МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» (ФГБОУ ВО «ВГТУ», ВГТУ)

Факультет информационных технологий и компьютерной безопасности

(факультет)

Кафедра автоматизированных и вычислительных систем

КУРСОВОЙ ПРОЕКТ

по дисциплине «Объектно-ориентированное программирование»

Тема «Разработка программ с применением объектно-ориентированного подхода»

Расчетно-пояснительная записка

Разработал студент М. Е. Федоров

гр. бИВТ-223 Подпись, дата Инициалы, фамилия

Руководитель А. А. Акинин Подпись, дата Инициалы, фамилия

Нормоконтролер

А. А. Акинин Подпись, дата Инициалы, фамилия

Защищена Оценка

дата

Воронеж 2023

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» (ФГБОУ ВО «ВГТУ», ВГТУ)

Кафедра автоматизированных и вычислительных систем

Задание на курсовой проект

по дисциплине «Объектно-ориентированное программирование»

Тема работы «Разработка программ с применением объектно-ориентированного подхода»

Вариант работы 31

Студент группы бИВТ-223, Федоров Михаил Евгеньевич

Фамилия, имя, отчество

Технические условия: AMD Ryzen 5 3600, RTX 2060, 16 ГБ ОЗУ, Microsoft Word 2016

Содержание и объем работы (графические работы, расчеты и прочее): 41 страница, 0 рисунков, 0 таблиц, 0 приложений

Сроки выполнения этапов Рассмотрение теоретических сведений (февраль-март 2023); оформление пояснительной записки (май 2023);

Срок защиты курсового проекта май-июнь 2023

Руководитель А. А. Акинин Подпись, дата Инициалы, фамилия

Задание принял студент 21.02.23 М. Е. Федоров

Подпись, дата Инициалы, фамилия

Замечания руководителя

Содержание

[Задание на курсовой проект 2](#_Toc136167085)

[Замечания руководителя 3](#_Toc136167086)

[Введение 5](#_Toc136167087)

[1 История развития ООП 6](#_Toc136167088)

[2 Структура объектно-ориентированного подхода 7](#_Toc136167089)

[3 Наследование 9](#_Toc136167090)

[4 Полиморфизм 11](#_Toc136167091)

[4.1 Полиморфизм на уровне классов 11](#_Toc136167092)

[4.2 Полиморфизм на уровне функций 13](#_Toc136167093)

[5 Инкапсуляция 14](#_Toc136167094)

[6 Задание на курсовой проект 16](#_Toc136167095)

[7 Процесс написания кода 17](#_Toc136167096)

[7.1 Листинг файла «Header.h» 18](#_Toc136167097)

[7.2 Листинг файла «Header.cpp» 20](#_Toc136167098)

[7.3 Листинг файла «Lightning.cpp» 39](#_Toc136167099)

[Заключение 40](#_Toc136167100)

[Список литературы 41](#_Toc136167101)

# Введение

Объектно-ориентированное программирование (ООП) — это парадигма программирования, основанная на понятии объектов, которые имеют свойства и методы, взаимодействующие друг с другом. Каждый объект представляет собой экземпляр определенного класса. В отличие от процедурного программирования, ООП позволяет более легко организовывать код в виде модулей, обеспечивает возможность повторного использования кода и упрощает использование программного обеспечения.

Основными преимуществами ООП являются:

* модульность: код можно разделить на небольшие модули, каждый из которых может быть изменен независимо от других модулей;
* повторное использование кода: объекты могут быть созданы повторно и использоваться в разных частях программы.

Однако у ООП есть и недостатки касательно начинающих разработчиков:

* усвоение: понимание концепций ООП может быть сложным для новых разработчиков;
* содержание программы: код, написанный с использованием ООП, должен тщательно продумываться для универсального использования.

