Частное учреждение образования

«Колледж бизнеса и права»

**ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЕ**

**Вопросы** **к экзамену для учащихся 2 курса дневной формы получения образования специальности 2-40 01 01 «Программное обеспечение информационных технологий»**

1. Дайте определение понятию «класс» и опишите его синтаксис. Приведите пример объявления.

Класс является абстрактным типом данных, определяемым программистом, и представляет собой модель реального объекта в виде данных и функций для работы с ними.

Class <имя>

{

Private:

<описание скрытых элементов>

Public:

<описание открытых элементов>

};

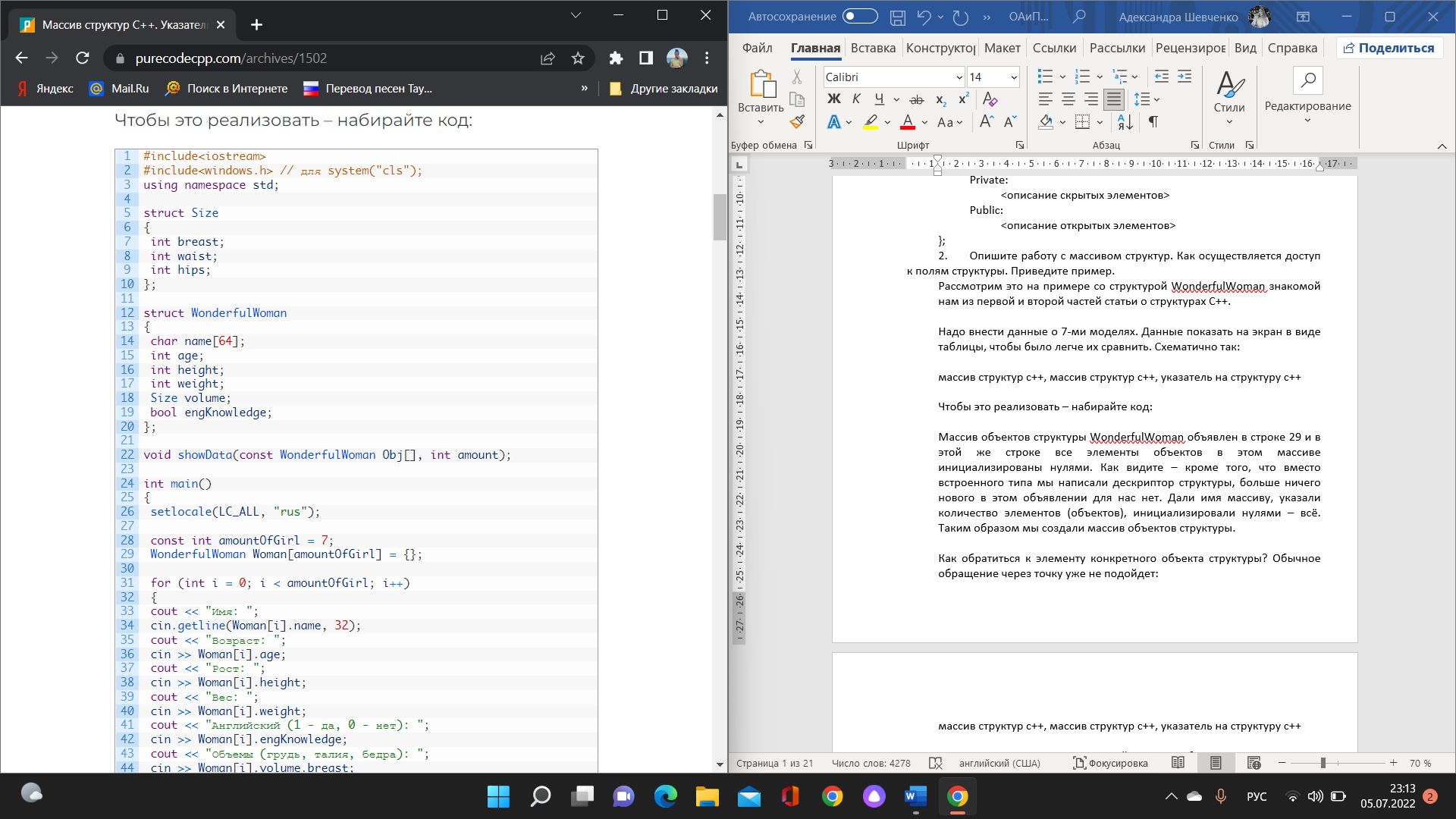
1. Опишите работу с массивом структур. Как осуществляется доступ к полям структуры. Приведите пример.

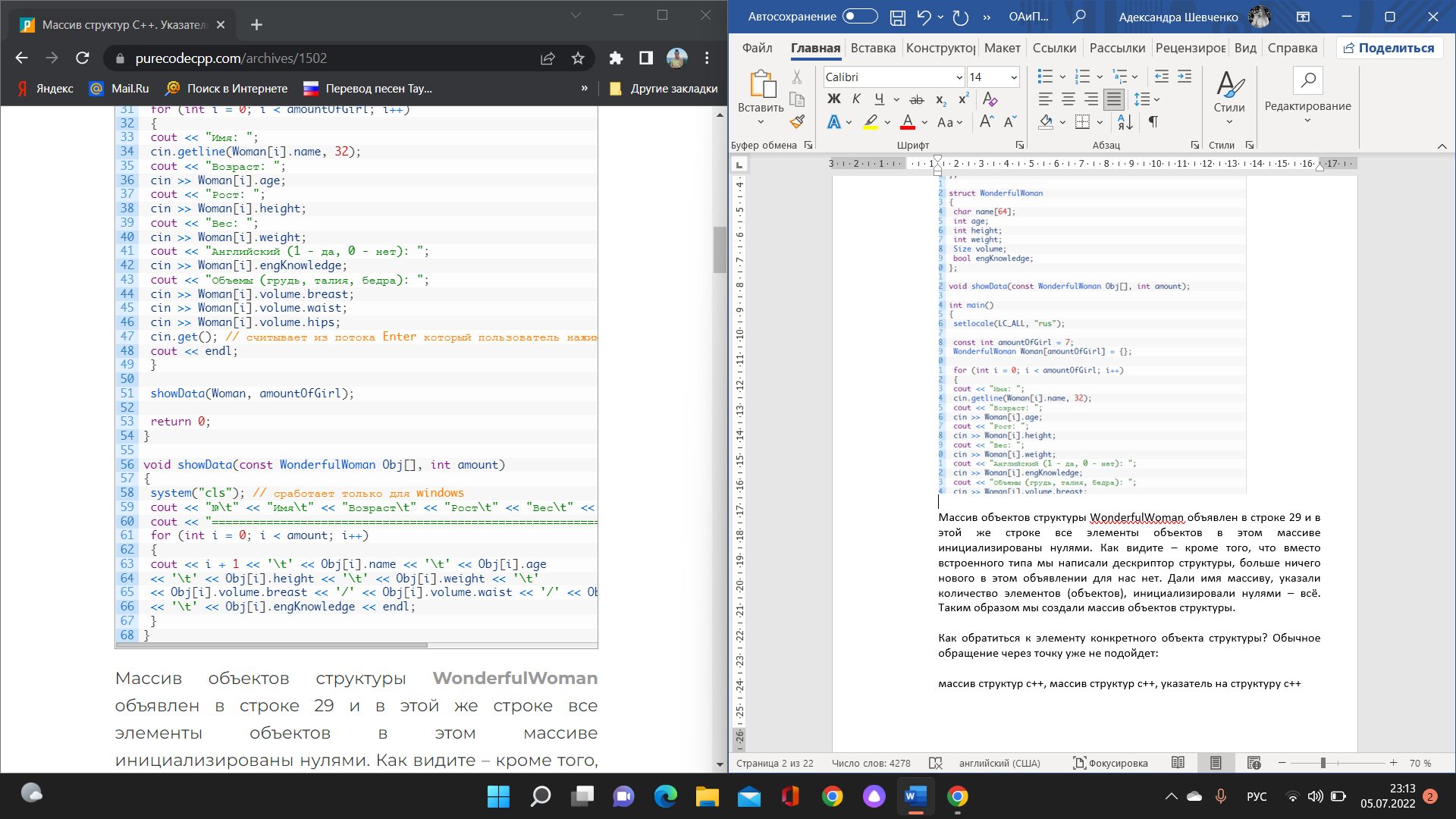
Рассмотрим это на примере со структурой WonderfulWoman знакомой нам из первой и второй частей статьи о структурах C++.

Надо внести данные о 7-ми моделях. Данные показать на экран в виде таблицы, чтобы было легче их сравнить. Схематично так:

массив структур с++, массив структур c++, указатель на структуру c++

Чтобы это реализовать – набирайте код:





Массив объектов структуры WonderfulWoman объявлен в строке 29 и в этой же строке все элементы объектов в этом массиве инициализированы нулями. Как видите – кроме того, что вместо встроенного типа мы написали дескриптор структуры, больше ничего нового в этом объявлении для нас нет. Дали имя массиву, указали количество элементов (объектов), инициализировали нулями – всё. Таким образом мы создали массив объектов структуры.

Как обратиться к элементу конкретного объекта структуры? Обычное обращение через точку уже не подойдет:

массив структур с++, массив структур c++, указатель на структуру c++

Конечно, ведь непонятно в какой именно объект из 7 надо внести данные. Так как каждый объект теперь является ячейкой массива структур, чтобы обратиться к его элементам надо сначала указать индекс объекта в квадратных скобках. Только потом точку и имя элемента:

массив структур с++, массив структур c++, указатель на структуру c++

В строках 31- 49 находится цикл for. С его помощью вносятся данные во все объекты массива структур. Обратите внимание на строку 47. Без этого оператора cin.get(); вы столкнетесь с ошибкой, о которой компилятор вам не сообщит. Она заключается в том, что после внесения данных в элемент Woman[i].volume.hips и нажатия Enter, в потоке ввода остаётся символ конца строки. Когда цикл продолжит работу – этот символ автоматически запишется в Woman[i].name следующего объекта и нам сразу будет предложено ввести возраст. То есть имя внести мы не сможем. Чтобы этого избежать, надо считать из потока этот символ с помощью cin.get(); .

