ЛАБОРАТОРНАЯ РАБОТА № 23
КЛАССЫ И СТРУКТУРЫ

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Понятие класса

В языке С широко используются структуры – создаваемый программистом новый тип данных. Структура представляет собой набор из одной или нескольких переменных, возможно даже разных типов, объединенных вместе под одним именем. Переменные, которые объединены структурой, называются полями и могут иметь любой тип, кроме типа этой же структуры, но могут быть указателями на него.

В языке С++ появилось новое понятие – класс. Класс – это тоже тип данных, определяемый пользователем, он, на первый взгляд, очень похож на структуру, но, в отличие от нее, содержит не только переменные, но и функции для работы с этими переменными. Этот принцип программирования называется **инкапсуляция** – объединение данных и кода (функций), манипулирующего этими данными.

Переменные, включенные в класс, называются полями (по аналогии с полями структуры), а также атрибутами, свойствами или просто членами-данными класса. Функции класса называются методами, операциями или членами-функциями. Члены данные и члены-функции, в свою очередь, называются элементами класса.

Описание класса состоит из ключевого слова **class,** после которого пишется имя класса. Далее идет **тело класса**, заключенное в фигурные скобки, после которых стоит точка с запятой. Внутри фигурных скобок располагают объявления членов-данных и членов-функций

```
class Point   // Объявление класса Point
{           // Тело класса
. . . . .
};
```

Важной особенностью класса является возможность скрыть детали его реализации от других классов и функций. Для этого все члены класса делят на **закрытые** и **открытые**. Видимостью элементов класса управляют модификаторы доступа **private** и **public**, которые определяют, что можно, а чего нельзя делать с объектом. Члены класса, описанные после служебного слова **private**, видимы только внутри класса (в методах класса). В данный раздел помещаются члены, доступ к которым извне *должен быть запрещен* и может осуществляться только через методы класса. Этот вид доступа принят в классе по умолчанию. *Открытые члены* класса описываются после спецификатора **public**. Сюда обычно помещают, так называемые, *интерфейсные функции*, с помощью которых можно осуществлять доступ к закрытым членам класса.

Действие любого спецификатора распространяется до следующего спецификатора или до конца класса. Можно задавать несколько секций **private** и **public**, порядок их следования значения не имеет.

Все методы класса имеют непосредственный доступ к его закрытым полям, т.е. в теле функций класса можно обращаться к любым элементам класса, объявленным в части **private**.

Класс – это только описание нового пользовательского типа данных. Чтобы создать конкретные экземпляры (объекты) класса нужно их объявить, как обычные переменные.

Доступ к элементам объекта аналогичен доступу к полям структуры. Для этого используются операция ’.’ (точка) при обращении к элементу через имя объекта и операция ’->’ при обращении через указатель.

Отличия структур и классов можно наглядно продемонстрировать на примере структуры и класса, в которых хранится время. Для описания времени необходимо три переменных, в которых хранится час, минута и секунда. Объединить их вместе удобно с помощью следующей структуры:

```
struct Time       // структура для хранения времени
{
  int Hour;  // поле для хранения часов
  int Min;   // поле для хранения минут
  int Sec;   // поле для хранения секунд
};
```

Поместив в программу данное объявление, мы создаем *новый тип переменных* – **Time**, предназначенный для хранения времени. Его размерность будет 12 байт. Чтобы **создать переменную** этого типа, нужно просто объявить ее в нужном месте программы:

```
Time t1;        // объявление переменной t1 типа Time
```

Чтобы задать значения полей структуры, используем *оператор доступа к членам структуры* (***точка***):

```
t1.Hour = 23;
t1.Min = 59;
t1.Sec = 59;
```

Структура никак не обеспечивает правильность задания времени, например, можно задать явно ошибочные значения полей:

```
t1.Hour = 25;
t1.Min = 61;
t1.Sec = 100;
```

