

Relatório das Práticas 4 e 5

Vítor Barbosa

4 de agosto de 2020

1 Introdução

Nesta prática, implementaremos um cadastro de Aluno e Professor, que herdarão da classe Pessoa. Inicialmente, faremos a leitura de dados pelo console e posteriormente criaremos um novo programa com uma janela de cadastro. Este foi o conteúdo abordado na aula 4.

Conforme solicitado na aula 5, estenderemos o programa de cadastro usando uma estrutura de dados para receber e armazenar os dados cadastrados. Também exibiremos os dados.

2 Cadastro com o Console

O primeiro programa é de linha de comando e conta com as classes Pessoa, Aluno, Professor e o *main.cpp*.

Tendo em mente que um bom código deve ser tão inteligível e bem comentado quanto possível, as explicações estão embutidas no próprio código.

Classe Pessoa

pessoa.h

```
#ifndef PESSOA_H
#define PESSOA_H
#include <string>

enum Tipo{GERAL,ALUNO,PROF};

class Pessoa{
protected:
```

```

std::string nome;
int idade;
Tipo tipo;

public:
Pessoa (std::string nome, int idade, Tipo tipo);
void definirNome(std::string nome);
std::string retornarNome();
void definirIdade(int idade);
int retornarIdade();
//Adicionei o Tipo para definir se é aluno ou prof ou qualquer outra coisa
Tipo retornarTipo();
virtual std::string retornarCurso();
virtual std::string retornarFormacao();
};
#endif

```

peessoa.cpp

```

#include <string>
#include "peessoa.h"

Pessoa::Pessoa (std::string nome, int idade, Tipo tipo): nome(nome),
    idade(idade), tipo(tipo){}

void Pessoa::definirNome(std::string nome){
    this->nome = nome;
}

std::string Pessoa::retornarNome(){
    return this->nome;
}

void Pessoa::definirIdade(int idade){
    //Lançar exceção se a idade for negativa
    if(idade<0) throw "Idade negativa";
    this->idade = idade;
}

int Pessoa::retornarIdade(){

```

```

return this->idade;
}

Tipo Pessoa::retornarTipo(){
return this->tipo;
}

/**
**Provi a implementação básica do método para que o compilador não acuse erro
ao tentarmos fazer uma chamada polimórfica.Como o método é virtual, o
compilador chamará a versão da subclasse se ele for sobreescrito.
**Por exemplo, para que possamos fazer o seguinte:
Pessoa p = new Aluno();
p.retornarCurso();
**Sem a implementação aqui, isso geraria um erro de método puramente virtual.
*/
std::string Pessoa::retornarCurso(){return ""};
std::string Pessoa::retornarFormacao(){return ""};

```

Classe Aluno

aluno.h

```

#ifndef ALUNO_H
#define ALUNO_H
#include <string>
#include "pessoa.h"

class Aluno: public Pessoa{

private:
std::string curso;

public:
Aluno(std::string nome, int idade, std::string curso);
Aluno(std::string nome, int idade);
void definirCurso(std::string curso);
//O especificador override é opcional
std::string retornarCurso() override;
};
#endif

```

aluno.cpp

```
#include "aluno.h"

//Este construtor inicializa a superclasse e a subclasse
//usando uma lista de inicialização
Aluno::Aluno(std::string nome, int idade, std::string curso):
    Pessoa(nome, idade, ALUNO), curso(curso){}
Aluno::Aluno(std::string nome, int idade):Pessoa(nome, idade, ALUNO){}

void Aluno::definirCurso(std::string curso){
    this->curso = curso;
}
std::string Aluno::retornarCurso() {
    return this->curso;
}
```

Classe Professor

professor.h

```
#ifndef PROFESSOR_H
#define PROFESSOR_H
#include "pessoa.h"

class Professor: public Pessoa{

private:
    std::string formacao;

public:
    Professor(std::string nome, int idade, std::string formacao);
    Professor(std::string nome, int idade);
    void definirFormacao(std::string);
    std::string retornarFormacao() override;
};
#endif
```

professor.cpp

```
#include "professor.h"

//Construtores com lista de inicialização
Professor::Professor(std::string nome, int idade, std::string formacao):
    Pessoa(nome,idade,PROF),formacao(formacao){}
Professor::Professor(std::string nome, int idade):Pessoa(nome,idade,PROF){}
void Professor::definirFormacao(std::string formacao){
    this->formacao = formacao;
}
std::string Professor::retornarFormacao(){
    return this->formacao;
}
```