Точно так же, как мы можем создавать массивы любых встроенных типов данных (int, char…), можно создавать массивы структур.

1. Дайте определение понятию «структура» и опишите его синтаксис. Как осуществляется доступ к полям. Приведите пример.

Структура – это составной тип данных, в котором под одним именем объединены данные различных типов. Отдельные данные структуры называются полями.

*struct имя*

{

*тип\_элемента\_1 имя\_элемента\_1*;

*тип\_элемента\_2 имя\_элемента\_2*;

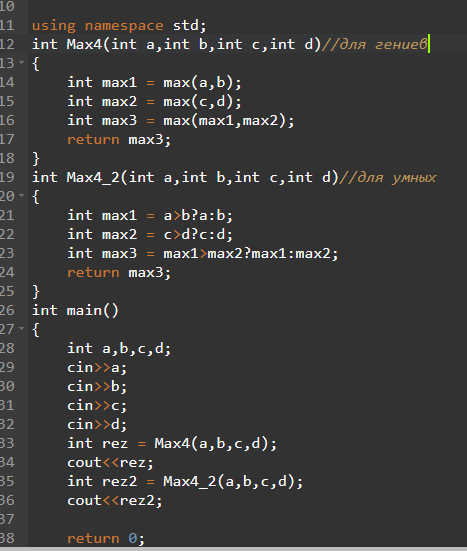
*тип\_элемента\_n имя\_элемента\_n*;

*имя\_struct\* указатель\_на\_следующий\_элемент*;

*имя\_struct\* указатель\_на\_предыдущий\_элемент*;

};

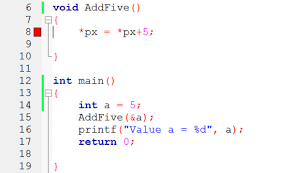
1. Опишите объявление, описание и вызов функции. Написать функцию *Max4(A,B,C,D)* вещественного типа, находящую максимальное из четырех вещественных чисел *A*, *B*, *C* и *D*.



1. Дайте определение понятию «указатель». Приведите пример объявления указателя. Как передать указатель в функцию.

**Указатель** — это переменная или константа, содержащая адрес другой переменной.

double \*py = &y; // взять адрес переменной у и поместить его (адрес переменной y) в указатель ру



для использования указателя в переменной используется \*

1. Опишите выполнение операторов выбора, их формы и синтаксис, приведите пример. Напишите таблицу истинности.

switch (***выражение***)

{

case *константа1*:

*последовательность\_операторов1*;

break;

case *константа2*:

*последовательность*\_*операторов2*;

break;

...

case *константаN*:

*последовательность\_операторовN*;

break;

default:

*последовательность\_операторов*;

}

1. Дайте определение понятию «список». Приведите структуру двухсвязного списка. Приведите пример объявления списка.

**Связный список** – одна из базовых структур данных. Ее часто сравнивают с массивом, так как многие другие структуры можно реализовать с помощью либо массива, либо связного списка.

Пример:

#include <list>

list < тип данных > <имя контейнера>;

list <int> this\_list = {4, 6, 3, 2};

1. Опишите работу с двумерным массивом. Представление в памяти. Способы сортировки массивов. Приведите пример объявления двухмерного массива вещественного типа размером 5\*7.

\*массив располагается в оперативной памяти, а не на внешнем устройстве, как файлы;

\*элементы массива занимают подряд идущие ячейки, в отличие, например, от списков.

double mas[5][7];

Выделяются несколько типов сортировок

Пузырьком

Сортировка выбором наименьшего элемента

Сортировка вставками

1. Дайте определение понятию «стек». Приведите структуру стека. Приведите пример объявления стека.

**Стек** – это базовая структура данных, которая позволяет добавлять или удалять элементы только в её начале. Она похожа на стопку книг: если вы хотите взглянуть на книгу в середине стека, сперва придется убрать лежащие сверху.

Struct stack

{

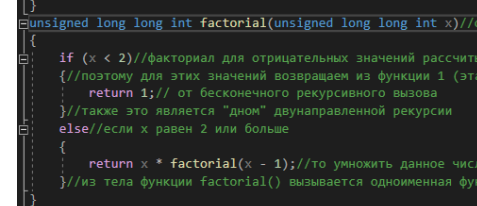
Int \* arr = new int[0];//размер стека

Int head = 0;//индекс крайнего элемента стека

};

1. Опишите виды и синтаксис рекурсии. Объясните различие косвенной и прямой рекурсией. Объясните различие с циклом. Приведите пример рекурсивной функции расчета факториала.

Рекурсия может быть как *прямой*, когда программа вызывает саму себя, так и непрямой( *косвенной*) , когда программа вызывает другую программу, а та в свою очередь, вызывает первую программу.



Цикл завершается в конце последовательности, в которой он выполняется. А вот рекурсивная функция может продолжаться бесконечно, потому что она может не иметь последовательности данных. Вместо этого у рекурсивной функции есть так называемое базовое условие. Базовое условие определяет, когда цикл должен завершится.

1. Дайте определение понятию «переменная», какое имя может быть у переменной. Приведите пример объявления переменной.

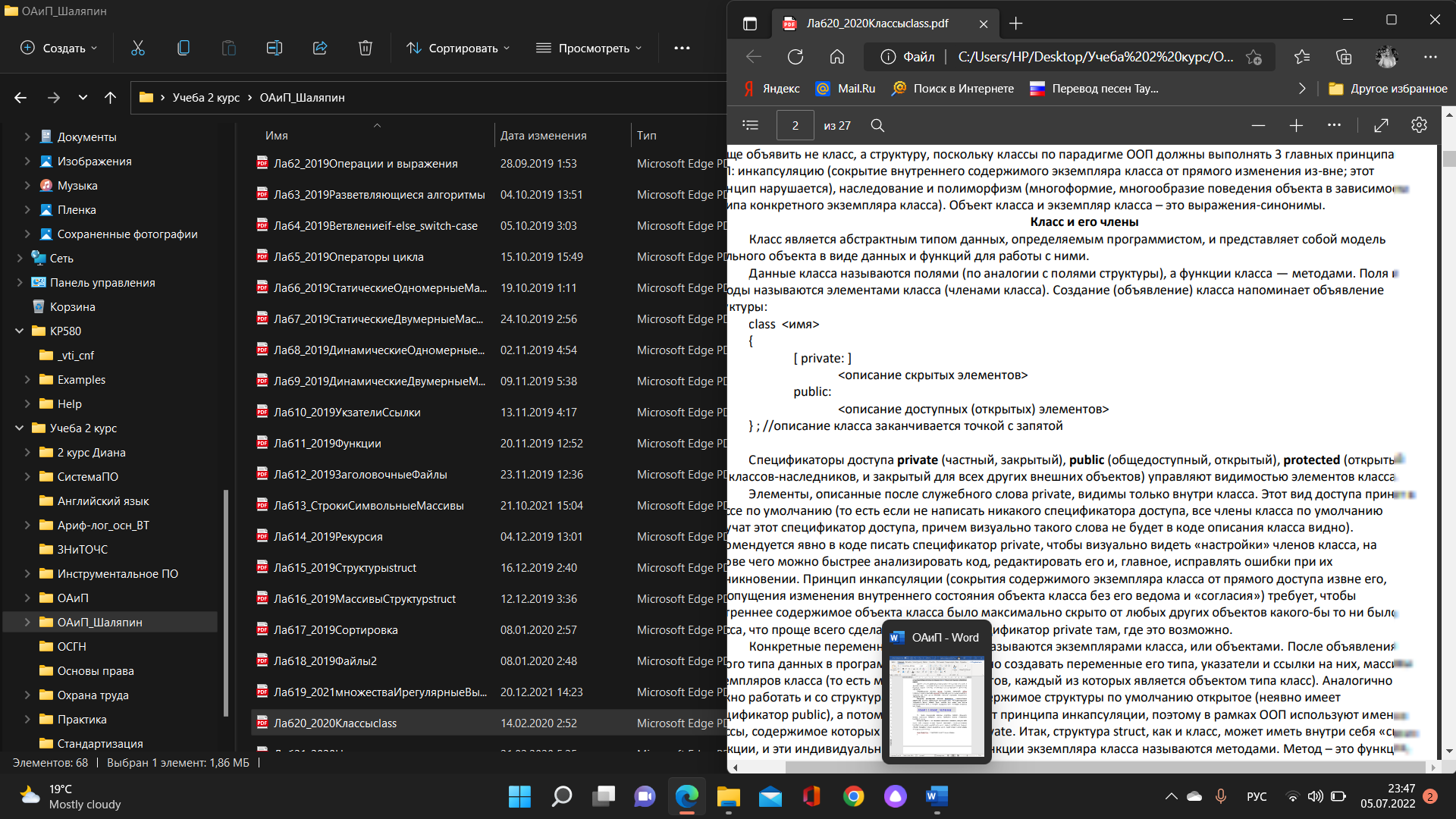
Переменная – это номер байта.

**Переменная** – это именованная область памяти, к которой мы имеем доступ из программы; туда можно помещать значения и затем извлекать их.

Int x = 666;

1. Опишите синтаксис класса. Объясните для чего используется оператор расширения области видимости “::”. Приведите пример объявления класса.

