiroPessoa = static_cast<Pessoa*>(ponteiroProfessor);
        LPessoas.incluaInfo (ponteiroPessoa, ponteiroPessoa->getNome());

        ponteiroProfessor = &Newton;
        ponteiroPessoa = static_cast<Pessoa*>(ponteiroProfessor);
        LPessoas.incluaInfo (ponteiroPessoa, ponteiroPessoa->getNome());
}
```

```
void Principal::MenuExe()
{   int op = -1;
    while ( op != 7 )
    {
        system ("cls");
        cout << "  Informe sua opção:       " << endl;
        cout << "  1 - Listar Disciplinas.  " << endl;
        cout << "  2 - Listar Departamentos.          " << endl;
        cout << "  3 - Listar Universidade. " << endl;
        cout << "  4 - Listar Alunos.                 " << endl;
        cout << "  5 - Listar Professores.  " << endl;
        cout << "  6 - Listar Pessoas.                " << endl;
        cout << "  7 – Sair.                " << endl;
        cin  >> op;

        switch ( op )
        {
                case 1:  {    Disciplinas.listeInfos();          system ("Pause");      } break;
                case 2:  {    LDepartamentos.listeInfos();       system ("Pause");      } break;
                case 3:  {    LUniversidades.listeInfos();       system ("Pause");      } break;
                case 4:  {    LAlunos.listeInfos();              system ("Pause");      } break;
                case 5:  {    DepOndeProfsTrabalham();           system ("Pause");      } break;

                case 6:  {
                              LPessoas.listeInfos();
                              system ("Pause");
                         } break;

                case 7:  {    cout << " FIM " << endl;                                  } break;

                default: {    cout << " Opção Inválida - Pressione uma tecla. " << endl;
                              getchar();
                         }
        }
    }
}
```


C:\WINDOWS\system32\cmd.exe
3 - Listar Universidade.
4 - Listar Alunos.
5 - Listar Professores.
6 - Listar Pessoas.
7 û Sair.
6
Elemento na lista AAA
Elemento na lista BBB
Elemento na lista CCC
Elemento na lista DDD
Elemento na lista EEE
Elemento na lista Jean Simòo
Elemento na lista Albert Einstein
Elemento na lista Isaac Newton
Elemento na lista joao
Pressione qualquer tecla para continuar. . .

```
void Principal::MenuExe()
{   int op = -1;
    while ( op != 7 )
    {
        system ("cls");
        cout << "  Informe sua opção:       " << endl;
        cout << "  1 - Listar Disciplinas.  " << endl;
        cout << "  2 - Listar Departamentos.          " << endl;
        cout << "  3 - Listar Universidade. " << endl;
        cout << "  4 - Listar Alunos.                 " << endl;
        cout << "  5 - Listar Professores.  " << endl;
        cout << "  6 - Listar Pessoas.                " << endl;
        cout << "  7 – Sair.                " << endl;
        cin  >> op;

        switch ( op )
        {
                case 1:  {    Disciplinas.listeInfos();          system ("Pause");      } break;
                case 2:  {    LDepartamentos.listeInfos();       system ("Pause");      } break;
                case 3:  {    LUniversidades.listeInfos();       system ("Pause");      } break;
                case 4:  {    LAlunos.listeInfos();              system ("Pause");      } break;
                case 5:  {    DepOndeProfsTrabalham();           system ("Pause");      } break;

                case 6:  {
                              LPessoas.listeInfos();
                              system ("Pause");
                         } break;

                case 7:  {    cout << " FIM " << endl;                                  } break;

                default: {    cout << " Opção Inválida - Pressione uma tecla. " << endl;
                              getchar();
                         }
        }
    }
}
```

Obs. a parte: o comando *system* nos permite introduzir o princípio de API, por não deixar de ser uma (mesmo que simplificadamente).

API (*Application Programming Interface*) permite ao programa acessar uma outra aplicação via uma 'interface'.

Neste caso, o *system* é a interface que permite acessar comandos do DOS.

Exercício 1

- Criar uma lista no sistema com todas as pessoas criadas independentemente de suas especializações (Professor, Aluno).

- **Na classe *Professor* criar um atributo *Salario,* criar um atributo *Bolsa Projeto* e um método *InformaProventos.***

- Criar uma classe *Estagiario* derivada de *Aluno*, com um atributo *BolsaEstudo* e um método *InformaProventos*.

```
#ifndef _PROFESSOR_H_
#define _PROFESSOR_H_

#include "Pessoa.h"
#include "Conhecimento.h"
#include "Universidade.h"

class Professor : public Pessoa
{
private:

   Universidade*   pUnivFiliado;
   Departamento*  pDptoFiliado;
   float              salario;
   float              bolsa_projeto;

public:

    Professor(int diaNa, int mesNa, int anoNa, char* nome = "");
    Professor();
    ~Professor();

    void inicializa();
    void setUnivFiliado(Universidade* pu);
    void setDepartamento(Departamento* pdep);
    void OndeTrabalho();
    void QualDepartamentoTrabalho();

     void setSalario ( float s );
     float getSalario ();

     void setBolsaProjeto( float s );
     float getBolsaProjeto ();

     void informaProventos();

};

#endif
```

```
#include “stdafx.h”
#include "Professor.h"

Professor::Professor(int diaNa, int mesNa, int anoNa, char* nome"):
Pessoa(diaNa, mesNa, anoNa, nome)
{
  inicializa();
}

Professor::Professor():
Pessoa()
{
  inicializa();
}

void Professor::inicializa()
{
  salario                    = 0.0;
  bolsa_projeto              = 0.0;
  . . .
}

//   …

void Professor::setSalario (float s)
{
    salario = s;
}

float Professor::getSalario ( )
{
    return salario;
}

void Professor::informaProventos ( )
{
   cout << "O valor dos proventos do Prof. "  << nomeP
        << "  é " <<  (salario+bolsa_projeto) << endl;
}
```

```
void Principal::InicializaProfessores ( )
{
   Pessoa*                 ponteiroPessoa;
   Professor*              ponteiroProfessor;

   Simao.Inicializa        ( 3, 10, 1976, "Jean Simão" );
   Einstein.Inicializa     ( 14, 3, 1879, "Albert Einstein" );
   Newton.Inicializa       ( 4, 1, 1643, "Isaac Newton" );

