Linguagens de Programação

Prof. Miguel Elias Mitre Campista

http://www.gta.ufrj.br/~miguel

Exercício 1

- Escreva um programa em C++ para armazenar cadastros profissionais ou pessoais. Para isso, crie uma classe Register que irá conter dados comuns aos cadastros profissionais e pessoais (Nome, CPF e Idade) e crie duas classes derivadas, a classe ProfessionalRegister e a classe PersonalRegister. A classe ProfessionalRegister adiciona os atributos profissão e área enquanto a classe PersonalRegister adiciona os atributos endereço e telefone. Todas as classes implementam funções públicas do tipo "get". Os cadastros são armazenados em um objeto da classe Agenda que contém um vector de objetos da classe Register. Logo, mesmo se os cadastros utilizados forem de classes derivadas, eles poderão ser armazenados no vector. Utilize o conceito de polimorfismo. A classe Agenda implementa uma função de inserção e de obtenção de elementos. Realize tratamento de exceção em ambas as funções para proteção de acesso a elementos inexistentes.

Exercício 1

```
#include "agenda.h"
#include <iomanip>

int main () {
        Agenda professionalAgenda (1);

        ProfessionalRegister *pr1 = new ProfessionalRegister ("Miguel", "123", 30, "Prof", "Electronic");
        professionalAgenda.insert (pr1);

        // Inserção de elemento fora do limite máximo
        // O objetivo é forçar uma exceção
        ProfessionalRegister *pr2 = new ProfessionalRegister ("Miguel", "123", 30, "Prof", "Electronic");
        professionalAgenda.insert (pr2);

        // Uso do conceito de polimorfismo para construção da agenda, obriga
        // o uso de um downcast para objetos da classe ProfessionalRegister
        ProfessionalRegister *pr3 = dynamic_cast <ProfessionalRegister *> (professionalAgenda.get (0));
        cout << left << endl;
        cout << setw (10) << "Data: " << setw(20) << "Content:" << endl;
        cout << setw (10) << "Name: " << setw(20) << pr3->getName () << "\n"
             << setw (10) << "CPF: " << setw(20) << pr3->getCPF () << "\n"
             << setw (10) << "Age: " << setw(20) << pr3->getAge () << "\n"
             << setw (10) << "Job: " << setw(20) << pr3->getJob () << "\n"
             << setw (10) << "Area: " << setw(20) << pr3->getArea () << "\n" << endl;

        // Objetivo é novamente forçar uma exceção
        ProfessionalRegister *pr4 = dynamic_cast <ProfessionalRegister *> (professionalAgenda.get (1));
        cout << endl;
```

Exercício 1

```
        // Desalocar da memória todos os objetos criados com o operador new
        delete pr1;
        delete pr2;

        // Criação de uma agenda pessoal
        // Atenção! Criação de um novo objeto da classe Agenda
        // Agora, porém, para colocar objetos de outra classe derivada de Register
        // Vantagem do polimorfismo! Permite usar a mesma classe Agenda para armazenar
        // objetos de classes diferentes derivadas de Register
        Agenda personalAgenda (1);

        PersonalRegister *pr5 = new PersonalRegister ("Miguel", "123", 30, "Rua A", "778899");
        personalAgenda.insert (pr5);

        PersonalRegister *pr6 = dynamic_cast <PersonalRegister *> (personalAgenda.get (0));
        cout << endl;
        cout << setw (10) << "Data:" << setw(20) << "Content:" << endl;
        cout << setw (10) << "Name:" << setw(20) << pr6->getName () << "\n"
             << setw (10) << "CPF: " << setw(20) << pr6->getCPF () << "\n"
             << setw (10) << "Age: " << setw(20) << pr6->getAge () << "\n"
             << setw (10) << "Address: " << setw(20) << pr6->getAddress () << "\n"
             << setw (10) << "Cell: " << setw(20) << pr6->getCell () << "\n" << endl;

        delete pr5;

        return 0;
}
```

Exercício 1

```
#include <string>
#include <iostream>
#include <iomanip>

#ifndef REGISTER_H
#define REGISTER_H

using namespace std;

class Register {
        public:
                Register (string, string, int);
                // Para ser polimórfica, a classe base tem que ter
                // pelo menos um método virtual
                virtual ~Register () {}

                string getName () const;
                string getCPF () const;
                int getAge () const;

        private:
                string name, cpf;
                int age;
};

#endif
```

Exercício 1

```
#include "register.h"

Register::Register (string n, string c, int a) : name (n), cpf (c), age (a) {}

string Register::getName () const { return name; }

string Register::getCPF () const { return cpf; }

int Register::getAge () const { return age; }
```

