МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра ВТ

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Алгоритмы и структуры данных»
Тема: Работа с иерархией объектов. Наследование и полиморфизм
Вариант 22

Студентка гр. 2308 _____ Рымарь М.И.

Преподаватель _____ Манирагена В.

Санкт-Петербург
2024 год

Цель работы.

Поработать с иерархией объектов, изучить и применить на практике основные принципы ООП – наследование и полиморфизм.

Задание (вариант 22).

Фигура 19 – прямоугольный треугольник с крестом (отражение – да, поворот – да).

Расположения: 2, 3 (бакенбарды), 14 (шишак).

То есть разработанной фигурой – прямоугольным треугольником с крестом – дополнить картинку в позициях бакенбарды и шишак.

Иерархия классов.

Иерархия классов представлена на рисунке 1.

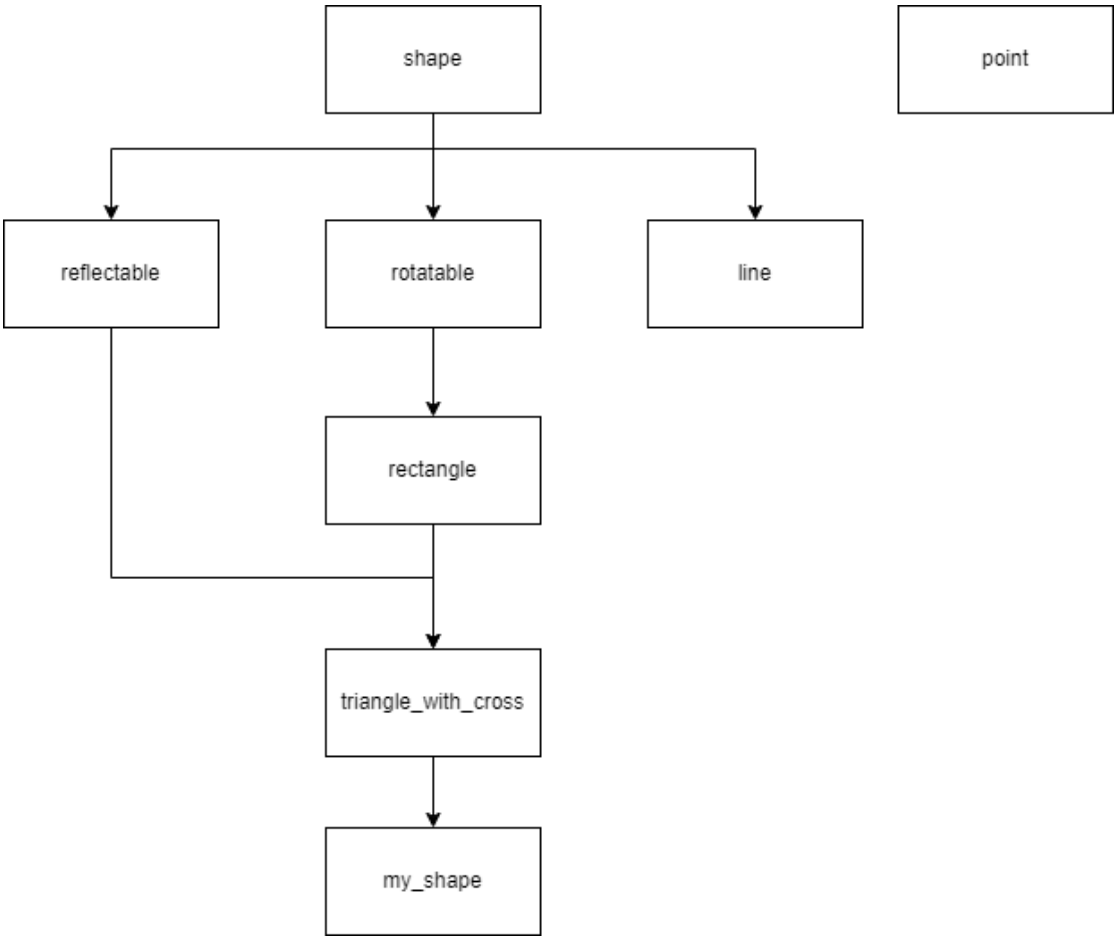

Рисунок 1 – Иерархия классов

Изменения в коде.