Отличительными чертами ООП являются:

* использование объектов: объекты — это экземпляры классов, которые содержат данные и методы, позволяющие работать с этими данными;
* наследование: классы могут наследовать свойства и методы других классов;
* полиморфизм: один и тот же метод может использоваться для работы с разными типами данных;
* инкапсуляция: объекты могут скрывать свои свойства и методы от внешнего блока программы, они представляют собой «черный ящик».

# История развития ООП

ООП возникло в результате развития идеологии процедурного программирования, где данные и подпрограммы (процедуры, функции) их обработки формально не связаны.

В 1967 году были предложены революционные идеи: объекты, классы, виртуальные методы и др., однако это всё не было воспринято современниками как нечто грандиозное. Немного позже понятие класса стало основообразующей идеей для многих конструкций языка.

Были разработаны три основополагающих принципа того, что потом стало называться объектно-ориентированным программированием:

* наследование;
* инкапсуляция;
* полиморфизм.

В настоящее время количество прикладных языков программирования (список языков), реализующих объектно-ориентированную парадигму, является наибольшим по отношению к другим парадигмам.

Наиболее распространённые в промышленности языки программирования:

* С++;
* Delphi;
* C#;
* Java.

# Структура объектно-ориентированного подхода

Структура объектно-ориентированного подхода включает в себя концепции классов и объектов, которые являются основными строительными блоками программы в ООП. Классы определяют свойства и поведение объекта, а объекты в свою очередь являются экземплярами определенных классов [1].

В C++ классы могут быть определены с использованием ключевого слова «class».

Пример определения класса Student в C++ приведён ниже.

class Student {

private:

//Свойства класса Student: имя и возраст студента

string name; int age;

public:

// Создание конструктора класса

Student (string nameX, int ageX) {name = nameX; age = ageX;}

// Геттер. Получение свойств экземпляра класса

void getStudent ()

{

cout << "Hello, my name is " << name << " and I am " << age << " years old.\n";

}

// Сеттер. При необходимости меняет свойство

void setAge (int ageX) {age = ageX;}

};

В этом примере класс Student имеет две переменные экземпляра «name» и «age», а также два вспомогательных метода и конструктор класса. Переменные-члены недоступны вне класса – они представляют собой черный ящик. Вспомогательные методы объявлены модификатором доступа «public», что обеспечивает возможность работу с ними вне класса.

Конструктор класса в C++ вызывается при создании нового объекта и инициализирует переменные экземпляра данного класса. Сеттер может изменять переменные экземпляра, а геттер позволяет узнать значения определенных свойств объекта.

В C++, модификаторы доступа public, private и protected используются для ограничения доступа к переменным-членам и функциям-членам класса. Модификатор доступа public означает, что переменные-члены и функции-члены доступны извне класса, private означает, что они доступны только внутри класса, а protected означает, что они доступны только внутри класса и его наследников.

# Наследование

Наследование — это механизм объектно-ориентированного программирования, который позволяет создавать новый класс на основе существующего. Класс, от которого производится наследование, называется родительским классом, а новый класс – дочерним классом.

Основы наследования заключаются в том, что дочерний класс получает все свойства и методы родительского класса, а также может добавлять свои собственные свойства и методы. Это позволяет избежать дублирования кода и повторного использования существующего функционала.

Существует три типа наследования.

Публичное наследование «public» - все открытые свойства и методы базового класса становятся открытыми свойствами и методами производного класса. Такой тип наследования наиболее распространен и используется по умолчанию.

Защищенное наследование «protected» - все открытые и защищенные свойства и методы базового класса становятся защищенными свойствами и методами производного класса.

Приватное наследование «private» - все открытые и защищенные свойства и методы базового класса становятся приватными свойствами и методами производного класса.