Arquivo main.cpp

```
/*
Compile com g++ main.cpp aluno.cpp pessoa.cpp professor.cpp* -o main.exe
Ou simplesmente: g++ *.cpp -o main.exe
*/
#include <iostream>
//Para usar a função toupper() do C
#include <ctype.h>
#include "pessoa.h"
#include "aluno.h"
#include "professor.h"
#include <typeinfo>

//O std::vector permite acesso aleatório.
//A std::list não, mas tem inserção e remoção em tempo constante
//O std::array é como o vector, mas tem tamanho fixo

#include <vector>

using namespace std;

int main (int argc, char **argv){
```

```

/*É obrigatório que o vetor armazene os ponteiros
* Em c++, se você armazenar uma cópia da subclasse numa variável do tipo
  superclasse, ocorre o problema de slicing. Ele consiste na perda das
  partes da subclasse, só o que é da super é armazenado.
*A solução é armazenarmos os ponteiros para a subclasse na variável do
  tipo da superclasse
*/
vector<Pessoa*> pessoas;
char opt;

while(true){
    //Comandos para limpar o stdin. Sem eles, pode ficar algum caractere
    //no buffer e as instruções serão impressas várias vezes
    cin.clear();
    fflush(stdin);

    cout<<"\nOla! Selecione uma opcao: \n";
    cout<<"A- Adicione aluno, P- Adicione professor,\n
        L- listar, S- Sair \n";
    cout<<"Ha "<<pessoas.size()<<" entradas\n";
    string nome;
    string extra;
    string opt_line;
    int idade ;
    getline(cin,opt_line);
    opt = opt_line[0];
    opt = toupper(opt);
    if(opt=='A' || opt=='P'){
        string s;
        cout<<"Digite o nome: \n";
        getline(cin,nome);
        cout<<"Digite a idade \n";
        getline(cin,s);
        bool except = false;
        try{idade = stoi(s);}catch(exception e){except = true;}
        if(idade<0 || except) cout<<"Idade invalida \n";
        else if(opt=='A'){
            cout<<"Digite o curso: \n";
            cin>>extra;
            Aluno *a = new Aluno(nome,idade,extra);
            //cout<<a->retornarCurso();

```

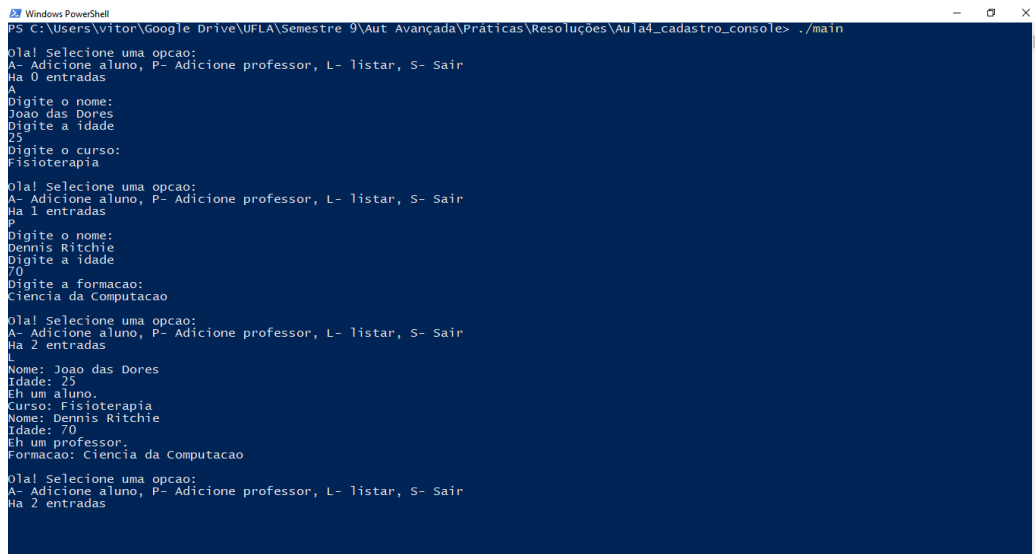
```

        pessoas.push_back(a);
    }
    else{
        cout<<"Digite a formacao: \n";
        getline(cin,extra);
        Professor *prof = new Professor(nome,idade,extra);
        pessoas.push_back(prof);
    }
}
else if (opt=='L'){
    for(Pessoa *p :pessoas){
        cout<<"Nome: "<<p->retornarNome()<<endl;
        cout<<"Idade: "<<p->retornarIdade()<<endl;
        if(p->retornarTipo()==ALUNO){
            cout<<"Eh um aluno."<<endl;
            cout<<"Curso: "<<p->retornarCurso()<<endl;
        }
        else if (p->retornarTipo()==PROF){
            cout<<"Eh um professor."<<endl;
            cout<<"Formacao: "<<p->retornarFormacao()<<endl;
        }
    }
}
else if(opt=='S'){
    pessoas.clear();
    break;
}
}
return 0;
}