Чтобы создать подобный данной структуре класс для хранения времени, нужно сделать следующее объявление:

```
class CTime       // класс для хранения времени
{
public:  // Далее идут открытые члены класса:
  int Hour;  // поле для хранения часов
  int Min;   // поле для хранения минут
  int Sec;   // поле для хранения секунд
};
```

Модификатор видимости `public` нужен, чтобы обеспечить прямой доступ к членам-данным класса. Чтобы **создать переменную** этого типа (эти переменные обычно называют ***объектами***), нужно тоже объявить ее:

```
CTime ct1;        // объявление переменной ct1 класса CTime
ct1.Hour = 25;
ct1.Min = 61;
ct1.Sec = 100;
```

При таком объявлении класс **CTime** и структура **TRime** совершенно равноправны, и члены-данные класса *также не защищены* от ошибочных значений. Но класс можно сделать более функциональным и, самое главное, можно легко обеспечить контроль за вводимыми значениями полей.

Для этого нужно поместить члены-данные класса **в закрытую часть** и разрешить задавать их значения **только с помощью методов**, которые нужно правильно написать и поместить в открытую часть класса.

```
class CTime       // класс для хранения времени
{
private:           // Далее идут закрытые члены класса:
  int Hour;  // поле для хранения часов
  int Min;   // поле для хранения минут
  int Sec;   // поле для хранения секунд
public:            // Далее идут открытые члены класса:
```

```
  // Сюда нужно поместить методы, с помощью которых
  // будем задавать члены-данные класса
};
```

При таком объявлении прямой доступ к полям объекта будет запрещен:

```
CTime ct1;      // объект класса CTime
ct1.Hour = 25; // !!! Компилятор выдаст ошибку
ct1.Min = 61;  // !!! Компилятор выдаст ошибку
ct1.Sec = 100; // !!! Компилятор выдаст ошибку
```

Метод, который задает, например, секунды, может выглядеть так:

```
void SetSec(int s)
{
  if (s >= 0 && s <= 59)  // проверка на корректность
    Sec = s;
}
```

Так как члены класса, описанные после служебного слова **private**, видимы в методах своего класса, то запись `Sec = s;` будет вполне корректна.

Этот метод, как и другие, можно поместить прямо в теле класса:

```
class CTime       // класс для хранения времени
{
private:          // Далее идут закрытые члены класса:
  int Hour;  // поле для хранения часов
  int Min;   // поле для хранения минут
  int Sec;   // поле для хранения секунд
public:           // Далее идут открытые члены класса:
  void SetSec(int s)  // задаем секунды
  {
    if (s >= 0 && s <= 59) // проверка на корректность
      Sec = s;
  }
  void SetMin(int m)  // задаем минуты
  {
    if (m >= 0 && m <= 59) // проверка на корректность
      Min = m;
  }
  void SetHour(int h)  // задаем часы
  {
    if (h >= 0 && h <= 23) // проверка на корректность
      Hour = h;
  }
};
```

Теперь задавать часы, минуты и секунды в объект класса **CTime** можно **только с помощью этих методов**, которые вызываются тоже с помощью оператор доступа к членам класса (**точка**):

```
CTime ct1;        // Объект класса CTime
ct1.SetHour(23); // Задаем часы
ct1.SetMin(59);  // Задаем минуты
ct1.SetSec(59);  // Задаем секунды
```

Если теперь задать ошибочные значения, то они не запишутся в объект, т.к. проверка на корректность, проводимая в методах, не позволит это сделать:

```
ct1.SetHour(-23); // Час не изменится
ct1.SetMin(61);   // Минуты не изменятся
ct1.SetSec(100);  // Секунды не изменятся
```

Для чтения значений полей тоже напишем соответствующие методы:

```
class CTime       // класс для хранения времени
```

```
{
private:     // Закрытые члены класса:
  int Hour;  // поле для хранения часов
  int Min;   // поле для хранения минут
  int Sec;   // поле для хранения секунд
public:
  void SetSec(int s)
  {
    if (s >= 0 && s <= 59) // задаем секунды
      Sec = s;
  }
  void SetMin(int m)       // задаем минуты
  {
    if (m >= 0 && m <= 59)
      Min = m;
  }
  void SetHour(int h)    // задаем часы
  {
    if (h >= 0 && h <= 23)
      Hour = h;
  }
  int GetSec()    // читаем секунды
  {
    return Sec;
  }
  int GetMin()    // читаем минуты
  {
    return Min;
  }
  int GetHour()    // читаем часы
  {
    return Hour;
  }
};
```

