Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе №18.5**

Дисциплина: основы алгоритмизации и программирования

Тема: “Классы и объекты. Полиморфизм.”

Вариант 10

Выполнил:

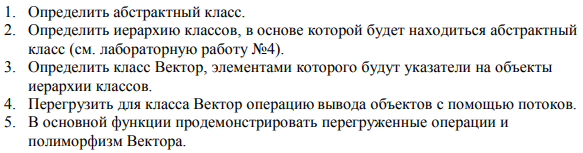
студент группы ИВТ-20-2Б Галинов О.Ю.

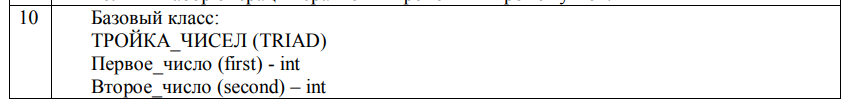
Проверила: доцент кафедры ИТАС

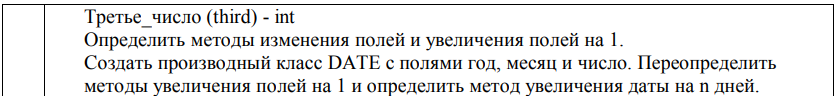
Полякова О.А.

Пермь, 2021

**Постановка задачи**







**Анализ задачи**

**1.** Для решения задачи необходимо:

**1.1.** Организовать класс Object, который будет содержать чисто виртуальную функцию PrintInf() для вывода объектов разных классов на консоль.

**1.2.** Организовать класс Triad для хранения первого числа (first), второго числа (second), третьего числа (third).

**1.3.** Организовать класс Date с полями типа int day, month, year, который является производным класса Triad.

**1.4.** Организовать класс Vector с полями Object\*\* startpos, \_size.

**1.5.** Организовать необходимые методы для ввода данных в поля first и second: гетторы, сетторы, конструкторы, деструктор.

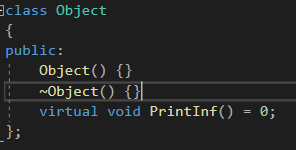
**1.6.** Организовать перегрузку оператора >> дружественной классу Triad.

**1.7.** Организовать перегрузку оператора << дружественной классу Vector.

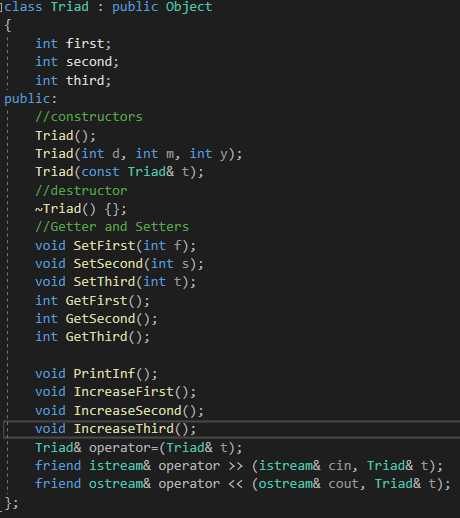
**1.8.** Организовать перегрузку оператора << дружественной классу Triad.

**2.** В ходе работы были использованы следующие типы данных:

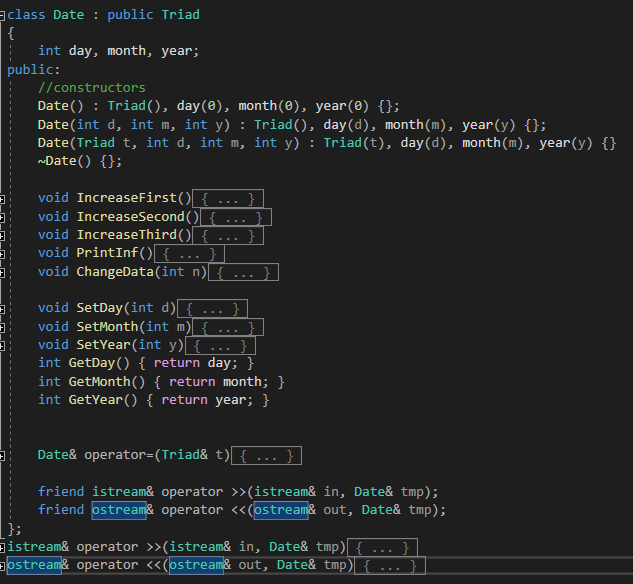
**2.1.** Класс Object, который будет содержать чисто виртуальную функцию PrintInf() для вывода объектов разных классов на консоль, в заголовочном файле Object.h.