```

3 Resultado no Console

Vejamos uma captura de tela da execução do programa abaixo.



```
PS C:\Users\vitor\Google Drive\UFLA\Semestre 9\Aut Avancada\Práticas\Resoluções\Aula4_cadastro_console> ./main
Olá! Selezione uma opcao:
A- Adicione aluno, P- Adicione professor, L- listar, S- Sair
Ha 0 entradas
Digite o nome:
Joao das Dores
Digite a idade:
25
Digite o curso:
Fisioterapia

Olá! Selezione uma opcao:
A- Adicione aluno, P- Adicione professor, L- listar, S- Sair
Ha 1 entradas
Digite o nome:
Dennis Ritchie
Digite a idade:
70
Digite a formacao:
Ciencia da Computacao

Olá! Selezione uma opcao:
A- Adicione aluno, P- Adicione professor, L- listar, S- Sair
Ha 2 entradas

Nome: Joao das Dores
Idade: 25
Eh um aluno.
Curso: Fisioterapia
Nome: Dennis Ritchie
Idade: 70
Eh um professor.
Formacao: Ciencia da Computacao

Olá! Selezione uma opcao:
A- Adicione aluno, P- Adicione professor, L- listar, S- Sair
Ha 2 entradas
```

Figura 1: Teste da Aplicação

4 Cadastro com Interface Gráfica

Todos os arquivos do programa para console foram mantidos, com exceção do *main.cpp*, e compuseram o que denominei *libCadastro*, como pode ser visto na figura 2.

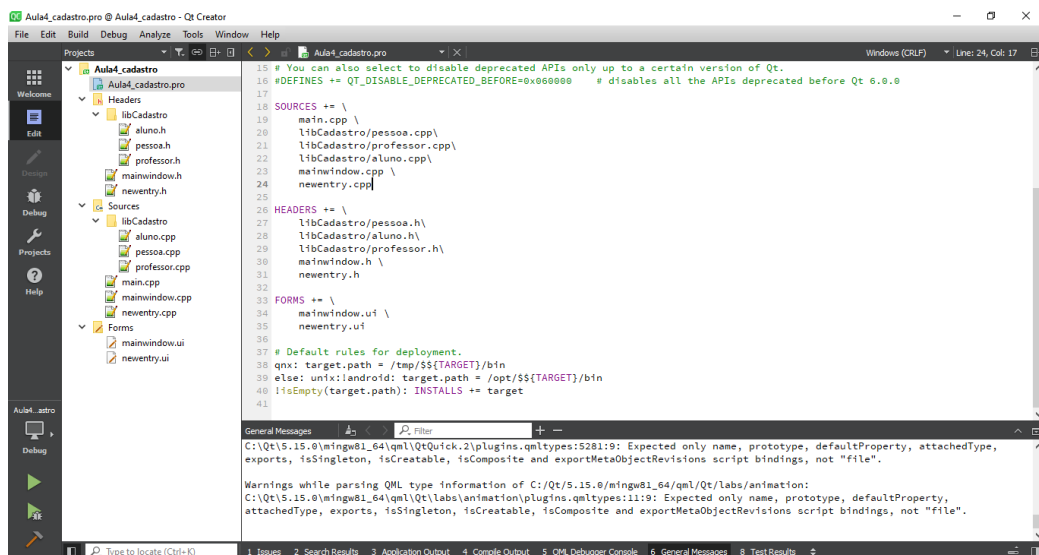


Figura 2: Arquivo .pro mostrando a inclusão das classes

Formulários de Cadastro e Exibição

Os formulários de cadastro e exibição(janela principal) foram feitos graficamente no Qt Creator, como pode ser visto nas figuras 3 e 4.

