| | |
|---|---|
| **Início** | quarta, 1 de junho de 2022 às 08:49 |
| **Estado** | Prova submetida |
| **Data de submissão:** | segunda, 6 de junho de 2022 às 17:21 |
| **Tempo gasto** | 5 dias 8 horas |
| **Nota** | **100** do máximo 100 |

## Pergunta 1

Correta Pontuou 20 de 20

Consider the definition of class `Person` given in header `Person.h`:

```
class Person {
public:
  Person(int id, const string& name)
    : id_(id), name_(name) { }
  int id() const { return id_; }
  const string& name() const { return name_; }
  virtual string to_string() const {
    ostringstream out;
    out << id_ << '/' << name_;
    return out.str();
  }
private:
  int id_;
  string name_;
};
```

As illustrated by the constructor and accessor functions, a `Person` object represents the identifier and the name of a person. In addition, `to_string()` returns a string representation for a `Person` object, where person attributes in text form are separated by the `/` character.

Write two classes, `Student` and `ErasmusStudent`, with the following requirements.

`Student` must extend `Person`, and define an attribute for the course in which a student is enrolled. The class must have the following functions defined:

- `Student(int id, const string& name, const string& course)`: the constructor;
- `const string& course() const`: an accessor function for the course information; and
- `string to_string() const`: overrides `Person::to_string()`, and returns a string in which the course information is appended to the person information (see the test cases for illustrative examples).

`ErasmusStudent` must extend `Student`, and define an attribute for the country of origin for the student. The class must have the following functions defined:

- `ErasmusStudent(int id, const string& name, const string& course, const string& country)`: the constructor:
- `const string& country() const`: an accessor function for the country information; and
- `string to_string() const`: overrides `Student::to_string()`, and returns a string in which the country information is appended to the student information.

**Por exemplo:**

| Teste | Resultado |
|---|---|
| `const Person& p = Student(123, "Manuel Dias", "LEIC");`<br>`cout << p.id() << ' ' << p.name() << '\n';` | `123 Manuel Dias` |
| `const Student& s = ErasmusStudent(124, "John Zorn", "LXPTO", "United States");`<br>`cout << s.id() << ' ' << s.name() << ' ' << s.course() << '\n';` | `124 John Zorn LXPTO` |
| `Student s(123, "Manuel Dias", "LEIC");`<br>`cout << s.id() << ' ' << s.name() << ' '  << s.course() << '\n';` | `123 Manuel Dias LEIC` |
| `ErasmusStudent es(124, "John Zorn", "LXPTO", "United States");`<br>`cout << es.id() << ' ' << es.name() << ' ' << es.course() << ' ' << es.country() << '\n';` | `124 John Zorn LXPTO United States` |
| `Person p(125, "Marie Curie");`<br>`Student s(126, "Ada Lovelace", "LEIC");`<br>`ErasmusStudent es(127, "Grace Hopper", "LXPTO", "United States");`<br>`cout << p.to_string() << ' ' << s.to_string() << ' ' << es.to_string() << '\n';` | `125/Marie Curie 126/Ada Lovelace/LEIC 127/Grace Hopper/LXPTO/United States` |

**Resposta:** (regime de penalização: 0, 0, 0, 0, 10, 20, 30, ... %)

Limpar resposta

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

using namespace std;

class Student: public Person{
    public:
        Student(int id, const string& name, const string& course): Person(id, name){
            course_ = course;
        }
        const string& course() const {return course_;}
        string to_string() const override{
            return Person::to_string() + "/" + course_;
        }
    private:
        string course_;
};