Теперь в программе можно создавать объекты класса **CTime**, задавать значения их полей, считывать значения полей:

```
CTime ct1, ct2;  // Два объекта класса CTime
ct1.SetHour(23); // Задаем часы
ct1.SetMin(59);  // Задаем минуты
ct1.SetSec(59);  // Задаем секунды
ct2.SetHour(12); // Задаем часы
ct2.SetMin(23);  // Задаем минуты
ct2.SetSec(0);  // Задаем секунды
```

Методы задают значения **тех объектов, для которых вызываются**, т.е. в объекте **ct1** будет записано время **23:59:59**, а в **ct2** – **12:23:00**.

```
int k;
k = ct1.GetHour();    // в k будет считано число 23 из ct1
k = ct2.GetHour();    // в k будет считано число 12 из ct2
```

Можно вывести даты на дисплей:

```
cout << ct1.GetHour() << ":" << ct1.GetMin() << ":" << ct1.GetSec();
cout << ct2.GetHour() << ":" << ct2.GetMin() << ":" << ct2.GetSec();
```

Этот пример показывает, что классы **более функциональны** и обеспечивают **защищенность хранимых данных**.

Таким образом, в классе есть две части: интерфейс и реализация. *Интерфейс* отражает *внешнее поведение* объектов данного класса. Внутренняя *реализация*

описывает механизмы достижения желаемого поведения объекта. Тот факт, что детали реализации класса скрыты, **позволяет его модифицировать**, не меняя интерфейса класса.

Например, в классе **CTime** тип **int** для членов-данных явно избыточен, гораздо лучше было бы использовать для них тип **unsigned char** (одного байта хватит для хранения возможных значений полей, а **unsigned** позволит не проверять вводимые значения на отрицательность).

Проведем модернизацию класса, для этого предварительно объявим синоним с помощью **typedef,** а затем заменим тип полей класса:

```
typedef unsigned char BYTE;
class CTime       // класс для хранения времени
{
private:      // Закрытые члены класса:
  BYTE Hour;  // поле для хранения часов
  BYTE Min;   // поле для хранения минут
  BYTE Sec;   // поле для хранения секунд
public:
  void SetSec(BYTE s)    // задаем секунды
  {
    if (s <= 59) Sec = s;
  }
  void SetMin(BYTE m)       // задаем минуты
  {
    if (m <= 59)  Min = m;
  }
  void SetHour(BYTE h)    // задаем часы
  {
    if (h <= 23)  Hour = h;
  }
  BYTE GetSec()  { return Sec;  }  // читаем секунды
  BYTE GetMin()  { return Min;  }  // читаем минуты
  BYTE GetHour() { return Hour; }  // читаем часы
};
```

При этом использование класса останется тем же самым:

```
CTime ct1, ct2;  // Два объекта класса CTime
ct1.SetHour(23); // Задаем часы
ct1.SetMin(59);  // Задаем минуты
ct1.SetSec(59);  // Задаем секунды
ct2.SetHour(12); // Задаем часы
ct2.SetMin(23);  // Задаем минуты
ct2.SetSec(0);  // Задаем секунды
cout << ct1.GetHour() << ":" << ct1.GetMin() << ":" << ct1.GetSec();
cout << ct2.GetHour() << ":" << ct2.GetMin() << ":" << ct2.GetSec();
```

Методы класса можно поместить прямо в теле класса, но можно и вне класса. В последнем случае в классе записывают только заголовок функции (его называют **объявлением метода**), а **определение** (заголовок и тело функции) помещают вне класса, указав его принадлежность данному классу с помощью **оператора расширения области видимости** (**::**).