В C++ для наследования используется ключевое слово class, за которым следует имя родительского класса, а затем - имя дочернего класса. Например, написанный ниже код создает производный класс CarX на основе базового класса Car [2].

class Car {

public:

void startEngine();

void stopEngine();

protected:

int speed;

};

class Car : public Car {

public:

void drive();

void stop();

};

В данном примере производный класс CarX наследует открытые свойства и методы базового класса Car с помощью ключевого слова public. Кроме того, производный класс CarX добавляет свои собственные открытые методы drive() и stop().

# Полиморфизм

Данное свойство ООП является одной из ключевых концепций объектно-ориентированного программирования. Этот принцип позволяет работать с объектами разных классов через общий интерфейс, что упрощает кодирование и повышает гибкость приложения [3].

Полиморфизм может быть реализован на двух уровнях: на уровне классов и на уровне функций.

## Полиморфизм на уровне классов

Полиморфизм на уровне классов может быть достигнут с помощью виртуальных функций. Виртуальная функция — это функция, объявленная в базовом классе, которая может быть переопределена в производных классах [4].

Для демонстрации полиморфизма на уровне классов, рассмотрим пример с классами «Мой инструмент» и её наследниками «Кирка» и «Топор». Допустим, что у нас есть несколько инструментов на выбор и мы хотим выполнить определенное действие в игре. В данном случае мы можем определить метод работы с инструментами. В родительском классе определяем метод работы без инструментов, а в дочерних – способы работы с инструментами.

#include <string>

#include <iostream>

using namespace std;

class myTool

{

public:

virtual void extraction()

{

cout << "You don't have the tools." << endl;

}

};

class pickaxe : public myTool

{

public:

void extraction() override

{

cout << "dig" << endl;

}

};

class axe : public myTool

{

public:

void extraction() override

{

cout << "hack" << endl;

}

};

int main()

{

//Создание экземпляра класса pickaxe. Выбранный инструмент - "Кирка"

pickaxe pickaxe;

//Создание указателя типа myToll, присваивание ссылки на объект класса pickaxe

myTool \*tool = &pickaxe;

//Вызов виртуальной функции extraction

tool->extraction();

return 0;

}

После выполнения кода программы на экране консоли будет написано «dig» – исполнен метод, определенный в классе pickaxe. То есть, если наш предмет «кирка», то мы будем копать. Если ссылаться на объект класса «axe», будет выполняться действие «hack». Таким образом, при помощи виртуальной функции мы можем переопределять метод работы с определенным объектом [5].

## Полиморфизм на уровне функций

Полиморфизм на уровне функций позволяет перегружать функции в зависимости от типа аргументов, которые они принимают [6]. Приведём пример.

void print(int n) { cout << "Integer: " << n << endl; }

void print(double d) { cout << "Double: " << d << endl; }

void print(string s) { cout << "String: " << s << endl; }

int main()

{

int n = 42;

double d = 3.14;

string s = "Hello, world!";

print(n); // Integer: 42

print(d); // Double: 3.14

print(s); // String: Hello, world!

}

В данном примере функция print() перегружена три раза для разных типов аргументов: int, double и string. В зависимости от типа аргумента будет вызвана соответствующая функция print().

# Инкапсуляция

Инкапсуляция — это один из основных принципов объектно-ориентированного программирования, который позволяет скрыть детали реализации объекта и предоставить только интерфейс для работы с ним. Основная цель инкапсуляции - обеспечить безопасность и целостность данных, предотвратить несанкционированный доступ к ним [7, 8].

В C++ инкапсуляция достигается с помощью модификаторов доступа: public, private и protected. Public члены класса доступны из любого места программы, private - только внутри класса, protected - внутри класса и его наследников.