Exercício 1

```
#include <iostream>
#include <string>
#include "register.h"

#ifndef PROFESSIONALREG_H
#define PROFESSIONALREG_H

using namespace std;

class ProfessionalRegister: public Register {
        public:
                ProfessionalRegister (string, string, int, string, string);
                virtual ~ProfessionalRegister () {}

                string getJob () const;
                string getArea () const;

        private:
                string job, area;
};

#endif
```

Exercício 1

```
#include "professionalregister.h"

ProfessionalRegister::ProfessionalRegister (string n, string c,
int a, string j, string ar) : Register (n, c, a), job (j), area (ar) {}

string ProfessionalRegister::getJob () const { return job; }
string ProfessionalRegister::getArea () const { return area; }
```

Exercício 1

```
#include <iostream>
#include <string>
#include "register.h"

#ifndef PERSONALREG_H
#define PERSONALREG_H

using namespace std;

class PersonalRegister : public Register {
        public:
                PersonalRegister (string, string, int, string, string);
                virtual ~PersonalRegister () {};

                string getAddress () const;
                string getCell () const;

        private:
                string address, cellphone;
};

#endif
```

Exercício 1

```
#include "personalregister.h"

PersonalRegister::PersonalRegister (string n, string c,
int a, string ad, string cell) :
Register (n, c, a), address (ad), cellphone (cell) {}

string PersonalRegister::getAddress () const { return address; }
string PersonalRegister::getCell () const { return cellphone; }
```

Exercício 1

```
#include "professionalregister.h"
#include "personalregister.h"
#include <stdexcept>
#include <vector>

#ifndef AGENDA_H
#define AGENDA_H

class Agenda {
        public:
                Agenda (int);

                void insert (Register *);
                Register * get (int) const;
        private:
                vector <Register *> v;
                int idx;
};

#endif
```

Exercício 1

```
#include "agenda.h"

Agenda::Agenda (int number) {
        idx = 0;
        v.resize (number);
}

void Agenda::insert (Register *r) {
        // Inserção em vector é uma operação que pode disparar exceção
        try {
                // Verifica se é possível inserir mais um objeto
                v.at (idx) = r;
        }
        catch (out_of_range &oor) {
                cout << "Insertion Error! " << oor.what () << endl;
                return;
        }
        cout << "Successful insertion!" << "\nAgenda size is: " << v.size () << endl;
        idx++;
}
```

Exercício 1

```
Register * Agenda::get (int pos) const {
        // Inserção em vector é uma operação que pode disparar exceção
        try {
                // Verifica se o objeto existe
                return v.at (pos);
        }
        catch (out_of_range &oor) {
                cout << "Recover Error! " << oor.what () << endl;
                return NULL;
        }
}
```

Exercício 1

```
miguel@pegasus-linux:~/UFRJ/disciplinas/linguagens/laboratorios/2$ ./l
Successful insertion!
Agenda size is: 1
Insertion Error! vector::_M_range_check

Data:     Content:
Name:     Miguel
CPF:      123
Age:      30
Job:      Prof
Area:     Electronic

Recover Error! vector::_M_range_check

Successful insertion!
Agenda size is: 1

Data:     Content:
Name:     Miguel
CPF:      123
Age:      30
Address:  Rua A
Cell:     778899
```

Exercício 2

- Repita o Exercício 1, mas imprima os atributos de cada classe usando o operador de << sobrecarregado. O operador sobrecarregado deve chamar uma função **`print`** virtual.

Exercício 2

```
#include "agenda2.h"
#include <iomanip>

int main () {
        Agenda professionalAgenda (1);

        ProfessionalRegister *pr1 = new ProfessionalRegister ("Miguel", "123", 30, "Prof", "Electronic");
        professionalAgenda.insert (pr1);

        // Inserção de elemento fora do limite máximo
        // O objetivo é forçar uma exceção
        ProfessionalRegister *pr2 = new ProfessionalRegister ("Miguel", "123", 30, "Prof", "Electronic");
        professionalAgenda.insert (pr2);

        Register *pr3 = professionalAgenda.get (0);

        cout << pr3;

        // Objetivo é novamente forçar uma exceção
        Register *pr4 = professionalAgenda.get (1);
        cout << endl;

        // Desalocar da memória todos os objetos criados com o operador new
        delete pr1;
        delete pr2;
```

