# 课设设计源代码合集

## 基础1.cpp

|  |
| --- |
| #include <iostream>  class matrix  {  private:      int \*\*matri;      int row;      int col;  public:      matrix(int row, int col) // 创建一个row行col列的矩阵      {          this->row = row;          this->col = col;          matri = new int \*[row];          for (int i = 0; i < row; i++)          {              matri[i] = new int[col];          }      }      ~matrix()      {          if (matri != nullptr)          {              for (int i = 0; i < row; i++)              {                  if (matri[i] != nullptr)                  {                      delete[] matri[i];                      matri[i] = nullptr;                  }              }              delete[] matri;              matri = nullptr;          }      }      void matrixinput()      {          for (int i = 0; i < this->row; i++)          {              for (int j = 0; j < this->col; j++)              {                  std::cin >> matri[i][j];              }          }      }      void matrixoutput()      {          for (int i = 0; i < this->row; i++)          {              for (int j = 0; j < this->col; j++)              {                  std::cout << matri[i][j] << " ";              }              std::cout << std::endl;          }      }      // 使用常量引用而杜绝拷贝构造函数的调用      void matrixplus(const matrix &m1, const matrix &m2)      {          if (m1.row != m2.row || m1.col != m2.col)          {              std::cout << "矩阵大小不同，无法相加" << std::endl;              return;          }          for (int i = 0; i < m1.row; i++)          {              for (int j = 0; j < m1.col; j++)              {                  matri[i][j] = m1.matri[i][j] + m2.matri[i][j];              }          }      }      void matrixminus(const matrix &m1, const matrix &m2)      {          if (m1.row != m2.row || m1.col != m2.col)          {              std::cout << "矩阵大小不同，无法相减" << std::endl;              return;          }          for (int i = 0; i < m1.row; i++)          {              for (int j = 0; j < m1.col; j++)              {                  matri[i][j] = m1.matri[i][j] - m2.matri[i][j];              }          }      }  };  int main()  {      matrix m(4, 5);      std::cout << "矩阵创建成功，请输入4\*5矩阵" << std::endl;      m.matrixinput();      std::cout << "矩阵为：" << std::endl;      m.matrixoutput();      std::cout << "请输入两个矩阵，用于相加和相减" << std::endl;      matrix m1(4, 5);      m1.matrixinput();      matrix m2(4, 5);      m2.matrixinput();      m2.matrixoutput();      matrix m3(4, 5);      m3.matrixminus(m, m1);      m3.matrixoutput();      m3.matrixplus(m, m1);      m3.matrixoutput();      // m1.~matrix();      // m2.~matrix();      // m3.~matrix();      int row, col;      std::cout << "请输入行数和列数：" << std::endl;      std::cin >> row >> col;      matrix \*A1 = new matrix(row, col);      matrix \*A2 = new matrix(row, col);      matrix \*A3 = new matrix(row, col);      std::cout << "请输入两个矩阵，用于相加和相减" << std::endl;      A1->matrixinput();      A2->matrixinput();      A3->matrixminus(\*A1, \*A2);      A3->matrixoutput();      A3->matrixplus(\*A1, \*A2);      A3->matrixoutput();      delete A1;      delete A2;      delete A3;      // A1.~matrix();      // A2.~matrix();      // A3.~matrix();      return 0;  } |