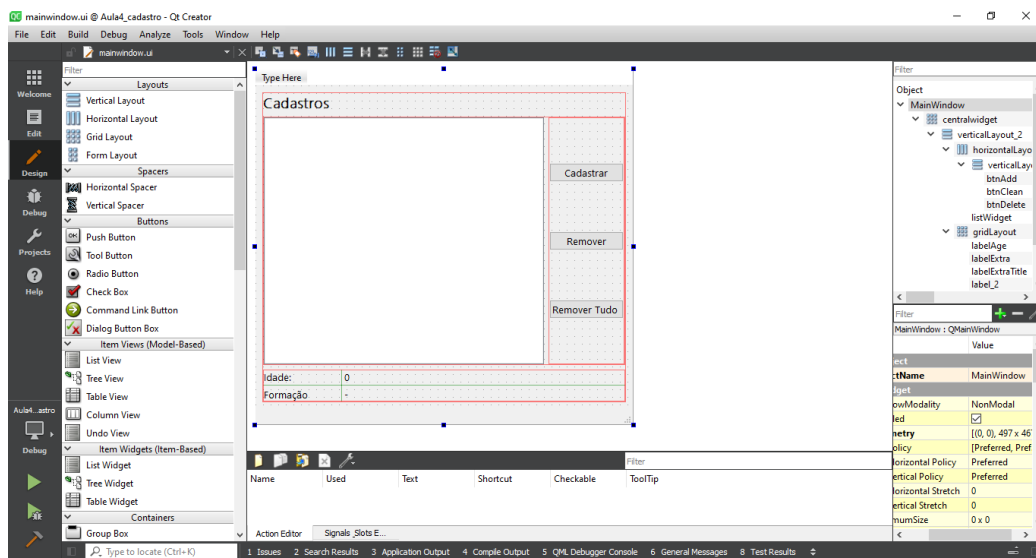


Figura 3: Criação do Formulário de Exibição

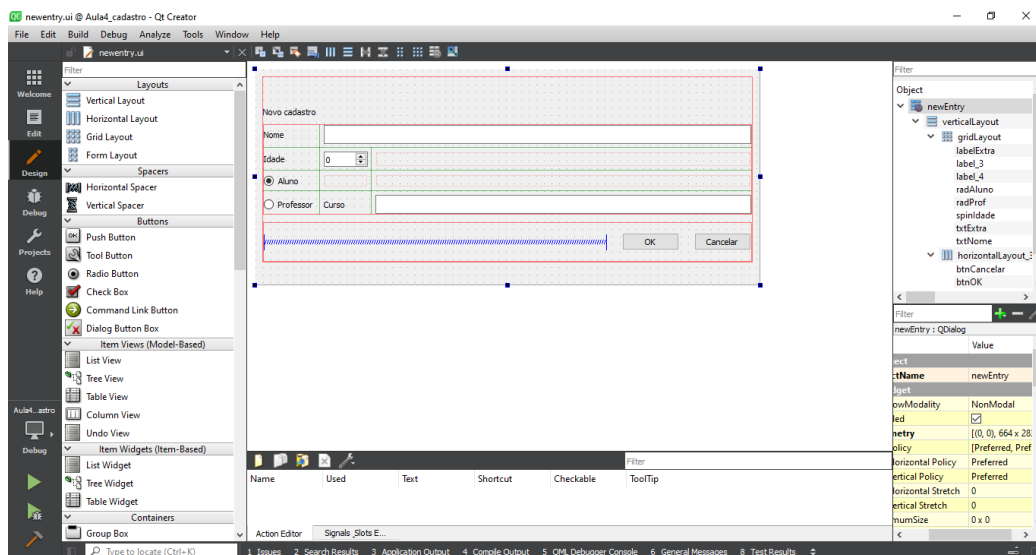


Figura 4: Criação do Formulário de Novo Cadastro

mainwindow.h

A Classe MainWindow contém a janela principal, que lista e exibe os cadastros. Este é seu arquivo de cabeçalho.

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "newentry.h"
#include "libCadastro/aluno.h"
#include "libCadastro/professor.h"

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow;}
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void on_btnAdd_clicked();
    void on_btnDelete_clicked();
    void on_btnClean_clicked();
    void on_listWidget_currentRowChanged(int currentRow);

private:
    Ui::MainWindow *ui;
    newEntry *janelaCadastro;
    QVector<Pessoa*> pessoas;