class ErasmusStudent: public Student{
    public:
```

| | Teste | Esperado | Recebido | |
|---|---|---|---|---|
| ✔ | `const Person& p = Student(123, "Manuel Dias", "LEIC");`<br>`cout << p.id() << ' ' << p.name() << '\n';` | `123 Manuel Dias` | `123 Manuel Dias` | ✔ |
| ✔ | `const Student& s = ErasmusStudent(124, "John Zorn", "LXPTO", "United States");`<br>`cout << s.id() << ' ' << s.name() << ' ' << s.course() << '\n';` | `124 John Zorn LXPTO` | `124 John Zorn LXPTO` | ✔ |
| ✔ | `Student s(123, "Manuel Dias", "LEIC");`<br>`cout << s.id() << ' ' << s.name() << ' '  << s.course() << '\n';` | `123 Manuel Dias LEIC` | `123 Manuel Dias LEIC` | ✔ |
| ✔ | `ErasmusStudent es(124, "John Zorn", "LXPTO", "United States");`<br>`cout << es.id() << ' ' << es.name() << ' ' << es.course() << ' ' << es.country() << '\n';` | `124 John Zorn LXPTO United States` | `124 John Zorn LXPTO United States` | ✔ |
| ✔ | `Person p(125, "Marie Curie");`<br>`Student s(126, "Ada Lovelace", "LEIC");`<br>`ErasmusStudent es(127, "Grace Hopper", "LXPTO", "United States");`<br>`cout << p.to_string() << ' ' << s.to_string() << ' ' << es.to_string() << '\n';` | `125/Marie Curie 126/Ada Lovelace/LEIC 127/Grace Hopper/LXPTO/United States` | `125/Marie Curie 126/Ada Lovelace/LEIC 127/Grace Hopper/LXPTO/United States` | ✔ |

Passou em todos os testes! ✔

## Solução do autor da pergunta (C):

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

using namespace std;

class Student : public Person {
public:
  Student(int id, const string& name, const string& course)
    : Person(id, name), course_(course) { }
  const string& course() const { return course_; }
  string to_string() const override {
     return Person::to_string() + "/" + course_;
  }
```

```
14 private:
15   string course_;
16 };
17
18 class ErasmusStudent : public Student {
19 public:
20   ErasmusStudent(int id, const string& name, const string& course, const string& country)
21     : Student(id, name, course), country_(country) { }
22   const string& country() const { return country_; }
```

Correta

Nota desta submissão: 20/20

## Pergunta 2

Correta Pontuou 20 de 20

Consider the definition of an abstract class for bidimensional geometric shapes given in header `Shape.h`:

```
struct point {
  double x, y;
};
class Shape {
public:
  Shape(const point& center) : center_(center) { }
  const point& get_center() const { return center_; }
  virtual double area() const = 0;
  virtual double perimeter() const = 0;
  virtual bool contains(const point& p) const = 0;
private:
  point center_;
};
```

A shape has a geometric center, returned by `get_center()`. Abstract functions `area()` and `perimeter()` should return a shape's area and perimeter, respectively. Finally, abstract function `contains()` should be used to determine if a shape contains a given point.

To represent circles and rectangles, write the corresponding definition of classes `Circle` and `Rectangle`, that should both be subclasses of `Shape`. `Circle` should have a constructor that takes two arguments: the `center` of the circle and its `radius` (a `double` value). `Rectangle` has a constructor that takes 3 arguments: the geometric `center` of the rectangle, the rectangle's `width` and the rectangle's `height` (`double` values).

Note: use the `M_PI` constant defined by header `<cmath>` for the value of Pi.

**Por exemplo:**

| Teste | Resultado |
|---|---|
| `Circle c({1, 2}, 1);`<br>`const point& p = c.get_center();`<br>`cout << fixed << setprecision(2)`<br>`  << '(' << p.x << ',' << p.y << ')' << ' '`<br>`  << c.area() << ' ' << c.perimeter() << '\n';` | `(1.00,2.00) 3.14 6.28` |
| `Rectangle r({3, 4}, 1, 2);`<br>`const point& p = r.get_center();`<br>`cout << fixed << setprecision(2)`<br>`  << '(' << p.x << ',' << p.y << ')' << ' '`<br>`  << r.area() << ' ' << r.perimeter() << '\n';` | `(3.00,4.00) 2.00 6.00` |
| `const Shape& s1 = Circle({1, 2}, 3);`<br>`const Shape& s2 = Rectangle({4, 5}, 6, 7);`<br>`cout << fixed << setprecision(2) << boolalpha`<br>`  << s1.area() << ' ' << s1.perimeter() << ' '`<br>`  << s1.contains({1, 2}) << ' ' << s1.contains({ 4, 5 }) << ' '`<br>`  << s2.area() << ' ' << s2.perimeter() <<  ' '`<br>`  << s2.contains({1, 2}) << ' ' << s2.contains({ 4, 5 }) << '\n';` | `28.27 18.85 true false 42.00 26.00 true true` |
| `Circle c({1, 2}, 3);`<br>`point a [] {`<br>`  { -2.1,  2.0 }, { -1.9,  2.0 }, { 4.1, 2.0}, { 3.9,  2.0},`<br>`  {  1.0,  4.9 }, {  1.0, -0.9 }, { 1.0, 5.1}, { 1.0, -1.1},`<br>`  {  1.2, -0.3},  {  5.2,  5.1 }, { 2.1, -0.5}, { 3.2,  5.5}`<br>`};`<br>`for (point& p : a)`<br>`  if (c.contains(p))`<br>`  cout << '(' << p.x << ',' << p.y << ')';`<br>`cout << '\n';` | `(-1.90,2.00)(3.90,2.00)(1.00,4.90)(1.00,-0.90)(1.20,-0.30)(2.10,-0.50)` |