Сам класс обычно помещают в так называемом *заголовочном файле* (они имеют расширение **h** или **hpp**).

```
                                        // Файл MyProg.h
typedef unsigned char BYTE;
class CTime       // класс для хранения времени
{
```

```
  private:      // Закрытые члены класса:
    BYTE Hour;  // поле для хранения часов
    BYTE Min;   // поле для хранения минут
    BYTE Sec;   // поле для хранения секунд
  public:
    void SetSec(BYTE s);   // задаем секунды
    void SetMin(BYTE m);   // задаем минуты
    void SetHour(BYTE h);  // задаем часы
    BYTE GetSec()  { return Sec; }  // читаем секунды
    BYTE GetMin()  { return Min; }  // читаем минуты
    BYTE GetHour() { return Hour;}  // читаем часы
  };
```

Методы класса помещают в файле с расширением **cpp**. в нем нужно не забыть подключить соответствующий заголовочный файл с помощью директивы препроцессора **#include**.

```
                                        // Файл MyProg.cpp
  #include "MyProg.h"
  void CTime::SetSec(BYTE s)    // задаем секунды
  {
    if (s <= 59) Sec = s;
  }
  void CTime::SetMin(BYTE m)    // задаем минуты
  {
    if (m <= 59)  Min = m;
  }
  void CTime::SetHour(BYTE h)    // задаем часы
  {
    if (h <= 23)  Hour = h;
  }
```

Запись **CTime::** указывает компилятору, что этот метод принадлежит классу **CTime** .

Функциональность класса легко расширять, добавляя по мере необходимости новые методы, например, дополнить класс методом, который позволил бы задавать часы, минуты и секунда одной функцией:

```
  void CTime::Set(BYTE h, BYTE m, BYTE s)
  {
    SetHour(h);
    SetMin(m);
    SetSec(s);
  }
```

В этом методе используем написанные ранее методы **SetHour**, **SetMin**, **SetSec**.

Можно добавить метод, который позволил бы задавать время строкой:

```
  void CTime::Set(const char *str)  // "23:59:59"
  {
  . . . .
  }
```

В этой функции нужно из строки выделить подстроки, соответствующие часам, минутам и секундам, затем преобразовать их в числа и записать их в соответствующие поля класса. Название функции можно сделать таким же, как и у предыдущей функции, т.к. в языке C++ в классе могут быть функции с одинаковыми именами, но отличающимися количеством или типом аргументов.

Можно добавить в класс метод, который позволил бы выводить время на дисплей:

```
  void CTime::Print()
  {
    cout << Hour << ":" << Min << ":" << Sec;
```

```
}
```

Пример использования этих методов:

```
CTime ct1, ct2;  // Два объекта класса CTime
ct1.Set(23, 59, 59); // Задаем часы, минуты, секунды
ct2.Set("12:23;00"); // Задаем часы часы, минуты, секунды строкой
. . . .
ct1.Print(); // Выводим время из ct1 на дисплей
ct2.Print(); // Выводим время из ct2 на дисплей
```

В языке C++ структуры (и объединения) тоже могут содержать методы, но не могут участвовать в наследовании. Поэтому, переходя к языку C++, лучше преобразовать структуру, созданную ранее, в класс, дополнив его методами, которые тоже переделываем из написанных ранее.

Например, пусть у нас была объявлена структура **Точка**:

```
 // структура для хранения параметров точки на плоскости
struct Point
{
   int x;
   int y;
};
```

Для работы с ней были разработаны функции:

```
float BegDistance(Point p) // Расстояние от p до начала координат
{
  return sqrt((float)p.x * p.x + p.y * p.y);
}
void Print(Point p)  // Вывод значений полей точки на дисплей
{                    // в виде: x = 10, y = 20
  cout << "x = " << p.x << ", y = " << p.y << endl;
}
```