## 基础2.1.cpp

|  |
| --- |
| #include <iostream>  #include <cmath>  #include <iomanip>  using namespace std;  class Point  {  private:      int x;      int y;  public:      Point(int x = 0, int y = 0)      {          this->x = x;          this->y = y;          cout << "构造点函数" << endl;      }      ~Point()      {          cout << "析构点函数，位置：（" << x << "，" << y << "）" << endl;      }      // 输入方法      void input()      {          std::cout << "请输入x坐标: ";          std::cin >> x;          std::cout << "请输入y坐标: ";          std::cin >> y;      }      // 静态工厂方法      static Point createFromInput()      {          Point p;          p.input();          return p;      }      // 获取坐标      int getX() const      {          return x;      }      int getY() const      {          return y;      }      double distance(const Point &p)      {          return sqrt((x - p.x) \* (x - p.x) + (y - p.y) \* (y - p.y));      }  };  class Circle  {  private:      Point center;      int radius;  public:      Circle(const Point &center)      {          this->center = center;          std::cout << "请输入半径：";          std::cin >> radius;      }      // 默认构造，通过输入创建      Circle()      {          std::cout << "是否要输入圆心坐标？(y/n): ";          char choice;          std::cin >> choice;          if (choice == 'y' || choice == 'Y')          {              int x, y;              std::cout << "请输入x坐标: ";              std::cin >> x;              std::cout << "请输入y坐标: ";              std::cin >> y;              center = Point(x, y);          }          else          {              center = Point();              std::cout << "使用默认圆心坐标(0,0)" << std::endl;          }          std::cout << "请输入半径：";          std::cin >> radius;      }      ~Circle()      {          std::cout << "析构圆函数，圆心：（" << center.getX() << "，" << center.getY()                    << "），半径：" << radius << std::endl;      }      bool isIntersect(const Circle &c)      {          return center.distance(c.center) < radius + c.radius; // 不包括相切      }  };  int main()  {      std::cout << "创建第一个点：" << std::endl;      // 使用静态工厂方法创建点      Point p1 = Point::createFromInput();      std::cout << "创建的点坐标为: (" << p1.getX() << ", " << p1.getY() << ")" << std::endl;      std::cout << "创建第二个点：" << std::endl;      Point p2;      p2.input();      std::cout << "创建的点坐标为: (" << p2.getX() << ", " << p2.getY() << ")" << std::endl;      std::cout << "使用第一个点创建圆：" << std::endl;      Circle c1(p1);      std::cout << "直接创建圆：" << std::endl;      Circle c2;      if (c1.isIntersect(c2))      {          std::cout << "两个圆相交" << std::endl;      }      else      {          std::cout << "两个圆不相交" << std::endl;      }      return 0;  } |