Приведем пример использования инкапсуляции в C++.

class BankAccount {

private:

float balance; // переменная, скрытая от других частей программы

public:

BankAccount(float initialBalance) { // конструктор класса

balance = initialBalance;

}

// публичный метод, который позволяет получить значение переменной

// balance

float getBalance()

{

return balance;

}

void deposit(float amount) { // публичный метод для пополнения баланса

balance += amount;

}

void withdraw(float amount) { // публичный метод для снятия средств //со счета

if (balance >= amount) {

balance -= amount;

} else {

cout << "Not enough funds" << endl;

}

}

};

int main() {

BankAccount myAccount(1000.0);

cout << "My balance is: " << myAccount.getBalance() << endl;

myAccount.deposit(500.0);

cout << "My balance is: " << myAccount.getBalance() << endl;

myAccount.withdraw(2000.0);

cout << "My balance is: " << myAccount.getBalance() << endl;

myAccount.withdraw(800.0);

cout << "My balance is: " << myAccount.getBalance() << endl;

return 0;

}

В этом примере переменная balance скрыта от других частей программы, и к ней можно получить доступ только через методы класса getBalance(), deposit() и withdraw(). Кроме того, метод withdraw() проверяет, достаточно ли средств на счете перед тем, как произвести снятие. Это позволяет обеспечить безопасность и целостность данных в объекте класса BankAccount.

# Задание на курсовой проект

Разработать классы для описанных ниже объектов. Для всех объектов в классе должны быть реализованы конструктор по умолчанию, конструктор с параметрами, конструктор копирования, деструктор и необходимые методы set (…), get (…), show (…). На основе разработанного класса необходимо создать массив экземпляров класса с возможностью добавления, удаления и редактирования элементов этого массива. А также реализовать описанный дополнительно для каждого типа объекта функционал.

Вариант 31. Класс Молния.

Свойства молнии:

* идентификатор молнии;
* длина молнии;
* мощность разряда;
* удаленность от наблюдателя;
* длительность разряда;
* фамилия наблюдателя.

Реализовать:

* вывод списка молний, зафиксированных указанным наблюдателем и ударивших от него дальше указанного значения;
* вывод списка молний, с длиной большей указанной и заданной длительностью.

# Процесс написания кода

Для разделения интерфейса и реализации были созданы три файла:

* Header.h – файл, содержащий описание класса и определение переменных;
* Header.cpp – файл, который содержит реализацию основных функций программы;
* Fedorov – файл запуска программы.

Первым шагом в работе является описание класса «Lightning» в файле «Header.h», в котором содержатся параметры молнии, защищенные модификатором доступа «private».

Во втором шаге для класса «Lightning» были объявлены конструктор по умолчанию, параметрический конструктор, конструктор копирования и деструктор.

Следующим действием является объявление методов класса:

* сеттеры (ввод определенного значения молнии);
* геттеры (получение определенного параметра молнии);
* метод «showInfoAboutLight» – отображение всех параметров молнии.

Для решения задания были объявлены следующие функции:

* вывод списка молний и их параметров
* добавление элемента;
* удаление элемента;
* изменение элемента;
* выполнение первого задания;
* выполнение второго задания;
* меню программы

Затем все предыдущие пункты были реализованы в файле «Header.cpp».

Опишем функцию, вызывающую меню программы. После её вызова пользователю предлагается выбрать действие со списком молний (добавление, удаление, изменение, показать решение первого задания, показать решение второго задания, вывести список молний и всех их параметров).

Функция добавления молнии запрашивает у пользователя количество новых молний, а затем их параметры.

Функция удаления молнии запрашивает тип удаления (все молнии с совпадающим параметром, молния с определенным индексом) и выводит список молний с их номерами и параметрами.

Функция изменения параметра молнии запрашивает параметр, который хочет изменить пользователь и номер молнии, а затем меняет его на новый.

