**ИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

Федеральное государственное бюджетное образовательное учреждение   
высшего образования

**«Сибирский государственный университет науки и технологий   
имени академика М.Ф. Решетнева»**

Институт информатики и телекоммуникаций

Кафедра информатики и вычислительной техники

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ**

Языки программирования

|  |
| --- |
| Наследование |

Руководитель А.В. Проскурин

подпись, дата инициалы, фамилия

Обучающийся БПИ22-02, 221219040 К.В. Трифонов

номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2023 г.

# ЦЕЛЬ РАБОТЫ

Получение практических навыков разработки и отладки программ, c использованием механизма наследования.

# порядок выполнения работы

1. Ознакомиться с общей постановкой задачи.

2. Ознакомится с вариантом задания – соответствует вашему номеру в списке группы (при нехватке заданий вариант задания вычисляется как номер\_в\_списке\_группы - количество\_заданий).

3. Разработать классы согласно варианту задания.

4. Написать и отладить программу на подготовленных наборах тестовых данных.

5. Подготовить отчет по лабораторной работе. Отчет должен включать в себя:

• титульный лист;

• цель лабораторной работы;

• постановку задачи;

• схему наследования классов (UML диаграмма классов);

• текст программы с комментариями;

• демонстрацию работы программы (Снимки экрана при выполнении действий программы с описанием).

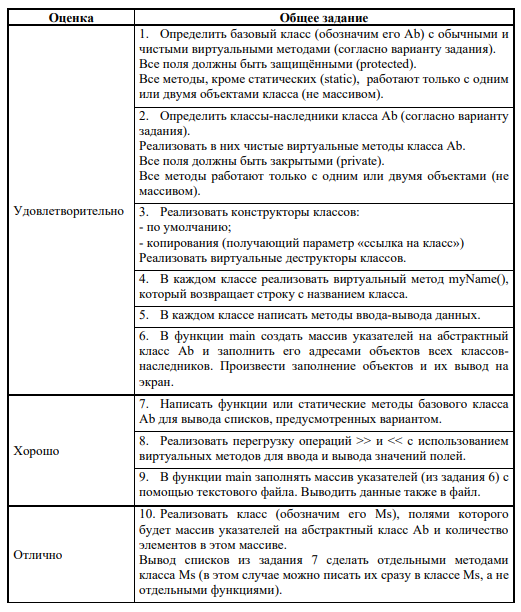
• краткие ответы на контрольные вопросы;

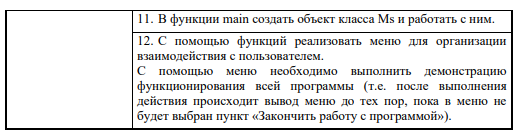
• выводы по лабораторной работе.

6. Защитить лабораторную работу перед преподавателем.

# постановка задачи

Необходимо разработать программу, в которой будет реализовано несколько классов (согласно варианту задания). В реализуемой программе необходимо предусмотреть возможность демонстрации работы конструкторов, деструкторов и методов созданных классов. В зависимости от оценки, на которую вы претендуете, необходимо выполнить следующие задания (Для каждой следующей оценки нужно выполнить ВСЕ предыдущие задания, если обратное не указано явно):





**Вариант №21.**

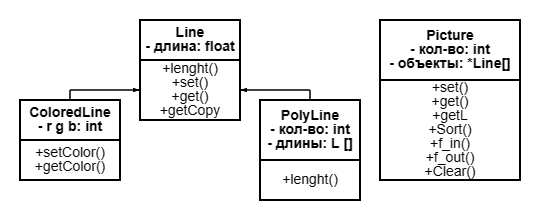
Создать класс Line (Линия) с виртуальным методом нахождения длины линии. На его основе создать классы:

- **ColoredLine** (Цветная линия) с полем «цвет»;

- **PolyLine** (Полилиния) с массивом длин участков ломаной линии (кроме первого). Для массива указателей на объекты этих классов предусмотреть возможность: - вывода характеристик объектов; - вывода объектов, длина которых находится в заданном диапазоне; - вывода массива, отсортированного по длине линий.