1. Был добавлен класс *triangle_with_cross*

Он является наследником классов *rectangle*, *reflectable*, и *rotatable*. В классе *triangle_with_cross* определены методы для изменения размера треугольника (*resize()*), отрисовки треугольника (*draw*), а также для отражения и поворота треугольника.

Конструктор

- Принимает две точки *a* и *b*, которые определяют прямоугольник, в который вписан треугольник.

- Принимает два дополнительных параметра: *re*, определяющий, нужно ли отражать треугольник (по умолчанию *true*), и *ro*, определяющий начальное положение треугольника (по умолчанию *0*).

- Вызывает конструктор базового класса *rectangle* с параметрами *a* и *b*, а также инициализирует переменные *reflected* и *rotate*.

Методы

- *resize(double d)*: Изменяет размер треугольника путем изменения координат его вершин.

- *draw()*: Отрисовывает треугольник и перекрестие внутри него в зависимости от его текущего состояния (поворота и отражения).

- *flip_horisontally()*: Отражает треугольник горизонтально путем изменения флага *reflected*.

- *rotate_left()*: Поворачивает треугольник на 90 градусов влево и устанавливает соответствующее значение переменной *rotate*.

- *rotate_right()*: Поворачивает треугольник на 90 градусов вправо и устанавливает соответствующее значение переменной *rotate*.

2. Метод *draw()* был переопределён, так как способ реализации форма треугольника отличается от предложенного способа реализации формы квадрата.

- Рисует стороны треугольника и перекрестие в зависимости от текущего значения переменной *rotate*.

В методе *draw()* для каждого возможного значения переменной *rotate* выполняются различные действия для отрисовки треугольника.

3. Все функции остались доступными, ограничивать доступ не потребовалось.

Исходный код программы вместе с выделенными дополнениями и изменениями представлен в Приложении А. Итоговый результат работы программы указан на рисунке 2.

На рисунке 3 представлен результат работы программы с реализованным заданием на защиту.