   Simao.setBolsaProjeto (1000);
   Simao.setSalario (4000);

   Einstein.setSalario (25000);

   Newton.setSalario (25000);

   Simao.setUnivFiliado     ( &UTFPR );
   Einstein.setUnivFiliado  ( &Princeton );
   Newton.setUnivFiliado    ( &Cambridge );

   Simao.setDepartamento ( &EletronicaUTFPR );
   Einstein.setDepartamento ( &FisicaPrinceton );
   Newton.setDepartamento ( &MatematicaCambridge );

   ponteiroProfessor = &Simao;
   ponteiroPessoa = static_cast<Pessoa*>(ponteiroProfessor);
   LPessoas.incluaInfo (ponteiroPessoa, ponteiroPessoa-> getNome());

   ponteiroProfessor = &Einstein;
   ponteiroPessoa = static_cast<Pessoa*>(ponteiroProfessor);
   LPessoas.incluaInfo (ponteiroPessoa, ponteiroPessoa->getNome());

   ponteiroProfessor = &Newton;
   ponteiroPessoa = static_cast<Pessoa*>(ponteiroProfessor);
   LPessoas.incluaInfo(ponteiroPessoa, ponteiroPessoa->getNome());
 }
```

Exercício 1

- Criar uma lista no sistema com todas as pessoas criadas independentemente de suas especializações (Professor, Aluno).

- Na classe *Professor* criar um atributo *Salario* e um método *InformaProventos.*

- **Criar uma classe *Estagiario* derivada de *Aluno*, com um atributo *BolsaEstudo* e um método *InformaProventos*.**

```
#ifndef _ESTAGIARIO_H_
#define _ESTAGIARIO_H_

#include "Aluno.h"

class Estagiario : public Aluno
{

protected:
  float BolsaEstudo;

public:
  Estagiario (int diaNa, int mesNa, int anoNa, char* nome = "");
  Estagiario();
  ~Estagiario();

  void inicializa();
  void setBolsaEstudo(float b);
  float getBolsaEstudo();

  void informaProventos();

};
#endif
```

```
#include "stdafx.h"
#include "Estagiario.h"

Estagiario::Estagiario (int diaNa, int mesNa, int anoNa, char* nome):
Aluno (diaNa, mesNa, anoNa, nome)
{
      inicializa();
}

Estagiario::Estagiario() : Aluno()
{
       inicializa();
}

Estagiario::~Estagiario()
{
}

void Estagiario::inicializa()
{
       BolsaEstudo = 0.0;
}

void Estagiario::setBolsaEstudo (float b)
{
        BolsaEstudo = b;
}

float Estagiario::getBolsaEstudo()
{
  return BolsaEstudo;
}

void Estagiario::informaProventos()
{
  cout << "O valor da bolsa de " << nomeP
       << " é " << BolsaEstudo << ".“  << endl;
}
```

```
#ifndef _PRINCIPAL_H_
#define _PRINCIPAL_H_

#include "Lista.h"
#include "Professor.h"
#include "Universidade.h"
#include "Departamento.h"
#include "Disciplina.h"
#include "Aluno.h"
#include "Estagiario.h"

class Principal
{
 private:
   . . .

   Aluno                AAA;
   Aluno                BBB;
   Aluno                CCC;
   Aluno                DDD;
   Aluno                EEE;

   Estagiario           Fulano;
   Estagiario           Ciclano;

   lista <Pessoa>       LPessoas;

  public:

   Principal ( );
   . . .
   void InicializaEstagiarios ();
   void Executar ();
   . . .
};
#endif
```

```
void Principal::InicializaEstagiarios ( )
{
        Pessoa*         ponteiroPessoa;
        Aluno*          ponteiroAluno;
        Estagiario*     ponteiroEstagiario;

        // ************************************
        Fulano.setNome      ("Fulano");
        Fulano.setBolsaEstudo (800);
        ponteiroEstagiario  = &Fulano;

        ponteiroAluno  = static_cast<Aluno*>(ponteiroEstagiario);
        LAlunos.incluaInfo ( ponteiroAluno, ponteiroAluno->getNome( ) );

        ponteiroPessoa = static_cast<Pessoa*>(ponteiroAluno);
        LPessoas.incluaInfo ( ponteiroPessoa, ponteiroPessoa->getNome( ) );

        // ************************************
        Ciclano.setNome     ("Ciclano");
        Ciclano.setBolsaEstudo (801);
        ponteiroEstagiario  = &Ciclano;

        ponteiroAluno  = static_cast<Aluno*>(ponteiroEstagiario);
        LAlunos.incluaInfo ( ponteiroAluno, ponteiroAluno->getNome( ) );

        ponteiroPessoa = static_cast<Pessoa*>(ponteiroAluno);
        LPessoas.incluaInfo ( ponteiroPessoa, ponteiroPessoa->getNome( ) );

}
```

Métodos ou ‘funções’ virtuais.

Métodos ou 'funções' virtuais.

```
#include <string>
#include <iostream>
using namespace std;
class Persona {
protected:
  string nomen;
public:
  Persona () { nomen = "Humanun";  }
  void getData ()  { cout << nomen << endl; }
  virtual void identificare ()  {  cout << "Persona!." << endl; }
};

class Alumnus : public Persona {
  int code;
public:
  Alumnus () :  Persona() {  code = 12; }
  void getData ()  { cout << nomen << " " << code << endl;  }
  void identificare () { cout << "Persona et alumnus." << endl; }
};

int main ( ) {
   Alumnus a;
   Persona* p = NULL;

   p = static_cast< Persona* >( &a );
   p->getData( );
   p->identificare ( );

   cout << "Pressione ENTER para continuar..." << endl;
   cin.get();
   return 0;
}
// Código adaptado de exemplo feito por Vagner Vengue.
```

Qualquer método pode ser redefinido em uma classe derivada. Por exemplo, o método *getData()* definido na classe *Persona* (ao lado) pode ser redefinido na classe derivada *Alumnus*. Assim, um objeto de *Alumnus* usaria o método *getData()* definido na sua classe. Entretanto, se ele fosse 'convertido' (*cast*) para *Persona* ou apontado como *Persona*, ai nesta dada conversão utilizaria o método *getData()* definido na sua superclasse (ou seja, na classe base).

Entretanto, se o método for virtual, como no caso de *identificare()*, mesmo se apotando por ponteiro de classe base, continua a ser executado o código da classe derivada.

Por exemplo, no código ao lado, na função *main*, primeiramente foi criado um objeto da classe *Alumnus* (derivada da classe *Persona*) e um ponteiro de objeto da classe base *Persona* (também chamada de superclasse). Subsequentemente, foi então feita a conversão de apontamento do endereço do objeto de *Alumnus* para um ponteiro *Persona*. Deste modo, dado que *identificare()* é método virtual, o programa deve imprimir no console o seguinte como resultado :

"Humanun"

"Persona et alumnus."

Caso a função *identificare()* não fosse virtual na classe *Alumnus*, o resultado seria:

"Humanun"

"Persona!"

Métodos ou ‘funções’ virtuais.

Aplicando no Exercício 1...

Métodos ou 'funções' virtuais.