## Листинг файла «Header.h»

#pragma once

#include <string>

#include <vector>

using namespace std;

class Lightning

{

private:

string idLight, surname;

int lenLight, powerLight, distantToLight;

double timeLight;

//Конструктор по умолчанию

Lightning();

//Конструктор c параметрами

Lightning(string idLight, int lenLight, int powerLight, int distantToLight, double timeLight, string surname);

//Конструктор копирования

Lightning(const Lightning& myLight);

//Деструктор

~Lightning();

//Сеттеры

void setIdLight(string idLight);

void setLenLight(int lenLight);

void setPowerLight(int powerLight);

void setDistantToLight(int distantToLight);

void setTimeLight(double timeLight);

void setSurname(string surname);

//Геттеры

string getIdLight();

int getLenLight();

int getPowerLight();

int getDistantToLight();

double getTimeLight();

string getSurname();

//Отобразить всю информацию о молнии

void showInfoAboutLight();

};

//Вывод информации о списке молний

void printAllList(vector<Lightning>& A);

//Добавление молний в список

void addElement(vector<Lightning>& A);

//Удаление молнии в списке

void delElement(vector<Lightning>& A);

//Изменение параметра молнии

void editElement(vector<Lightning>& A);

//Решение задания 1

void taskOne(vector<Lightning>& A);

//Решение задания 2

void taskTwo(vector<Lightning>& A);

//Запуск программы

void menu();

## Листинг файла «Header.cpp»

#include <string>

#include <vector>

#include <iostream>

#include "Header.h"

using namespace std;

class Lightning

{

private:

string idLight, surname;

int lenLight, powerLight, distantToLight;

double timeLight;

public:

//Конструктор по умолчанию

Lightning()

{

idLight = "";

surname = "";

lenLight = 0;

powerLight = 0;

distantToLight = 0;

timeLight = 0;

}

//Конструктор с параметрами

Lightning(string idLight, int lenLight, int powerLight, int distantToLight, double timeLight, string surname)

{

this->idLight = idLight;

this->lenLight = lenLight;

this->powerLight = powerLight;

this->distantToLight = distantToLight;

this->timeLight = timeLight;

this->surname = surname;

}

//Конструктор копирования

Lightning(const Lightning& myLight)

{

idLight = myLight.idLight;

lenLight = myLight.lenLight;

powerLight = myLight.powerLight;

distantToLight = myLight.distantToLight;

timeLight = myLight.timeLight;

surname = myLight.surname;

}

//Деструктор

~Lightning() {}

//Сеттеры

void setIdLight(string idLight) { this->idLight = idLight; }

void setLenLight(int lenLight) { this->lenLight = lenLight; }

void setPowerLight(int powerLight) { this->powerLight = powerLight; }

void setDistantToLight(int distantToLight) { this->distantToLight = distantToLight; }

void setTimeLight(double timeLight) { this->timeLight = timeLight; }

void setSurname(string surname) { this->surname = surname; }

//Геттеры

string getIdLight() { return idLight; }

int getLenLight() { return lenLight; }

int getPowerLight() { return powerLight; }

int getDistantToLight() { return distantToLight; }

double getTimeLight() { return timeLight; }

string getSurname() { return surname; }

//Отобразить всю информацию о молнии

void showInfoAboutLight()

{

cout << "Id lightning: " << idLight;

cout << "\nLength lightning: " << lenLight;

cout << "\nPower lightning: " << powerLight;

cout << "\nDistant to lightning: " << distantToLight;

cout << "\nFlash time: " << timeLight;

cout << "\nSurname spectator: " << surname << endl;

}

};

//Вывод информации о списке молний

void printAllList(vector<Lightning>& A)

{

for (Lightning i : A)

{

i.showInfoAboutLight();

cout << endl << endl;

}

}

//Добавление молний

void addElement(vector<Lightning>& A)

{

//Запрос на количество добавлений

int count;

cout << "How many lightnings do you want to add? - ";

cin >> count;