## 基础2.2.cpp

|  |
| --- |
| #include <iostream>  class matrix  {  private:      int \*\*matri;      int rows;      int cols;  public:      matrix(int rows, int cols) // 创建一个rows行lines列的矩阵      {          this->rows = rows;          this->cols = cols;          matri = new int \*[rows];          for (int i = 0; i < rows; i++)          {              matri[i] = new int[cols];          }          for (int i = 0; i < rows; i++) // 默认初始化          {              for (int j = 0; j < cols; j++)              {                  matri[i][j] = 0;              }          }      }      matrix()      {          std::cout << "请输入行数和列数：" << std::endl;          std::cin >> rows >> cols;          matri = new int \*[rows];          for (int i = 0; i < rows; i++)          {              matri[i] = new int[cols];          }          std::cout << "请输入矩阵元素：" << std::endl;          matrixinput();      }      matrix(const matrix &m) // 拷贝构造      {          this->rows = m.rows;          this->cols = m.cols;          matri = new int \*[rows];          for (int i = 0; i < rows; i++)          {              matri[i] = new int[cols];          }          for (int i = 0; i < rows; i++)          {              for (int j = 0; j < cols; j++)              {                  matri[i][j] = m.matri[i][j];              }          }      }      ~matrix() // 析构函数，释放内存      {          if (matri != nullptr)          {              for (int i = 0; i < rows; i++)              {                  if (matri[i] != nullptr)                  {                      delete[] matri[i];                      matri[i] = nullptr;                  }              }              delete[] matri;              matri = nullptr;          }      }      void matrixinput()      {          for (int i = 0; i < this->rows; i++)          {              for (int j = 0; j < this->cols; j++)              {                  std::cin >> matri[i][j];              }          }      }      void matrixoutput()      {          for (int i = 0; i < this->rows; i++)          {              for (int j = 0; j < this->cols; j++)              {                  std::cout << matri[i][j] << " ";              }              std::cout << std::endl;          }      }      // 友元函数方法可以需要两个参数（左右操作数）      // 也可以放置在类外部，作为全局函数      friend matrix operator+(const matrix &m1, const matrix &m2)      {          if (m1.rows != m2.rows || m1.cols != m2.cols)          {              std::cout << "矩阵大小不同，无法相加" << std::endl;              return matrix(m1);          }          matrix m3(m1.rows, m1.cols);          for (int i = 0; i < m1.rows; i++)          {              for (int j = 0; j < m1.cols; j++)              {                  m3.matri[i][j] = m1.matri[i][j] + m2.matri[i][j];              }          }          return m3;      }      matrix operator-(const matrix &m1)      {          if (rows != m1.rows || cols != m1.cols)          {              std::cout << "矩阵大小不同，无法相减" << std::endl;              return matrix(\*this);          }          matrix m3(rows, cols);          for (int i = 0; i < rows; i++)          {              for (int j = 0; j < cols; j++)              {                  m3.matri[i][j] = matri[i][j] - m1.matri[i][j];              }          }          return m3;      }      void matrixplus(const matrix &m1, const matrix &m2)      {          if (m1.rows != m2.rows || m1.cols != m2.cols)          {              std::cout << "矩阵大小不同，无法相加" << std::endl;              return;          }          for (int i = 0; i < m1.rows; i++)          {              for (int j = 0; j < m1.cols; j++)              {                  matri[i][j] = m1.matri[i][j] + m2.matri[i][j];              }          }      }        void matrixminus(const matrix &m1, const matrix &m2)      {          if (m1.rows != m2.rows || m1.cols != m2.cols)          {              std::cout << "矩阵大小不同，无法相减" << std::endl;              return;          }          for (int i = 0; i < m1.rows; i++)          {              for (int j = 0; j < m1.cols; j++)              {                  matri[i][j] = m1.matri[i][j] - m2.matri[i][j];              }          }      }      matrix &operator=(const matrix &m) // 复制并交换？      {          if (this == &m) // 自赋值检查          {              return \*this;          }          if (matri != nullptr) // 旧内存释放          {              for (int i = 0; i < rows; i++)              {                  if (matri[i] != nullptr)                  {                      delete[] matri[i];                      matri[i] = nullptr;                  }              }              delete[] matri;              matri = nullptr;          }          rows = m.rows;          cols = m.cols;          matri = new int \*[rows];          for (int i = 0; i < rows; i++)          {              matri[i] = new int[cols];              for (int j = 0; j < cols; j++)              {                  matri[i][j] = m.matri[i][j];              }          }          return \*this;      }  };  int main()  {      matrix A1 = matrix();      matrix A2 = matrix();      matrix A3 = A1 + A2;      std::cout << "相加后的矩阵：" << std::endl;      A3.matrixoutput();      matrix A4 = matrix(A3);      std::cout << "拷贝构造相加的矩阵：" << std::endl;      A4.matrixoutput();      int rows, cols;      std::cout << "请输入行数和列数：" << std::endl;      std::cin >> rows >> cols;      matrix B1(rows, cols);      matrix B2(rows, cols);      std::cout << "请输入第一个矩阵：" << std::endl;      B1.matrixinput();      std::cout << "请输入第二个矩阵：" << std::endl;      B2.matrixinput();      matrix B3 = B1 + B2;      std::cout << "相加后的矩阵：" << std::endl;      B3.matrixoutput();      matrix B4 = B1 - B2;      std::cout << "相减后的矩阵：" << std::endl;      B4.matrixoutput();      matrix \*C1 = new matrix(2, 2);      matrix \*C2 = new matrix(2, 2);      std::cout << "请输入第一个矩阵：" << std::endl;      C1->matrixinput();      std::cout << "请输入第二个矩阵：" << std::endl;      C2->matrixinput();      matrix C3 = \*C1 + \*C2;      std::cout << "相加后的矩阵：" << std::endl;      C3.matrixoutput();      matrix C4 = \*C1 - \*C2;      std::cout << "相减后的矩阵：" << std::endl;      C4.matrixoutput();      delete C1;      delete C2;      return 0;  } |