```
#ifndef _PESSOA_H_
#define _PESSOA_H_
class Pessoa
{
protected:
   int      diaP;
   int      mesP;
   int      anoP;
   int      idadeP;
   char     nomeP[30];

public:
   Pessoa ( int diaNa, int mesNa, int anoNa, char* nome = "" );
   Pessoa ( ) ;
   void   Inicializa (int diaNa, int mesNa, int anoNa, char* nome = "");
   void   Calc_Idade ( int diaAT, int mesAT, int anoAT );
   void   Calc_Idade ( int anoAT);
   int    informaIdade ( );

   void     setNome ( char* n );
   char*    getNome ( );
   virtual void informaProventos();

};
#endif
```

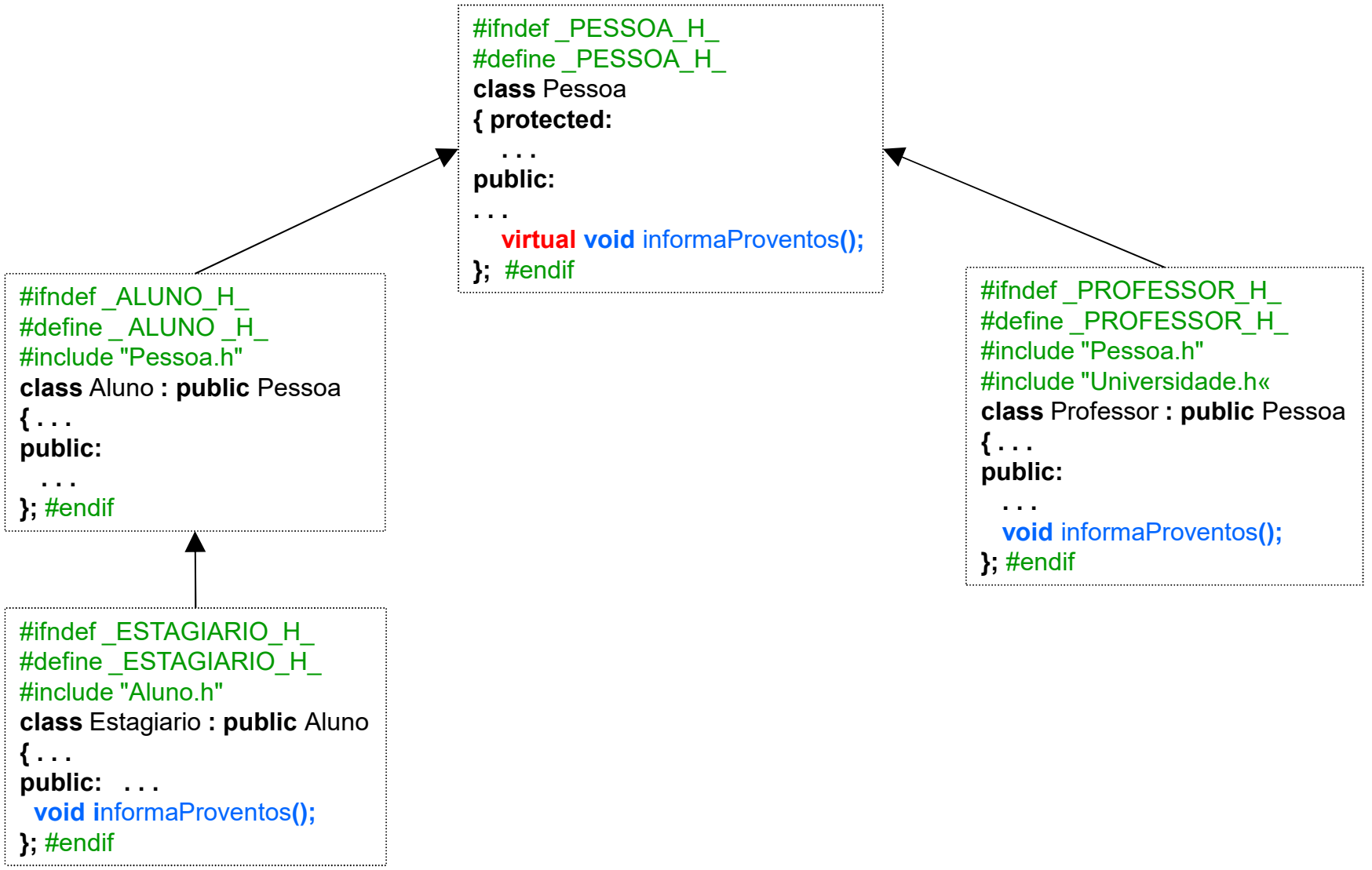
Novamente:

Qualquer método pode ser redefinido em uma classe derivada. Por exemplo, o método *informaIdade()* poderia ser redefinido na classe *Professor*.

Assim um objeto *Professor* usaria o método (ou função-membro) *informaIdade()* definida na sua classe. Entretanto, se ele fosse 'convertido' (*cast*) para *Pessoa* ou apontado como *Pessoa*, ele utilizaria o método *informaIdade()* definida na sua superclasse.

Já no caso de uma 'função' ou método virtual, mesmo que o objeto seja apontado como sua superclasse, ele continuará utilizando o método de sua classe!