Создать класс **Picture**, содержащий массив указателей на объекты этих классов. Написать демонстрационную программу, в которой будут использоваться все методы классов.

# Схема наследования классов



# ХОД РАБОТЫ

**Определение базового класса Line:**

*class* Line{

*protected:*

*float* l;

*public:*

        Line(){}

        Line(*const* *float* *L*):l(*L*){};

        Line(*const* Line *&B*);

*virtual* ~Line(){}

*virtual* *float* length() *const*;

*virtual* string myName() *const*{return "Line";}

*//Ввод-вывод*

*virtual* *void* set(istream*&* *s\_in*);

*virtual* ostream*&* get(ostream*&* *s\_out*) *const*;

*void* operator=(*const* Line *B*);

*bool* operator>(*const* Line *B*)*const*;

*bool* operator<(*const* Line *B*)*const*;

*virtual* Line*\** getCopy() *const*;

};

ostream*&* operator<<(ostream*&* *s\_out*, *const* Line*&* *D*);

istream*&* operator>>(istream*&* *s\_in*, Line*&* *D*);

Описание конструкторов и методов:

*void* Line::set(istream*&* *s\_in*){

    if (&*s\_in* == &cin){

        cout<<"Введите длину линии:\n";

*s\_in*>>l;

    }

    else{

*s\_in*>>l;

    }

}

ostream*&* Line::get(ostream*&* *s\_out*) *const*{

    if (&*s\_out* == &cout){

        cout<<"Длина линии:\n";

*s\_out*<<l<<"\n";

    }

    else{

*s\_out*<<l<<"\n";

    }

    return *s\_out*;

}

*//Операторы*

*void* Line::operator=(*const* Line *B*){

    l = *B*.l;

}

*bool* Line::operator>(*const* Line *B*)*const*{

    return *B*.l > l;

}

*bool* Line::operator<(*const* Line *B*)*const*{

    return *B*.l < l;

}

Line*\** Line::getCopy() *const* {

    return new Line(\**this*);

}

*// Перегрузка ввода в поток*

ostream *&*operator<<(ostream *&s\_out*, *const* Line *&D*){

*D*.get(*s\_out*);

    return *s\_out*;

}

*// Перегрузка вывода из потока*

istream *&*operator>>(istream *&s\_in*, Line *&D*){

*D*.set(*s\_in*);

    return *s\_in*;

}

**Определение производных классов ColoredLine и PolyLine класса:**

Класс цветная линия с дополнительными полями цвета.

*class* ColoredLine : *public* Line{

*int* r,g,b;

*public:*

        ColoredLine(){}

        ColoredLine(*const* *float* *L*) : Line(*L*) {}

        ColoredLine(*const* ColoredLine *&B*);

*virtual* ~ColoredLine(){}

        string myName() *const* *override*  {return "ColoredLine";}

*//ввод-вывод*

*void* setColor();

*void* getColor() *const*;

*void* set(istream*&* *s\_in*) *override*;

        ostream*&* get(ostream*&* *s\_out*) *const* *override*;

*virtual* *void* operator=(*const* ColoredLine *B*) ;

};

Описание методов:

*void* ColoredLine::setColor(){

    cout<<"Укажите цвет r g b:\n";

    cin>>r>>g>>b;

}

*void* ColoredLine::getColor()*const* {

    cout<<r<<" "<<g<<" "<<b<<"\n";

}

ColoredLine::ColoredLine(*const* ColoredLine &*B*){

    l = *B*.l;

    r = *B*.r;

    g = *B*.g;

    b = *B*.

    b;

}

*void* ColoredLine::set(istream*&* *s\_in*) {

    Line::set(*s\_in*);

*s\_in*>>r>>g>>b;

}

ostream*&* ColoredLine::get(ostream*&* *s\_out*) *const*{

    Line::get(*s\_out*);

    if (&*s\_out* == &cout){

*s\_out*<<"Цвет линии r g b: \n"

        << r <<" "<<g<<" "<<b<<"\n";

    }

    else{

*s\_out*<<r<<" "<<g<<" "<<b<<"\n";

    }

    return *s\_out*;

}

*void* ColoredLine::operator=(*const* ColoredLine *B*){

    l = *B*.l;

    r = *B*.r;

    g = *B*.g;

    b = *B*.b;

}

Класс кривой линии с массивом длин.