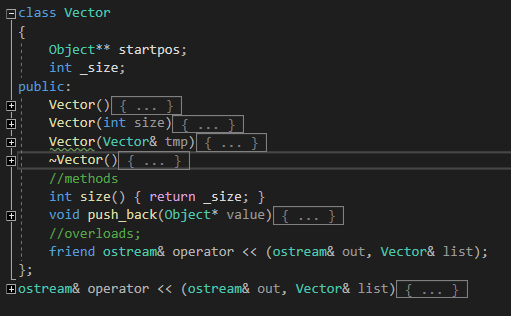
**2.2.** Класс Triad, который является производным классу Object, для хранения первого числа (first), второго числа (second), третьего числа (third) в заголовочном файле Triad.h.



**2.3.** Класс Date с полями типа int date, month, year который является производным класса Triad в заголовочном файле Date.h.



**2.4.** Класс Vector с полями Object\*\* startpos, \_size.



**2.5.** Объект A класса Triad. Объект B принадлежит классу Date.

**2.6.** Указатель pointer1типа Object, который будет указывать на объекты разных классов.

**2.7.** Объект list типа Vector – массив объектов разных классов.

**3.** Для решения задачи данные были представлены в следующем виде:

**3.1.** Объект A класса Triad. Объект B принадлежит классу Date.

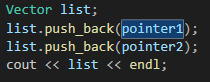
Triad A;

Date B;

**3.2.** Указатель pointer1типа Object, который будет указывать на объекты разных классов.

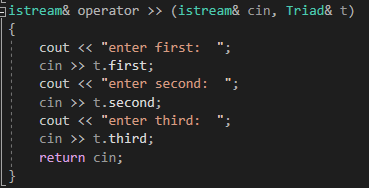


**3.3.** Объект list типа Vector – массив объектов разных классов.

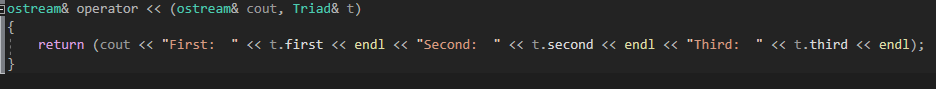


**4.** Для операций ввода и вывода использовались следующие операторы и функции:

**4.1.** Ввод чисел происходит при помощи перегрузки оператора >>.

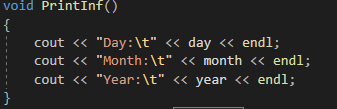


**4.2.** Так же вывод на консоль может быть осуществлён с помощью перегрузки оператора <<.

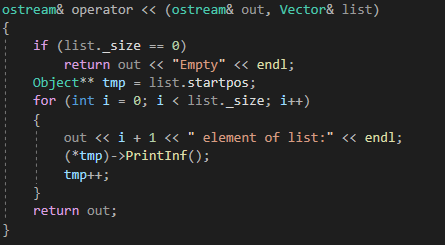
****

**4.3.** Кроме того, вывод введённых чисел, вычисленных в классе Date осуществляется через метод PrintInf() класса Object по принципу полиморфизма.



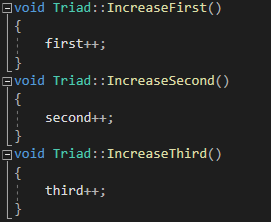


**4.4.** Элементы класса Vector выводятся через перегрузку оператора <<

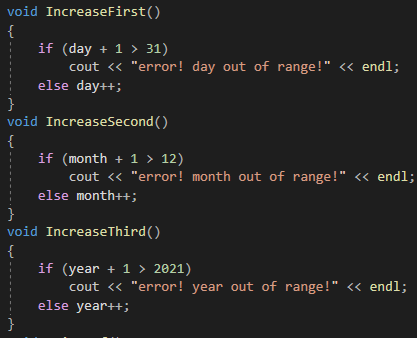


**5.** Поставленные задачи будут решены следующими действиями:

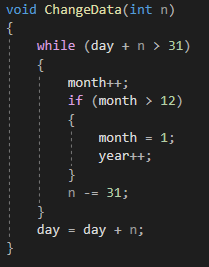
**5.1.** В методах IncreaseFirst, IncreaseSecond, IncreaseThird описанных в классе Triad происходит увеличение всех полей класса на единицу.