| Teste | Resultado |
|---|---|
| `Rectangle r({1, 2}, 6, 8);`<br>`point a [] {`<br>`  { -2.1,  2.0 }, { -1.9,  2.0 }, { 4.1,`<br>`2.0}, { 3.9,  2.0},`<br>`  {  1.0,  4.9 }, {  1.0, -0.9 }, { 1.0,`<br>`5.1}, { 1.0, -1.1},`<br>`  {  1.2, -0.3},  {  5.2,  5.1 }, { 2.1,`<br>`-0.5}, { 3.2,  5.5}`<br>`};`<br>`for (point& p : a)`<br>`  if (r.contains(p))`<br>`    cout << '(' << p.x << ',' << p.y << ')';`<br>`cout << '\n';` | `(-1.90,2.00)(3.90,2.00)(1.00,4.90)(1.00,-0.90)(1.00,5.10)`<br>`(1.00,-1.10)(1.20,-0.30)(2.10,-0.50)(3.20,5.50)` |

**Resposta:** (regime de penalização: 0, 0, 0, 0, 10, 20, 30, ... %)

Limpar resposta

```
#include <iostream>
#include <iomanip>
#include <cmath>
#include "Shape.h"


using namespace std;

double distance (const point p1, const point p2){
    return ( sqrt (pow(p1.x - p2.x,2) + pow(p1.y - p2.y,2)));
}

class Circle: public Shape{
    public:

        Circle(point c, double radius): Shape(c)
        {
            radius_ = radius;
        }
        double area() const override {return M_PI * pow(radius_,2);}
        double perimeter() const override {return M_PI * 2*radius_;}
        bool contains(const point& p) const override {
```

| | Teste | Esperado | Recebido | |
|---|---|---|---|---|
| ✔ | `Circle c({1, 2}, 1);`<br>`const point& p =`<br>`c.get_center();`<br>`cout << fixed <<`<br>`setprecision(2)`<br>`  << '(' << p.x <<`<br>`',' << p.y << ')' << ' '`<br>`  << c.area() << ' '`<br>`<< c.perimeter() <<`<br>`'\n';` | `(1.00,2.00) 3.14 6.28` | `(1.00,2.00) 3.14 6.28` | ✔ |
| ✔ | `Rectangle r({3, 4}, 1,`<br>`2);`<br>`const point& p =`<br>`r.get_center();`<br>`cout << fixed <<`<br>`setprecision(2)`<br>`  << '(' << p.x <<`<br>`',' << p.y << ')' << ' '`<br>`  << r.area() << ' '`<br>`<< r.perimeter() <<`<br>`'\n';` | `(3.00,4.00) 2.00 6.00` | `(3.00,4.00) 2.00 6.00` | ✔ |