    void updateList();
    void changeEvent(QEvent *event);
};
#endif // MAINWINDOW_H
```

mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "newentry.h"
#include <QTextStream>

MainWindow::MainWindow(QWidget *parent): QMainWindow(parent),
    ↪ ui(new Ui::MainWindow){
    ui->setupUi(this);
    janelaCadastro = nullptr;
}

MainWindow::~MainWindow(){delete ui;}

void MainWindow::on_btnAdd_clicked()
{
    //Não use delete ou o programa vai dar crash, o Qt já
    ↪ gerencia a memória de forma transparente
    //if(janelaCadastro!=nullptr) delete janelaCadastro;

    //MUITO IMPORTANTE
    //Precisamos passar o endereço de memória do nosso vetor
    ↪ de pessoas, para que ele seja modificado pela outra
    ↪ janela
    //Use ponteiros sempre que possível, e evite variáveis
    ↪ referência. Ex.: use Pessoa *p em vez de Pessoa &p
    janelaCadastro = new newEntry(this,&(this->pessoas));
    janelaCadastro->show();
    this->hide();
}

void MainWindow::updateList(){
ui->listWidget->clear();
    for(Pessoa *p: pessoas){

        ↪ ui->listWidget->addItem((QString::fromStdString(p->retornarNome()))
        .prepend(p->retornarTipo()==PROF?"Professor -
        ↪ ":(p->retornarTipo()==ALUNO?"Aluno --- ":"")));
    }
}
```

```

    }
}

void MainWindow::changeEvent(QEvent *e){

    QWidget::changeEvent(e);
    if(e->type()==QEvent::ActivationChange)
        if(this->isActiveWindow()){
            updateList();
        }
}

void MainWindow::on_btnDelete_clicked(){
    if(!pessoas.isEmpty())
        pessoas.pop_back(),updateList();
}

void MainWindow::on_btnClean_clicked(){
    if(!pessoas.isEmpty())
        pessoas.clear(),updateList();
}

void MainWindow::on_listWidget_currentRowChanged(int
↪ currentRow){
    //Limite o índice ou teremos um bug ao acessar o vetor
    currentRow = currentRow>(pessoas.size()-1)? 0 :
    ↪ (currentRow<0? 0:currentRow);

    Pessoa* p = pessoas.at(currentRow);

    ↪ ui->labelAge->setText(QString::number(pessoas.at(currentRow)->retornarId
if(p->retornarTipo()==PROF){
    ui->labelExtraTitle->setText("Prof. Formação: ");

    ↪ ui->labelExtra->setText(QString::fromStdString(p->retornarFormacao())
}
else if(p->retornarTipo()==ALUNO){
    ui->labelExtraTitle->setText("Aluno. Curso: ");

    ↪ ui->labelExtra->setText(QString::fromStdString(p->retornarCurso()));
}
}

```

```
}
```

newentry.h

A Classe NewEntry contém as rotinas para criação de novos cadastros. Este é seu cabeçalho.

```
#ifndef NEWENTRY_H
#define NEWENTRY_H

#include <QDialog>
#include "libCadastro/aluno.h"
#include "libCadastro/professor.h"
#include <QMessageBox>

namespace Ui {
class newEntry;
}

class newEntry : public QDialog{
    Q_OBJECT
public:
    newEntry(QWidget *parent, QVector<Pessoa*> *pessoas);
    ~newEntry();

private slots:
    void on_btnOK_clicked();
    void on_btnCancelar_clicked();
private:
    Ui::newEntry *ui;
    QWidget *parent;
    QVector <Pessoa*> *pessoas;

    void init();
    void goBack(); //Volta à janela principal
};
#endif // NEWENTRY_H
```

newentry.cpp

```
#include "newentry.h"
#include "ui_newentry.h"
#include "container.h"

void newEntry::init(){
    QString strAluno = "Curso ";
    QString strProf = "Formação ";

    ↪ QObject::connect(ui->radProf,&QRadioButton::toggled,this,[=]{

        ↪ ui->labelExtra->setText(ui->radProf->isChecked()?strProf:strAluno);}
}

// Note que pessoas precisa ser um ponteiro para um vetor de
↪ ponteiros de Pessoa
// Os ponteiros de Pessoa são para evitar o slicing (perda de
↪ dados ao passar um tipo subclasse para a superclasse
// O ponteiro para o vetor é porque desejamos modificar e
↪ repassar os dados no lugar
//Evite armazenar referências, pois elas são confusas e
↪ causam erros
newEntry::newEntry(QWidget *parent,QVector<Pessoa*>
↪ *pessoas):QDialog(parent),
    ui(new Ui::newEntry),pessoas(pessoas){

    ui->setUpUi(this);
    this->parent = parent;
    init();
    //QMessageBox::information(this,"Info","Pessoas passado");
}

newEntry::~newEntry(){ delete ui; }

void newEntry::goBack(){
    this->close();
    parent->show();
}
```

```
}
```

```
void newEntry::on_btnOK_clicked()
{
    bool dadosOK = true;
    QString nome = ui->txtNome->toPlainText();
    int idade = ui->spinIdade->value();
    QString extra = ui->txtExtra->toPlainText();

    // Validando os dados
    if(nome.isEmpty()) dadosOK = false;
    if(dadosOK){
        if(ui->radAluno->isChecked())
            pessoas->push_back(new
                ↳ Aluno(nome.toStdString(),idade,extra.toStdString()));
        else if(ui->radProf->isChecked())
            pessoas->push_back(new
                ↳ Professor(nome.toStdString(),idade,extra.toStdString()));

        ↳ QMessageBox::about(this,QString::fromStdString(pessoas->last()->retornar
        ↳ com Sucesso!"));
    }
    else QMessageBox::warning(this,"Erro","O nome não pode
        ↳ estar em branco");

    if(dadosOK){
        goBack();
    }
}
```

```
void newEntry::on_btnCancelar_clicked(){ goBack(); }
```

main.cpp

Código padrão gerado pelo Qt creator, igual ao da prática anterior e não será incluído aqui.