Рисунок 2 – Результат работы программы

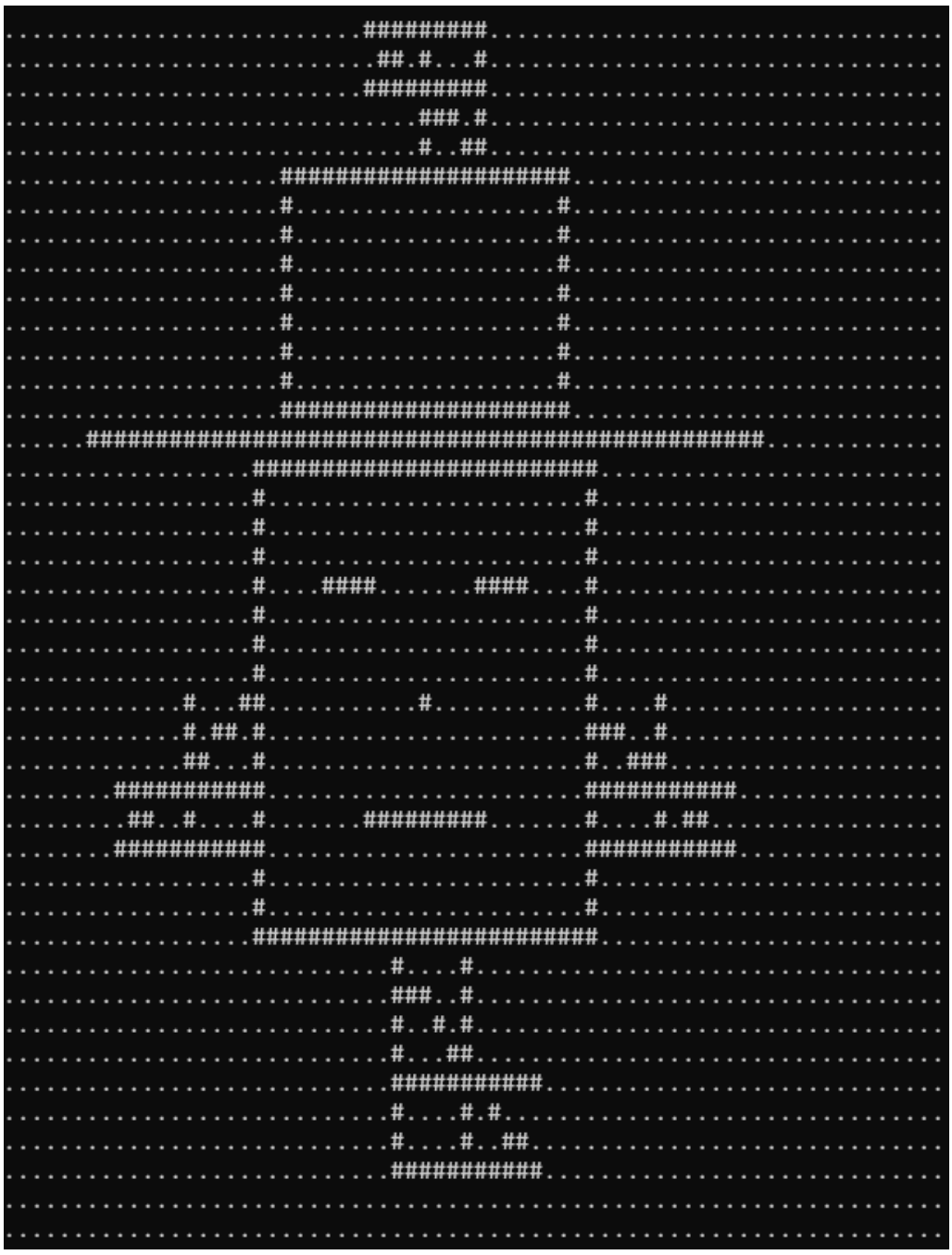

Рисунок 3 – Результат работы программы с добавленным заданием

Контрольные вопросы.

1. Какой базовый класс лучше использовать для производного класса «треугольник»? *rectangle*

2. То же для класса «кружок». *rectangle*

3. То же для класса «крестик». *line*

4. Какой тип наследования стоит выбрать: private, public или protected?

Выбор типа наследования (private, public или protected) зависит от того, как требуется, чтобы члены базового класса были доступны в производном классе и вне него:

1). Public наследование:

- Публичное наследование делает все публичные члены базового класса также публичными в производном классе. Это означает, что они будут доступны как внутри класса-наследника, так и за его пределами.

- Public наследование обеспечивает наиболее прямой доступ к членам базового класса, и оно часто используется, когда требуется полная функциональность базового класса в производном классе.

2). Protected наследование:

- Защищенное наследование делает все публичные и защищенные члены базового класса защищенными в производном классе. Это означает, что они будут доступны только внутри класса-наследника и его производных классов, но не снаружи.

- Protected наследование часто используется, когда требуется, чтобы только классы-наследники имели доступ к членам базового класса, но не пользовательский код.

3). Private наследование:

- Приватное наследование делает все публичные и защищенные члены базового класса приватными в производном классе. Это означает, что они будут доступны только внутри класса-наследника, но не в его производных классах или снаружи.

- Приватное наследование редко используется, поскольку оно ограничивает доступ к членам базового класса только внутри производного класса.

Выбор типа наследования зависит от вашей конкретной ситуации и требований проектирования. В общем, public наследование наиболее распространено и используется тогда, когда нужно предоставить полный доступ к интерфейсу базового класса в производном классе.

5. Можно ли вообще не указывать тип наследования?