| | Teste | Esperado | Recebido | |
|---|---|---|---|---|
| ✔ | `const Shape& s1 = Circle({1, 2}, 3);`<br>`const Shape& s2 = Rectangle({4, 5}, 6, 7);`<br>`cout << fixed << setprecision(2) << boolalpha`<br>`  << s1.area() << ' ' << s1.perimeter() << ' '`<br>`  << s1.contains({1, 2}) << ' ' << s1.contains({ 4, 5 }) << ' '`<br>`  << s2.area() << ' ' << s2.perimeter() << ' '`<br>`  << s2.contains({1, 2}) << ' ' << s2.contains({ 4, 5 }) << '\n';` | `28.27 18.85 true false 42.00 26.00 true true` | `28.27 18.85 true false 42.00 26.00 true true` | ✔ |
| ✔ | `Circle c({1, 2}, 3);`<br>`point a [] {`<br>`  { -2.1,  2.0 }, { -1.9,  2.0 }, { 4.1, 2.0}, { 3.9,  2.0},`<br>`  {  1.0,  4.9 }, { 1.0, -0.9 }, { 1.0, 5.1}, { 1.0, -1.1},`<br>`  {  1.2, -0.3},  { 5.2,  5.1 }, { 2.1, -0.5}, { 3.2,  5.5}`<br>`};`<br>`for (point& p : a)`<br>`  if (c.contains(p))`<br>`  cout << '(' << p.x << ',' << p.y << ')';`<br>`cout << '\n';` | `(-1.90,2.00)(3.90,2.00)(1.00,4.90)(1.00,-0.90)(1.20,-0.30)(2.10,-0.50)` | `(-1.90,2.00)(3.90,2.00)(1.00,4.90)(1.00,-0.90)(1.20,-0.30)(2.10,-0.50)` | ✔ |
| ✔ | `Rectangle r({1, 2}, 6, 8);`<br>`point a [] {`<br>`  { -2.1,  2.0 }, { -1.9,  2.0 }, { 4.1, 2.0}, { 3.9,  2.0},`<br>`  {  1.0,  4.9 }, { 1.0, -0.9 }, { 1.0, 5.1}, { 1.0, -1.1},`<br>`  {  1.2, -0.3},  { 5.2,  5.1 }, { 2.1, -0.5}, { 3.2,  5.5}`<br>`};`<br>`for (point& p : a)`<br>`  if (r.contains(p))`<br>`  cout << '(' << p.x << ',' << p.y << ')';`<br>`cout << '\n';` | `(-1.90,2.00)(3.90,2.00)(1.00,4.90)(1.00,-0.90)(1.00,5.10)(1.00,-1.10)(1.20,-0.30)(2.10,-0.50)(3.20,5.50)` | `(-1.90,2.00)(3.90,2.00)(1.00,4.90)(1.00,-0.90)(1.00,5.10)(1.00,-1.10)(1.20,-0.30)(2.10,-0.50)(3.20,5.50)` | ✔ |

Passou em todos os testes! ✔

## Solução do autor da pergunta (C):

```
#include <cmath>
#include <iostream>
#include <iomanip>
#include "Shape.h"

using namespace std;

class Circle : public Shape {
public:
  Circle(const point& c, double r) : Shape(c),  radius_(r) { }
  double radius() const { return radius_; }
```

```
11    double radius() const { return radius_; }
12    double area() const override {
13      return M_PI * radius_ * radius_;
14    }
15    double perimeter() const override {
16      return 2 * M_PI * radius_;
17    }
18    bool contains(const point &p) const override {
19      point c = get_center();
20      double dx = p.x - c.x;
21      double dy = p.y - c.y;
22      return dx * dx + dy * dy <= radius_ * radius_;
```

Correta

Nota desta submissão: 20/20

## Pergunta 3

Correta Pontuou 20 de 20

Consider the following interface of a class named `Point` given in header `Point.h`:

```
class Point {
public:
  Point(); // builds (0,0)
  Point(int x, int y); // builds (x,y)
  Point(const Point& p); // copy constructor
  int get_x() const; // get x coordinate
  int get_y() const; // get y coordinate
  Point& operator=(const Point& p); // assignment operator
  Point operator+(const Point& p) const; // sum
  Point& operator+=(const Point& p); // composed assignment and sum
  Point operator*(int v) const; // "right" multiplication by scalar
private:
  int x_, y_;
};

Point operator*(int x, const Point& p); // "left" multiplication by scalar
std::ostream& operator<<(std::ostream& os, const Point& p);
```

Provide an implementation of all functions, in particular those related to operator overloading. For an output stream `os`, points `a` and `b`, and an `int` value `v`:

- `os << a` should output to `os` the coordinates x and y of a with the format `(x,y)` and (as usual) return `os` as a result (for chained calls using the `<<` operator).
- `a = b` should assign the coordinates of `a` to `b` and (as usual) the implementation of `operator=` should return `*this` (for chained assignments);
- `a += b` should assign to `a` the sum of its coordinates with `b` and (as usual) the implementation of `operator+=` should return `*this` (for chained assignments);
- `a + b` should return a point with the coordinates corresponding to the sum of `a` and `b`;
- `a * v` and `v * a` should return a point with the coordinates of `a` both multiplied by `v`.