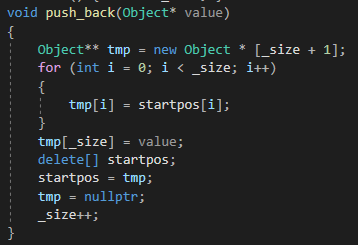
**5.2.** В этих же методах класса Date также происходит увеличение полей, но с проверками.



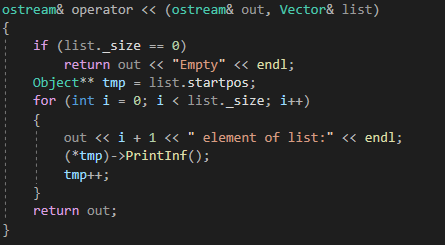
**5.3.** В методе ChangeData выполняется изменение даты на n-дней (может быть и отрицательным).



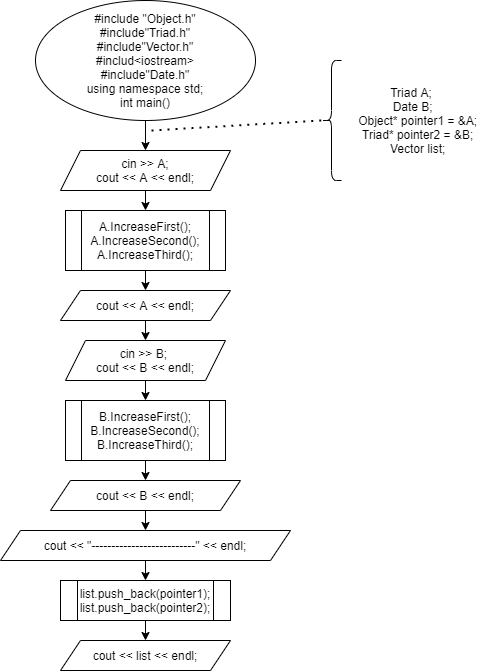
**5.4.** Создается новый элемент класса Vector, в который записываются все элементы из старого, а затем добавляется новый, который передается как параметр.

