PCLP2

Laboratorul 10

1. **Funcții virtuale și polimorfism.** Polimorfismul este implementat prin funcțiile virtuale. Atunci când programul cere folosirea unei funcții printr-un pointer sau o referință la o clasă de bază, C++ alege suprascrierea corectă din clasa derivată corespunzatoare obiectului care o apelează. Programul de mai jos implementează clasa de bază Shape și clasa derivată Point. Ambele clase conțin câte o versiune a funcției `printShapeName()` care are comportament polimorfic si câte o versiune a funcției `init()` care nu are comportament polimorfic. Rulați programul de mai jos și studiați rezultatul.

shape.cpp

```
#include <iostream>
using namespace std;
class Shape{
  public:
    virtual void printShapeName() const
    {
      cout << "Shape::printShapeName()" << endl;
    }
    void init() const
    {
      cout << "Shape::init()" << endl;
    }
};

class Point : public Shape{
  public:
    void printShapeName() const
    {
      cout << "Point::printShapeName()" << endl;
    }
    void init() const
    {
      cout << "Point::init()" << endl;
    }
};
int main()
{
  cout << "Functii apelate prin pointer la Shape:" << endl;
  Shape* shapePtr = new Shape();
  shapePtr->printShapeName();
  shapePtr->init();

  cout << "\nFunctii apelate prin pointer la Shape "
       << "initializat prin pointer la Point:" << endl;
  Point* pointPtr = new Point();
  shapePtr = pointPtr;
  cout << "Comportament polimorfic: ";
     shapePtr->printShapeName();
  shapePtr->init();
```

```
    cout << "\nFunctii apelate prin obiect de tip Shape:"
         << endl;
    Shape shape;
    shape.printShapeName();
    shape.init();

    cout << "\nFunctii apelate prin obiect de tip Point:"
         << endl;
    Point point;
    point.printShapeName();
    point.init();

    cout << "\nFunctie non-virtuala apelata prin pointer la "
         << "Shape:" << endl;
    shapePtr = &point;
    cout << "Comportament non-polimorfic: ";
    shapePtr->init();

    return 0;
}
```

2. Scrieți un program care să implementeze o nouă versiune a claselor `Point`, `Circle` și `Cylinder`. Datele membre ale acestor clase sunt aceleași ca cele din capitolul 6.4 al cursului. Pentru aceste clase implementați în clasa `Point` o funcție virtuală pură numită `area()` și implementări ale sale în clasele `Circle` și `Cylinder`. Demonstrați funcționalitatea polimorfică declarând în `main` câte un pointer la `Circle` și `Cylinder` și apelând apoi pentru fiecare dintre aceste obiecte funcția `area()`, similar cu programul de la punctul 1.