**Por exemplo:**

| Teste | Resultado |
|---|---|
| `Point a, b(1,2), c(b), d(3,4);`<br>`b = a;`<br>`cout << a << ' ' << b << ' ' << c << ' ' << d << '\n';` | `(0,0) (0,0) (1,2) (3,4)` |
| `Point a, b(1,2), c(b), d(3,4);`<br>`c = b = a;`<br>`cout << a << ' ' << b << ' ' << c << ' ' << d << '\n';` | `(0,0) (0,0) (0,0) (3,4)` |
| `Point a(1,2), b(3,4), c = a + b, d(5,6);`<br>`b += d;`<br>`cout << a << ' ' << b << ' ' << c << ' ' << d << '\n';` | `(1,2) (8,10) (4,6) (5,6)` |
| `Point a(1,2), b(3,4), c = a * 2, d(5,6);`<br>`b = 2 * d;`<br>`cout << a << ' ' << b << ' ' << c << ' ' << d << '\n';` | `(1,2) (10,12) (2,4) (5,6)` |
| `Point a(1,1), b(0,1), c(1,0), d(1,1);`<br>`d += c += b += a += {1,2};`<br>`d = 2 * d * 2;`<br>`cout << a << ' ' << b << ' ' << c << ' ' << d << '\n';` | `(2,3) (2,4) (3,4) (16,20)` |

**Resposta:** (regime de penalização: 0, 0, 0, 0, 10, 20, 30, ... %)

Limpar resposta

```
#include "Point.h"
#include <sstream>

using namespace std;

Point::Point(){
    x_ = 0;
    y_ = 0;
}

```

```
Point::Point(int x, int y){
    x_ = x;
    y_ = y;
}

Point::Point(const Point& p){
    x_ = p.x_;
    y_ = p.y_;
}
int Point::get_x() const{
    return x_;
```

| | Teste | Esperado | Recebido | |
|---|---|---|---|---|
| ✔ | `Point a, b(1,2), c(b), d(3,4);`<br>`b = a;`<br>`cout << a << ' ' << b << ' ' << c << ' ' << d << '\n';` | `(0,0) (0,0) (1,2) (3,4)` | `(0,0) (0,0) (1,2) (3,4)` | ✔ |
| ✔ | `Point a, b(1,2), c(b), d(3,4);`<br>`c = b = a;`<br>`cout << a << ' ' << b << ' ' << c << ' ' << d << '\n';` | `(0,0) (0,0) (0,0) (3,4)` | `(0,0) (0,0) (0,0) (3,4)` | ✔ |
| ✔ | `Point a(1,2), b(3,4), c = a + b, d(5,6);`<br>`b += d;`<br>`cout << a << ' ' << b << ' ' << c << ' ' << d << '\n';` | `(1,2) (8,10) (4,6) (5,6)` | `(1,2) (8,10) (4,6) (5,6)` | ✔ |
| ✔ | `Point a(1,2), b(3,4), c = a * 2, d(5,6);`<br>`b = 2 * d;`<br>`cout << a << ' ' << b << ' ' << c << ' ' << d << '\n';` | `(1,2) (10,12) (2,4) (5,6)` | `(1,2) (10,12) (2,4) (5,6)` | ✔ |
| ✔ | `Point a(1,1), b(0,1), c(1,0), d(1,1);`<br>`d += c += b += a += {1,2};`<br>`d = 2 * d * 2;`<br>`cout << a << ' ' << b << ' ' << c << ' ' << d << '\n';` | `(2,3) (2,4) (3,4) (16,20)` | `(2,3) (2,4) (3,4) (16,20)` | ✔ |

Passou em todos os testes! ✔

## Solução do autor da pergunta (C):

```
#include "Point.h"

using namespace std;

Point::Point() : x_(0), y_(0) { }

Point::Point(int x, int y) : x_(x), y_(y) { }

Point::Point(const Point& p) : x_(p.x_), y_(p.y_) { }

int Point::get_x() const { return x_; }

int Point::get_y() const { return y_; }

Point& Point::operator=(const Point& p) {
  x_ = p.x_;
  y_ = p.y_;
  return *this;
}

Point& Point::operator+=(const Point& p) {
  x_ += p.x_;
```

Correta

Nota desta submissão: 20/20

## Pergunta 4

Correta Pontuou 20 de 20

Consider that you want to develop an application for drawing figures, for simplicity only rectangles and circles.