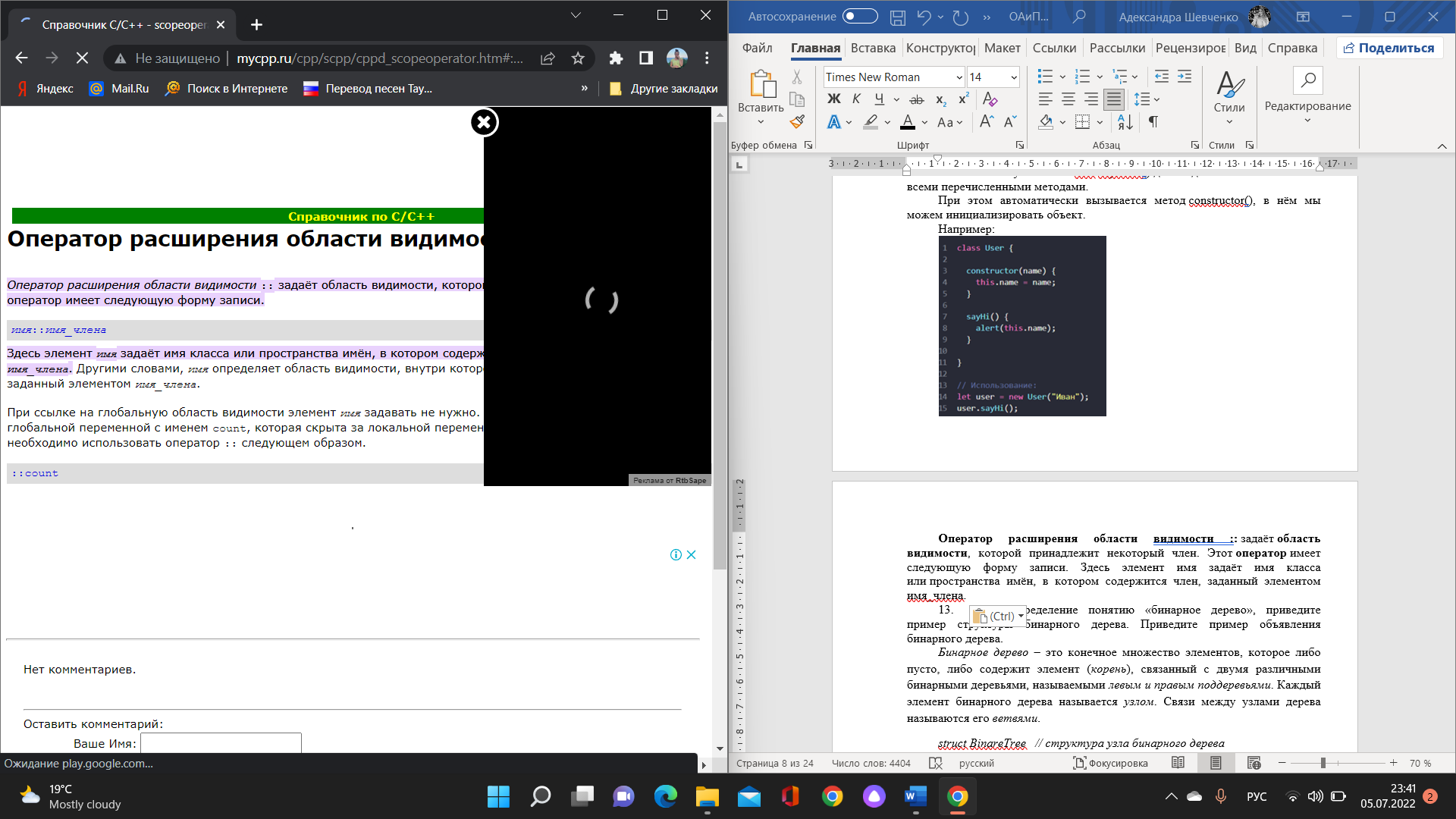
Данные класса называются полями (по аналогии с полями структуры), а функции класса — методами. Поля и методы называются элементами класса (членами класса). Создание (объявление) класса напоминает объявление структуры:



Спецификаторы доступа private (частный, закрытый), рubliс (общедоступный, открытый), protected (открытый для классов-наследников, и закрытый для всех других внешних объектов) управляют видимостью элементов класса

Каждый класс может состоять из членов. Члены класса, это: 1) поле – переменная, массив, структура для хранения данных определенного типа: int number; double weight; 2) свойство – метод, предназначенный для обеспечения доступа к полю класса (то есть через свойство можно получить значение конкретного поля (get) и/или присвоить конкретному полю значение (set). Получить значение конкретного поля можно почти всегда, но если поле константное или его значение вычисляется, зависит от других значений и условий, то присвоить ему значение будет нельзя (то есть свойство set в таком случае создавать программисту нельзя). Не надо создавать свойства для присвоения значений полям, которые получают конкретные смысловые значения, а не любые (в классе Калькулятор поле Результат нет смысла делать со свойством, устанавливающим ему значение, поскольку лучше, чтобы значение поля Результат устанавливал метод, вычисляющий результат из двух чисел (см. ниже)). 3) конструктор – особый метод, который создает экземпляры своего одноименного класса (объекты своего класса, переменные типа своего класса). Класс – это общая конструкция, схема, идея, общий шаблон, «матрица» некого явления, процесса или предмета (машина, компьютер, человек). Мы описываем класс подобно структуре struct, а после можем создавать объекты данного класса (например, конкретная машина Honda Accord III 1989 года выпуска в кузове седан с двигателем В20А I4 весом 1255 кг и регистрационным номером BY6426IP-1, принадлежащая водителю Ильину И.К. – то есть конкретный автомобиль, реально существовавший, существующий сейчас или могущий быть созданным (обнаруженным) в будущем; объект обычно можно потрогать руками, а сам класс как таковой – нет). Конструктор класса Машина создает этот объект – конкретный автомобиль Honda Accord III, экземпляр класса Машина. Конструктор пишется программистом. Конструктор имеет имя, всегда совпадающее с именем его класса. Конструктор – это метод, поэтому пишется со скобками(), может принимать ни одного или несколько входных параметров и никогда ничего не возвращает (не имеет return'a). Конструктором можно создавать переменные типа своего класса (объекты), а также указатели и ссылки типа своего класса. Переменные типа некого одного класса можно объединять в массивы. То есть класс – это пользовательский (программистский) тип данных, которым можно пользоваться как базовым типом данных, но создаваемая переменная будет иметь свою внутреннюю структуру из своих полей и у нее можно вызвать ее личные методы. 4) метод – член класса, представляющий собой функцию, которая находится в данном классе (принадлежит ему и его объектам, может быть вызвана только у экземпляров данного класса). Метод, подобно функции, может принимать ни одного или несколько входных параметров и одно значение возвращать return'ом или ни одного (void). Метод видит содержимое поля своего экземпляра класса (даже закрытые от других экземпляров классов – private, поскольку метод сам находится в этом же (своем) экземпляре класса) и может изменять значения неконстантных полей своего класса. Методы обычно оформляются как внешние: в классе описывается прототип метода, а за классом с указанием пространства имен данного класса (ИмяКласса:: ) дается полное определение метода (тело метода). 5) деструктор – особый метод класса, который уничтожает объекты своего класса, когда они уже, по мнению программиста, не нужны. Деструктор очищает память от неиспользуемых объектов своего класса, аналогично коду delete. Если вызов конструктора «рождает» объект класса, то вызов для него деструктора – уничтожает этот объект, то есть означает смерть этого объекта.

**Оператор расширения области видимости::** задаёт **область видимости**, которой принадлежит некоторый член. Этот **оператор** имеет следующую форму записи. Здесь элемент имя задаёт имя класса или пространства имён, в котором содержится член, заданный элементом имя\_члена.



1. Дайте определение понятию «бинарное дерево», приведите пример структуры бинарного дерева. Приведите пример объявления бинарного дерева.

*Бинарное дерево* – это конечное множество элементов, которое либо пусто, либо содержит элемент (*корень*), связанный с двумя различными бинарными деревьями, называемыми *левым и правым поддеревьями*. Каждый элемент бинарного дерева называется *узлом*. Связи между узлами дерева называются его *ветвями*.

*struct BinareTree // структура узла бинарного дерева*

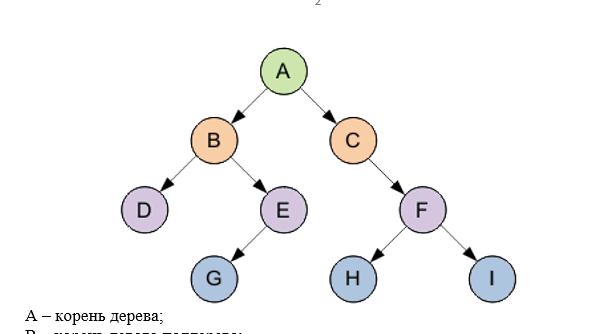
*{*

*DataField data\_field; //поле для данных (информационной части)*

*BinareTree\* left = nullptr; // левый потомок*

*BinareTree\* right = nullptr; // правый потомок*

*};*



1. Перечислите логические операции, составьте таблицу истинности.

Существуют несколько типов логических операций

& и

| или

^ исключающее или

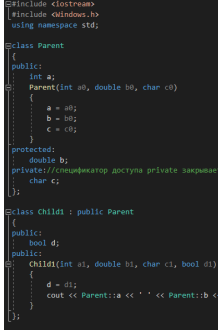
|  |  |  |
| --- | --- | --- |
| & | + | - |
| + | + | - |
| - | - | - |

|  |  |  |
| --- | --- | --- |
| | | + | - |
| + | + | + |
| - | + | - |

|  |  |  |
| --- | --- | --- |
| ^ | + | - |
| + | - | + |
| - | + | - |

1. Дайте определение понятию «наследование». Какие поля и методы наследуются. Перечислите типы наследования. Приведите пример наследования.