*class* PolyLine : *public* Line{

*int* n = 0;

*float* \*L;

*public:*

        PolyLine(){}

        PolyLine(*const* *int* *N*):n(*N*){};

        PolyLine(*const* PolyLine *&B*);

*virtual* ~PolyLine(){delete[] L;}

        string myName() *const* *override* {return "PolyLine";}

*//ввод-вывод*

*float* length() *const* *override* {

*float* LM = 0;

            for (*int* i = 0; i<n; i++){

                LM+=L[i];

            }

            LM += Line::length();

            return LM;

        }

*void* set(istream*&* *s\_in*) *override*;

        ostream*&* get(ostream*&* *s\_out*) *const* *override*;

*void* operator=(*const* PolyLine *B*);

};

Описание конструкторов и методов:

PolyLine::PolyLine(*const* PolyLine &*B*){

    l = *B*.l;

    n = *B*.n;

    L = new *float*[n];

    for (*int* i=0; i<n; i++){

        L[i] = *B*.L[i];

    }

}

*void* PolyLine::set(istream*&* *s\_in*){

    Line::set(*s\_in*);

    if (n>0){

        L = new *float* [n];

        for (*int* i = 0; i<n; i++){

            if (&*s\_in* == &cin){

                cout<<"Укажите длину "<<i+1<<" линии: ";

*s\_in*>>L[i];

            }

            else{

*s\_in*>>L[i];

            }

        }

    }

}

ostream*&* PolyLine::get(ostream*&* *s\_out*) *const*{

    Line::get(*s\_out*);

    if (n>0){

        for (*int* i = 0; i<n; i++){

            if (&*s\_out* == &cout){

                cout<<"Длина "<<i+1<<" линии: ";

*s\_out*<<L[i]<<"\n";

            }

            else{

*s\_out*<<L[i]<<"\n";

            }

        }

    }

    return *s\_out*;

}

*void* PolyLine::operator=(*const* PolyLine *B*){

    l = *B*.l;

    n = *B*.n;

    L = new *float*[n];

    for (*int* i=0; i<n; i++){

        L[i] = *B*.L[i];

    }

}

**Определение класса Picture, содержащего массив указателей на объекты типа Line:**

*class* Picture{

*int* size = 0;

        Line \*\*objects = nullptr;

*public:*

        Picture(){}

        Picture(*const* *int* *S*): size(*S*), objects(new Line \* [size]){};

        Picture(*const* Picture *&B*);

        ~Picture(){Clear();};

*//Ввод-вывод*

        istream*&* set(istream*&* *s\_in*);

        ostream*&* get(ostream*&* *s\_out*) *const*;

*void* getL(*const* *float* *l1*, *const* *float* *l2*);

*void* Sort();

*void* f\_in(string *filename*);

*void* f\_out(string *filename*) *const*;

*void* Clear();

};

Описание конструкторов и методов:

Picture::Picture(*const* Picture &*B*){

    size = *B*.size;

    objects = new Line\*[size];

    for (*int* i = 0; i < size; i++){

        objects[i] = *B*.objects[i]->getCopy();

    }

}

При помощи метода **getCopy** класса Line происходит заполнение целевого массива целевого объекта.