```
. . .
void Pessoa::informaProventos()
{
   cout << "Nenhuma informação sobre proventos de " << nomeP << "." << endl;
}
```


#ifndef _PESSOA_H_
#define _PESSOA_H_
class Pessoa
{ protected:
. . .
public:
. . .
virtual void informaProventos();
}; #endif
#ifndef _ALUNO_H_
#define _ ALUNO _H_
#include "Pessoa.h"
class Aluno : public Pessoa
{ . . .
public:
. . .
}; #endif
#ifndef _PROFESSOR_H_
#define _PROFESSOR_H_
#include "Pessoa.h"
#include "Universidade.h«
class Professor : public Pessoa
{ . . .
public:
. . .
void informaProventos();
}; #endif
#ifndef _ESTAGIARIO_H_
#define _ESTAGIARIO_H_
#include "Aluno.h"
class Estagiario : public Aluno
{ . . .
public: . . .
void informaProventos();
}; #endif

Métodos ou 'funções' virtuais.

```
#ifndef _PESSOA_H_
#define _PESSOA_H_
class Pessoa
{
protected:
   int      diaP;
   int      mesP;
   int      anoP;
   int      idadeP;
   char     nomeP[30];

public:
   Pessoa ( int diaNa, int mesNa, int anoNa, char* nome = "" );
   Pessoa ( ) ;
   void   Inicializa (int diaNa, int mesNa, int anoNa, char* nome = "");
   void   Calc_Idade ( int diaAT, int mesAT, int anoAT );
   void   Calc_Idade ( int anoAT);
   int    informaIdade ( );

   void     setNome ( char* n );
   char*    getNome ( );
   virtual void informaProventos();

};
#endif
```

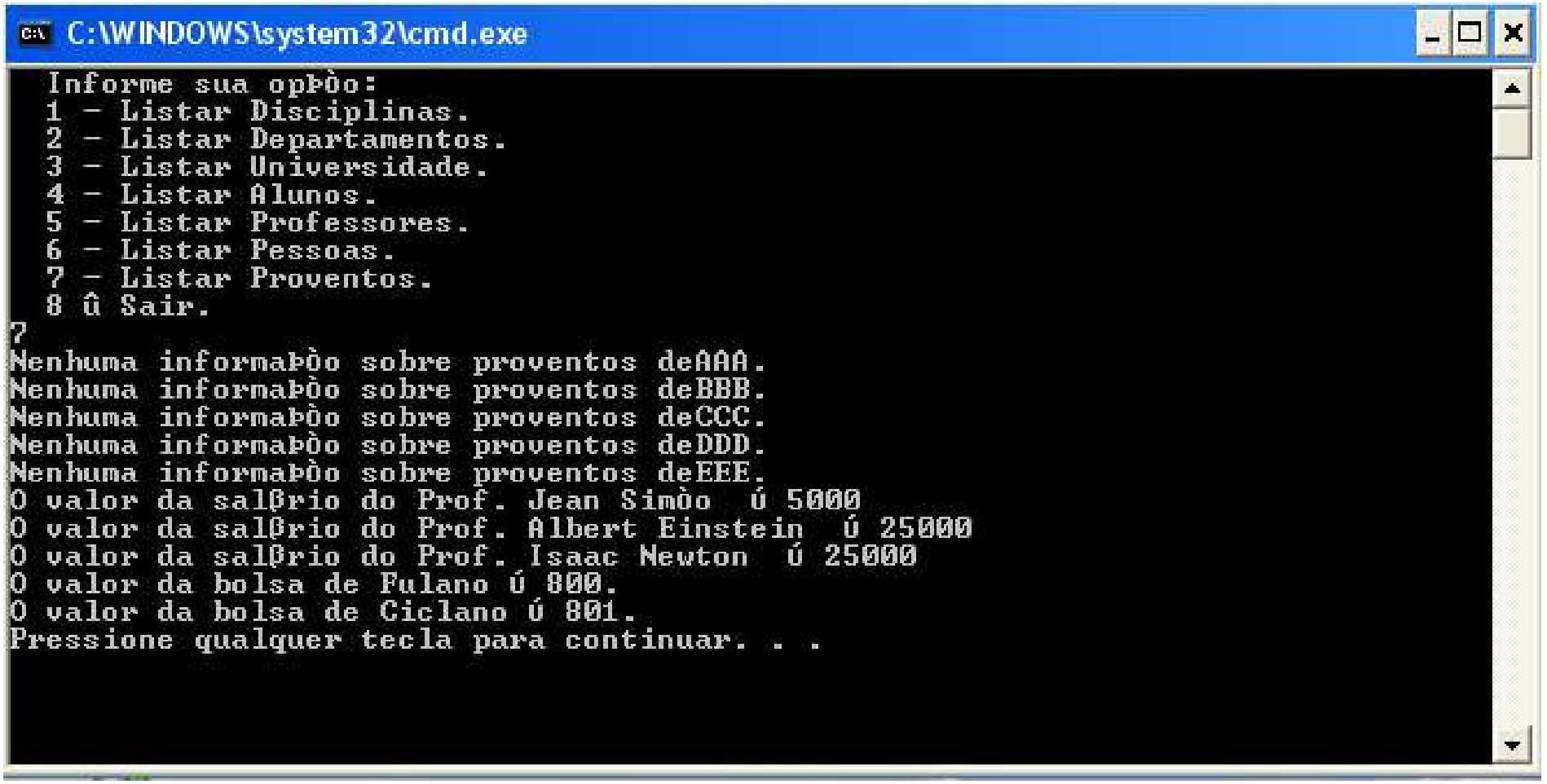
Tornar o método informaProventos() virtual possibilitará às classes derivadas redefinir o método e manter esta redefinição mesmo que o objeto seja "convertido" ou apontado para a classe base (Pessoa) por meio de ponteiros apropriados.