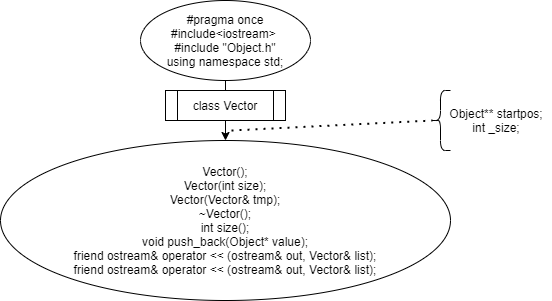


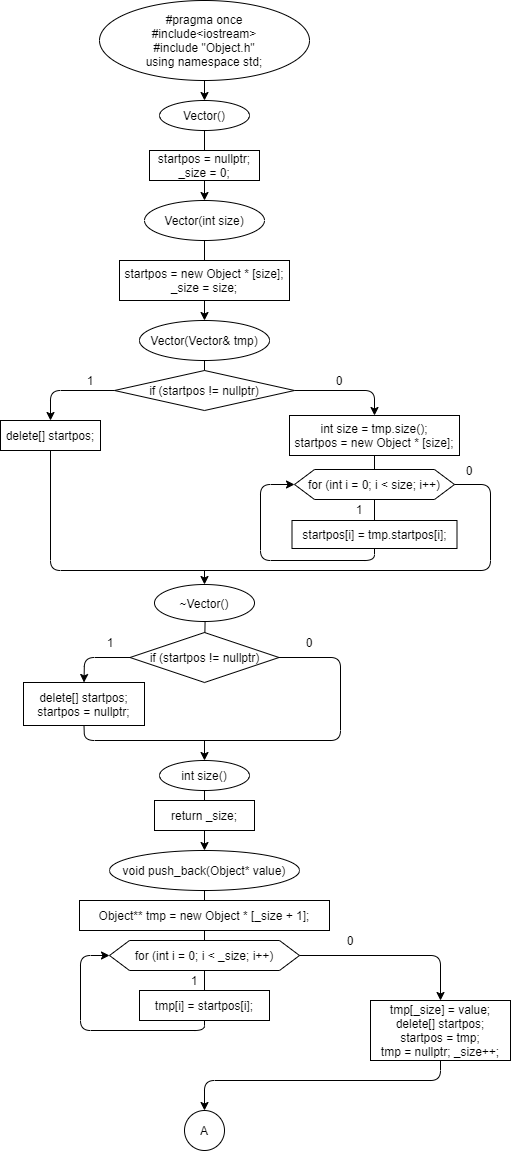
**5.5.** В перегрузке оператора << происходит итерирование по циклу for от 0 до количества элементов в векторе. В цикле происходит вывод объектов через указатель и сдвиг к следующему элементу через инкремент.