5 Execução do Programa com Interface Gráfica

O programa com janelas funcionou como esperado e alguns testes podem ser vistos a seguir.

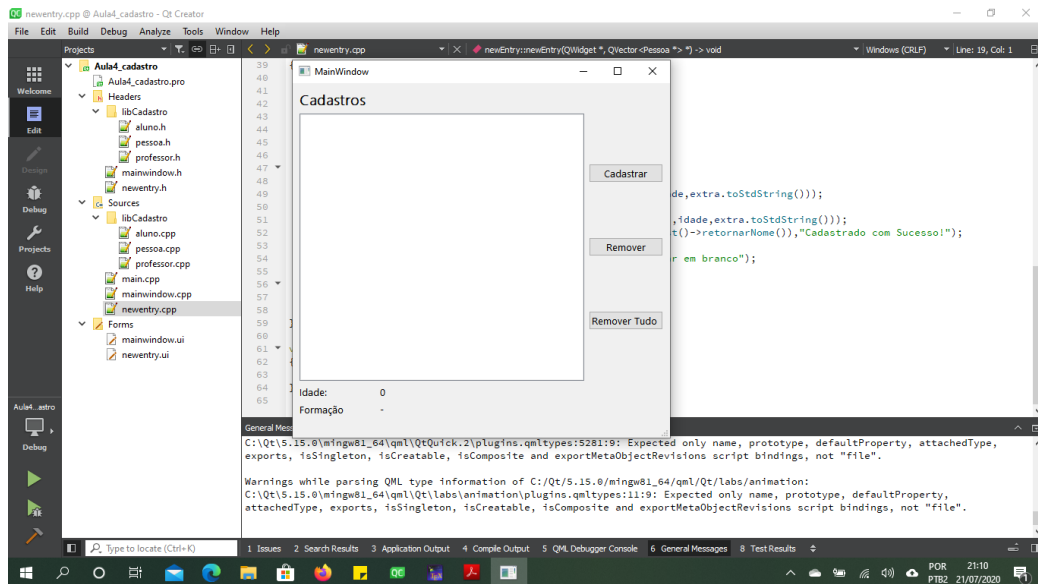


Figura 5: A Janela Principal logo após o início do programa

As figuras 6 e 7 mostram o cadastro de um aluno e um professor, respectivamente.

As figuras 8 e 9 mostram a listagem de todos os cadastros e a exibição dos detalhes de um aluno e um professor, respectivamente.