Exercício 2

```
        // Criação de uma agenda pessoal
        // Atenção! Criação de um novo objeto da classe Agenda
        // Agora, porém, para colocar objetos de outra classe derivada de Register
        // Vantagem do polimorfismo! Permite usar a mesma classe Agenda para armazenar
        // objetos de classes diferentes derivadas de Register
        Agenda personalAgenda (1);

        PersonalRegister *pr5 = new PersonalRegister ("Miguel", "123", 30, "Rua A", "778899");
        personalAgenda.insert (pr5);

        Register *pr6 = personalAgenda.get (0);

        cout << pr6;

        delete pr5;

        return 0;
}
```

Exercício 2

```
#include <string>
#include <iostream>
#include <iomanip>

#ifndef REGISTER_H
#define REGISTER_H

using namespace std;

class Register {
        friend ostream &operator << (ostream &, Register *);
        public:
                Register (string, string, int);
                // Para ser polimórfica, a classe base tem que ter
                // pelo menos um método virtual
                virtual ~Register () {}

                string getName () const;
                string getCPF () const;
                int getAge () const;

                virtual void print () const;

        private:
                string name, cpf;
                int age;
};

#endif
```

Exercício 2

```
#include "register.h"

Register::Register (string n, string c, int a) : name (n), cpf (c), age (a) {}

string Register::getName () const { return name; }

string Register::getCPF () const { return cpf; }

int Register::getAge () const { return age; }

void Register::print () const {
        cout << setw (10) << "Data:" << setw(20) << "Content:" << endl;
        cout << setw (10) << "Name:" << setw(20) << getName () << "\n"
               << setw (10) << "CPF: " << setw(20) << getCPF () << "\n"
               << setw (10) << "Age: " << setw(20) << getAge () << "\n";
}

ostream &operator << (ostream &output, Register *r) {
        r->print ();
        return output;
}
```

Exercício 2

```
#include <iostream>
#include <string>
#include "register.h"

#ifndef PROFESSIONALREG_H
#define PROFESSIONALREG_H

using namespace std;

class ProfessionalRegister: public Register {
        public:
                ProfessionalRegister (string, string, int, string, string);
                virtual ~ProfessionalRegister () {}

                string getJob () const;
                string getArea () const;

                virtual void print () const;

        private:
                string job, area;
};
```

Exercício 2

```
#include "professionalregister.h"

ProfessionalRegister::ProfessionalRegister (string n, string c,
int a, string j, string ar) : Register (n, c, a), job (j), area (ar) {}

string ProfessionalRegister::getJob () const { return job; }
string ProfessionalRegister::getArea () const { return area; }

void ProfessionalRegister::print () const {
        Register::print ();
        cout << setw (10) << "Job: " << setw(20) << getJob () << "\n"
             << setw (10) << "Area: " << setw(20) << getArea () << "\n" << endl;

}
```

Exercício 2

```
#include <iostream>
#include <string>
#include "register.h"

#ifndef PERSONALREG_H
#define PERSONALREG_H

using namespace std;

class PersonalRegister : public Register {
        public:
                PersonalRegister (string, string, int, string, string);
                virtual ~PersonalRegister () {};

                string getAddress () const;
                string getCell () const;

                virtual void print () const;

        private:
                string address, cellphone;
};

#endif
```

Exercício 2

```
#include "personalregister.h"

PersonalRegister::PersonalRegister (string n, string c,
int a, string ad, string cell) :
Register (n, c, a), address (ad), cellphone (cell) {}

string PersonalRegister::getAddress () const { return address; }
string PersonalRegister::getCell () const { return cellphone; }

void PersonalRegister::print () const {
        Register::print ();
        cout << setw (10) << "Address: " << setw(20) << getAddress () << "\n"
             << setw (10) << "Cell: " << setw(20) << getCell () << "\n" << endl;

}
```

Exercício 2

```
#include "professionalregister.h"
#include "personalregister.h"
#include <stdexcept>
#include <vector>

#ifndef AGENDA_H
#define AGENDA_H

class Agenda {
        public:
                Agenda (int);

                void insert (Register *);
                Register * get (int) const;
        private:
                vector <Register *> v;
                int idx;
};

#endif
```

Exercício 2

```
#include "agenda.h"

Agenda::Agenda (int number) {
        idx = 0;
        v.resize (number);
}

void Agenda::insert (Register *r) {
        // Inserção em vector é uma operação que pode disparar exceção
        try {
                // Verifica se é possível inserir mais um objeto
                v.at (idx) = r;
        }
        catch (out_of_range &oor) {
                cout << "Insertion Error! " << oor.what () << endl;
                return;
        }
        cout << "Successful insertion!" << "\nAgenda size is: " << v.size () << endl;
        idx++;
}
```