```
. . .
void Pessoa::informaProventos()
{
   cout << "Nenhuma informação sobre proventos de " << nomeP << "." << endl;
}
```

Ligação Dinâmica

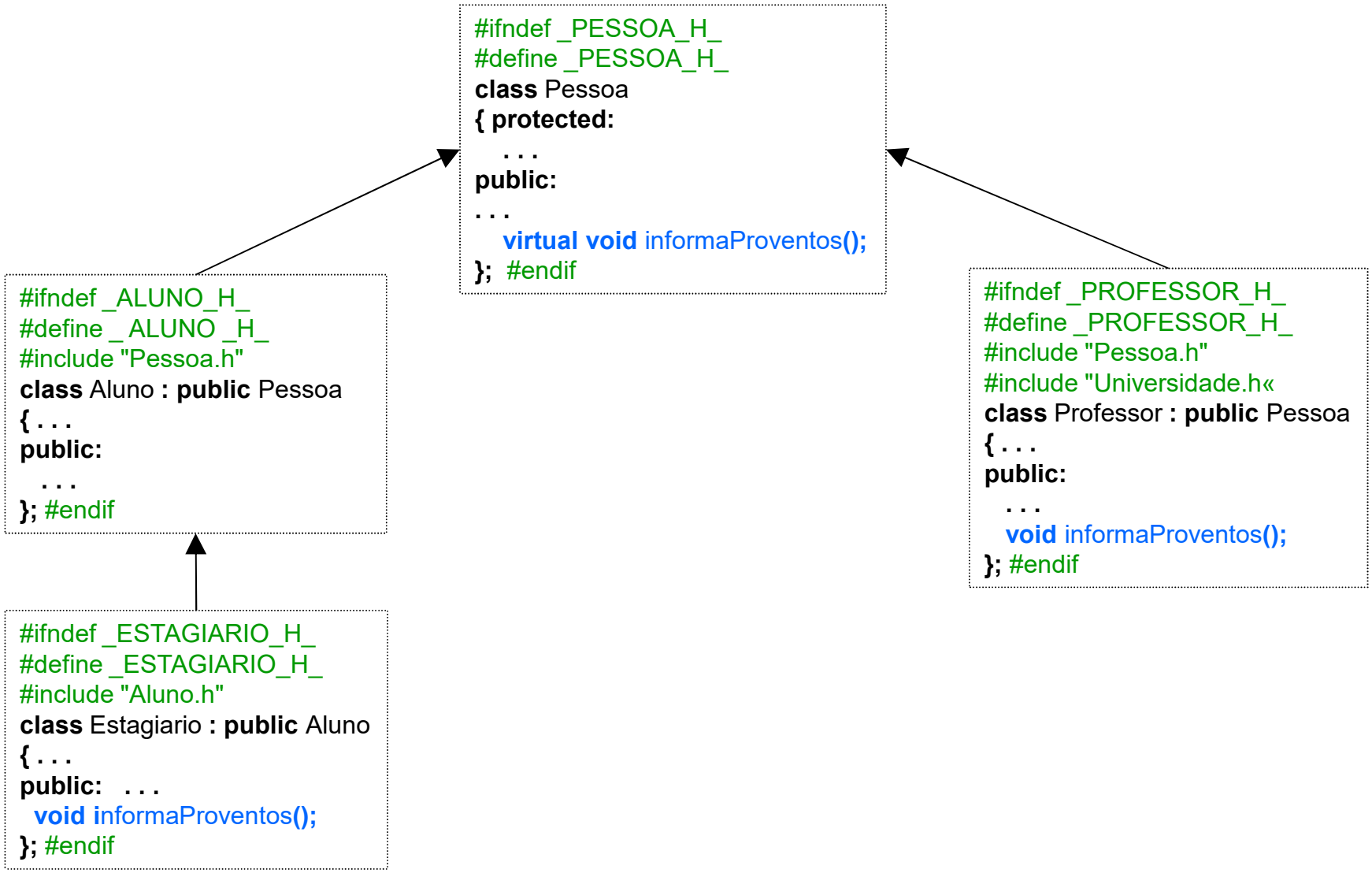
Exemplo de uso de função virtual.

```
void Principal::ListeProventosPessoas()
{
        Elemento <Pessoa>*   ponteiroElementoPessoa;
        Pessoa*              ponteiroPessoa;

        ponteiroElementoPessoa  = LPessoas.getPrimeiro();

        while ( ponteiroElementoPessoa != NULL )
        {
                ponteiroPessoa = ponteiroElementoPessoa->getInfo();

                ponteiroPessoa->informaProventos();

