Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное учреждение высшего образования  
«Пермский национальный исследовательский политехнический университет»

Электротехнический факультет  
Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т  
по лабораторной работе**

Дисциплина: «Основы алгоритмизации и программирования»

Выполнил  
студент группы ИВТ-21-1б  
Мифтахов И.Г

Проверил  
ст. п. кафедры ИТАС  
Яруллин Д.В.

Пермь 2022

Постановка задачи:

1. Определить абстрактный класс.
2. Определить иерархию классов, в основе которой будет находиться абстрактный класс (см. лабораторную работу №4).
3. Определить класс Вектор, элементами которого будут указатели на объекты иерархии классов.
4. Перегрузить для класса Вектор операцию вывода объектов с помощью потоков.
5. В основной функции продемонстрировать перегруженные операции и полиморфизм Вектора.

Анализ задачи:

Определен абстрактный класс abstract, имеющий чистую виртуальную ф-ию show(). Пользовательский класс-наследник – triad, три числа типа int, которые вводятся при помощи потокового ввода, выведены аналогичным методом. Пользовательский класс-наследник – date, который считает увеличение даты на n дней и может вывести значение как при помощи потокового вывода, так и при помощи виртуальной функции базового класса(triad тоже так может). Объекты обоих классов-наследников записаны в отдельный пользовательский класс Vector и выведены на экран с помощью потокового вывода.

Ответы на вопросы:

1. Чисто виртуальному присваивается значение 0. Класс становится абстрактным.
2. Абстрактный класс – класс, содержащий в себе хотя бы 1 чистую виртуальную ф-ию.
3. Для дальнейшей реализации указанных методов в классах-наследниках.
4. Способность функции обрабатывать разные типы данных.
5. При полиморфизме ф-ия может изменять свой функционал.

class Figure

{

public:

    virtual double getSquare() =0;

    virtual double getPerimeter() =0;

    virtual void showFigureType()=0;

};

class Rectangle : public Figure

{

private:

    double width;

    double height;

public:

    Rectangle(double w, double h) : width(w), height(h)

    {

    }

    double getSquare() override

    {

        return width \* height;

    }

    double getPerimeter() override

    {

        return width \* 2 + height \* 2;

    }

    void showFigureType()

    {

        std::cout << "Rectangle" << std::endl;

    }

};

class Circle : public Figure

{

private:

    double radius;

public:

    Circle(double r) : radius(r)

    {

    }

    double getSquare() override

    {

        return radius \* radius \* 3.14;

    }

    double getPerimeter() override

    {

        return 2 \* 3.14 \* radius;

    }

    void showFigureType()

    {

        std::cout << "Circle" << std::endl;

    }

};

int main()

{

    Rectangle rect(30, 50);

    Circle circle(30);

    std::cout << "Rectangle square: " << rect.getSquare() << std::endl;

    std::cout << "Circle square: " << circle.getSquare() << std::endl;

    return 0;

}

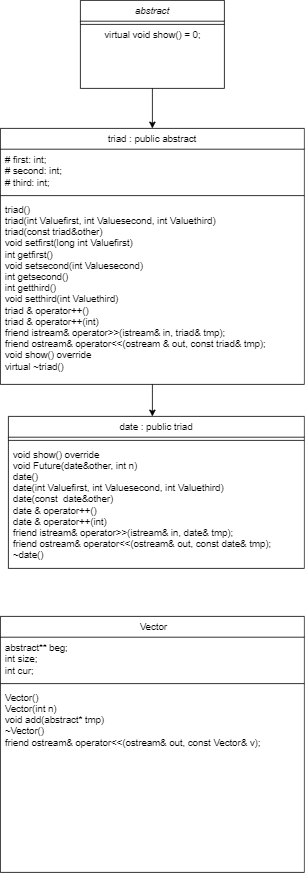
1. class Abstract{ //Абстрактный класс

public:virtual void print\_msg()=0;

};

1. Позднее связывание означает, что объект связывается с вызовом функции только во время ис­полнения программы, а не раньше. Позднее связывание достигается в С++ с помощью использо­вания виртуальных функций и производных классов.

Блок-схема:



Код программы:

#include<iostream>

using namespace std;

class abstract

{

public:

virtual void show() = 0;

};

class triad : public abstract

{

protected:

int first;

int second;

int third;

public:

triad()

{

first = 0;

second = 0;

third = 0;

}

triad(int Valuefirst, int Valuesecond, int Valuethird)

{

first = Valuefirst;

second = Valuesecond;

third = Valuethird;

}

triad(const triad&other)

{

this->first = other.first;

this->second = other.second;

this->third = other.third;

}

void setfirst(long int Valuefirst)

{ first = Valuefirst;}

int getfirst()

{ return first;}

void setsecond(int Valuesecond)

{ second = Valuesecond;}

int getsecond()

{ return second;}

int getthird()

{ return third;}

void setthird(int Valuethird)

{ third = Valuethird;}

triad & operator++()

{

this->first += 1;

this->second += 1;

this->third += 1;

return \*this;