Exercício 2

```
Register * Agenda::get (int pos) const {
        // Inserção em vector é uma operação que pode disparar exceção
        try {
                // Verifica se o objeto existe
                return v.at (pos);
        }
        catch (out_of_range &oor) {
                cout << "Recover Error! " << oor.what () << endl;
                return NULL;
        }
}
```

Exercício 2

```
miguel@pegasus-linux:~/UFRJ/disciplinas/linguagens/laboratorios/2$ ./l
Successful insertion!
Agenda size is: 1
Insertion Error! vector::_M_range_check
     Data:            Content:
     Name:              Miguel
     CPF:                  123
     Age:                   30
     Job:                 Prof
    Area:           Electronic

Recover Error! vector::_M_range_check

Successful insertion!
Agenda size is: 1
     Data:            Content:
     Name:              Miguel
     CPF:                  123
     Age:                   30
 Address:                Rua A
    Cell:               778899
```

Exercício 3

- Repetir o Exercício 1 utilizando uma estrutura **`map`** para armazenar os registros. Utilize o nome dos registros como chave. A inserção deve disparar exceção caso o nome já esteja presente e a função do tipo **`get`** para recuperar o registro deve disparar uma exceção caso o registro não exista.

Exercício 3

```
#include <string>
#include <iostream>
#include <iomanip>

#ifndef REGISTER_H
#define REGISTER_H

using namespace std;

class Register {
        friend ostream &operator << (ostream &, Register *);
        public:
                Register (string, string, int);
                // Para ser polimórfica, a classe base tem que ter
                // pelo menos um método virtual
                virtual ~Register () {}

                string getName () const;
                string getCPF () const;
                int getAge () const;

                virtual void print () const;

        private:
                string name, cpf;
                int age;
};

#endif
```

Exercício 3

```
#include "register.h"

Register::Register (string n, string c, int a) : name (n), cpf (c), age (a) {}

string Register::getName () const { return name; }

string Register::getCPF () const { return cpf; }

int Register::getAge () const { return age; }

void Register::print () const {
        cout << setw (10) << "Data:" << setw(20) << "Content:" << endl;
        cout << setw (10) << "Name:" << setw(20) << getName () << "\n"
               << setw (10) << "CPF: " << setw(20) << getCPF () << "\n"
               << setw (10) << "Age: " << setw(20) << getAge () << "\n";
}

ostream &operator << (ostream &output, Register *r) {
        r->print ();
        return output;
}
```

Exercício 3

```
#include <iostream>
#include <string>
#include "register.h"

#ifndef PROFESSIONALREG_H
#define PROFESSIONALREG_H

using namespace std;

class ProfessionalRegister: public Register {
        public:
                ProfessionalRegister (string, string, int, string, string);
                virtual ~ProfessionalRegister () {}

                string getJob () const;
                string getArea () const;

                virtual void print () const;

        private:
                string job, area;
};
```

Exercício 3

```
#include "professionalregister.h"

ProfessionalRegister::ProfessionalRegister (string n, string c,
int a, string j, string ar) : Register (n, c, a), job (j), area (ar) {}

string ProfessionalRegister::getJob () const { return job; }
string ProfessionalRegister::getArea () const { return area; }

void ProfessionalRegister::print () const {
        Register::print ();
        cout << setw (10) << "Job: " << setw(20) << getJob () << "\n"
             << setw (10) << "Area: " << setw(20) << getArea () << "\n" << endl;

}
```

Exercício 3

```
#include <iostream>
#include <string>
#include "register.h"

#ifndef PERSONALREG_H
#define PERSONALREG_H

using namespace std;

class PersonalRegister : public Register {
        public:
                PersonalRegister (string, string, int, string, string);
                virtual ~PersonalRegister () {};

                string getAddress () const;
                string getCell () const;

                virtual void print () const;

        private:
                string address, cellphone;
};

#endif
```

Exercício 3

```
#include "personalregister.h"

PersonalRegister::PersonalRegister (string n, string c,
int a, string ad, string cell) :
Register (n, c, a), address (ad), cellphone (cell) {}

string PersonalRegister::getAddress () const { return address; }
string PersonalRegister::getCell () const { return cellphone; }

void PersonalRegister::print () const {
        Register::print ();
        cout << setw (10) << "Address: " << setw(20) << getAddress () << "\n"
             << setw (10) << "Cell: " << setw(20) << getCell () << "\n" << endl;

}
```

Exercício 3

```
#include "professionalregister3.h"
#include "personalregister3.h"
#include "insertionexception.h"
#include "getexception.h"
#include <stdexcept>
#include <string>
#include <map>
#include <iterator>

using namespace std;

#ifndef AGENDA_H
#define AGENDA_H

typedef map <string, Register *, less <string> > mymap;

class Agenda {
        public:
                void insert (Register *);
                Register * get (string) const;

                void showAll ();
        private:
                mymap m;
};

#endif
```