                ponteiroElementoPessoa = ponteiroElementoPessoa->getProximo();
        }
}
```

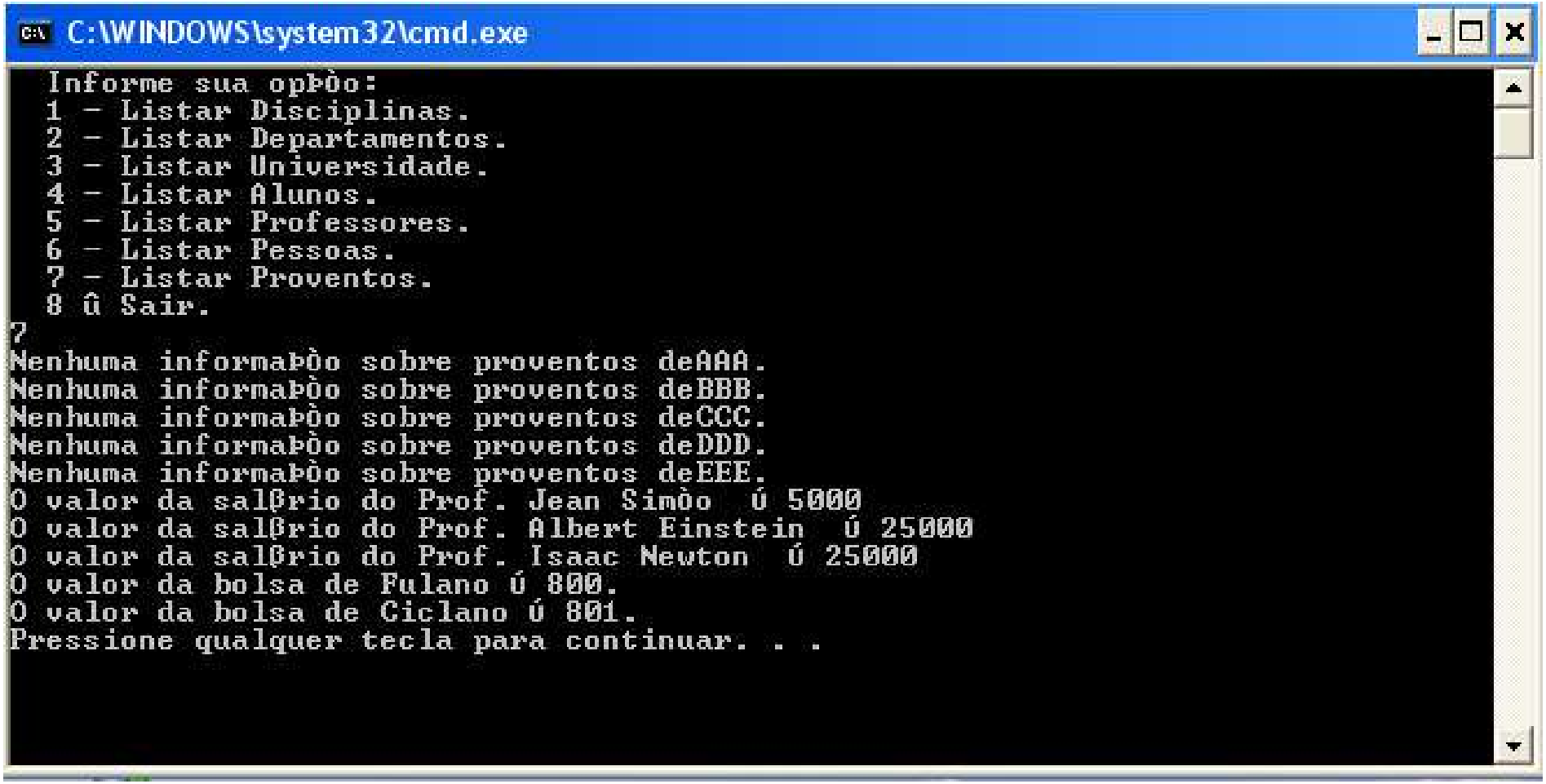

C:\WINDOWS\system32\cmd.exe
Informe sua opÞÒo:
1 - Listar Disciplinas.
2 - Listar Departamentos.
3 - Listar Universidade.
4 - Listar Alunos.
5 - Listar Professores.
6 - Listar Pessoas.
7 - Listar Proventos.
8 Û Sair.
7
Nenhuma informaÞÒo sobre proventos deAAA.
Nenhuma informaÞÒo sobre proventos deBBB.
Nenhuma informaÞÒo sobre proventos deCCC.
Nenhuma informaÞÒo sobre proventos deDDD.
Nenhuma informaÞÒo sobre proventos deEEE.
O valor da salßrio do Prof. Jean Simòo Ú 5000
O valor da salßrio do Prof. Albert Einstein Ú 25000
O valor da salßrio do Prof. Isaac Newton Ú 25000
O valor da bolsa de Fulano Ú 800.
O valor da bolsa de Ciclano Ú 801.
Pressione qualquer tecla para continuar. . .

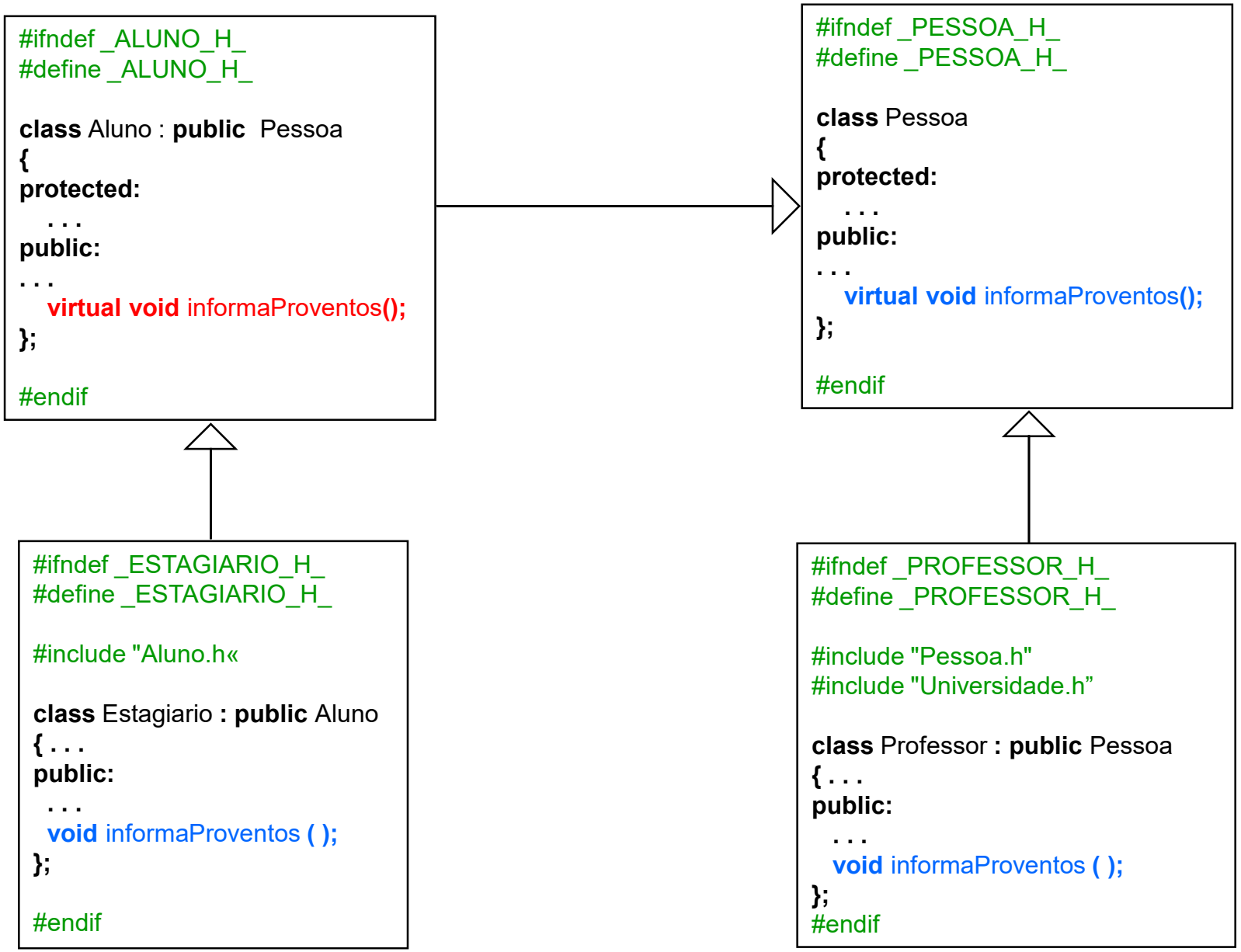
Base para o Polimorfismo

Métodos (funções-membro) polimórficos: Dois métodos em classes diferentes com mesma assinatura (i.e., mesma forma), mas com implementações diferentes, sendo estas classes derivadas de uma mesma classe base e estes métodos reimplementações de um método virtual definido nesta classe base.



Polimorfismo ocorrendo




#ifndef _ALUNO_H_
#define _ALUNO_H_
class Aluno : public Pessoa
{
protected:
. . .
public:
. . .
virtual void informaProventos();
};
#endif
#ifndef _PESSOA_H_
#define _PESSOA_H_
class Pessoa
{
protected:
. . .
public:
. . .
virtual void informaProventos();
};
#endif
#ifndef _ESTAGIARIO_H_
#define _ESTAGIARIO_H_
#include "Aluno.h«
class Estagiario : public Aluno
{ . . .
public:
. . .
void informaProventos ( );
};
#endif
#ifndef _PROFESSOR_H_
#define _PROFESSOR_H_
#include "Pessoa.h"
#include "Universidade.h”
class Professor : public Pessoa
{ . . .
public:
. . .
void informaProventos ( );
};
#endif

Sobrecarga de métodos