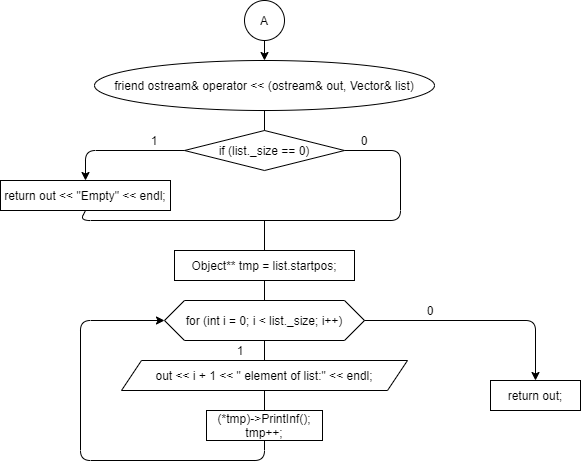


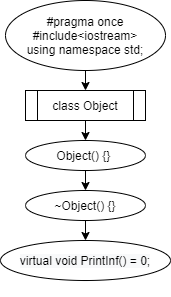
**Блок-схема**

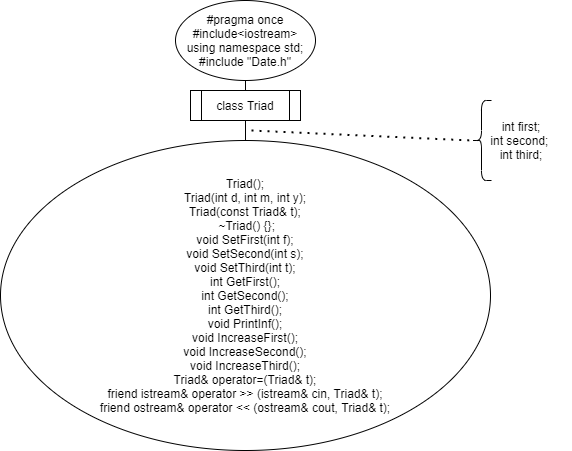


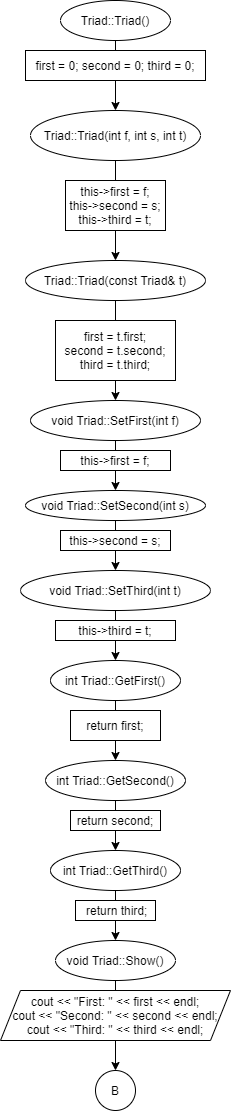


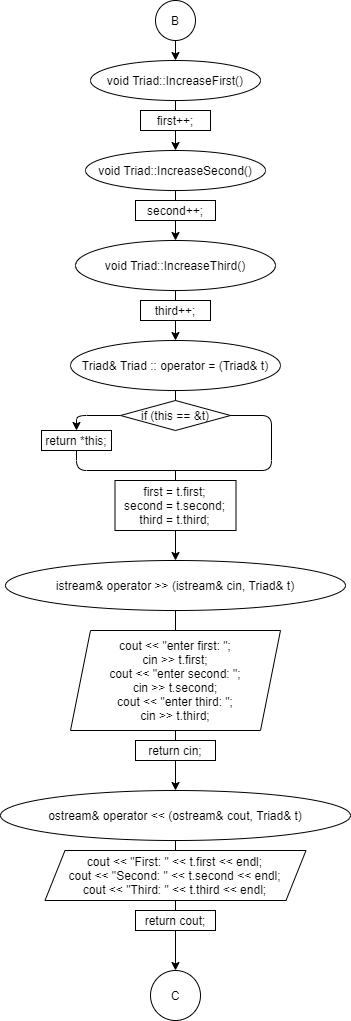


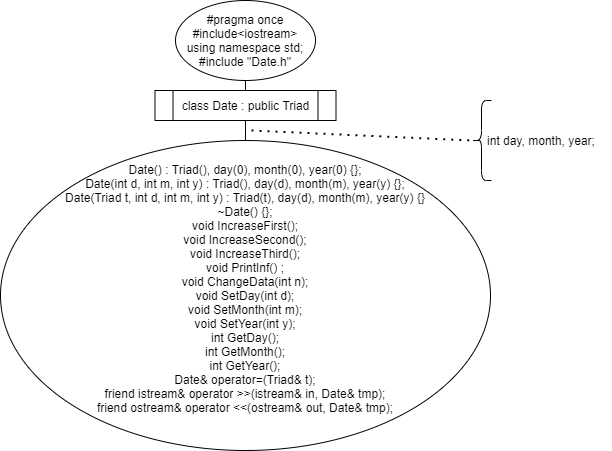


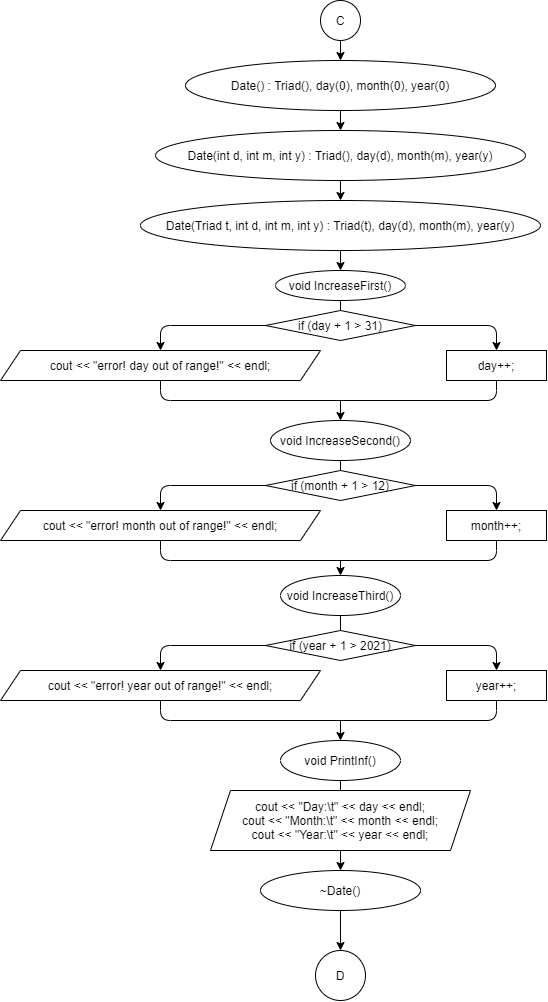


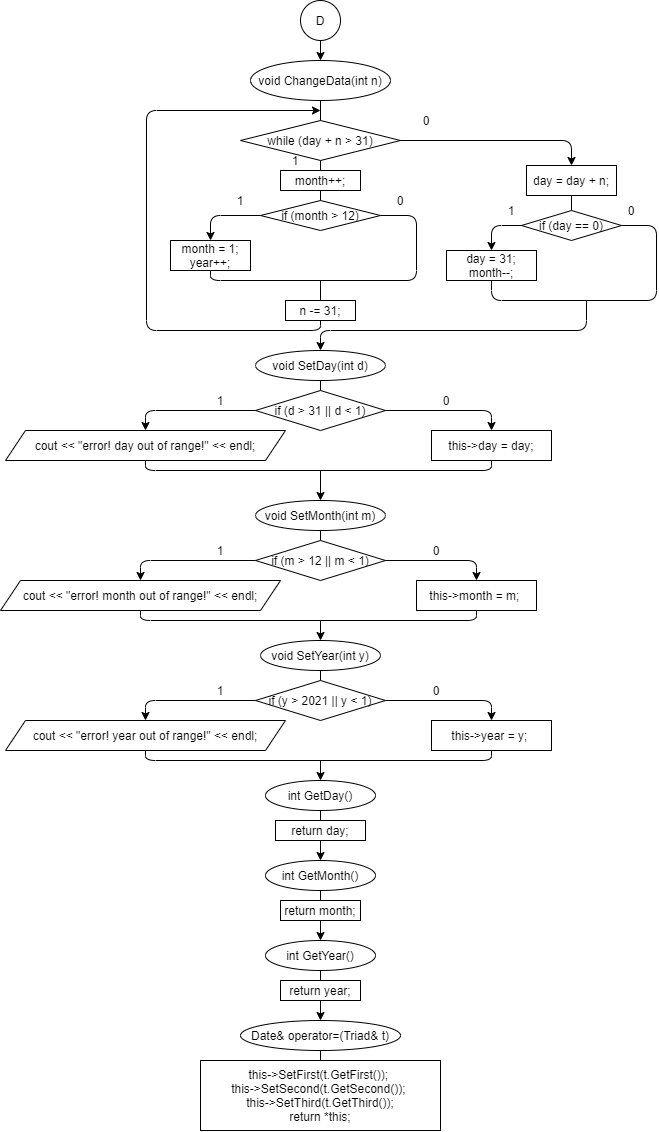












**Код**

#include "Object.h"

#include"Triad.h"

#include"Vector.h"

#include <iostream>

#include"Date.h"

using namespace std;

int main()

{

Triad A;

cin >> A;

cout << A << endl;

A.IncreaseFirst();

A.IncreaseSecond();

A.IncreaseThird();

cout << A << endl;

Date B;

cin >> B;

cout << B << endl;

B.IncreaseFirst();

B.IncreaseSecond();

B.IncreaseThird();

cout << B << endl;

cout << "--------------------------" << endl;

Object\* pointer1 = &A;

Triad\* pointer2 = &B;

Vector list;

list.push\_back(pointer1);

list.push\_back(pointer2);

cout << list << endl;

}

#pragma once

#include <iostream>

#include "Object.h"

using namespace std;

class Vector

{

Object\*\* startpos;

int \_size;

public:

Vector()

{

startpos = nullptr;

\_size = 0;

}

Vector(int size)

{

startpos = new Object \* [size];

\_size = size;

}

Vector(Vector& tmp)

{

if (startpos != nullptr)

delete[] startpos;

else

{

int size = tmp.size();

startpos = new Object \* [size];

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

{

startpos[i] = tmp.startpos[i];

}

}

}

~Vector()

{

if (startpos != nullptr)

{

delete[] startpos;

startpos = nullptr;

}

}

//methods

int size() { return \_size; }

void push\_back(Object\* value)

{

Object\*\* tmp = new Object \* [\_size + 1];

for (int i = 0; i < \_size; i++)

{

tmp[i] = startpos[i];

}

tmp[\_size] = value;

delete[] startpos;

startpos = tmp;

tmp = nullptr;

\_size++;

}

//overloads;

friend ostream& operator << (ostream& out, Vector& list)

{

if (list.\_size == 0)

return out << "Empty" << endl;

Object\*\* tmp = list.startpos;

for (int i = 0; i < list.\_size; i++)