Класс можно сделать по примеру данной структуры, но функции поместить прямо в класс. Поля x и y помещаем в закрытую часть класса, в открытой части помещаем методы, позволяющие задавать и читать значения полей (атрибутов), а также методы для вывода на дисплей и вычисления расстояния от начала координат.

```
class MPoint
{
private:
  int X, Y;
public:
  void SetX(int x);
  void SetY(int y);
  void SetXY(int x, int y);
  int GetX();
  int GetY();
  float BegDistance()
  void Print();
};
```

Имя класса

Видимость

Объявление атрибутов

Видимость

Объявление методов

Методам **BegDistance()** и **Print()** не нужны аргументы, т.к. они будут вызываться через имя конкретного объекта, с атрибутами этого объекта они и будут работать. Напишем класс с полностью определенными функциями:

```
class MPoint
{
private:
  int x;
  int y;
public:
```

```
    void SetX(int i) // задаем x
    {  // Здесь можно вставить, если нужно, проверку на корректность
      x = i;
    }
    void SetY(int i) // задаем y
    {  // Здесь можно вставить, если нужно, проверку на корректность
      y = i;
    }
    void SetXY(int i, int j)
    {
      SetX(i);
      SetY(j);
    }
    int GetX() {return x;}
    int GetY() {return y;}
    float BegDistance() // Расстояние от точки, для которой
    {              // будет вызван этот метод, до начала координат
      return sqrt((float)x * x + y * y);
    }
    void Print()  // Вывод значений полей точки, для которой
    { // будет вызван этот метод, на дисплей в виде: x = 10, y = 20
      cout << "x = " << x << ", y = " << y ;
    }
  };
```

Теперь в программах можно использовать данный класс, объявляя объекты, массивы объектов и т.д., вызывая разработанные методы для этих объектов.

```
  MPoint p1, p2;
  float f;
  p1.SetXY(10, 20);
  p2.SetXY(25, 125);

  cout << "Точка 1: ";
  p1.Print();
  f = p1.BegDistance();
  cout << " Расстояние до начала координат = " << f << endl;
  cout << "Точка 2: ";
  p2.Print();
  f = p2.BegDistance();
  cout << " Расстояние до начала координат = " << f << endl;
```

Класс можно легко модифицировать, добавлять функциональность, например, добавить функции сдвига точки по оси X и Y.

```
  class MPoint
  {
  private:
    int x;
    int y;
  public:
    void SetX(int i) { x = i; }
    void SetY(int i) { y = i; }
    void SetXY(int i, int j) { SetX(i); SetY(j); }
    int GetX() { return x; }
    int GetY() { return y; }
    float BegDistance() { return sqrt((float)x * x + y * y); }
    void Print() { cout << "x = " << x << ", y = " << y ; }
    void ShiftX(int i) { x += i; } // Сдвиг по оси x на i пикселов
```

```
  void ShiftY(int i) { y += i; } // Сдвиг по оси y на i пикселов
};
```

Функции, которые были написаны для структур и имели по два аргумента нецелесообразно делать членами класса (их можно объявить в качестве перегруженных операций, об этом будет рассказано в последующих лабораторных работах), поэтому их лучше оставить глобальными функциями. Например, функция, сравнивающая две точки по расстоянию от начала координат может выглядеть так:

```
bool CmpP(MPoint a, MPoint b) // a дальше б
{
  return a.BegDistance() > b.BegDistance();
}
```

У функции два аргумента – по числу сравниваемых объектов, а возвращает функция значение **true**, если точка **a** находится дальше от начала координат, и **false** – в противном случае. Возвращаемое значение типа **bool** позволит использовать эту функцию, например, при сортировке массива точек.

```
void SwapP(MPoint *pa, MPoint *pb) // Перестановка точек
{
  MPoint c = *pa;
  *pa = *pb;
  *pb = c;
}
void OrderP(MPoint *pa, MPoint *pb) // Сравнение и перестановка
{
  if (CmpP(*pa, *pb))
    SwapP(pa, pb);
}
void BubbleSortP(MPoint *A, int N) // Сортировка точек
{
  for (int i = 0; i < N - 1; i++)
    for (int j = N - 2; j >= i; j--)
      OrderP(&A[j], &A[j + 1]);
}
```