The application must use three classes, `Figure`, `Rectangle` and `Circle`, such that:

- Classes `Rectangle` and `Circle` must be derived from `Figure`
- `Figure` is an abstract class whose definition is given in `Figure.h`
- The data members of class `Figure` are the coordinates of the center of the figure `x_center_`, `y_center_`

Define classes `Rectangle` and `Circle`, taking into account that:

- `Rectangle` has two additional data members, the width and height of the rectangle `width_`, `height_`
- `Circle` has one additional data member, the radius of the circle `radius_`

Member functions `draw()` do not effectively draws the figures on the screen, they are just "stubs" that write a message indicating the type (`R` or `C`), the attributes of the rectangle or circle (coordinates of the center) and, for the rectangle, the length of the sides, for the circle, its radius. For example:

- `R(10,20)(200,100)` — rectangle centered at (x=10,y=20) with (width=200, height=100)
- `C(-10,0)(50)` — circle rectangle centered at (x=-10,y=0) with (radius=50)

**Por exemplo:**

| Teste | Resultado |
|---|---|
| `Rectangle r(10, 10, 200, 100);`<br>`r.draw(); cout << endl;` | `R(10,10)(200,100)` |
| `Circle c (20, 20, 500);`<br>`c.draw(); cout << endl;` | `C(20,20)(500)` |
| `Rectangle * r = new Rectangle(10, 10, 200, 100);`<br>`r->draw(); cout << endl; delete r;` | `R(10,10)(200,100)` |
| `vector<Figure*> figures = {`<br>`  new Rectangle(10,10,200,100),`<br>`  new Circle(20,20,500),`<br>`  new Rectangle(-10,-20,150,250),`<br>`  new Circle(0,0,100) };`<br>`for (const auto& f : figures) f->draw();`<br>`cout << endl; for (const auto& f : figures) delete f;` | `R(10,10)(200,100)C(20,20)(500)R(-10,-20)(150,250)C(0,0)(100)` |
| `vector<Figure*> figures = {`<br>`  new Rectangle(-10,-10,5,15),`<br>`  new Rectangle(0,0,20,10),`<br>`  new Circle(5,0,25) };`<br>`for (const auto& f : figures) f->draw();`<br>`cout << endl; for (const auto& f : figures) delete f;` | `R(-10,-10)(5,15)R(0,0)(20,10)C(5,0)(25)` |

**Resposta:** (regime de penalização: 0, 0, 0, 0, 10, 20, 30, ... %)

Limpar resposta

```
#include <iostream>
#include <vector>
#include "Figure.h"

using namespace std;

class Rectangle: public Figure{
    public:
        Rectangle(double xc, double yc, double w, double h) : Figure(xc, yc){w_ = w; h_ = h;}
        void draw() const override {
            cout << "R(" << x_center_ << "," << y_center_ << ")(" << w_ << "," << h_ << ")";
        }
    private:
        double w_, h_;
};

class Circle: public Figure{
    public:
        Circle(double xc, double yc, double r) : Figure(xc, yc){r_ = r;}
        void draw() const override {
            cout << "C(" << x_center_ << "," << y_center_ << ")(" << r_ << ")";
        }
```

| | Teste | Esperado | Recebido | |
|---|---|---|---|---|
| ✔ | `Rectangle r(10, 10, 200, 100); r.draw(); cout << endl;` | `R(10,10)(200,100)` | `R(10,10)(200,100)` | ✔ |
| ✔ | `Circle c (20, 20, 500); c.draw(); cout << endl;` | `C(20,20)(500)` | `C(20,20)(500)` | ✔ |
| ✔ | `Rectangle * r = new Rectangle(10, 10, 200, 100); r->draw(); cout << endl; delete r;` | `R(10,10)(200,100)` | `R(10,10)(200,100)` | ✔ |
| ✔ | `vector<Figure*> figures = { new Rectangle(10,10,200,100), new Circle(20,20,500), new Rectangle(-10,-20,150,250), new Circle(0,0,100) }; for (const auto& f : figures) f->draw(); cout << endl; for (const auto& f : figures) delete f;` | `R(10,10)(200,100)C(20,20)(500)R(-10,-20)(150,250)C(0,0)(100)` | `R(10,10)(200,100)C(20,20)(500)R(-10,-20)(150,250)C(0,0)(100)` | ✔ |
| ✔ | `vector<Figure*> figures = { new Rectangle(-10,-10,5,15), new Rectangle(0,0,20,10), new Circle(5,0,25) }; for (const auto& f : figures) f->draw(); cout << endl; for (const auto& f : figures) delete f;` | `R(-10,-10)(5,15)R(0,0)(20,10)C(5,0)(25)` | `R(-10,-10)(5,15)R(0,0)(20,10)C(5,0)(25)` | ✔ |