Наследование – это один из трех основных принципов ООП, предполагающий создание иерархии классов (объектов классов), в которой объекты-потомки наследуют все члены (свойства и поведение) своих классов-предков. Описывать заново явно в коде унаследованные от родителей члены у потомков не нужно. Потомки имеют унаследованные и свои «личные» свойства и поведения.



В C ++ есть несколько типов наследования:

* публичный (public)- публичные (public) и защищенные (protected) данные наследуются без изменения уровня доступа к ним
* защищенный (protected) — все унаследованные данные становятся защищенными
* приватный (private) — все унаследованные данные становятся приватными

1. Опишите выполнение операторов цикла, их формы и синтаксис. Приведите пример вычисления суммы чисел от 1 до 100 включительно.

Int sum = 0;

For(int I = 0;i<=100;i++)

{

Sum+=I;

}

1. Дайте определение понятию «хеширование». Приведите пример хеш-функции.

Процесс преобразования данных называется хешированием.

Int hash(int k)

{

Return (k%10);

}

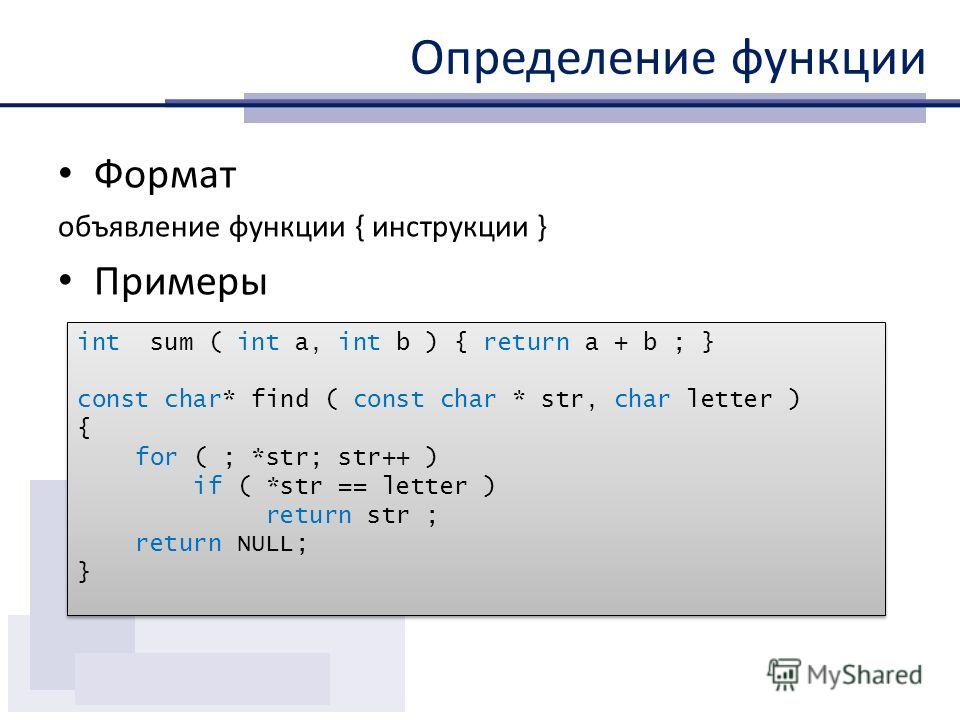
1. Перечислите типы данных, правила записи идентификаторов. Приведите пример.

Существую такие типы данных как: bool, int , double, char, float.

**Идентификатор** может состоять только из букв (нижнего и верхнего регистра) латинского алфавита, цифр и символов подчёркивания. Это означает, что все другие символы и пробелы — запрещены.

1. Дайте определение понятию «функция». Укажите синтаксис объявления и определения функции. Приведите пример объявления и определения функции.

Функция – изолированный именованный блок кода, имеющий определенное назначение.



1. Дайте определение понятию «класс». Опишите принцип работы с конструктором (с параметрами, без параметров, копирования, по умолчанию).

Класс является абстрактным типом данных, определяемым программистом, и представляет собой модель реального объекта в виде данных и функций для работы с ними.

Class Human

{

private:

Int age;

Public:

Human()//конструктор без параметров

{

}

Human(int x)//конструктор с параметрами

{

SetAge(x);

}

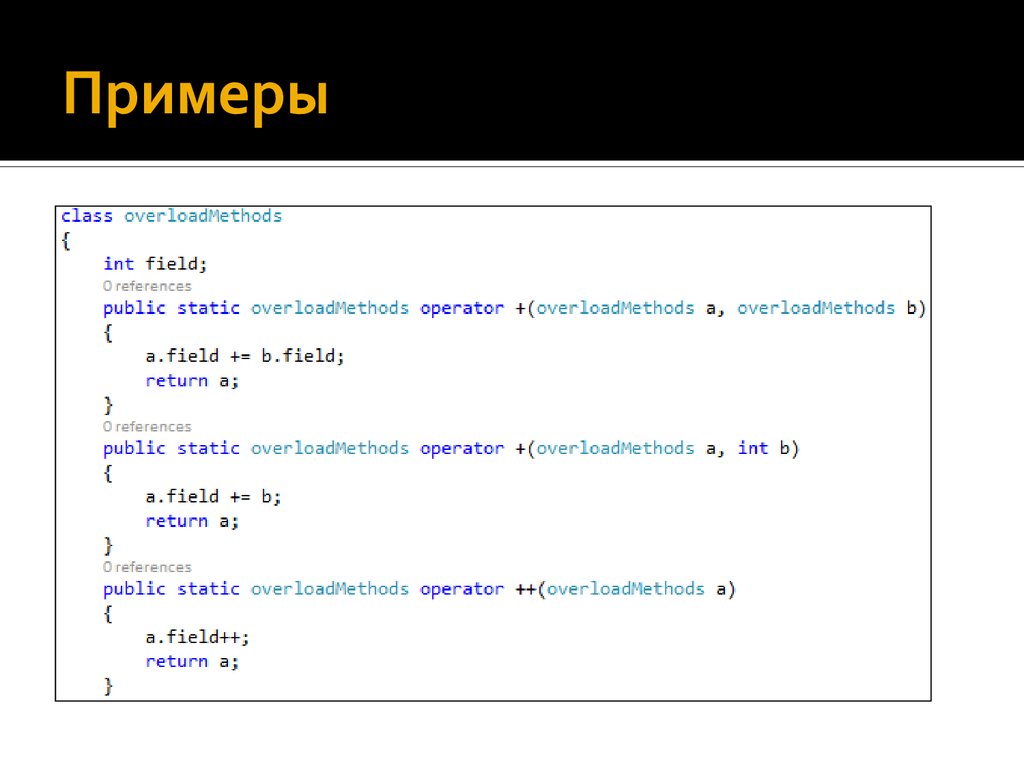
};

В классе допустимо создавать несколько конструкторов, если это необходимо. Имена, согласно пункту 2 нашего списка, будут одинаковыми. Компилятор будет их различать по передаваемым параметрам (как при перегрузке функций). Если мы не передаем в конструктор параметры, он считается конструктором по умолчанию;

1. Дайте определение понятию «полиморфизм». Объясните принцип полиморфизма, опишите синтаксис. Приведите пример.

Полиморфизм – (от греч. poli – много, morpheo – форма, поведение) – один из трех основных принципов ООП, заключающийся в возможности определения единого по имени действия (поведения), применимого ко всем объектам иерархии наследования (иерархии классов), причем каждый объект может реализовать это действие (поведение) собственным способом (в зависимости от того, к какому именно классу из данной иерархии этот объект относится).

Виртуальные методы – это методы, объявляемые в родительском классе с ключевым словом virtual и переопределяемые (перегружаемые) в дочерних классах.



1. Дайте определение понятию «тип данных». Перечислите типы данных по типу возвращаемых значений функций.

Каждая переменная имеет определенный тип. Тип данных определяет какие значение может принимать переменная.

Существую такие типы данных как: bool, int , double, char, float

Функция может возвращать любые типы данных

1. Дайте определение понятию «перегрузка функций». Приведите пример перегрузки функции.