Exercício 3

```
#include "agenda3.h"

void Agenda::insert (Register *r) {
        // Inserção em vector é uma operação que pode disparar exceção
        pair <mymap::const_iterator, bool> p;
        p = m.insert (mymap::value_type (r->getName (), r));
        try {
                // Verifica se um elemento novo foi inserido
                if (!p.second)
                        throw InsertionException ();
        }
        catch (InsertionException &ie) {
                cout << "Insertion Fails! " << ie.what () << endl;
                return;
        }
        cout << "Successful insertion!" << "\nAgenda size is: " << m.size () << endl;
}
```

Exercício 3

```
Register * Agenda::get (string s) const {
        // Vou disparar exceção se o elemento não for encontrado
        mymap::const_iterator it = m.find (s);
        try {
                if (it == m.end ())
                        throw GetException ();
        }

        catch (GetException &ge) {
                cout << "Get Fails! " << ge.what ();
                return NULL;
        }
        return it->second;
}

void Agenda::showAll () {
        mymap::iterator it;

        cout << setw (25) << "**** Show All: ****\n" << endl;
        for (it = m.begin (); it != m.end (); it++)
                cout << (*it).second;
        cout << setw (25) << "**********************" << endl;
}
```

Exercício 3

```
#include <iostream>
#include <stdexcept>

class GetException : public runtime_error {
        public:
                GetException () : runtime_error ("Element does not exist.") {}
};
```

```
#include <iostream>
#include <stdexcept>

class InsertionException : public runtime_error {
        public:
                InsertionException () : runtime_error ("Element already exists.") {}
};
```

Exercício 3

```
#include "agenda3.h"
#include <iomanip>

int main () {
        Agenda professionalAgenda;

        ProfessionalRegister *pr1 = new ProfessionalRegister ("Miguel", "123", 30, "Prof", "Electronic");
        professionalAgenda.insert (pr1);

        ProfessionalRegister *pr2 = new ProfessionalRegister ("Joao", "321", 20, "Prof", "Computer Science");
        professionalAgenda.insert (pr2);

        // Inserção de elemento fora do limite máximo
        // O objetivo é forçar uma exceção
        ProfessionalRegister *pr3 = new ProfessionalRegister ("Miguel", "123", 30, "Prof", "Electronic");
        cout << endl;
        professionalAgenda.insert (pr3);

        // Mostra todos os elementos em ordem de nome
        cout << endl;
        professionalAgenda.showAll ();



        Register *pr4 = professionalAgenda.get ("Miguel");

        cout << endl;
        cout << pr4;
```

Exercício 3

```
        // Objetivo é novamente forçar uma exceção
        Register *pr5 = professionalAgenda.get ("Marcos");
        cout << endl;

        // Desalocar da memória todos os objetos criados com o operador new
        delete pr1;
        delete pr2;
        delete pr3;

        // Criação de uma agenda pessoal
        // Atenção! Criação de um novo objeto da classe Agenda
        // Agora, porém, para colocar objetos de outra classe derivada de Register
        // Vantagem do polimorfismo! Permite usar a mesma classe Agenda para armazenar
        // objetos de classes diferentes derivadas de Register
        Agenda personalAgenda;

        PersonalRegister *pr6 = new PersonalRegister ("Miguel", "123", 30, "Rua A", "778899");
        cout << endl;
        personalAgenda.insert (pr6);

        Register *pr7 = personalAgenda.get ("Miguel");

        cout << endl;
        cout << pr7;

        delete pr6;

        return 0;
}
```

Exercício 3

```
miguel@pegasus-linux:~/UFRJ/disciplinas/linguagens/laboratorios/2$ ./l
Successful insertion!
Agenda size is: 1
Successful insertion!
Agenda size is: 2

Insertion Fails! Element already exists.

     **** Show All: ****

     Data:            Content:
     Name:                Joao
     CPF:                  321
     Age:                   20
     Job:                 Prof
    Area:     Computer Science

     Data:            Content:
     Name:              Miguel
     CPF:                  123
     Age:                   30
     Job:                 Prof
    Area:           Electronic

      ******************
```

Exercício 3

```
     Data:            Content:
     Name:              Miguel
     CPF:                  123
     Age:                   30
     Job:                 Prof
    Area:           Electronic

Get Fails! Element does not exist.

Successful insertion!
Agenda size is: 1

     Data:            Content:
     Name:              Miguel
     CPF:                  123
     Age:                   30
 Address:                Rua A
    Cell:               778899
```