Passou em todos os testes! ✔

## Solução do autor da pergunta (C):

```
#include <iostream>
#include <vector>
#include "Figure.h"

using namespace std;

class Rectangle : public Figure {
public:
  Rectangle(double x_center=0, double y_center=0, double width=0, double height=0)
    : Figure(x_center, y_center), width_(width), height_(height) { }
  void draw() const {
    std::cout << "R(" << x_center_ << ',' << y_center_ << ")(" << width_ << ',' << height_ << ')
  }
private:
  double width_;
  double height_;
};

class Circle : public Figure {
public:
  Circle(double x_center=0, double y_center=0, double radius=0)
    : Figure(x_center, y_center), radius_(radius) { }
```

Correta

Nota desta submissão: 20/20

## Pergunta 5

Correta Pontuou 20 de 20

Consider the classes partially defined in the files `Employee.h`, `SalariedEmployee.h`, `HourlyEmployee.h`, `Employee.cpp`, `SalariedEmployee.cpp`, and `HourlyEmployee.cpp`, given in the ex5.zip:

- Classes `SalariedEmployee` and `HourlyEmployee` are derived from class `Employee`
- A `SalariedEmployee` receives a fixed monthly amount for payment
- An `HourlyEmployee` receives an amount that is given by the number of worked hours times a wage rate (payment per hour)
- The program that calculates the payment that each employee receives at the end of the month, uses a `vector<Employee*>` to store the data about all the employees, both salaried and hourly

Complete the implementation of the classes `SalariedEmployee` (with code for `calculate_net_pay()`) and `HourlyEmployee` and write the code of the following global functions:

- `void read_hours_worked(vector<Employee*> &employees)` to read hours worked for all `HourlyEmployees`; this function must scan the employees vector and read *from the standard input* the hours, as a `double`, worked by each of the `HourlyEmployees` stored in the vector (but only for these employees)
- `void calculate_pay(vector<Employee*>& employees)`, to calculate net pay for all employees
- `void print_checks(const vector<Employee*>& employees)`, to print all checks for all employees

**Por exemplo:**

| Teste | Entrada | Resultado |
|---|---|---|
| `vector<Employee*> employees = { new SalariedEmployee("John", 2000), new HourlyEmployee("Mary", 10) }; read_hours_worked(employees); calculate_pay(employees); print_checks(employees); for (const auto& e : employees) delete e;` | `100` | `\| John:SE(2000.00)=2000.00 \| Mary:HE(100.00,10.00)=1000.00 \|` |
| `vector<Employee*> employees = { new HourlyEmployee("Peter", 10.31), new SalariedEmployee("Ann", 1957.4) }; read_hours_worked(employees); calculate_pay(employees); print_checks(employees); for (const auto& e : employees) delete e;` | `175.5` | `\| Peter:HE(175.50,10.31)=1809.41 \| Ann:SE(1957.40)=1957.40 \|` |
| `vector<Employee*> employees = { new HourlyEmployee("Philip", 10.75), new HourlyEmployee("Elisabeth", 9.5), new SalariedEmployee("Charles", 5000) }; read_hours_worked(employees); calculate_pay(employees); print_checks(employees); for (const auto& e : employees) delete e;` | `200.25 199.75` | `\| Philip:HE(200.25,10.75)=2152.69 \| Elisabeth:HE(199.75,9.50)=1897.62 \| Charles:SE(5000.00)=5000.00 \|` |

| Teste | Entrada | Resultado |
|---|---|---|
| vector<Employee*> employees = {<br>  new SalariedEmployee("John", 2123.5),<br>  new SalariedEmployee("Peter", 1999.9) };<br>read_hours_worked(employees);<br>calculate_pay(employees);<br>print_checks(employees);<br>for (const auto& e : employees) delete e; | | \| John:SE(2123.50)=2123.50 \| Peter:SE(1999.90)=1999.90 \| |