Перегрузка операторов (перегрузка операций) – случай полиморфизма в С++, когда общеизвестный допустимый оператор приобретает новый функционал взамен старого при условии, что этот оператор применяется с объектом класса, для которого он перегружен.

class Counter

{

public:

Counter operator ++ (int)//перегрузка оператора # для класса ААА

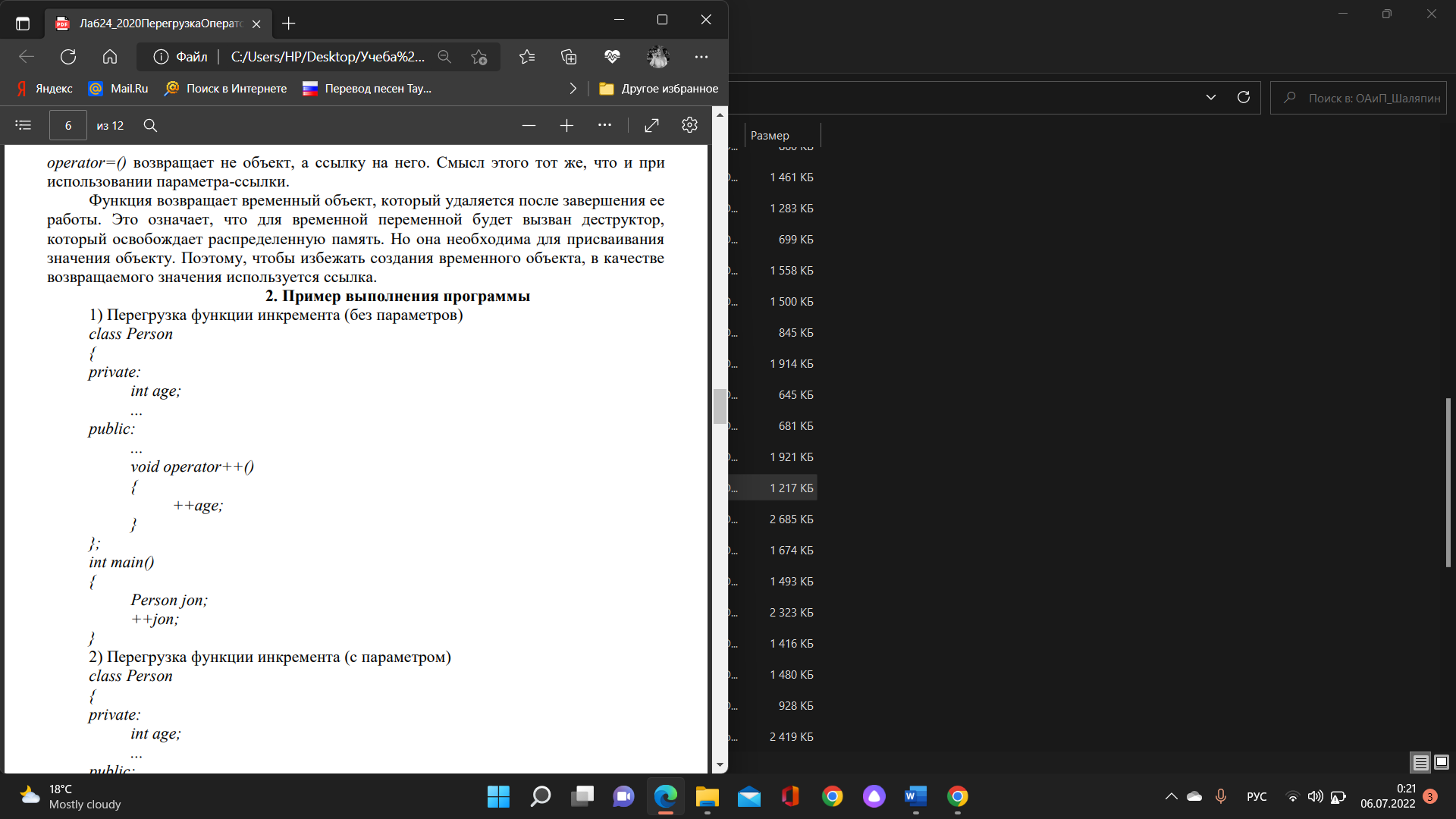
{

Count++;

Return \*this;

}

};

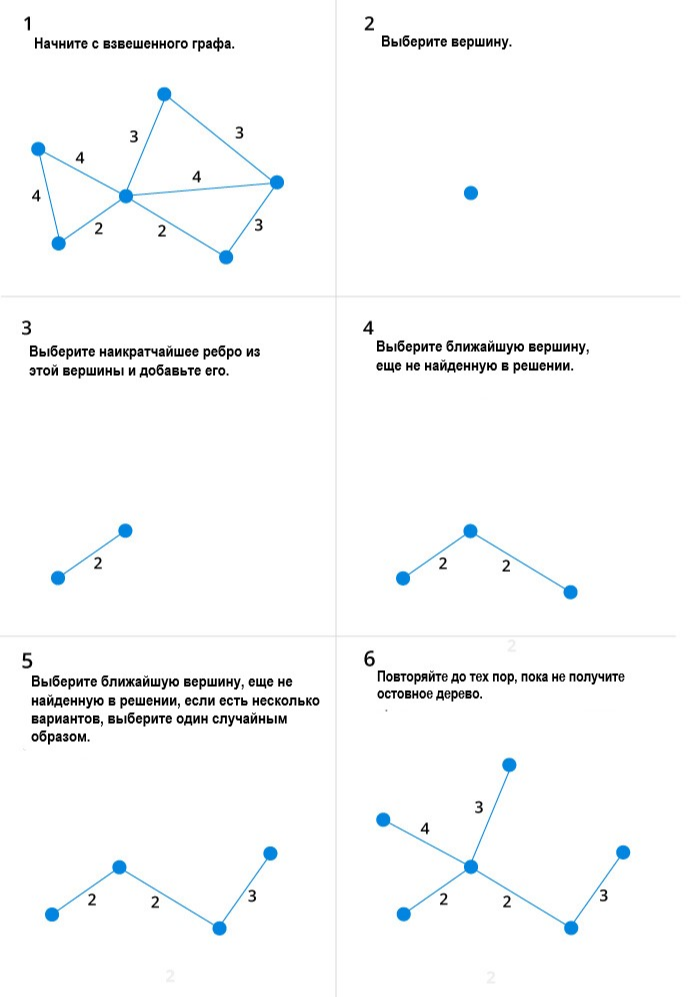


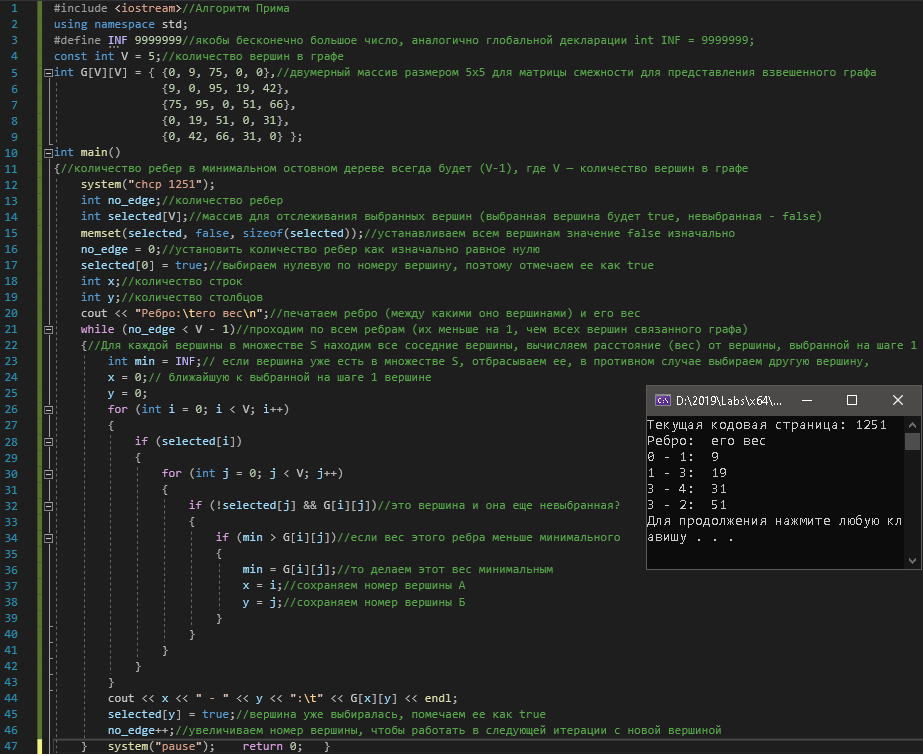
1. Дайте определение понятию «минимального остовного дерева». Опишите алгоритм Р. Прима. Приведите пример данного алгоритма.

**Минимальное остовное дерево** (или **минимальное** покрывающее **дерево**) в связанном взвешенном неориентированном графе — это **остовное дерево** этого графа, имеющее минимальный возможный вес, где под весом **дерева** понимается сумма весов входящих в него рёбер.

Неформальное описание алгоритма:

1. Выбрать корневую вершину V;
2. Пометить вершину V как посещенную;
3. Для каждой смежной с V вершины установить затраты кратчайшего ребра;
4. Выбрать не посещённую вершину с наименьшим весом ребра в качестве текущей вершины;
5. Пометить вершину V как посещеннуюи добавить связывающее ребро в остовное дерево;
6. Повторить все шаги со второго, пока не будут посещены все вершины.
7. Продолжайте добавлять ребра, пока не достигнете всех вершин

[](https://evileg.com/users/mafulechka/albums/photo/827/)



1. Дайте определение понятию «граф» и приведите способы его хранения.

**Граф** – совокупность точек, соединенных линиями. Точки называются **вершинами**, или узлами, а линии – **ребрами**, или дугами.

Граф может быть представлен (сохранен) несколькими способами:

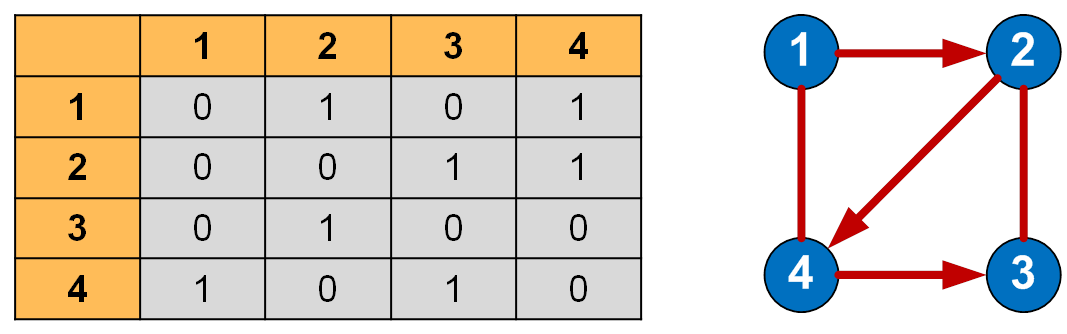
1. матрица смежности;
2. матрица инцидентности;
3. список смежности (инцидентности);
4. список ребер.

