```cpp
// W12_1.cpp : Defines the entry point for the console application.
//             Funkcje wirtualne

#include "stdafx.h"
#include <iostream>
using namespace std;

class Figura;

class Figura
{
protected:
    double s;  //area
public:
    Figura() { s = 0.0; }
    double get_s() { return s; }
    virtual void area() = 0;  //to jest funkcja abstrakcyjna - w klasie bazowej
      brakuje jej realizacji
    void disp() { cout << " s = " << s << endl; }
};

class Circle: public Figura
{
    double r;
    const double PI;
public:
    Circle(double rr) : r(rr), PI(3.14159236) {}
    Circle() : r(0), PI(3.14159236) {}
    void area();
    void disp() { cout << "circle: s = " << s << endl; }
};

void Circle::area()
{
    s = PI*r*r;
}

class Trojkat:public Figura
{
    double a, h;
public:
    Trojkat(double aa, double hh) : a(aa), h(hh) {}
    Trojkat() { a = h = 0.0; }
    void area();
    void disp() { cout << "trojkat: s = " << s << endl; }
};

void Trojkat::area()
{
    s = 0.5*a*h;
}


int _tmain(int argc, _TCHAR* argv[])
{
    //--------------test 1-------------------------//
```

```
    //obiekt klas pochodnych
    Circle c1(10);
    Trojkat t1(2, 4);

    //wskaznik do typu klasy bazowej
    Figura *ptr_fig;

    //ustawiamy na obiekt klasy Circle
    ptr_fig = &c1;

    //liczymy pole Circle
    ptr_fig->area();

    //ustawiamy na obiekt klasy Trojkat
    ptr_fig = &t1;

    //liczymy pole Trojkat
    ptr_fig->area();
    //Uwaga! interfejs dla obliczenia pola jest dokladnie taki samy,
    //ale realizacje funkcji area sa rozne

    cout << "pole kola  : ";
    c1.disp();
    cout << "pole trojkota: ";
    t1.disp();

    //------------test 2-----------------------------//
    //Jeden interfejs nadaje taka mozliwosc:
    Circle c2(20);
    Trojkat t2(6, 8);
    Figura *ptr_f[4] =
    {
        &c1, &c2, &t1, &t2
    };  //tablica wskaznikow

    //liczymy pole kazdej figury
    cout << "pole c1, c2, t1, t2\n";
    int i;
    for(i=0; i<4; i++)
    {
        ptr_f[i]->area();
        ptr_f[i]->disp();
    }

    //to jest to samo, tylko bez uwzglednenia
    //polimorfizmu dynamicznego
    c1.area();
    c2.area();
    t1.area();
    t2.area();

    //wyswietlamy na monitorze
    cout << "pole c1, c2, t1, t2\n";
    c1.disp();
    c2.disp();
    t1.disp();
```

```
    t2.disp();

    //wywolanie funkcji wirtualnej przy uzycju obiektów klas pochodnych
    cout << "pole c1, c2, t1, t2\n";
    for(i=0; i<4; i++)
    {
        //pobieramy obiekt przez wskaznik
        (*ptr_f[i]).area();
        (*ptr_f[i]).disp();
    }

    system("pause");
    return 0;
}
```