{

out << i + 1 << " element of list:" << endl;

(\*tmp)->PrintInf();

tmp++;

}

return out;

}

};

#pragma once

#include<iostream>

using namespace std;

class Object

{

public:

Object() {}

~Object() {}

virtual void PrintInf() = 0;

};

#pragma once

#include "Object.h"

#include <iostream>

using namespace std;

class Triad : public Object

{

int first;

int second;

int third;

public:

//constructors

Triad();

Triad(int d, int m, int y);

Triad(const Triad& t);

//destructor

~Triad() {};

//Getter and Setters

void SetFirst(int f);

void SetSecond(int s);

void SetThird(int t);

int GetFirst();

int GetSecond();

int GetThird();

void PrintInf();

void IncreaseFirst();

void IncreaseSecond();

void IncreaseThird();

Triad& operator=(Triad& t);

friend istream& operator >> (istream& cin, Triad& t);

friend ostream& operator << (ostream& cout, Triad& t);

};

#include "Triad.h"

#include <iostream>

using namespace std;

Triad::Triad() { first = 0; second = 0; third = 0; }

Triad::Triad(int f, int s, int t) {

this->first = f;

this->second = s;

this->third = t;

}

Triad::Triad(const Triad& t)

{

first = t.first;

second = t.second;

third = t.third;

}

void Triad::SetFirst(int f)

{ this->first = f; }

void Triad::SetSecond(int s)

{ this->second = s; }

void Triad::SetThird(int t)

{ this->third = t; }

int Triad::GetFirst()

{ return first; }

int Triad::GetSecond()

{ return second; }

int Triad::GetThird()

{ return third; }

//methods

void Triad::PrintInf()

{

cout << "First: " << first << endl;

cout << "Second: " << second << endl;

cout << "Third: " << third << endl;

}

void Triad::IncreaseFirst()

{

first++;

}

void Triad::IncreaseSecond()

{

second++;

}

void Triad::IncreaseThird()

{

third++;

}

Triad& Triad :: operator = (Triad& t)

{

if (this == &t)

return \*this;

first = t.first;

second = t.second;

third = t.third;

}

istream& operator >> (istream& cin, Triad& t)

{

cout << "enter first: ";

cin >> t.first;

cout << "enter second: ";

cin >> t.second;

cout << "enter third: ";

cin >> t.third;

return cin;

}

ostream& operator << (ostream& cout, Triad& t)

{

return (cout << "First: " << t.first << endl << "Second: " << t.second << endl << "Third: " << t.third << endl);

}

#pragma once

#include "Triad.h"

#include <iostream>

using namespace std;

class Date : public Triad

{

int day, month, year;

public:

//constructors

Date() : Triad(), day(0), month(0), year(0) {};

Date(int d, int m, int y) : Triad(), day(d), month(m), year(y) {};

Date(Triad t, int d, int m, int y) : Triad(t), day(d), month(m), year(y) {}

~Date() {};

void IncreaseFirst()

{

if (day + 1 > 31)

cout << "error! day out of range!" << endl;

else day++;

}

void IncreaseSecond()

{

if (month + 1 > 12)

cout << "error! month out of range!" << endl;

else month++;

}

void IncreaseThird()

{

if (year + 1 > 2021)

cout << "error! year out of range!" << endl;

else year++;

}

void PrintInf()

{

cout << "Day:\t" << day << endl;

cout << "Month:\t" << month << endl;

cout << "Year:\t" << year << endl;

}

void ChangeData(int n)

{

while (day + n > 31)

{

month++;

if (month > 12)

{

month = 1;

year++;

}

n -= 31;

}

day = day + n;

}

void SetDay(int d)

{

if (d > 31 || d < 1)

{

cout << "error! day out of range!" << endl;

}

else this->day = d;

};

void SetMonth(int m)

{

if (m > 12 || m < 1)

{

cout << "error! month out of range!" << endl;

this->month = 0;

}

this->month = m;

};

void SetYear(int y)

{

if (y > 2021 || y < 1)

{

cout << "error! year out of range!" << endl;

}

this->year = y;

};

int GetDay() { return day; }

int GetMonth() { return month; }

int GetYear() { return year; }

Date& operator=(Triad& t)