В некоторых случаях можно не указывать тип наследования, и компилятор будет использовать тип наследования по умолчанию. Например, если хотите, чтобы класс наследовался публично, можете просто написать class Triangle : public Shape, где Shape является базовым классом. Однако в явном виде указание типа наследования является хорошей практикой для ясности кода.

6. В чём смысл объявления функций в базовом классе как виртуальных?

Смысл объявления функций в базовом классе как виртуальных заключается в том, что это позволяет производным классам переопределять эти функции. Таким образом, виртуальные функции обеспечивают полиморфизм в объектно-ориентированном программировании.

7. Нужно ли объявлять виртуальной функцию в производном классе, если в базовом она уже объявлена таковой?

Нет, не обязательно. Если виртуальная функция уже объявлена в базовом классе, производный класс может переопределить ее без явного объявления как виртуальной. Однако, для ясности кода и предотвращения ошибок рекомендуется явно объявить ее как виртуальную.

8. Можно ли потребовать от компилятора проверить корректность объявления виртуальной функции в производном классе и как это сделать?

Да, можно использовать ключевое слово override в объявлении виртуальной функции в производном классе. Это заставит компилятор проверить, переопределена ли эта функция из базового класса. Если функция не была переопределена, компилятор выдаст ошибку.

9. Можно ли запретить виртуальную функцию в классах-наследниках?

Да, можно запретить виртуальную функцию в производном классе, добавив к ее объявлению ключевое слово final. Это делает функцию завершенной в иерархии наследования, и невозможно переопределить эту функцию в классах-наследниках.

10. Что такое «чисто виртуальная функция»?

«Чисто виртуальная функция» (или абстрактная виртуальная функция) - это виртуальная функция в базовом классе, которая не имеет определения. Она объявляется с помощью добавления = 0 в конце объявления в базовом классе. Класс, содержащий чисто виртуальную функцию, считается абстрактным классом, и он не может быть инстанциирован.

11. Обязательно ли переопределять все функции-члены базового класса в производном классе?

Нет, не обязательно переопределять все функции-члены базового класса в производном классе. Однако это зависит от вашей конкретной ситуации и требований вашего проекта.

Случаи, когда можно переопределить только некоторые функции-члены базового класса:

1). Необходимость только частичной функциональности: Если нужно изменить только часть функциональности, предоставляемой базовым классом, можно переопределить только те функции, которые нужны, не трогая остальные.

2). Простое наследование без изменений: Если производный класс не добавляет никакой новой функциональности и просто наследует функциональность базового класса, то нет необходимости переопределять функции-члены базового класса.

3). Использование виртуальных функций: Если функции в базовом классе являются виртуальными, то переопределение их в производном классе зависит от необходимости изменения их поведения. Можно переопределить только те функции, для которых требуется другое поведение в производном классе.

12. Зачем может понадобиться создание набора (массива или списка) указателей на разные типы объектов в пределах некоторой иерархии?

Создание набора указателей на разные типы объектов в пределах некоторой иерархии может быть полезным во многих ситуациях:

1). Полиморфизм и обобщенное программирование: Возможность работать с объектами различных классов, производных от одного базового

класса, позволяет писать более обобщенный и гибкий код. Можно использовать одинаковые интерфейсы для работы с объектами разных типов, что упрощает код и делает его более легким для поддержки и расширения.

2). Хранение в контейнерах: Использование указателей на базовый класс позволяет хранить объекты разных типов в контейнерах, таких как массивы, списки или векторы. Это особенно полезно, если нужно обрабатывать большое количество объектов и иметь к ним доступ по индексу или итератору.

3). Реализация виртуальных функций: Если есть набор объектов с общим базовым классом и переопределенными виртуальными функциями, то хранение указателей на эти объекты позволяет вызывать соответствующие функции в зависимости от типа объекта, что осуществляет полиморфизм и обеспечивает правильное выполнение кода во время выполнения.

4). Управление памятью: Использование указателей на базовый класс позволяет эффективно управлять памятью и избегать избыточной копировании объектов. Вместо хранения самих объектов в контейнерах хранить указатели на них, что позволяет сократить расход памяти и повысить производительность.