//Запрос списка параметров молнии

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

{

string idLightX, surnameX;

int lenLightX, powerLightX, distantToLightX, timeLightX;

cout << "Enter id lightning " << i + 1 << ": ";

cin >> idLightX;

cout << "Enter length lightning " << i + 1 << ": ";

cin >> lenLightX;

cout << "Enter power lightning " << i + 1 << ": ";

cin >> powerLightX;

cout << "Enter distant to lightning " << i + 1 << ": ";

cin >> distantToLightX;

cout << "Enter flash time lightning " << i + 1 << ": ";

cin >> timeLightX;

cout << "Enter surname spectator " << i + 1 << ": ";

cin >> surnameX;

//Добавление молнии в список

A.push\_back(Lightning(idLightX, lenLightX, powerLightX, distantToLightX, timeLightX, surnameX));

cout << "!!! Lightning number " << i + 1 << " is added in your list!!! \n\n";

}

}

//Удаление молнии

void delElement(vector<Lightning>& A)

{

/\*Параметры, по которым отобразить список молний

1 - по идентефикатору

2 - по длине

3 - по мощности

4 - по дистанции до молнии

5 - по времени вспышки

6 - по фамилии наблюдателя\*/

cout << "Enter parametr lightnings:\n";

cout << "1 - identificator\n";

cout << "2 - length\n";

cout << "3 - power\n";

cout << "4 - distant to lightning\n";

cout << "5 - flash time\n";

cout << "6 - surname spectator\n";

//Запрос номера действия

char numAct;

cout << "Your action: ";

cin >> numAct;

cout << endl;

cout << "Number\t\tparametr\n";

switch (numAct)

{

case'1':

{

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

{

cout << i + 1 << "\t\t" << A[i].getIdLight() << endl;

}

break;

}

case'2':

{

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

{

cout << i + 1 << "\t\t" << A[i].getLenLight() << endl;

}

break;

}

case'3':

{

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

{

cout << i + 1 << "\t\t" << A[i].getPowerLight() << endl;

}

break;

}

case'4':

{

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

{

cout << i + 1 << "\t\t" << A[i].getDistantToLight() << endl;

}

break;

}

case'5':

{

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

{

cout << i + 1 << "\t\t" << A[i].getTimeLight() << endl;

}

break;

}

case'6':

{

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

{

cout << i + 1 << "\t\t" << A[i].getSurname() << endl;

}

break;

}

}

char typeDel;

string delParametr;

cout << "Enter value parameter: ";

cin >> delParametr;

cout << endl;

cout << "Deleting an element\n";

cout << "1 - everything related to this value\n";

cout << "2 - defined number\n";

cout << "Type of deletion: ";

cin >> typeDel;

cout << endl;

switch (typeDel)

{

case'1':

{

//Удаление по параметру всех элементов

switch (numAct)

{

case'1':

{

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

{

if (A[i].getIdLight() == delParametr)

{

A.erase(A.begin() + i);

}

}

break;

}

case'2':

{

int k = stoi(delParametr);

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

{

if (A[i].getLenLight() == k)

{

A.erase(A.begin() + i);

}

}

break;

}

case'3':

{

int k = stoi(delParametr);

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

{

if (A[i].getPowerLight() == k)

{

A.erase(A.begin() + i);

}

}

break;

}

case'4':

{

int k = stoi(delParametr);

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

{

if (A[i].getDistantToLight() == k)

{

A.erase(A.begin() + i);

}

}

break;

}

case'5':

{

int k = stoi(delParametr);

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

{

if (A[i].getTimeLight() == k)

{

A.erase(A.begin() + i);

}

}

break;

}

case'6':

{

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

{

if (A[i].getSurname() == delParametr)

{

A.erase(A.begin() + i);

}

}

break;

}

default:

{

cout << "Error in the parameter\n";

}

}

break;

}

case'2':

{

int numLight = 0;

cout << "Enter num lightning - ";

cin >> numLight;

A.erase(A.begin() + numLight - 1);

break;

}

default:

{

cout << "The type of deletion is not defined";

}

}

}

//Изменение параметра молнии

void editElement(vector<Lightning>& A)