Использование двух первых методов предполагает хранение графа в виде двумерного массива (матрицы). Размер массива зависит от количества вершин и (или) ребер в конкретном графе.

**Матрица смежности графа** – это квадратная матрица, в которой каждый элемент принимает одно из двух значений: 0 или 1. Число строк матрицы смежности равно числу столбцов и соответствует количеству вершин графа:

0 – соответствует отсутствию ребра,

1 – соответствует наличию ребра.



Когда из одной вершины в другую проход свободен (имеется ребро), в ячейку заносится 1, иначе – 0. Все элементы на главной диагонали равны 0, если граф не имеет петель.

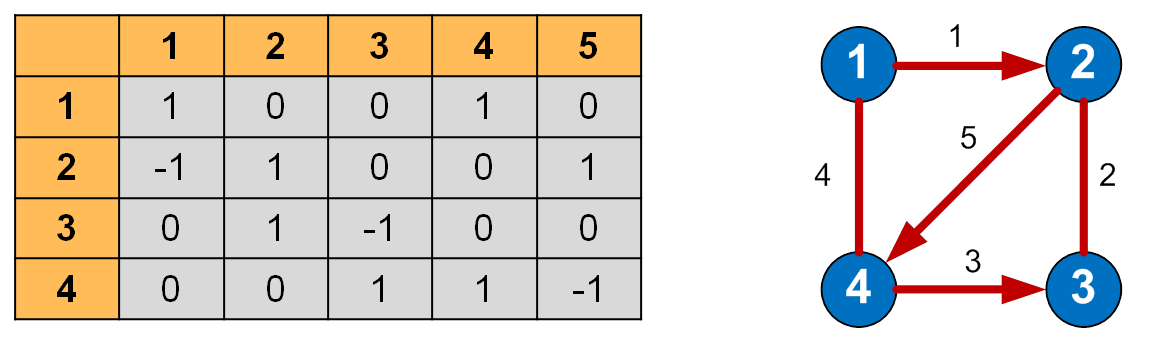
**Матрица инцидентности**

**Матрица инцидентности (инциденции) графа** – это матрица, количество строк в которой соответствует числу вершин, а количество столбцов – числу рёбер. В ней указываются связи между инцидентными элементами графа: ребро (дуга) и вершина.

В неориентированном графе, если вершина инцидентна ребру, то соответствующий элемент равен 1, в противном случае – элемент равен 0.

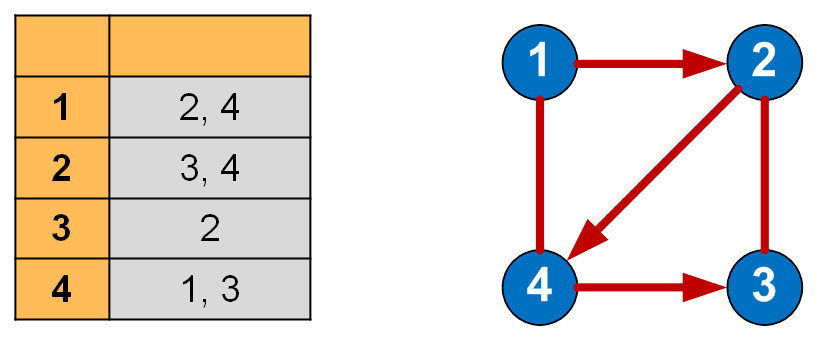
В ориентированном графе, если ребро выходит из вершины, то соответствующий элемент равен 1, если ребро входит в вершину, то соответствующий элемент равен -1, если ребро отсутствует, то элемент равен 0.

Матрица инцидентности для своего представления требует нумерации рёбер, что не всегда удобно.



**Список смежности (инцидентности)**

Если количество ребер графа по сравнению с количеством вершин невелико, то значения большинства элементов матрицы смежности будут равны 0. В таком случае использование данного метода нецелесообразно. Для подобных графов имеются более оптимальные способы их представления. По отношению к памяти компьютера **списки смежности** менее требовательны, чем матрицы смежности. Такой список можно представить в виде таблицы, столбцов в которой – 2, а строк – не больше, чем вершин в графе. В каждой строке в первом столбце указана вершина выхода, а во втором столбце – список вершин, в которые входят ребра из текущей вершины.



**Преимущества списка смежности**:

1. Рациональное использование памяти.
2. Позволяет быстро перебирать соседей у данной вершины.
3. Позволяет проверять наличие ребра и удалять его.

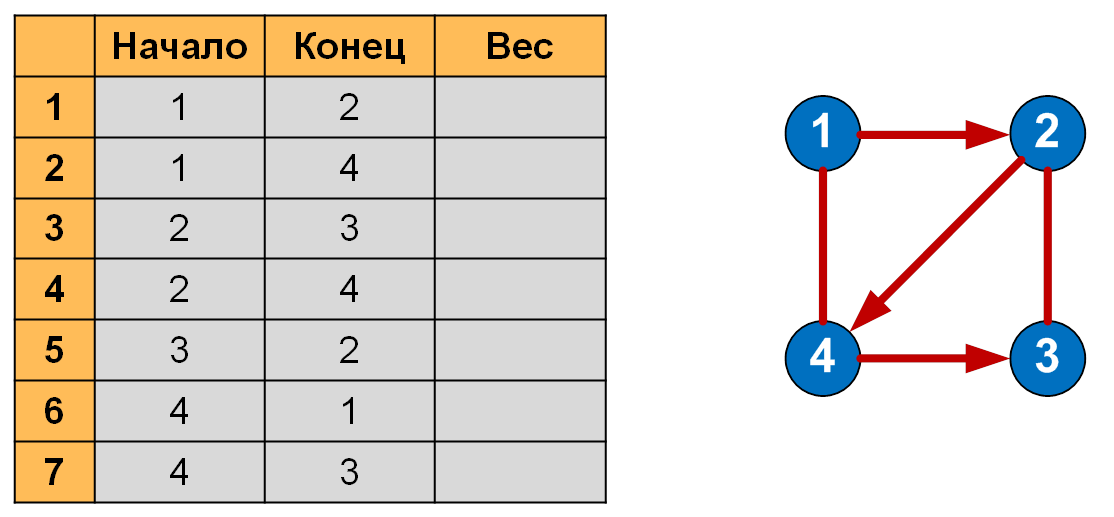
**Недостатки списка смежности**:

1. При работе с **насыщенными графами** (с большим количеством рёбер) скорости может не хватать для обработки всего большого графа.
2. Нет быстрого способа проверить, существует ли ребро между двумя вершинами.
3. Количество вершин графа должно быть известно заранее.
4. Для взвешенных графов приходится хранить список, элементы которого должны содержать два значащих поля (см. далее), что усложняет код:
   1. номер вершины, с которой соединяется текущая вершина;
   2. вес ребра (число).

**Список рёбер**

В **списке рёбер** в каждой строке записываются две смежные вершины и вес соединяющего их ребра (для взвешенного графа).

Количество строк в списке ребер всегда должно быть равно величине, получающейся в результате сложения ориентированных рёбер с удвоенным количеством неориентированных рёбер.



Какой способ представления графа лучше? Ответ зависит от отношения между числом вершин и числом рёбер. Число ребер может быть довольно малым (такого же порядка, как и количество вершин) или довольно большим (если граф является полным). Графы с большим числом рёбер называют **плотными**, с малым – **разреженными**. Плотные графы удобнее хранить в виде матрицы смежности, разреженные – в виде списка смежности.

1. Дайте определение понятию «функция». Объясните различие передачи параметров по значению и ссылке. Приведите пример передачи параметров по значению и ссылке.

Функция – изолированный именованный блок кода, имеющий определенное назначение.

При передаче параметров по ссылке передается ссылка на объект, через которую мы можем манипулировать самим объектов, а не просто его значением.

Передача ссылки происходит при помощи & перед именем переменной

void square(int &a, int &b);

1. Опишите работу комментариев и способы их записей. Для чего нужны комментарии, приведите пример их в коде.

**Комментарий** – это заметка, удобная для чтения программисту, которая вставляется непосредственно в исходный код программы. Комментарии игнорируются компилятором и предназначены только для использования программистом.

Комментарии можно оставлять в конце строки после символа // или комбинацией /\* <текст> \*/для созданий многострочного комментария

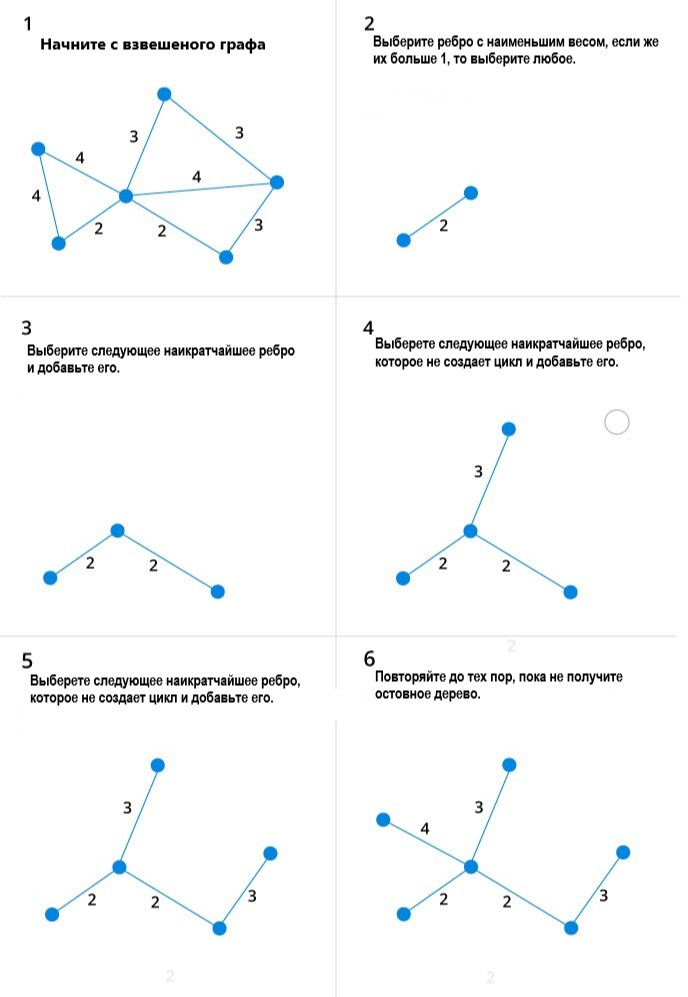
1. Опишите алгоритм Дж. Краскала. Дайте определение понятию «минимального остовного дерева». Приведите пример данного алгоритма.