*void* Picture::Clear(){

    for (*int* i = 0; i < size; i++){

        delete objects[i];

        cout<<"!\n";

    }

    delete[] objects;

    size = 0;

    objects = nullptr;

}

Метод чистки динамической памяти и обнуления полей.

istream*&* Picture::set(istream*&* *s\_in*){

    Clear();

    if (&*s\_in* == &cin){

        if (size == 0){

            cout<<"Укажите кол-во элементов:\n";

            cin>>size;

            objects = new Line\*[size];

        }

        for (*int* i = 0; i < size; i++){

            cout<<"Линия "<<i<<":\n";

            cout<<"Укажите тип:\n";

            string type;

*s\_in* >> type;

            if (type == "Line"){

                objects[i] = new Line();

            }

            else if (type == "ColoredLine"){

                objects[i] = new ColoredLine();

            }

            else if (type == "PolyLine"){

                objects[i] = new PolyLine();

            }

            objects[i]->set(*s\_in*);

        }

    }

    else{

*s\_in*>>size;

        objects = new Line\*[size];

        for (*int* i = 0; i < size; i++){

            string type;

*s\_in* >> type;

            if (type == "Line"){

                objects[i] = new Line;

            }

            else if (type == "ColoredLine"){

                objects[i] = new ColoredLine;

            }

            else if (type == "PolyLine"){

                objects[i] = new PolyLine;

            }

            objects[i]->set(*s\_in*);

        }

    }

    return *s\_in*;

}

Спрашивается у пользователя (или берётся сразу ответ из потока) тип объекта указатель которого будет храниться в ячейке, затем происходит заполнение через виртуальный метод set этого класса.

ostream*&* Picture::get(ostream*&* *s\_out*)*const* {

    if (&*s\_out* == &cout){

        for (*int* i = 0; i<size; i++){

            cout<<"Линия "<<i<<" типа "<<objects[i]->myName()<<":\n";

            objects[i]->get(*s\_out*);

        }

    }

    else{

*s\_out*<<size<<"\n";

        for (*int* i = 0; i<size; i++){

*s\_out*<<objects[i]->myName()<<"\n";

            objects[i]->get(*s\_out*);

        }

    }

    return *s\_out*;

}

Выводится в поток (с форматированием для cout) название класса методом myName и его содержимое через get.

*void* Picture::getL(*const* *float* *l1*, *const* *float* *l2*){

    cout<<"Удовлетворяют условию:\n";

*int* c = 0;

    for (*int* i = 0; i < size; i++)

        {

            if (objects[i]->length()>*l1* && objects[i]->length()<*l2*){

                cout << "Линия № " << i + 1 << ":\n";

                objects[i]->get(cout);

                c++;

            }

        }

    cout<<"Всего: "<< c <<"\n";

}

Метод выборки по длине.

*//Работа с файлами*

*void* Picture::f\_in(string *filename*){

    Clear();

    ifstream fin(*filename*);

    set(fin);

    cout<<"Запись завершена.\n";

}

*void* Picture::f\_out(string *filename*) *const* {

    ofstream fout(*filename*);

    get(fout);

    cout<<"Запись завершена.\n";

}

Работа с файловым потоком.

*void* Picture::Sort(){

    for (*int* i = 0; i < size; i++)

    {

*bool* flag = true;

        for (*int* j = 0; j < size - 1; j++)

        {

            if (\*objects[j] < \*objects[j + 1])

            {

                flag = false;

                swap(objects[j],objects[j+1]);

            }

        }

        if (flag)

        {

            break;

        }

    }

}

Пузырьковая сортировка по полю длины.

# Тестирование

Для тестирования программы было создано меню:

while (f){

        cout<<"\*Главное меню\*\n"

        <<"1) Работа с объектом\n"

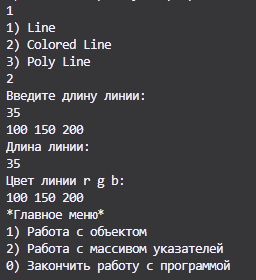
        <<"2) Работа с массивом указателей\n"

        <<"0) Закончить работу с программой\n";

        cin>>sw;

        switch (sw){

Создание одного объекта



Работа с массивом указателей:

                while (f2){

                    cout

                    <<"1) Заполнить массив указателей\n"

                    <<"2) Вывести массив указателей\n"

                    <<"3) Отсортировать по возрастанию\n"

                    <<"4) Вывести в диапазоне\n"

                    <<"5) Заполнить из файла\n"