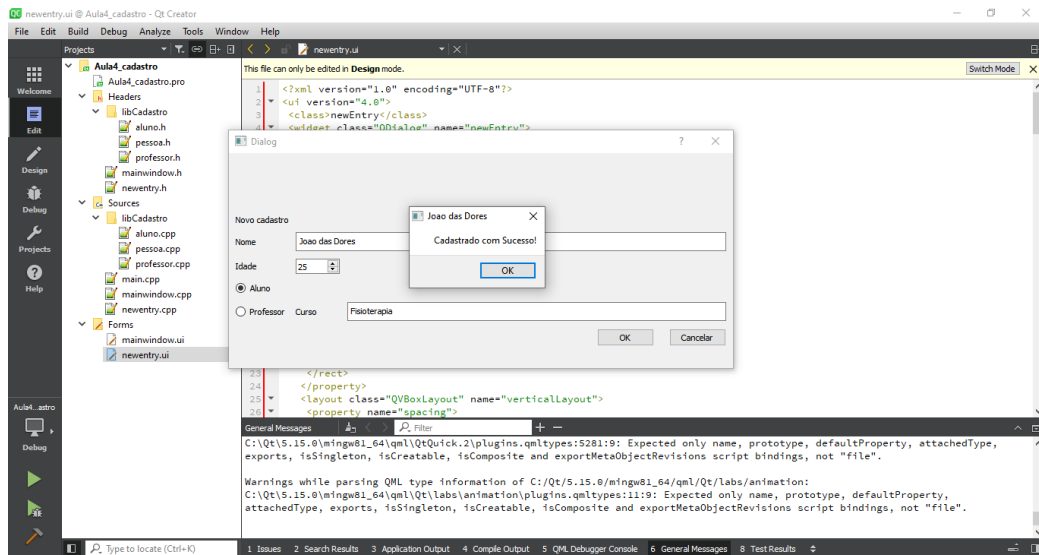


Figura 6: Cadastro de um Aluno

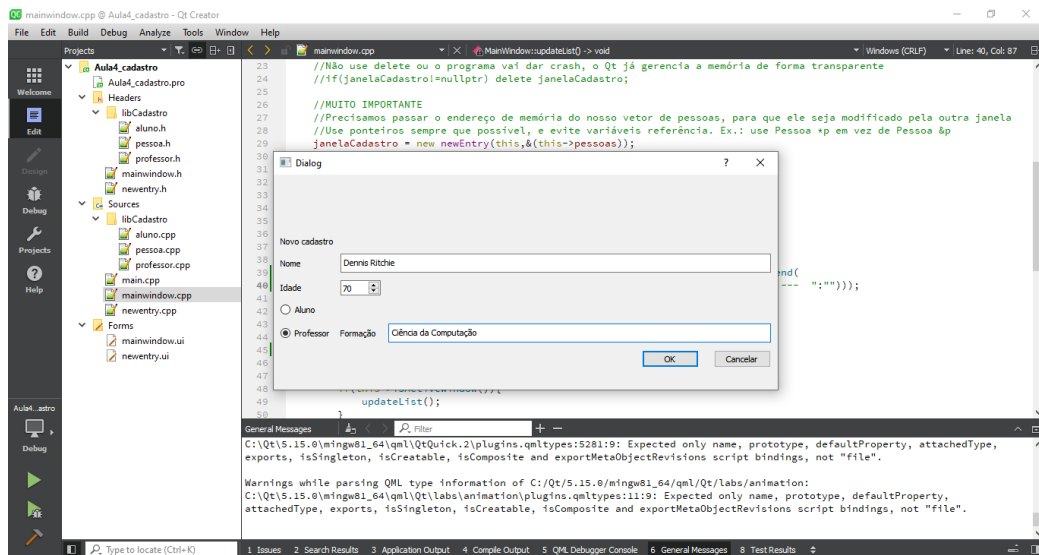


Figura 7: Cadastro de um Professor