## 基础3.cpp

|  |
| --- |
| #include <iostream>  #include <cmath>  class shape  {  private:  public:      shape()      {          std::cout << "shape 构造" << std::endl;      }      virtual ~shape()      {          std::cout << "shape 析构" << std::endl;      }      virtual double area() = 0;      // 重载1：打印信息      void print()      {          std::cout << "这是一个形状" << std::endl;      }      // 重载2：打印信息(带信息)      void print(const std::string &message)      {          std::cout << "形状信息: " << message << std::endl;      }  };  class circle : public shape  {  private:      double radius;  public:      circle(double r) : radius(r)      {          std::cout << "circle 构造函数被调用，半径=" << radius << std::endl;      }      // 析构函数      ~circle() override      {          std::cout << "circle 析构函数被调用，半径=" << radius << std::endl;      }      double area() override      {          return M\_PI \* radius \* radius;      }      double area(int r) const      {          return M\_PI \* r \* r;      }      // 重写：基类的print方法      void print()      {          std::cout << "这是一个圆形，半径=" << radius << std::endl;      }  };  class rectangle : public shape  {  private:      double a;      double b;  public:      // 构造函数      rectangle(double w, double h) : a(w), b(h)      {          std::cout << "rectangle 构造函数被调用，宽=" << w << "，高=" << h << std::endl;      }      // 析构函数      virtual ~rectangle() override      {          std::cout << "rectangle 析构函数被调用" << std::endl;      }      virtual double area() override      {          return a \* b;      }      double area(int x, int y)      {          return x \* y;      }  };  class square : public rectangle  {  private:      double a;  public:      square(double a) : rectangle(a, a), a(a) // 首先初始化基类，然后初始化自己的成员      {          std::cout << "square 构造函数被调用，边长=" << a << std::endl;      }      ~square() override      {          std::cout << "square 析构函数被调用" << std::endl;      }      double area() override      {          return a \* a;      }      double area(int r)      {          return r \* r;      }  };  int main()  {      std::cout << "\n===== 测试1: 单独创建各种形状对象 =====\n"                << std::endl;      // 创建圆形对象      std::cout << "创建圆形对象:" << std::endl;      circle c(5.0);      std::cout << "圆形面积: " << c.area() << std::endl;      c.print();      std::cout << std::endl;      // 创建矩形对象      std::cout << "创建矩形对象:" << std::endl;      rectangle r(4.0, 6.0);      std::cout << "矩形面积: " << r.area() << std::endl;      r.print();      std::cout << std::endl;      // 创建正方形对象      std::cout << "创建正方形对象:" << std::endl;      square s(3.0);      std::cout << "正方形面积: " << s.area() << std::endl;      s.print();      std::cout << std::endl;      std::cout << "\n===== 测试2: 观察对象生命周期 =====\n"                << std::endl;      std::cout << "创建局部作用域..." << std::endl;      {          std::cout << "进入局部作用域" << std::endl;          circle localCircle(2.0);          rectangle localRect(2.0, 3.0);          square localSquare(4.0);          std::cout << "即将离开局部作用域" << std::endl;      } // 局部对象在这里销毁      std::cout << "已离开局部作用域，局部对象已被销毁" << std::endl;      std::cout << "\n===== 测试3: 使用基类指针实现多态 =====\n"                << std::endl;      // 使用基类指针数组存储不同类型的对象      shape \*shapes[3];      std::cout << "创建形状数组:" << std::endl;      shapes[0] = new circle(3.0);      shapes[1] = new rectangle(5.0, 4.0);      shapes[2] = new square(2.0);      // 通过基类指针调用虚函数      std::cout << "\n计算不同形状的面积:" << std::endl;      for (int i = 0; i < 3; i++)      {          std::cout << "形状 " << (i + 1) << " 的面积: " << shapes[i]->area() << std::endl;          shapes[i]->print(); // 多态调用      }      // 释放动态分配的对象      std::cout << "\n释放动态创建的对象:" << std::endl;      for (int i = 0; i < 3; i++)      {          delete shapes[i];      }      std::cout << "\n===== 测试4: 重载方法测试 =====\n"                << std::endl;      // 测试重载的area方法      std::cout << "调用重载的area方法:" << std::endl;      std::cout << "圆形: 指定半径为4的圆面积 = " << c.area(4) << std::endl;      std::cout << "矩形: 指定宽3高7的矩形面积 = " << r.area(3, 7) << std::endl;      std::cout << "正方形: 指定边长5的正方形面积 = " << s.area(5) << std::endl;      // 测试重载的print方法      std::cout << "\n调用重载的print方法:" << std::endl;      c.print();      r.print("这是一个矩形对象");      s.print("这是一个正方形对象");      std::cout << "\n===== 程序结束, 剩余对象将被销毁 =====\n"                << std::endl;      // 使用system("pause")暂停程序(仅适用于Windows)      system("pause");      return 0;  } |