5). Обработка групп объектов: Если есть несколько объектов с общими свойствами, и нужно обработать их как единое целое, использование массива или списка указателей на базовый класс позволяет легко итерироваться по этим объектам и выполнять различные операции над ними.

13. Как запретить для объекта вызов конструктора по умолчанию?

Если нужно запретить вызов конструктора по умолчанию (конструктора без параметров), можно удалить его явным образом, добавив ключевое слово delete. Это предотвратит создание объектов без передачи параметров конструктору.

14. Как запретить вызов конструктора для использования в качестве неявного преобразователя типа?

Для того чтобы запретить неявное преобразование типа с использованием конструктора, можно сделать конструктор явным, добавив ключевое слово

explicit. Добавление ключевого слова explicit к конструктору означает, что этот конструктор больше не может использоваться для неявного преобразования типа. Теперь его нужно вызывать явно, указывая ключевое слово static_cast или явное приведение типа.

15. Каким образом можно установить значение переменных объекта, объявленных с модификаторами const?

Значение переменных объекта с модификаторами const может быть объявлено при их инициализации и не может быть изменено после.

16. Каким образом следует инициализировать объект базового класса в конструкторе производного класса? Всегда ли это нужно делать?

Конструкторы базового класса вызываются автоматически, если у них нет аргумента. Если требуется вызвать конструктор базового класса с аргументом, нужно использовать список инициализации конструктора производного класса. C ++ поддерживает множественное наследование, поэтому к базовому классу следует обращаться по имени.

17. Каким требованиям должны удовлетворять классы, чтобы между ними можно было сформировать отношение «является»?

Должно быть наследование свойств основного класса. Логическая связь между классами.

18. Как установить между двумя классами отношение «содержит»?

Один класс является свойством другого класса.

19. Каким образом на экране появляются глаза и рот, если функция myshape :: draw( ) не содержит кода для выдачи этих элементов фигуры?

Потому что для отрисовывания глаз и рта используется конструктор класса line.

20. Почему данные в классах line и rectangle сделаны protected?

Мы можем использовать переменную в производном классе, но извне базового и производного классов к ней обратиться нельзя.

Выводы.

В ходе выполнения лабораторной работы была рассмотрена иерархия объектов, была реализована дополнительная фигура и внесена в консольную картинку. Были применены принципы ООП.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