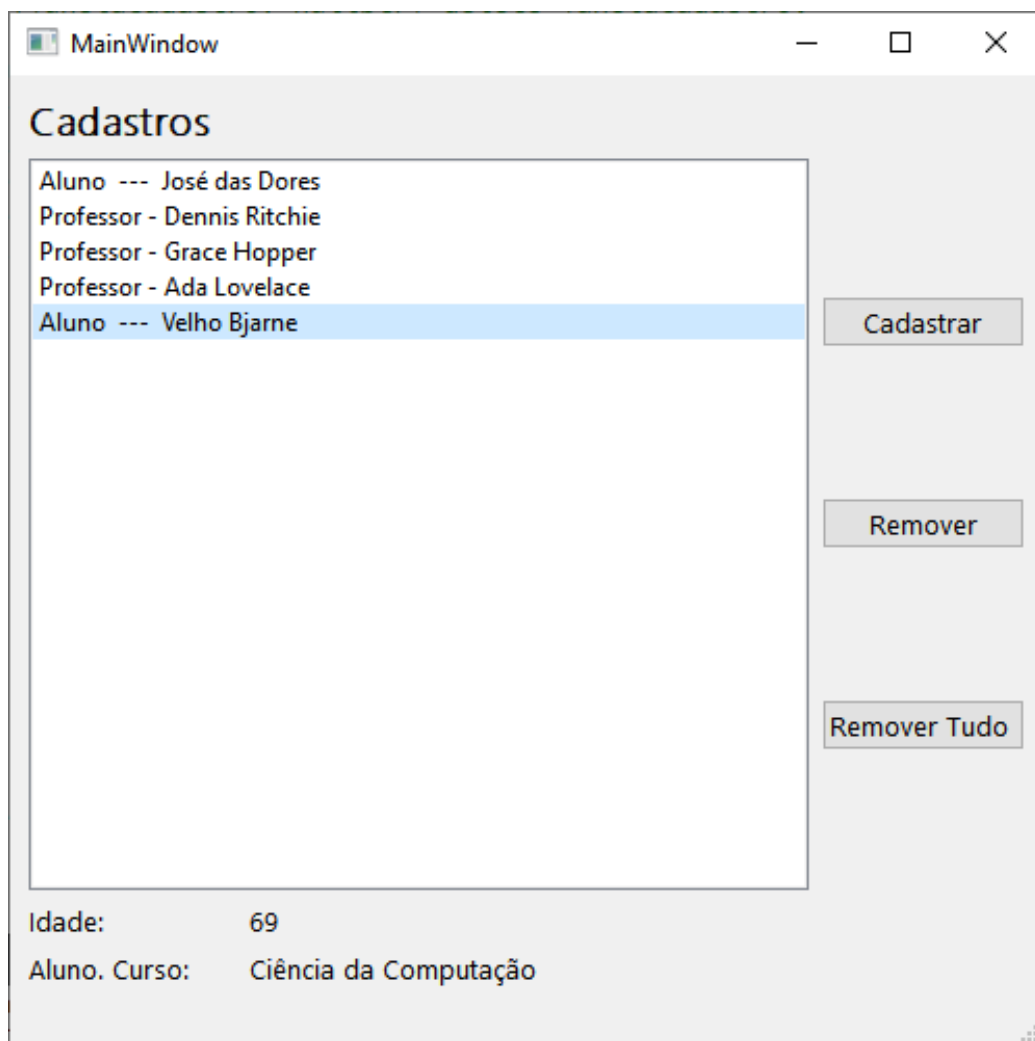


Figura 8: Listagem e exibição de um Aluno

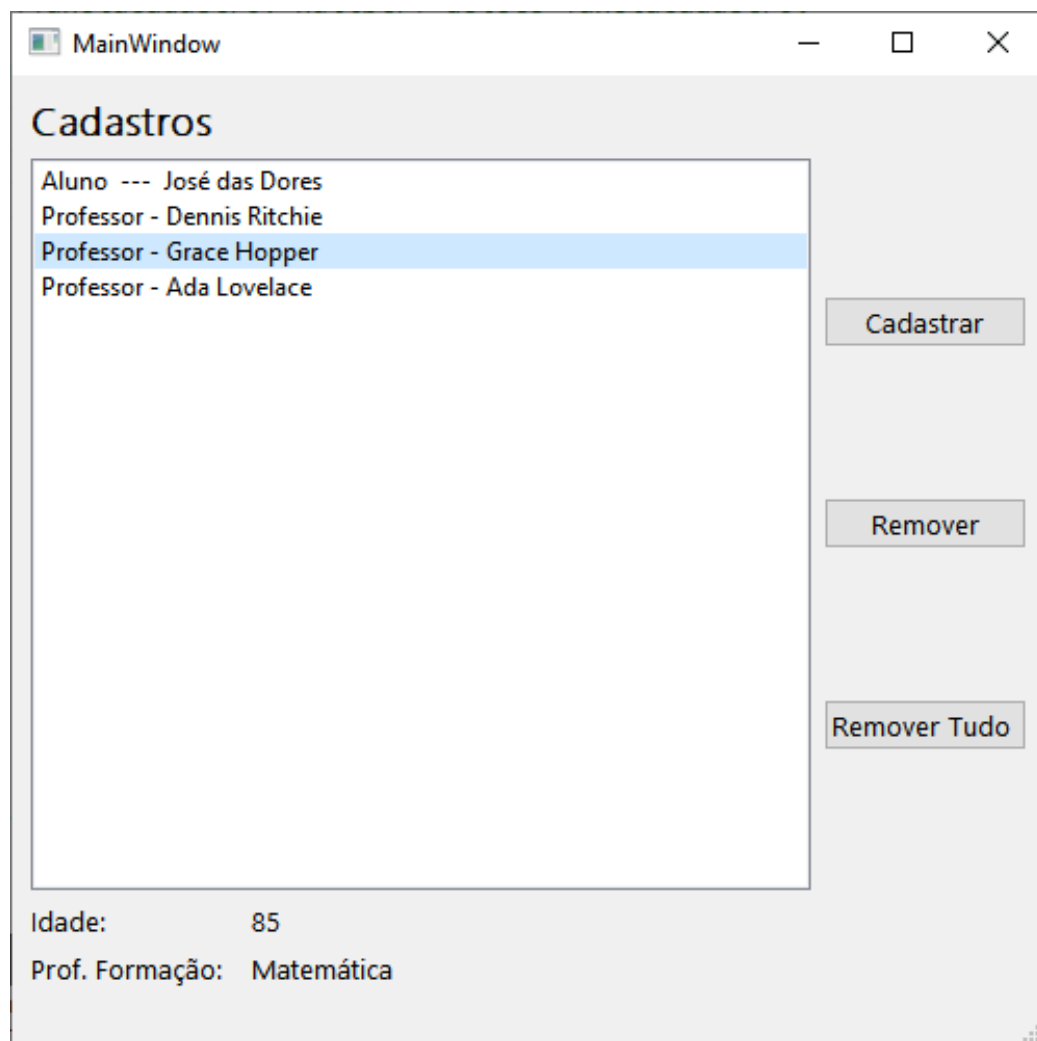


Figura 9: Listagem e exibição de um Professor