## 基础4.cpp

|  |
| --- |
| #include <iostream>  #include <cstdlib>  #include <ctime>  #include <limits> // 用于 std::numeric\_limits  using namespace std;  int main()  {      int num = rand() % 1000 + 1;      int guess = 0;      while (guess != num)      {          cout << "请输入一个1-1000的整数：" << endl;          if (!(cin >> guess))          {              cout << "输入错误，请输入整数" << endl;              cin.clear();                                         // 清除错误标志              cin.ignore(numeric\_limits<streamsize>::max(), '\n'); // 清除输入缓冲区              continue;                                            // 继续循环，重新输入          }          if (guess < 1 || guess > 1000)          {              cout << "请输入1-1000范围内的整数！" << endl;              continue;          }          if (guess > num)          {              cout << "猜大了" << endl;          }          else if (guess < num)          {              cout << "猜小了" << endl;          }          else          {              cout << "猜对了" << endl;          }      }  } |

## 基础5.cpp

|  |
| --- |
| #include <iostream>  #include <cmath>  class Point  {  private:      int x;      int y;  public:      Point(int x = 0, int y = 0)      {          this->x = x;          this->y = y;          std::cout << "构造点函数" << std::endl;      }      ~Point()      {          std::cout << "析构点函数，位置：（" << x << "，" << y << "）" << std::endl;      }      // 输入方法      void input()      {          std::cout << "请输入x坐标: ";          std::cin >> x;          std::cout << "请输入y坐标: ";          std::cin >> y;      }      // 静态工厂方法      static Point createFromInput()      {          Point p;          p.input();          return p;      }      // 获取坐标      int getX() const      {          return x;      }      int getY() const      {          return y;      }      double distance(const Point &p)      {          return sqrt((x - p.x) \* (x - p.x) + (y - p.y) \* (y - p.y));      }      // 重载运算符,修改x和y      Point operator+(const Point &p) const      {          return Point(x + p.x, y + p.y);      }      Point operator-(const Point &p) const      {          return Point(x - p.x, y - p.y);      }      Point operator++() // 不使用const表示修改自身      {                  // 前置改变自身          ++x;          ++y;          return \*this;      }      Point operator++(int) // 自身修改但返回之前的值      {                     // 后置改变自身          Point temp = \*this;          ++x;          ++y;          return temp;      }      Point operator--()      {          --x;          --y;          return \*this;      }      Point operator--(int)      {          Point temp = \*this;          --x;          --y;          return temp;      }      Point &operator=(const Point &p)      {          if (this != &p) // 防止自赋值          {              x = p.x;              y = p.y;          }          // 无内存分配，故无需判断是否需要释放旧内存          return \*this;      }  };  class shape  {  private:  public:      shape()      {          std::cout << "shape 构造" << std::endl;      }      virtual ~shape()      {          std::cout << "shape 析构" << std::endl;      }      virtual double area() = 0;      // 重载1：打印信息      void print()      {          std::cout << "这是一个形状" << std::endl;      }      // 重载2：打印信息(带信息)      void print(const std::string &message)      {          std::cout << "形状信息: " << message << std::endl;      }  };  class circle : public shape  {  private:      double radius;      Point center; // 圆心坐标  public:      circle(double r, double x = 0, double y = 0) : radius(r), center(x, y)      {          std::cout << "circle 构造函数被调用，半径=" << radius                    << "，圆心=(" << center.getX() << "," << center.getY() << ")" << std::endl;      }      circle()      {          std::cout << "是否要输入圆心坐标？