{

/\*Параметры, по которым отобразить список молний

1 - по идентефикатору

2 - по длине

3 - по мощности

4 - по дистанции до молнии

5 - по времени вспышки

6 - по фамилии наблюдателя\*/

cout << "Enter the change parameter:\n";

cout << "1 - identificator\n";

cout << "2 - length\n";

cout << "3 - power\n";

cout << "4 - distant to lightning\n";

cout << "5 - flash time\n";

cout << "6 - surname spectator\n";

//Запрос номера действия

char numAct;

cout << "Your action: ";

cin >> numAct;

cout << endl;

cout << "Number\t\tparametr\n";

switch (numAct)

{

case'1':

{

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

{

cout << i + 1 << "\t\t" << A[i].getIdLight() << endl;

}

break;

}

case'2':

{

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

{

cout << i + 1 << "\t\t" << A[i].getLenLight() << endl;

}

break;

}

case'3':

{

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

{

cout << i + 1 << "\t\t" << A[i].getPowerLight() << endl;

}

break;

}

case'4':

{

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

{

cout << i + 1 << "\t\t" << A[i].getDistantToLight() << endl;

}

break;

}

case'5':

{

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

{

cout << i + 1 << "\t\t" << A[i].getTimeLight() << endl;

}

break;

}

case'6':

{

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

{

cout << i + 1 << "\t\t" << A[i].getSurname() << endl;

}

break;

}

}

int numLight;

string edParametr;

cout << "Enter num lightning: ";

cin >> numLight;

cout << "Enter new value parameter: ";

cin >> edParametr;

cout << endl;

//cout << "Number\t\tparametr\n";

switch (numAct)

{

case'1':

{

A[numLight - 1].setIdLight(edParametr);

break;

}

case'2':

{

A[numLight - 1].setLenLight(stoi(edParametr));

break;

}

case'3':

{

A[numLight - 1].setPowerLight(stoi(edParametr));

break;

}

case'4':

{

A[numLight - 1].setDistantToLight(stoi(edParametr));

break;

}

case'5':

{

A[numLight - 1].setTimeLight(stoi(edParametr));

break;

}

case'6':

{

A[numLight - 1].setSurname(edParametr);

break;

}

}

}

//Решение задания 1

void taskOne(vector<Lightning>& A)

{

string surnameX;

cout << "Enter surname spectator: ";

cin >> surnameX;

int distantToLightX;

cout << "Enter distant to lightning: ";

cin >> distantToLightX;

for (Lightning i : A)

{

if (i.getSurname() == surnameX)

{

if (i.getDistantToLight() > distantToLightX)

{

i.showInfoAboutLight();

cout << endl << endl;

}

}

}

}

//Решение задания 2

void taskTwo(vector<Lightning>& A)

{

int timeLightX;

cout << "Enter flash time: ";

cin >> timeLightX;

int lenLightX;

cout << "Enter lenght lightning: ";

cin >> lenLightX;

for (Lightning i : A)

{

if (i.getTimeLight() == timeLightX)

{

if (i.getLenLight() > lenLightX)

{

i.showInfoAboutLight();

cout << endl << endl;

}

}

}

}

//Запуск программы

void menu()

{

//Создание начального списка молний

//Параметры молнии: идентификатор, длина, мощность, удаленность, длительность, фамилия наблюдателя

vector<Lightning> vecLightning;

vecLightning.push\_back(Lightning("A", 1750, 15000, 1000, 1, "Fedorov"));

vecLightning.push\_back(Lightning("B", 1300, 18000, 1500, 0.75, "Ivanov"));

vecLightning.push\_back(Lightning("C", 1200, 20000, 200, 1.1, "Komarov"));

vecLightning.push\_back(Lightning("D", 1250, 22500, 1355, 0.83, "Mikhailov"));

vecLightning.push\_back(Lightning("B", 1250, 17500, 270, 1.1, "Denisov"));

vecLightning.push\_back(Lightning("C", 1800, 16500, 1200, 0.85, "Fedorov"));

while (true)