Пример использования:

```
#include <cstdlib> // Для подключения генератора случайных чисел
#include <time.h>
#include <iostream>
#include <iomanip>
#include <windows.h>

#define NMAX 100
#define NMIN -100
#define SIZE 10

// Сюда вставить объявление класса и глобальных функций. . . .

int main()
{
  MPoint ArrP[SIZE];
  int kx, ky;

  srand(time(0));// Начальная инициализация генератора случ. чисел
  SetConsoleCP(1251);
  SetConsoleOutputCP(1251);
```

```
  for (int i = 0; i < SIZE; i++)
  {              // Заполняем случ. числами
    kx = (float)rand()/RAND_MAX * (NMAX - NMIN) + NMIN;
    ky = (float)rand()/RAND_MAX * (NMAX - NMIN) + NMIN;
    ArrP[i].SetXY(kx, ky);
  }
  cout << "До сортировки:" << endl;
  for (int i = 0; i < SIZE; i++)
  {
    cout << "Точка " << i+1 << ": ";
    ArrP[i].Print();
    cout << ", Расстояние = " << ArrP[i].BegDistance() << endl;
  }
  BubbleSortP(ArrP, SIZE);
  cout << "После сортировки:" << endl;
  for (int i = 0; i < SIZE; i++)
  {
    cout << "Точка " << i+1 << ": ";
    ArrP[i].Print();
    cout << ", Расстояние = " << ArrP[i].BegDistance() << endl;
  }
  cin.get();
  return 0;
}
```

Результат работы программы:

```
До сортировки:
Точка 1: x = 49, y = -4, Расстояние = 49.163
Точка 2: x = 95, y = -2, Расстояние = 95.021
Точка 3: x = -65, y = -59, Расстояние = 87.7838
Точка 4: x = -68, y = 54, Расстояние = 86.8332
Точка 5: x = -69, y = 60, Расстояние = 91.4385
Точка 6: x = 30, y = 40, Расстояние = 50
Точка 7: x = -57, y = 47, Расстояние = 73.8783
Точка 8: x = -13, y = 63, Расстояние = 64.3273
Точка 9: x = 86, y = -47, Расстояние = 98.0051
Точка 10: x = -51, y = 60, Расстояние = 78.7464
После сортировки:
Точка 1: x = 49, y = -4, Расстояние = 49.163
Точка 2: x = 30, y = 40, Расстояние = 50
Точка 3: x = -13, y = 63, Расстояние = 64.3273
Точка 4: x = -57, y = 47, Расстояние = 73.8783
Точка 5: x = -51, y = 60, Расстояние = 78.7464
Точка 6: x = -68, y = 54, Расстояние = 86.8332
Точка 7: x = -65, y = -59, Расстояние = 87.7838
Точка 8: x = -69, y = 60, Расстояние = 91.4385
Точка 9: x = 95, y = -2, Расстояние = 95.021
Точка 10: x = 86, y = -47, Расстояние = 98.0051
```

ЭКСПЕРИМЕНТАЛЬНАЯ ЧАСТЬ

В начале программы ОБЯЗАТЕЛЬНО выводить:
ФИО, группа, номер лаб. работы, номер варианта.

Задание. На базе структуры, созданной в лабораторной работе 12, разработать класс и глобальные функции для работы с объектами данного класса, а также программу, демонстрирующую их работу. Создать массив из 6 объектов класса и показать работу его методов и глобальных функций. Вариант задания выдает преподаватель.

Для получения максимального балла добавить в программу:
– меню и возможность проводить повторные вычисления;
– возможность задавать параметры объектов вручную и случайными числами.

1. Разработать класс «Прямоугольник», прямоугольник задается координатами его верхнего левого и правого нижнего углов. Написать методы класса, которые вычисляют и возвращают длину диагонали прямоугольника, площадь прямоугольника, периметр прямоугольника. Написать методы класса для ввода данных прямоугольника и вывода на дисплей. Написать программу, демонстрирующую работу с массивом объектов класса.