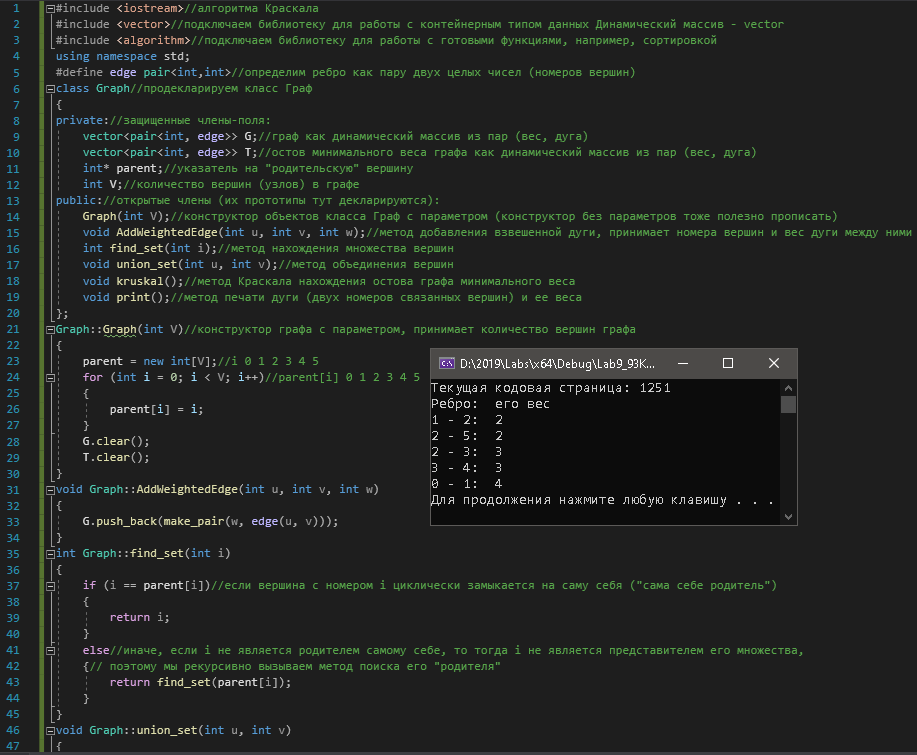
Шаги для реализации алгоритма Краскала следующие:

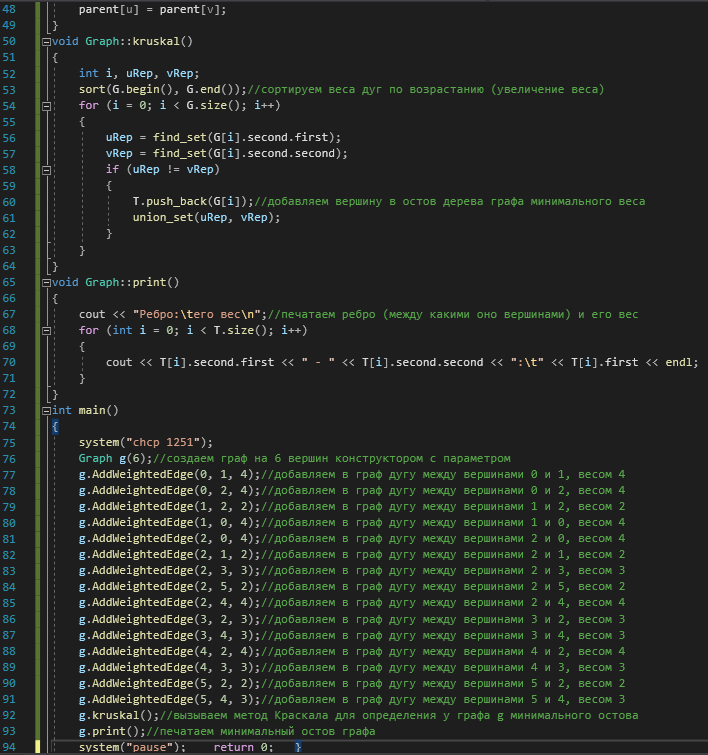
1. Сортировать все ребра от малого веса до высокого.

2. Возьмите ребро с наименьшим весом и добавьте его в остовное дерево. Если добавление ребра создало цикл, то отклоните это ребро.

3. Продолжайте добавлять ребра, пока не достигнете всех вершин.

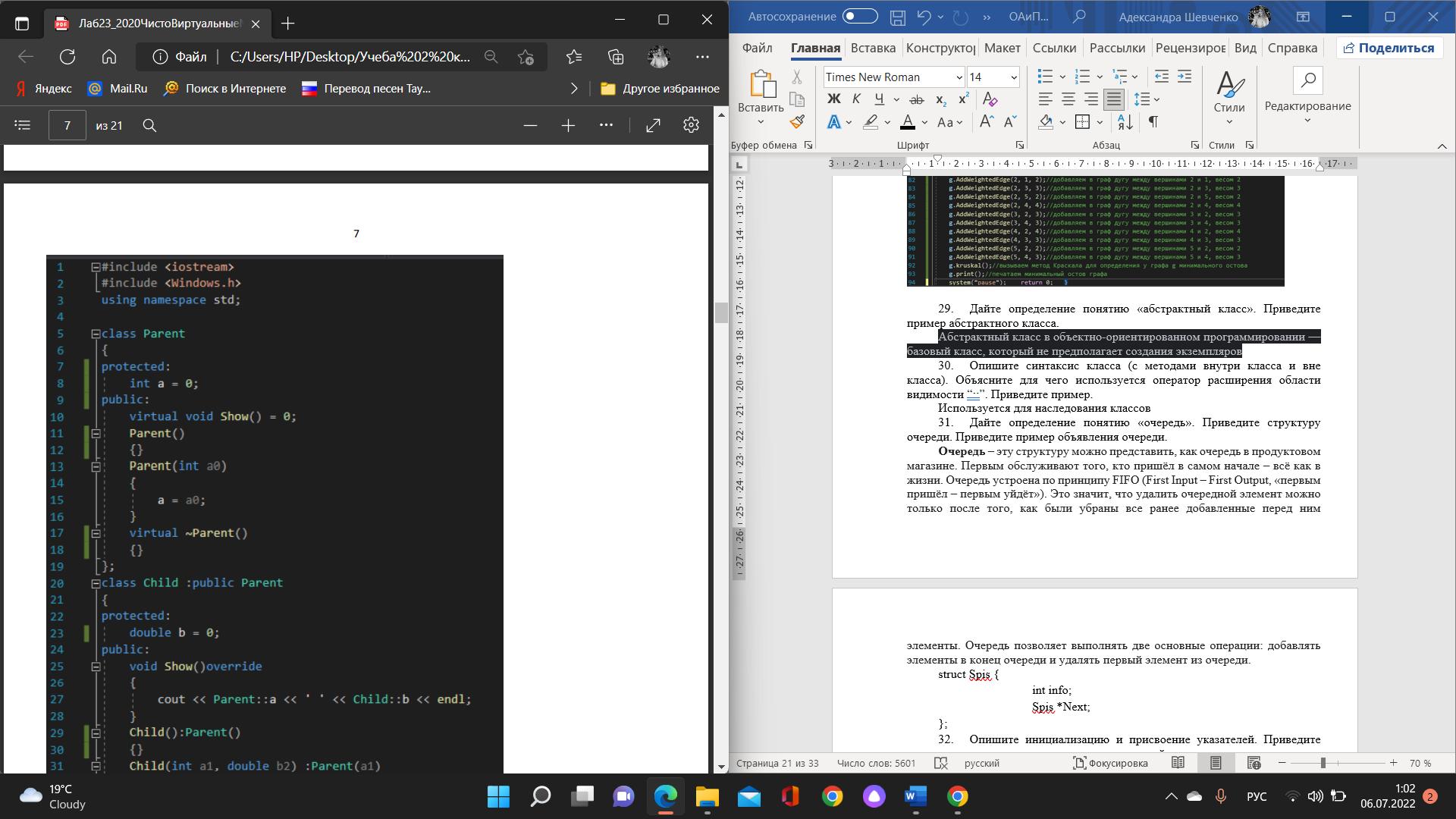
[](https://evileg.com/users/mafulechka/albums/photo/825/)





1. Дайте определение понятию «абстрактный класс». Приведите пример абстрактного класса.

Абстрактный класс в объектно-ориентированном программировании — базовый класс, который не предполагает создания экземпляров



1. Опишите синтаксис класса (с методами внутри класса и вне класса). Объясните для чего используется оператор расширения области видимости “::”. Приведите пример.

Выше

1. Дайте определение понятию «очередь». Приведите структуру очереди. Приведите пример объявления очереди.