{

this->SetFirst(t.GetFirst());

this->SetSecond(t.GetSecond());

this->SetThird(t.GetThird());

return \*this;

}

friend istream& operator >>(istream& in, Date& tmp)

{

cout << "Input day:\t";

int d;

in >> d;

tmp.SetDay(d);

cout << "Input month:\t";

int month;

in >> month;

tmp.SetMonth(month);

cout << "Input year:\t";

int year;

in >> year;

tmp.SetYear(year);

return in;

}

friend ostream& operator <<(ostream& out, Date& tmp)

{

out << "Day:\t" << tmp.GetDay() << endl;

out << "Month:\t" << tmp.GetMonth() << endl;

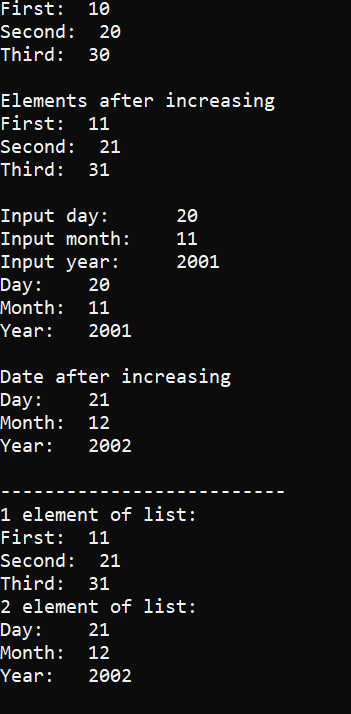
out << "Year:\t" << tmp.GetYear() << endl;

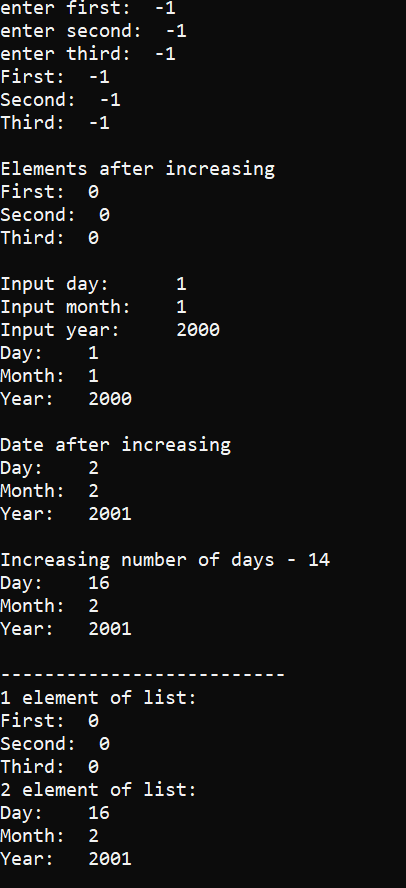
return out;

}

};

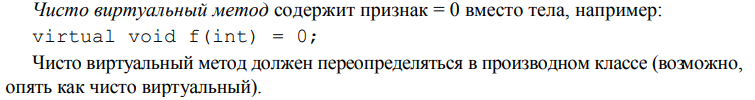
**Скриншоты**





**Контрольные вопросы**

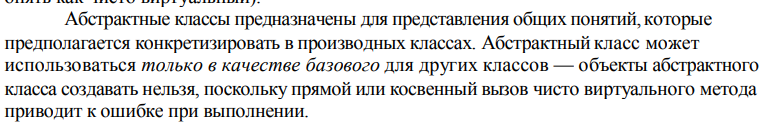


1. 



2. 



3. 



4. Полиморфные функции – это функции, которые работают с объектом любого типа в пределах одной иерархии.



5. При полиморфизме невозможно создать объект, так как базовым классом является абстрактный, когда в принципе подстановки используется наследование, что означает, что везде, где может быть использован объект базового класса, может быть и использован объект производного класса.



6. Пример: класс собака и класс кошка имеют одну из функций: говорить, но делают они это по-разному, так как собака гавкает, а кошка мяукает.



7. Метод PrintInf, который наследуется от абстрактного класса, для вывода на консоль объектов разных классов.