```
#ifndef _PESSOA_H_
#define _PESSOA_H_

class Pessoa
{
protected:
   int diaP;
   int mesP;
   int anoP;
   int idadeP;
   char nomeP[30];

public:

   Pessoa(int diaNa, int mesNa, int anoNa, char* nome = " ");
   Pessoa();

   ~Pessoa();

   . . .
  // Sobrecarga de função NÃO é polimorfismo!

   void Calc_Idade(int diaAT, int mesAT, int anoAT);
   void Calc_Idade(int anoAT);

   int  informaIdade();
   virtual void informaProventos();
};
#endif
```

ATENÇÃO: Reestudar sobrecarga de operadores! Parte da literatura chama sobrecarga de operadores como “polimorfismo de método”, o que não deve ser confundido com o polimorfismo propriamente dito, estudado nos *slides* anteriores, logo antes deste.

```
. . .
Pessoa::Pessoa(int diaNa, int mesNa, int anoNa, char* nome)
{
  Inicializa(diaNa, mesNa, anoNa, nome);
}

Pessoa::Pessoa()
{
}
. . .

void Pessoa::Calc_Idade(int diaAT, int mesAT, int anoAT)
{
   idadeP = anoAT - anoP;
    if ( mesP > mesAT )
    {
       idadeP = idadeP - 1;
    }
    else
    {
       if ( mesP == mesAT )
       {
           if ( diaP > diaAT )
           {
             idadeP = idadeP - 1;
           }
       }
    }
    printf("\n A idade da Pessoa %s é %d \n", nomeP, idadeP);
}

void Pessoa::Calc_Idade(int anoAT)
{
     idadeP = anoAT - anoP;
      printf("\n A idade da Pessoa %s é %d \n", nomeP, idadeP);
}
. . .
```

Herança Múltipla

Uma classe pode herdar o conteúdo de mais de uma classe.

Exemplo

```
#ifndef _CONHECIMENTO_H_
#define _CONHECIMENTO_H_
class Conhecimento
{ private:
  protected:
    // int id;
     char domino[150];
  public:
     Conhecimento ( int i = 0 );
     ~Conhecimento ( );
     char* getDominio ( );
     void  setDominio ( char* d );
};
#endif
```

```
#ifndef _LIVRO_H_
#define _LIVRO_H_

#include "Conhecimento.h"

class Livro : public Conhecimento
{
  private:
  protected:
     char nome[150];

  public:
     Livro ( int i = 0 );
     ~Livro();

     void setNome(char* n);
     char* getNome();
};


#endif
```

```
#ifndef _PROFESSOR_H_
#define _PROFESSOR_H_
#include "Pessoa.h"
#include "Universidade.h"
#include "Conhecimento.h"
class Professor : public Pessoa, public Conhecimento
{ private:
     Universidade* UnivFiliado;
     Departamento* DptoFiliado;
     . . .
 public:
     Professor ( int diaNa, int mesNa, int anoNa, char* nome = "" );
     Professor ( int i = 0 );
     ~Professor();
     void setUnivFiliado(Universidade* u);
     void setDepartamento(Departamento* dep);
     void OndeTrabalho();
     void QualDepartamentoTrabalho();
    . . .
 };
#endif
```

Exemplo

```
#ifndef _CONHECIMENTO_H_
#define _CONHECIMENTO_H_
class Conhecimento
{ private:
  protected:
     // int id;
     char domino[150];
  public:
     Conhecimento();
     ~Conhecimento();
     char* getDominio();
     void  setDominio(char* d);
};
#endif
```

```
#ifndef _PESSOA_H_
#define _PESSOA_H_
class Pessoa
{
protected:
    . . .
public:
. . .
    virtual void informaProventos();
};
#endif
```

```
#ifndef _LIVRO_H_
#define _LIVRO_H_
#include "Conhecimento.h“
class Livro : public Conhecimento
{
  private:
  protected:
     char nome[150];

  public:
     Livro(( int i = 0 );
     ~Livro();

     void setNome(char* n);
     char* getNome();
};
#endif
```

```
#ifndef _PROFESSOR_H_
#define _PROFESSOR_H_
#include "Pessoa.h"
#include "Universidade.h"
#include "Conhecimento.h"
class Professor : public Pessoa, public Conhecimento
{ private:
     Universidade* UnivFiliado;
     Departamento* DptoFiliado;
     . . .
 public:
     Professor(int diaNa, int mesNa, int anoNa, char* nome = "");
     Professor();
     ~Professor();
     void setUnivFiliado(Universidade* u);
. . .
 };
#endif
```

```
void Principal::InicializaProfessores()
{
   Pessoa*              ponteiroPessoa;
   Professor*           ponteiroProfessor;

   Simao.Inicializa     ( 3, 10, 1976, "Jean Simão" );
   Einstein.Inicializa  ( 14, 3, 1879, "Albert Einstein" );
   Newton.Inicializa    ( 4, 1, 1643, "Isaac Newton" );

   Simao.setBolsaProjeto (1000);
   Simao.setSalario (4000);

   Einstein.setSalario (25000);

   Newton.setSalario (25000);

   Simao.setUnivFiliado    ( &UTFPR );
   Einstein.setUnivFiliado  ( &Princeton );
   Newton.setUnivFiliado  ( &Cambridge );

   . . .