2. Разработать класс «Равносторонний треугольник», треугольник задается координатами его центра и длиной стороны. Написать методы класса, которые вычисляют и возвращают площадь треугольника, периметр треугольника. Написать методы класса для ввода данных треугольника и вывода на дисплей. Написать программу, демонстрирующую работу с массивом объектов класса.

3. Разработать класс «Квадрат», квадрат задается координатами его центра и длиной стороны. Написать методы класса, которые вычисляют и возвращают длину диагонали квадрата, площадь квадрата, площадь описанной вокруг квадрата окружности. Написать методы класса для ввода данных квадрата и вывода на дисплей. Написать программу, демонстрирующую работу с массивом объектов класса.

4. Разработать класс «Окружность», окружность задается координатами ее центра и радиусом. Написать методы класса, которые вычисляют и возвращают площадь круга, длину окружности, длину стороны квадрата, вписанного в окружность. Написать методы класса для ввода данных круга и вывода на дисплей. Написать программу, демонстрирующую работу с массивом объектов класса.

5. Разработать класс «Прямоугольник», прямоугольник задается координатами его верхнего левого и правого нижнего углов. Написать метод класса, который вычисляет и возвращает площадь прямоугольника. Написать функцию, которая сравнивает два прямоугольника по площади и возвращает тот прямоугольник, площадь которого больше. Написать функцию, которая вычисляет и возвращает разность площадей двух прямоугольников. Написать методы класса для ввода данных прямоугольника и вывода на дисплей. Написать программу, демонстрирующую работу с массивом объектов класса и функциями.

6. Разработать класс «Квадрат», квадрат задается координатами его центра и длиной стороны. Написать метод класса, который вычисляет и возвращает площадь квадрата, и метод, который определяет, находится ли начало координат внутри квадрата. Написать функцию, которая сравнивает два квадрата по площади и возвращает площадь того, который больше. Написать методы класса для ввода данных квадрата и вывода на дисплей. Написать программу, демонстрирующую работу с массивом объектов класса и функциями.

7. Разработать класс «Окружность», окружность задается координатами ее центра и радиусом. Написать методы класса, которые вычисляют и возвращают площадь окружности и расстояние от ее центра до начала координат. Написать функцию, которая сравнивает две окружности по площади и возвращает площадь той, которая меньше. Написать методы класса для ввода данных окружности и вывода на дисплей. Написать программу, демонстрирующую работу с массивом объектов класса и функциями.

8. Разработать класс «Отрезок», отрезок задается координатами на плоскости его начала и конца. Написать метод класса, который вычисляет и возвращает длину отрезка. Написать функцию, которая сравнивает два отрезка на плоскости и возвращает тот, длина которого больше. Написать методы класса для ввода данных отрезка и вывода на дисплей. Написать программу, демонстрирующую работу с массивом объектов класса и функциями.

9. Разработать класс «Отрезок», отрезок задается координатами на плоскости его начала и конца. Написать метод класса, который вычисляет и возвращает длину отрезка, и метод, который определяет, пересекает ли отрезок оси координат. Написать методы класса для ввода данных отрезка и вывода на дисплей. Написать программу, демонстрирующую работу с массивом объектов класса и функциями.

10. Разработать класс «Равносторонний треугольник», треугольник задается координатами его центра и длиной стороны. Написать метод класса, который вычисляет и возвращает периметр треугольника. Написать функцию, которая сравнивает два треугольника по площади и возвращает тот треугольник, площадь которого больше. Написать методы класса для ввода данных треугольника и вывода на дисплей. Написать программу, демонстрирующую работу с массивом объектов класса и функциями.

11. Разработать класс «Треугольник», треугольник задается координатами его вершин. Написать метод класса, который вычисляет и возвращает периметр треугольника. Написать функцию, которая сравнивает периметры двух треугольников и возвращает тот треугольник, периметр которого больше. Написать методы класса для ввода данных треугольника и вывода на дисплей. Написать программу, демонстрирующую работу с массивом объектов класса и функциями.