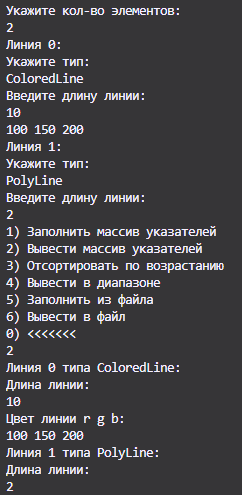
                    <<"6) Вывести в файл\n"

                    <<"0) <<<<<<<\n";

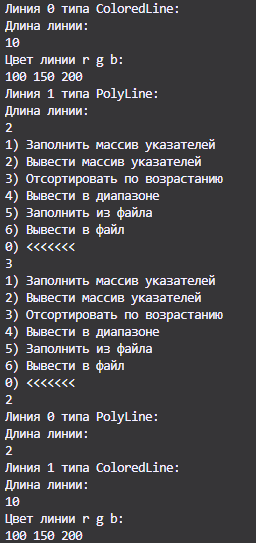
                    cin>>sw3;

                    switch (sw3){

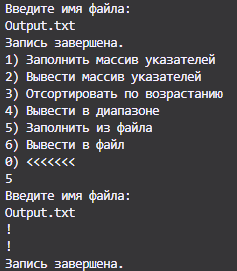
Заполнение и вывод массива указателей длинной 2:



Сортировка массива



Чтение и запись из файла



# ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ

1. **Сущность механизма наследования в ООП:**

Механизм наследования в объектно-ориентированном программировании (ООП) представляет собой принцип, позволяющий новому классу (производному) использовать свойства и методы уже существующего класса (базового). Это способствует повторному использованию кода и созданию иерархий классов.

1. **Объявление производного класса:**

В C++ объявление производного класса выглядит так:

*class* ColoredLine : *public* Line{

Дочерний класс цветная линия наследуется от родительского класса линия.

1. **Доступ к элементам базового класса:**

Из производного класса можно получить доступ к элементам базового класса с использованием оператора **::**:

LM += Line::length()

1. **Наследование производного класса:**

Производный класс наследует от базового класса его члены (поля и методы), за исключением приватных членов.

1. **Ненаследуемые элементы производного класса:**

Производный класс не наследует приватные члены базового класса.

1. **Инициализация наследуемых элементов:**

Инициализация наследуемых элементов базового класса обычно выполняется через конструктор производного класса.

1. **Множественное наследование:**

Множественное наследование позволяет классу наследовать от нескольких базовых классов. Конструкторы базовых классов вызываются в порядке, указанном при объявлении производного класса.

1. **Порядок вызова конструкторов и деструкторов:**

При создании объекта производного класса вызываются конструкторы базовых классов в порядке наследования. При удалении объекта вызываются деструкторы в обратном порядке.

1. **Принцип виртуальных функций:**

Механизм виртуальных функций реализует принцип полиморфизма, позволяя вызывать методы производного класса через указатель или ссылку на базовый класс.

1. **Использование виртуальных функций:**

Виртуальные функции следует использовать, когда необходима динамическая диспетчеризация и возможность переопределения методов в производных классах.

1. **Динамическое связывание:**

Суть метода динамического связывания заключается в том, что вызов метода происходит во время выполнения программы, что позволяет использовать полиморфизм.

1. **Идея "один интерфейс – множество методов":**

Эта идея подразумевает, что интерфейс (абстрактный класс) определяет набор методов, которые должны быть реализованы в производных классах, обеспечивая единообразный интерфейс для работы с различными классами.

1. **Игнорирование виртуальных функций:**

Механизм виртуальных функций может быть проигнорирован, если в производном классе не предусмотрено переопределение виртуальных методов.

1. **Абстрактный класс:**

Абстрактный класс содержит чисто виртуальные функции и не может быть инстанциирован. Его задача – предоставить интерфейс для производных классов. Объекты абстрактных классов создавать нельзя.

# ВЫВОД

В ходе лабораторной работы был изучен и применен механизм наследования и виртуальных функций в объектно-ориентированном программировании. Полученные навыки позволяют эффективно использовать принципы ООП для создания иерархий классов и обеспечения гибкости программной архитектуры.