shape.cpp:

```
#include "screen.h"
#include "shape.h"
// Сборная пользовательская фигура – физиономия
class myshape : public triangle_with_cross {      // Моя фигура ЯВЛЯЕТСЯ
    int w, h;                                     //        треугольником
    line l_eye;    // левый глаз – моя фигура СОДЕРЖИТ линию
    line r_eye;   // правый глаз
    line mouth;  // рот
public:
    myshape(point, point);
    void draw( );
    void move(int, int);
    void resize(double);
};
myshape :: myshape(point a, point b) : triangle_with_cross(a, b),
     //Инициализация базового класса
                                       w(neast( ).x - swest( ).x + 1), //
Инициализация данных
                                       h(neast( ).y - swest( ).y + 1), //
– строго в порядке объявления!
                                       l_eye(point(swest( ).x + 2, swest(
).y + h * 3 / 4), 2),
                                       r_eye(point(swest( ).x + w - 4,
swest( ).y + h * 3 / 4), 2),
                                       mouth(point(swest( ).x + 2, swest(
).y + h / 4), w - 4)
{ }
void myshape :: draw( )
{
    rectangle :: draw( );      //Контур лица (глаза и нос рисуются сами!)
    int a = (swest( ).x + neast( ).x) / 2;
    int b = (swest( ).y + neast( ).y) / 2;
    put_point(point(a, b));   // Нос – существует только на рисунке!
}
void myshape :: move(int a, int b)
```

```
{
    rectangle :: move(a, b);
    l_eye.move(a, b);
    r_eye.move(a, b);
    mouth.move(a, b);
}
void myshape :: resize(double d) {
    l_eye.resize(d+1);
    r_eye.resize(d+1);
    mouth.resize(d);
    l_eye.move((int)d*2+1, (int)((d-1)* h * 3 / 4));
    r_eye.move((int)d*4+1, (int)((d-1)* h * 3 / 4));
    mouth.move((int)((d-1)*(w-4))+1, (int)((d-1)* h / 4));
    triangle_with_cross :: resize(d);
}

int main( )
{   setlocale(LC_ALL, "Rus");
    screen_init( );
//== 1. Объявление набора фигур ==
    rectangle hat(point(22, 28), point(30, 33));
    line brim(point(14,27),25);
    myshape face(point(20,18), point(32,26));
    triangle_with_cross shishak(point(20, 40), point(30, 45));
    triangle_with_cross l_sideburn(point(60, 40), point(70, 45));
    triangle_with_cross r_sideburn(point(60, 20), point(70, 25));
    shape_refresh( );
    std::cout << "=== Generated... ===\n";
    std::cin.get(); //Смотреть исходный набор
//== 2. Подготовка к сборке ==
    hat.resize(1.5);
    hat.rotate_right( );
    brim.resize(1.5);
    face.resize(1.5);
    shishak.resize(0.8);
    l_sideburn.resize(0.8);
    r_sideburn.resize(0.8);
    shishak.rotate_right();
```

```
    r_sideburn.rotate_left();
    shape_refresh( );
    std::cout << "=== Prepared... ===\n";
    std::cin.get(); //Смотреть результат поворотов/отражений
//== 3. Сборка изображения ==
    up(brim, face);
    up(hat, brim);
    left(l_sideburn, face);
    right(r_sideburn, face);
    up(shishak, hat);
    shape_refresh( );
    std::cout << "=== Ready! ===\n";
    std::cin.get();       //Смотреть результат
    screen_destroy( );
    return 0; }
```

triangle_with_cross:

```
class triangle_with_cross: public rectangle, public reflectable, virtual
public rotatable {
    bool reflected;
    short rotate; //0-ничего 1- повернут вправо 2 - повернут влево на 90
    public:
    triangle_with_cross(point a, point b, bool re=true, short ro = 0) :
rectangle(a, b),

reflected(re),

rotate(ro){ }
    void resize(double d);
    void draw();
    void flip_horisontally() {reflected = !reflected;};
    void flip_vertically() {};
    void rotate_left(){
        rotate = 2;
        rectangle :: rotate_left();
    };
```

```
    void rotate_right(){
        rotate = 1;
        rectangle :: rotate_right();
    };
};
void triangle_with_cross ::resize(double d) {
    ne.x = sw.x + (ne.x - sw.x) * d;
    ne.y = sw.y + (ne.y - sw.y) * d;
}
void triangle_with_cross :: draw()
{
    put_line((sw.x+ne.x)/2, sw.y, (sw.x+ne.x)/2, ne.y);
    put_line(sw.x, (sw.y+ne.y)/2, ne.x, (sw.y+ne.y)/2);
    if(rotate == 0) {
        put_line(sw.x, sw.y, sw.x, ne.y);
        put_line(sw.x,sw.y, ne.x, sw.y);
        put_line(sw.x, ne.y, ne.x, sw.y);
    } else if(rotate == 1){
        put_line(sw.x,ne.y,ne.x,ne.y);
        put_line(ne.x,ne.y,ne.x,sw.y);
        put_line(sw.x,ne.y, ne.x,sw.y);
    } else if(rotate == 2){
        put_line(sw.x, sw.y, ne.x, ne.y);
        put_line(sw.x, sw.y, ne.x, sw.y);
        put_line(ne.x, ne.y, ne.x, sw.y);
    } else if(rotate == 3){
        put_line(sw.x, ne.y, ne.x, ne.y);
        put_line(sw.x, ne.y, ne.x, sw.y);
        put_line(ne.x, ne.y, ne.x, sw.y);
    }
}
```