(y/n): ";          char choice;          std::cin >> choice;          std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n');          if (choice == 'y' || choice == 'Y')          {              int x, y;              std::cout << "请输入x坐标: ";              std::cin >> x;              std::cout << "请输入y坐标: ";              std::cin >> y;              center = Point(x, y);          }          else          {              center = Point();              std::cout << "使用默认圆心坐标(0,0)" << std::endl;          }          std::cout << "请输入半径：";          std::cin >> radius;          std::cout << "半径为：" << radius << std::endl;      }      // 析构函数      ~circle() override      {          std::cout << "circle 析构函数被调用，半径=" << radius << std::endl;      }      double area() override      {          return M\_PI \* radius \* radius;      }      double area(int r) const      {          return M\_PI \* r \* r;      }      // 重写：基类的print方法      void print()      {          std::cout << "这是一个圆形，半径=" << radius << std::endl;      }      void printXY()      {          std::cout << "圆心坐标： (" << center.getX() << ", " << center.getY() << ")" << std::endl;      }      void centerpluspre() // 前置++重载      {          center++;      }      void centerpluspost() // 后置++重载      {          center++;      }      void centerminuspre() // 前置--重载      {          center--;      }      void centerminuspost() // 后置--重载      {          center--;      }  };  class rectangle : public shape  {  private:      double a;      double b;  public:      // 构造函数      rectangle(double w, double h) : a(w), b(h)      {          std::cout << "rectangle 构造函数被调用，宽=" << w << "，高=" << h << std::endl;      }      // 析构函数      virtual ~rectangle() override      {          std::cout << "rectangle 析构函数被调用" << std::endl;      }      virtual double area() override      {          return a \* b;      }      double area(int x, int y)      {          return x \* y;      }  };  class square : public rectangle  {  private:      double a;  public:      square(double a) : rectangle(a, a), a(a) // 首先初始化基类，然后初始化自己的成员      {          std::cout << "square 构造函数被调用，边长=" << a << std::endl;      }      ~square() override      {          std::cout << "square 析构函数被调用" << std::endl;      }      double area() override      {          return a \* a;      }      double area(int r)      {          return r \* r;      }  };  int main()  {      std::cout << "\n===== 测试1: 创建各种形状对象并计算面积 =====\n"                << std::endl;      // 创建圆形对象      std::cout << "创建圆形对象:" << std::endl;      circle c1(5.0, 2.0, 3.0); // 半径为5，圆心为(2, 3)      std::cout << "圆形面积: " << c1.area() << std::endl;      c1.print();      std::cout << std::endl;      // 创建矩形对象      std::cout << "创建矩形对象:" << std::endl;      rectangle r1(4.0, 6.0); // 宽为4，高为6      std::cout << "矩形面积: " << r1.area() << std::endl;      r1.print();      std::cout << std::endl;      // 创建正方形对象      std::cout << "创建正方形对象:" << std::endl;      square s1(3.0); // 