Lab_9 Virtual functions.

The Figure class is the base class for Circle, Triangle, Rectangle flat figures. Contains the abstract void *area*() and void *disp*() functions. The *area*() function is designed to calculate the area of a plane figure, and *disp*() – to display this area on the monitor.

```cpp
class Figura
{
public:
      static size_t alloc;  // how many times memory was dynamically allocated for objects
                            //of derived classes
protected:
      double s;             // Flat figure area
public:
      …..................................
      //class method's declaration
};
```

Create classes derived from *Circle*, *Triangle*, *Rectangle* that represent the appropriate flat circle, triangle and rectangle figures. The data should be provided by parameterized constructors. Override the *area* and *disp* methods in each class so that the *area* method counts the area of the appropriate figure and assigns the result to the variable *s*, and *disp* – outputs the figure name. The Rectangle class in the constructor dynamically allocates memory for the *dat* variable and increments the *alloc* variable. The destructor should free this memory and decrement the *alloc* variable.

```cpp
class Rectangle : public Figura
{
      double *dat; //dat[0] - a, dat[1] – b; a, b – side lengths
public:
      // class methods
};
```

The *main* function executes the code:

```cpp
int _tmain(int argc, _TCHAR* argv[])
{
      Figura *ptr = NULL, *ptr_rect = NULL;
      //create a Rectangle object and assign its pointer to the base class pointer
      ptr_rect = new Rectangle (2, 3);

      {
            //create the objects
            Circle cl(2), cl1(3);
            Triangle tr(2, 4);

            srand(time(NULL));

            for(int it=0; it<10; ++it)
            {
                  int ind = rand()%4; // now ind is 0, 1, 2, 3 randomly
                  switch(ind)
                  {
                  case 0: ptr = &cl;
                        break;
                  case 1: ptr = &cl1;
                        break;
```

```cpp
                case 2: ptr = &tr;
                        break;
                case 3: ptr = ptr_rect;
                        break;
                };

                // here ptr - base class pointer, randomly points to one of
                // derived class objects. For which objects will the functions area, disp
                //be called ?
                ptr->area();
                ptr->disp();
        }
    }//now cl, cl1 and tr go beyond a scope resolution area, destructors of these objects
     //should be called

    delete ptr_rect;
    ptr_rect = NULL;

    // if we allocate and deallocate memory correctly, alloc should be zero.
    if(Figura::alloc)
            cout << "!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!\n" << "error: leak of memory\n";

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

//

Exercise 2.

The base class Rectangle calculates the area of the rectangle S = a*h, where a - side length, h - height.

```cpp
class Rectangle
{
protected:
        double S;      // Area
        double a;      // side length
        double h;      // height
public:
        Rectangle(double hh, double aa) : h(hh), a (aa) { S = 0; }
        void Calc() { Disp_T(); Calc_A(); Disp(); }
        void Disp_T() { cout << "Rectangle Area : "; }
        void Calc_A() {     S = a*h; }
        void Disp() { cout << S << endl; }
};
```

An algorithm implemented by the Calc method is used:
1. We output the name of a flat figure - Disp_T ();.
2. We count area of it – Calc_A ();.
3. Output the result – Disp();

Create a Triangle derived class that would call the base class's Calc () method (general field calculation algorithm remains the same), but after calling Calc it should use its function called Disp_T () – outputs "Triangle Area", and Calc_A () – computes the area of a triangle with the base length *a* and height *h*). **You must not change anything in the base class methods, but you can change a declarations of methods. Please, use a dynamic polymorphism. In main() function you must call of methods of the derived class using a pointer to the base class.**