{

/\*Варианты действий со списком молний

1 - добавить молнию

2 - удалить молнию

3 - изменить параметр молнии

4 - выполнение задания один

5 - выполнение задания два

6 - вывыод информации о молниях\*/

cout << "Table of actions\n";

cout << "1 - add lightning\n";

cout << "2 - delete lightning\n";

cout << "3 - edit info about lightning\n";

cout << "4 - task one\n";

cout << "5 - task two\n";

cout << "6 - info about list\n\n";

//Запрос номера действия

char numAct;

cout << "Enter an action: ";

cin >> numAct;

cout << endl;

switch (numAct)

{

case '1':

{

addElement(vecLightning);

break;

}

case '2':

{

delElement(vecLightning);

break;

}

case '3':

{

editElement(vecLightning);

break;

}

case '4':

{

taskOne(vecLightning);

break;

}

case '5':

{

taskTwo(vecLightning);

break;

}

case '6':

{

printAllList(vecLightning);

break;

}

default:

{

cout << "The end of the program\n";

break;

}

}

}

}

## Листинг файла «Lightning.cpp»

#include <iostream>

#include <string>

#include <vector>

#include "Header.h"

int main()

{

menu();

}

# Заключение

Объектно-ориентированное программирование является эффективным подходом к разработке программного обеспечения. Оно позволяет создавать простые, модульные и легко расширяемые системы. ООП является важным инструментом в программировании, который помогает создавать качественное программное обеспечение. Благодаря изучению ООП становится возможным разрабатывать сложные системы, которые можно проще поддерживать и расширять в будущем.

Однако, нужно научиться правильно применять этот подход, так как он подходит не ко всем типам задач, поскольку разделение объявления и определения для малой программы будет не эффективно.

В ходе работы над курсовым проектом, были приобретены знания и умения, включающие в себя:

* разработку класса;
* работу с адресами;
* разделение интерфейса и реализации;
* создание многофайловых проектов.

# Список литературы

1. Ашарина И.В., Крупская Ж.Ф. Язык С++ и объектно-ориентированное программирование в С++.: Лабораторный практикум. Учебное пособие для вузов. /И.В. Аширина, Ж.Ф. Крупская – М.: Горячая линия – Телеком, 2016. – 232 с.

2. Мейер Б. Почувствуй класс.: учебное издание. / Б. Мейер; Перевод под ред. В.А. Биллига. — М.: Национальный Открытый Университет «ИНТУИТ»: БИНОМ. Лаборатория знаний, 2011. —775 с.

3. Иванова Г.С, Ничушкина Т.Н., Пугачев Е.К. И21 Объектно-ориентированное программирование: Учеб. для вузов/ Под ред. Г.С. Ивановой. - М.: Изд-во МГТУ им. Н.Э. Баумана, 2001. - 320 с, ил.

4. Лафоре Р. Л29 Объектно-ориентированное программирование в С++ / Р. Лафоре - 4-е изд. – СПб.: Санкт-Петербург,2011 – 928с.: ил.

5. Колесов, Ю. Б. Моделирование систем. Объектно-ориентированный подход / Ю.Б. Колесов, Ю.Б. Сениченков. - М.: БХВ-Петербург, 2006. - 192 c.

6. Пол, Айра Объектно-ориентированное программирование на С++ / Айра Пол. - М.: Не указано, Бином, Невский Диалект, 2001. - 464 c.

7. Лесневский, А. С. Объектно-ориентированное программирование для начинающих (+ CD-ROM) / А.С. Лесневский. - М.: Бином. Лаборатория знаний, 2010. - 232 c.

8. Хорев, П.Б. Объектно-ориентированное программирование: Учебное пособие для студ. учреждений высш. проф. образования / П.Б. Хорев. - М.: ИЦ Академия, 2012. - 448 c.