**Resposta:** (regime de penalização: 0, 0, 0, 0, 10, 20, 30, ... %)

Limpar resposta

```
#include <iostream>
#include <vector>
#include "Employee.h"
#include "HourlyEmployee.h"
#include "SalariedEmployee.h"

using namespace std;

void HourlyEmployee::calculate_net_pay(){
  set_net_pay(wage_rate_*hours_);
}

void SalariedEmployee::calculate_net_pay(){
  set_net_pay(salary_);
}

void read_hours_worked(vector<Employee*> &employees){
    for(Employee* worker : employees){
        HourlyEmployee* w = dynamic_cast<HourlyEmployee*>(worker);
        if(w != nullptr){
            double h;
            cin >> h;
```

| | Teste | Entrada | Esperado | Recebido | |
|---|---|---|---|---|---|
| ✔ | vector<Employee*> employees = {<br>  new SalariedEmployee("John", 2000),<br>  new HourlyEmployee("Mary", 10) };<br>read_hours_worked(employees);<br>calculate_pay(employees);<br>print_checks(employees);<br>for (const auto& e : employees) delete e; | 100 | \| John:SE(2000.00)=2000.00 \| Mary:HE(100.00,10.00)=1000.00 \| | \| John:SE(2000.00)=2000.00 \| Mary:HE(100.00,10.00)=1000.00 \| | ✔ |
| ✔ | vector<Employee*> employees = {<br>  new HourlyEmployee("Peter", 10.31),<br>  new SalariedEmployee("Ann", 1957.4) };<br>read_hours_worked(employees);<br>calculate_pay(employees);<br>print_checks(employees);<br>for (const auto& e : employees) delete e; | 175.5 | \| Peter:HE(175.50,10.31)=1809.41 \| Ann:SE(1957.40)=1957.40 \| | \| Peter:HE(175.50,10.31)=1809.41 \| Ann:SE(1957.40)=1957.40 \| | ✔ |

| | Teste | Entrada | Esperado | Recebido | |
|---|---|---|---|---|---|
| ✔ | `vector<Employee*> employees = { new HourlyEmployee("Philip", 10.75), new HourlyEmployee("Elisabeth", 9.5), new SalariedEmployee("Charles", 5000) }; read_hours_worked(employees); calculate_pay(employees); print_checks(employees); for (const auto& e : employees) delete e;` | 200.25<br>199.75 | `\| Philip:HE(200.25,10.75)=2152.69 \| Elisabeth:HE(199.75,9.50)=1897.62 \| Charles:SE(5000.00)=5000.00 \|` | `\| Philip:HE(200.25,10.75)=2152.69 \| Elisabeth:HE(199.75,9.50)=1897.62 \| Charles:SE(5000.00)=5000.00 \|` | |
| ✔ | `vector<Employee*> employees = { new SalariedEmployee("John", 2123.5), new SalariedEmployee("Peter", 1999.9) }; read_hours_worked(employees); calculate_pay(employees); print_checks(employees); for (const auto& e : employees) delete e;` | | `\| John:SE(2123.50)=2123.50 \| Peter:SE(1999.90)=1999.90 \|` | `\| John:SE(2123.50)=2123.50 \| Peter:SE(1999.90)=1999.90 \|` | |

Passou em todos os testes! ✔

## Solução do autor da pergunta (C):

```
#include <iostream>
#include <vector>
#include "Employee.h"
#include "HourlyEmployee.h"
#include "SalariedEmployee.h"

using namespace std;

//! calculate net pay of Hourly
void HourlyEmployee::calculate_net_pay() {
  set_net_pay(wage_rate_ * hours_);
}

//! calculate net pay of Salaried
void SalariedEmployee::calculate_net_pay() {
  set_net_pay(salary_);
}

//! read hours worked for all HourlyEmployees
void read_hours_worked(vector<Employee*> &employees) {
  for (auto& e : employees)   {
    HourlyEmployee* he = dynamic_cast<HourlyEmployee*> (e);
```

Correta

Nota desta submissão: 20/20

◄ T10 24/05

Ir para...

T11 31/05 ►