边长为3      std::cout << "正方形面积: " << s1.area() << std::endl;      s1.print();      std::cout << std::endl;      std::cout << "\n===== 测试2: 观察对象生命周期 =====\n"                << std::endl;      std::cout << "创建局部作用域..." << std::endl;      {          std::cout << "进入局部作用域" << std::endl;          circle localCircle(2.0, 1.0, 1.0); // 半径为2，圆心为(1, 1)          rectangle localRect(2.0, 3.0);     // 宽为2，高为3          square localSquare(4.0);           // 边长为4          std::cout << "即将离开局部作用域" << std::endl;      } // 局部对象在这里销毁      std::cout << "已离开局部作用域，局部对象已被销毁" << std::endl;      std::cout << "\n===== 测试3: 使用基类指针实现多态 =====\n"                << std::endl;      // 使用基类指针数组存储不同类型的对象      shape \*shapes[3];      std::cout << "创建形状数组:" << std::endl;      shapes[0] = new circle(3.0, 0.0, 0.0); // 半径为3，圆心为(0, 0)      shapes[1] = new rectangle(5.0, 4.0);   // 宽为5，高为4      shapes[2] = new square(2.0);           // 边长为2      // 通过基类指针调用虚函数      std::cout << "\n计算不同形状的面积:" << std::endl;      for (int i = 0; i < 3; i++)      {          std::cout << "形状 " << (i + 1) << " 的面积: " << shapes[i]->area() << std::endl;          shapes[i]->print(); // 多态调用      }      // 释放动态分配的对象      std::cout << "\n释放动态创建的对象:" << std::endl;      for (int i = 0; i < 3; i++)      {          delete shapes[i];      }      std::cout << "\n===== 测试4: Point 类的运算符重载 =====\n"                << std::endl;      circle c2; // 默认构造函数      std::cout << "创建圆形对象 c2:" << std::endl;      c2.printXY();         // 打印圆心坐标      c2.centerminuspost(); // 后置--重载      c2.printXY();         // 打印圆心坐标      c2.centerminuspre();  // 前置--重载      c2.printXY();         // 打印圆心坐标      c2.centerpluspost();  // 后置++重载      c2.printXY();         // 打印圆心坐标      c2.centerpluspre();   // 前置++重载      c2.printXY();         // 打印圆心坐标      Point p1(1, 2), p2(3, 4);      std::cout << "初始点 p1: (" << p1.getX() << ", " << p1.getY() << ")" << std::endl;      std::cout << "初始点 p2: (" << p2.getX() << ", " << p2.getY() << ")" << std::endl;      // 测试加法运算符      Point p3 = p1 + p2;      std::cout << "p1 + p2 = (" << p3.getX() << ", " << p3.getY() << ")" << std::endl;      // 测试减法运算符      Point p4 = p1 - p2;      std::cout << "p1 - p2 = (" << p4.getX() << ", " << p4.getY() << ")" << std::endl;      // 测试前置 ++      ++p1;      std::cout << "前置 ++p1: (" << p1.getX() << ", " << p1.getY() << ")" << std::endl;      // 测试后置 ++      p1++;      std::cout << "后置 p1++: (" << p1.getX() << ", " << p1.getY() << ")" << std::endl;      // 测试前置 --      --p2;      std::cout << "前置 --p2: (" << p2.getX() << ", " << p2.getY() << ")" << std::endl;      // 测试后置 --      p2--;      std::cout << "后置 p2--: (" << p2.getX() << ", " << p2.getY() << ")" << std::endl;      std::cout << "\n===== 测试5: 重载方法测试 =====\n"                << std::endl;      // 测试重载的 area 方法      std::cout << "调用重载的 area 方法:" << std::endl;      std::cout << "圆形: 指定半径为4的圆面积 = " << c1.area(4) << std::endl;      std::cout << "矩形: 指定宽3高7的矩形面积 = " << r1.area(3, 7) << std::endl;      std::cout << "正方形: 指定边长5的正方形面积 = " << s1.area(5) << std::endl;      // 测试重载的 print 方法      std::cout << "\n调用重载的 print 方法:" << std::endl;      c1.print();      r1.print("这是一个矩形对象");      s1.print("这是一个正方形对象");      std::cout << "\n===== 程序结束, 剩余对象将被销毁 =====\n"                << std::endl;      return 0;  } |

## 综合1.cpp

## 综合2.cpp

## 综合3.cpp