12. Разработать класс «Прямоугольник», прямоугольник задается координатами его верхнего левого и правого нижнего углов. Написать методы класса, которые вычисляют и возвращают площадь прямоугольника и периметр прямоугольника. Написать функцию, которая вычисляет и возвращает сумму площадей двух прямоугольников. Написать методы класса для ввода данных прямоугольника и вывода на дисплей. Написать программу, демонстрирующую работу с массивом объектов класса и функциями.

13. Разработать класс «Квадрат», квадрат задается координатами его центра и длиной стороны. Написать методы класса, которые вычисляют и возвращают периметр и площадь квадрата. Написать функцию, которая сравнивает периметры двух квадратов, и возвращает тот квадрат, периметр которого больше. Написать функцию, которая вычисляет и возвращает сумму площадей трех квадратов. Написать методы класса для ввода данных квадрата и вывода на дисплей. Написать программу, демонстрирующую работу с массивом объектов класса и функциями

14. Разработать класс «Окружность», окружность задается координатами ее центра и радиусом. Написать метод класса, который вычисляет и возвращает площадь окружности. Написать функцию, которая сравнивает площади двух окружностей и

возвращает ту окружность, площадь которой меньше. Написать функцию, которая вычисляет и возвращает сумму площадей трех окружностей. Написать методы класса для ввода данных окружности и вывода на дисплей. Написать программу, демонстрирующую работу с массивом объектов класса и функциями

15. Разработать класс «Ромб», ромб задается координатами его центра и длиной диагоналей. Написать метод класса, который вычисляет и возвращает площадь ромба. Написать функцию, которая сравнивает площади двух ромбов и возвращает тот ромб, площадь которого меньше. Написать методы класса для ввода данных ромба и вывода на дисплей. Написать программу, демонстрирующую работу с массивом объектов класса и функциями

16. Разработать класс «Окружность», окружность задается координатами ее центра и радиусом. Написать метод класса, который вычисляет и возвращает площадь круга, и метод, который определяет, захватывает ли окружность начало координат. Написать методы класса для ввода данных окружности и вывода на дисплей. Написать программу, демонстрирующую работу с массивом объектов класса и функциями

17. Разработать класс «Квадрат», квадрат задается координатами его центра и длиной стороны. Написать методы класса, которые вычисляют и возвращают площадь квадрата и расстояние от его центра до начала координат. Написать функцию, которая сравнивает два квадрата по площади и возвращает площадь того, который меньше. Написать функцию, которая определяет, центр какого квадрата находится ближе к началу координат, и возвращает его. Написать методы класса для ввода данных квадрата и вывода на дисплей. Написать программу, демонстрирующую работу с массивом объектов класса и функциями

18. Разработать класс «Окружность», окружность задается координатами ее центра и радиусом. Написать метод класса, который вычисляет и возвращает площадь окружности. Написать функцию, которая определяет, пересекаются две окружности или нет. Написать методы класса для ввода данных окружности и вывода на дисплей. Написать программу, демонстрирующую работу с массивом объектов класса и функциями

19. Разработать класс «Окружность», окружность задается координатами ее центра и радиусом. Разработать класс «Точка на плоскости». Написать функцию, которая вычисляет расстояние между двумя точками. Написать метод класса «Окружность», который определяет, находится ли заданная точка внутри нее. Написать методы классов для ввода данных и вывода на дисплей. Написать программу, демонстрирующую работу с массивом объектов класса и функциями

20. Разработать класс «Квадрат», квадрат задается координатами его центра и длиной стороны. Разработать класс «Точка на плоскости». Написать функцию, которая вычисляет расстояние между двумя точками. Написать метод класса «Квадрат», который определяет, находится ли заданная точка внутри него. Написать методы классов для ввода данных и вывода на дисплей. Написать программу, демонстрирующую работу с массивом объектов класса и функциями