   // Area de Conhecimento.
   Simao.setDominio("Computação");
   Einstein.setDominio("Física");
   Newton.setDominio("Matemática-Física");

}
```

```
void Principal::ConhecProfs()
{

   // Conhecimento da Pessoa

   cout << Simao.getDominio() << endl;

   cout << Einstein.getDominio() << endl;

   cout << Newton.getDominio() << endl;

   printf("\n");

}
```

```
void Principal::InicializaProfessores()
{
   . . .
}
```

```
void Principal::ConhecProfs()
{
   . . .
}
```

Caso múltiplas superclasses tenham algum método com a mesma assinatura e deseje-se utilizar uma destas funções, basta identificar o método com o nome da superclasse. Exemplo:

```
class Conhecimento
{

 private:

 protected:

    int id;
    . . .

 public:

    . . .

};
```

```
class Professor : public Pessoa, public Conhecimento
{

. . .

public:

   . . .

      Professor(int RG = -1, int IdGeral = -1)
      {

            Pessoa::setId ( RG );

            Conhecimento::setId ( IdGeral );

            // muita atenção com isto tipo de situação.
      }

};

// código adaptado de exemplo feito por Vagner Vengue
```

Obs.: estudar o chamado “losango ou diamante da morte” nos livros de C++.

Função virtual pura

Métodos ou ‘funções’ virtuais puras.

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

class Persona
{ . . .
public:  . . .
        Persona ( ) { . . .  }
        // função virtual pura
        virtual void identificare( ) = 0;
};

class Alumnus : public Persona
{ . . .
public:  . . .
        Alumnus ( ) : Persona ( ) { . . . }
        void identificare ( )        {
                cout << "Alumnus." << endl;
        }
};

class Professor : public Persona
{ . . .
public: . . .
        Professor ( ) : Persona ( ) { . . .  }
        void identificare ( )
        {
                cout << "Professor." << endl;
        }
};
// Código adaptado de exemplo feito por Vagner Vengue.
```

```
int main ( )
{
        // Persona pes;                    // Erro;

        Persona* pes1; Persona* pes2;
        Alumnus alu;
        Professor pro;

        pes1 = static_cast< Persona* > ( &alu );
        pes1->identificare();

        pes2 = static_cast< Persona* > ( &pro );
        pes2->identificare();

        cout << "Pressione ENTER para continuar..." << endl;
        cin.get();   return 0;
}
```

Um método ou função virtual pura não é implementável na classe em questão e deve obrigatoriamente ser implementada nas classes derivadas! No código de exemplo ao lado, foi criada uma função virtual pura na classe *Persona* e depois implementada nas classes derivadas.

O modo de uso destas funções virtuais puras é igual ao da função virtual normal. Entretanto, pode-se utilizá-las para criar classes sem a necessidade de implementar todos os métodos, apenas as suas interfaces, podendo-se criar padrões para projetos, obrigando todas as classes derivadas a redefinirem os métodos desejados.

Classe Abstrata

Uma classe abstrata não pode ser instanciada (i.e., não se cria objetos a partir dela), mas pode ser derivada. Define-se uma classe abstrata por meio de pelo menos uma função virtual pura!

```
#ifndef _CONHECIMENTO_H_
#define _CONHECIMENTO_H_

class Conhecimento
{
  private:
  protected:
     int id;
     char domino[150];

  public:
     Conhecimento ( int i = 0  );
     ~Conhecimento();

     char* getDominio();
     void  setDominio(char* d);

     virtual void informaAntiguidade ( ) = 0;

};
#endif
```

Uma função virtual pura não é implementável na classe em questão e deve obrigatoriamente ser implementada nas classes derivadas!

```
#include "Conhecimento.h"
#include "stdio.h"

Conhecimento::Conhecimento ( int i = 0 )
{
   id = i;
}

Conhecimento::~Conhecimento()
{

}

void Conhecimento::setDominio(char* d)
{
   strcpy(domino, d);
}

char* Conhecimento::getDominio()
{
   return domino;
}
```

```
#ifndef _LIVRO_H_
#define _LIVRO_H_
#include "Conhecimento.h"

class Livro : public Conhecimento
{
  private:
  protected:
    char nome[150];
    int ano_impressao;
  public:
    Livro ( int i = 0,  int ano_imp = 0 ) ;
    ~Livro();

    void setNome ( char* n );
    char* getNome ( );

    void informaAntiguidade ( );
};

#endif
```

```
. . .
void Livro:: informaAntiguidade ( )
{
   cout << ano_impressao << endl;
}
```

```
#ifndef _PROFESSOR_H_
#define _PROFESSOR_H_
#include "Pessoa.h"
#include "Conhecimento.h"
#include "Universidade.h"
class Professor : public Pessoa, public Conhecimento
{
private:
   Universidade*   UnivFiliado;
   Departamento*  DptoFiliado;
   float          salario;
   . . .
public:
    Professor ( int diaNa, int mesNa, int anoNa, char* nome = "");
    Professor ( int i = 0 );
   ~Professor();
    void inicializa();
    void setUnivFiliado (Universidade* u);
    void setDepartamento(Departamento* dep);
    void OndeTrabalho();
    void QualDepartamentoTrabalho();
    void MeuConhecimento();
    void setSalario(float s);
    float getSalario();
    void informaProventos();
    void informaAntiguidade ( );
    . . .
};
#endif
```

```
. . .
void Professor:: informaAntiguidade ( )
{
   cout  << idade << endl;
}
```

Exercício Proposto

a) Aplicar no código desenvolvido neste presente grupo de slides, a solução apresentada no grupo de slides 10 – parte B.

b) Localizar em livros ou páginas ou afins sobre C++ o chamado "losango da morte" ou "diamante da morte", reimplementado o exemplo lá disponível.

Itens para revisar ou estudar:

- Sobrecarga de métodos (ou funções membros ou ainda operações).
- Sobrecarga de operadores.
- Objetos Constantes (*const*).
- Classe Pré-definida *String*.
- Tipos de cast (*static cast*, *dynamic cast*, ...)
- Espaço de nomes (*namespace*).
- Classes Aninhadas.
- Atributos e métodos estáticos (*static*).