}

triad & operator++(int)

{

this->first += 1;

this->second += 1;

this->third += 1;

return \*this;

}

friend istream& operator>>(istream& in, triad& tmp);

friend ostream& operator<<(ostream & out, const triad& tmp);

void show() override

{

cout << " Числа: " << first << "." << second << "." << third << endl;

}

virtual ~triad()

{

cout << "Вызван деструктор для обьекта " << this << endl;

}

};

istream& operator>>(istream& in, triad& tmp)

{

cout << "Введите первое число: "; in >> tmp.first;

cout << "Введите второе число: "; in >> tmp.second;

cout << "Введите третье число: "; in >> tmp.third;

return in;

}

ostream& operator<<(ostream& out, const triad& tmp)

{

return (out << "Числа: " << tmp.first << "." << tmp.second << "." << tmp.third << endl);

}

class date : public triad

{

public:

void show() override

{

cout << " Числа: " << first << "." << second << "." << third << endl;

}

void Future(date&other, int n)

{

int day; int month; int year; int countM = 0; int countY = 0;

day = other.first + n;

while (day > 31)

{

day = day - 31;

countM++;

}

month = other.second + countM;

while (month > 12)

{

month = month - 12;

countY++;

}

year = other.third + countY;;

other.first = day; other.second = month; other.third = year;

}

date()

{

first = 0;

second = 0;

third = 0;

}

date(int Valuefirst, int Valuesecond, int Valuethird)

{

first = Valuefirst;

second = Valuesecond;

third = Valuethird;

}

date(const date&other)

{

this->first = other.first;

this->second = other.second;

this->third = other.third;

}

date & operator++()

{

this->first += 1;

this->second += 1;

this->third += 1;

return \*this;

}

date & operator++(int)

{

this->first += 1;

this->second += 1;

this->third += 1;

return \*this;

}

friend istream& operator>>(istream& in, date& tmp);

friend ostream& operator<<(ostream& out, const date& tmp);

~date()

{

cout << "Удалили date..." << endl << endl;

}

};

istream& operator>>(istream& in, date& tmp)

{

cout << "Введите дни: "; in >> tmp.first;

cout << "Введите месяцы: "; in >> tmp.second;

cout << "Введите годы: "; in >> tmp.third;

return in;

}

ostream& operator<<(ostream& out, const date& tmp)

{

return (out << "Дата: " << tmp.first << "." << tmp.second<<"."<<tmp.third<<endl);

}

class Vector

{

private:

abstract\*\* beg;

int size;

int cur;

public:

Vector()

{

beg = 0;

size = 0;

cur = 0;

}

Vector(int n)

{

beg = new abstract \*[n];

cur = 0;

size = n;

}

void add(abstract\* tmp)

{

if (cur < size)

{

beg[cur] = tmp;

cur++;

}

}

~Vector()

{

if (beg != 0)

{

delete[] beg;

}

beg = 0;

}

friend ostream& operator<<(ostream& out, const Vector& v);

};

ostream& operator<<(ostream& out, const Vector& v)

{

cout << "Вектор:" << endl;

if (v.size == 0)

{

out << "Empty" << endl;

}

abstract\*\* tmp = v.beg;

for (int i = 0; i < v.cur; i++)

{

(\*tmp)->show();

tmp++;

}

return out;

}

int main()

{

setlocale(LC\_ALL, "Rus");

Vector v(2);

triad a;

cin >> a;

cout << a;

date b;

cin >> b;

cout << b;

abstract\* tmp = &a;

v.add(tmp);

tmp = &b;

b.Future(b, 1500);

tmp->show();

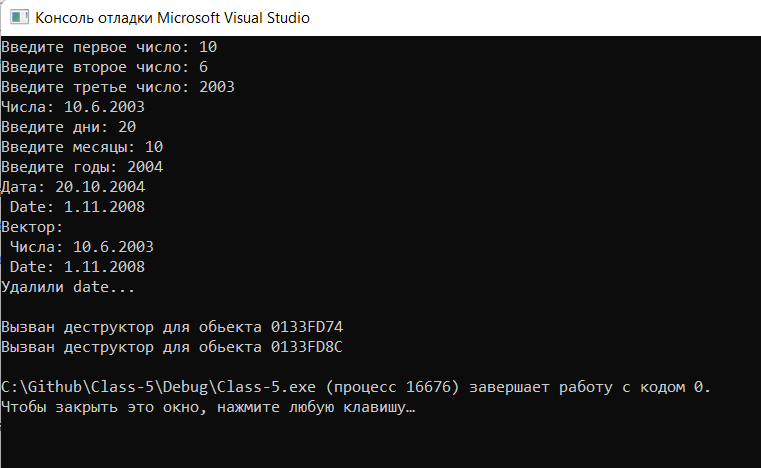
v.add(tmp);

cout << v;

return 0;

}

Скриншоты результатов:



Анализ результатов:

Программа работает корректно. Выводятся введенные числа при помощи потокового вывода, дата увеличивается на n дней для второго обьекта-наследника.