**Очередь** – эту структуру можно представить, как очередь в продуктовом магазине. Первым обслуживают того, кто пришёл в самом начале – всё как в жизни. Очередь устроена по принципу FIFO (First Input – First Output, «первым пришёл – первым уйдёт»). Это значит, что удалить очередной элемент можно только после того, как были убраны все ранее добавленные перед ним элементы. Очередь позволяет выполнять две основные операции: добавлять элементы в конец очереди и удалять первый элемент из очереди.

struct Spis {

int info;

Spis \*Next;

};

1. Опишите инициализацию и присвоение указателей. Приведите пример инициализации и присвоение указателей

Выше

1. Дайте определение понятию «массив». Приведите пример объявления динамического массива.

Массив — это сложный (составной, структурированный) тип данных, который характеризуется следующим:

• элементы массива имеют одинаковый тип в отличие от структур struct, поэтому каждый элемент массива занимает одинаковый объём памяти;

• массив располагается в оперативной памяти, а не на внешнем устройстве, как файлы;

• элементы массива занимают подряд идущие ячейки, в отличие, например, от списков

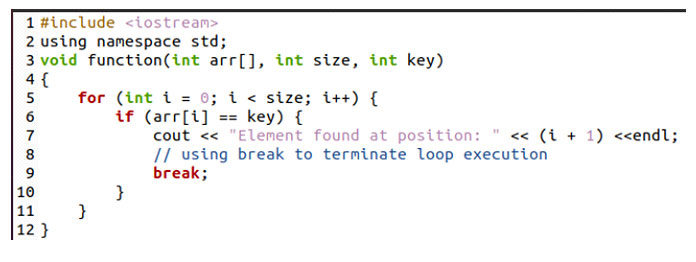
Int I;

Cin>n;

Int \* mas = new int [n];

1. Перечислите ситуации использования оператора break. Приведите пример ситуации использования оператора break.

Break используется для входа из цикла. Может применяться для выхода при достижении определенного значения в цикле



1. Дайте определение понятию «циклический (кольцевой) список». Приведите структуры списка. Приведите пример объявления кольцевого списка.

**Кольцевой список** (кольцо) – список, в котором последний элемент хранит указатель не на NULL, а на первый элемент этого же списка, в результате чего список становится «закольцованным» (циклическим, круговым) и по нему можно бесконечно ходить, поскольку у него конец переходит в начало.

Struct list

{

Int field;//поле данных

List\* next;//следующий элемент

List\* prev;//предыдущий

};

1. Перечислите типы констант и способы их определения. Приведите пример.

**Константы** бывают числовые, символьные и строковые

Const char text = “Q”;

Const int I = 0;

Const double r = 3.5;

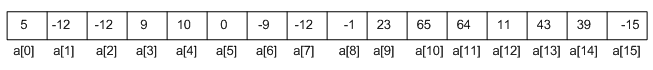
1. Дайте определение понятию «инкапсуляция». Приведите пример инкапсуляция.

**Инкапсуляция** – это один из трех основных принципов ООП, предполагающий сокрытие деталей реализации класса от доступа извне данного класса (и, соответственно, извне данного объекта класса) и недопущение изменения внутреннего состояния объекта класса (значений, хранимых в его полях) без ведома самого данного объекта, а также недопущение изменений в состоянии объекта класса, которые приведут к его некорректной работе и/или уничтожению (не путать с деструктором, который является членом экземпляра класса и для того и существует, чтобы очищать оперативную память от данного объекта класса, уничтожая тем самым этот объект).

1. Опишите работу с одномерным массивом. Представление в памяти. Способы сортировки массивов. Приведите пример.

Сортировки могут быть: «пузырьковой», выбором минимального, вставками, Хоара и Шелла.

Одномерный массив — массив, с одним параметром, характеризующим количество элементов одномерного массива. Фактически одномерный массив — это массив, у которого может быть только одна строка, и n-е количество столбцов. Столбцы в одномерном массиве — это элементы массива. На рисунке 1 показана структура целочисленного одномерного массива **a**. Размер этого массива — 16 ячеек.



Заметьте, что максимальный индекс одномерного массива **a** равен 15, но размер массива 16 ячеек, потому что нумерация ячеек массива всегда начинается с 0. Индекс ячейки – это целое неотрицательное число, по которому можно обращаться к каждой ячейке массива и выполнять какие-либо действия над ней (ячейкой).

1. Дайте определение понятию «алгоритм». Какие есть способы записи алгоритма, Приведите пример алгоритма.

Алгоритм – последовательность действия приводящие к решению задач.

Существуют несколько типов записи алгоритмов

Человеческий

Код программы

Графический

Виды алгоритмов

Линейный

Разветвляющий

циклический

1. Опишите принцип работы перегрузки операций, приведите синтаксис. Приведите пример перегрузки операции «+».

Перегрузка операторов (перегрузка операций) – случай полиморфизма в С++, когда общеизвестный допустимый оператор приобретает новый функционал взамен старого при условии, что этот оператор применяется с объектом класса, для которого он перегружен.

Counter operator + (Counter c1, Counter c2)

{

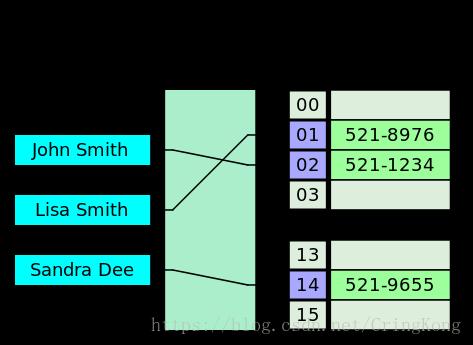
    return Counter(c1.seconds + c2.seconds);

}

Выше

1. Дайте определение понятию «хеш-таблица». Опишите структуру хеш-таблицы. Приведите пример объявления хеш-таблицы.

**Хеш**-**таблица** (hash table) — это специальная структура данных для хранения пар ключей и их значений. По сути это ассоциативный массив, в котором ключ представлен в виде **хеш**-функции



Как показано выше, теперь есть набор пар ключ-значение имен и телефонных номеров, **посредством хеш-функции имя может быть сопоставлено с индексом массива, а соответствующая позиция массива хранит номер телефона пользователя.**

1. Опишите способы доступа к элементам массива. Приведите пример объявления двухмерно массива и проинициализируйте элементы массива при помощи датчика случайных чисел в диапазоне (-10, 10).

Доступ к элементам массива в языке С++ осуществляется при помощи индекса или указателем

#include <stdlib>

#include <time.h>

Srand(time(NULL));

Const int m = 10;

Const int n = 10;

Int a = -10;

Int b = 10;

Int mas[m][n];

For(int I = 0;i<m;i++)

{

For(int j = 0;j < n;j++)

{

Mas[i][j] = a+rand()%(b-a+1);

}

}

1. Дайте определение понятию «виртуальная функция». Приведите пример.

Виртуальные методы – это методы, объявляемые в родительском классе с ключевым словом virtual и переопределяемые (перегружаемые) в дочерних классах

virtual void Show()

{

cout << a << endl;

}

1. Объясните разницу между классом и структурой. Приведите пример объявления класса «Студент».

Class Stud

{

private:

Int age;

Int mark;

public:

Stud(int a, int m)

{

SetAge(a);

SetMark(m);

}

GetAge()//геттер

{

return age;

}

SetAge(int a)//сеттер

{

age = a;

}

Getmark()

{

return mark;

}

SetMark(int m)

{

mark = m;

}

};

Если делать без геттера и сеттера

Public:

Stud(int a, int m)

{

age= a;

Mark = m;

}

1. Дайте определение понятию «кратчайший путь в графе» Опишите алгоритм Флойда.

Кратчайший путь в графе – это путь от точки а к точке б с минимальным весом ребер.

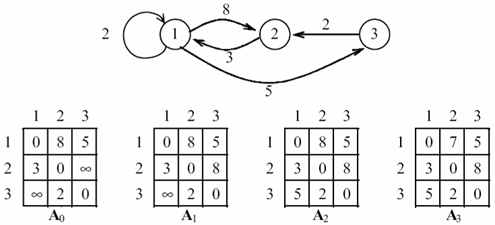
Метод Флойда непосредственно основывается на том факте, что в графе с положительными весами ребер всякий неэлементарный (содержащий более 1 ребра) кратчайший путь состоит из других кратчайших путей.

Этот алгоритм более общий по сравнению с алгоритмом Дейкстры, так как он находит кратчайшие пути между любыми двумя вершинами графа. В алгоритме Флойда используется матрица A размером n на n, в которой вычисляются длины кратчайших путей. Элемент A[i, j] равен расстоянию от вершины i к вершине j, которое имеет конечное значение, если существует ребро (i, j), и равен бесконечности в противном случае.

Основная идея алгоритма Флойда: пусть есть три вершины i, j, k и заданы расстояния между ними. Если выполняется неравенство A[i, k] + A[k, j] < A[i, j], то целесообразно заменить путь i->j путем i->k->j. Такая замена выполняется систематически в процессе выполнения данного алгоритма.

Шаг 0. Определяем начальную матрицу расстояния A0 и матрицу последовательности вершин S0. Каждый диагональный элемент обеих матриц равен 0, таким образом показывая, что эти элементы в вычислениях не участвуют. Полагаем k = 1.

Основной шаг k. Задаем строку k и столбец k как ведущую строку и ведущий столбец. Рассматриваем возможность применения замены описанной выше, ко всем элементам A[i,j] матрицы Ak-1. Если выполняется неравенство A[i,k]+A[k,j]<A[i,j], (i\ne k, j\ne k, i\ne j), тогда выполняем следующие действия: 1) создаем матрицу Ak путем замены в матрице Ak-1 элемента A[i, j] на сумму A[i,k] + A[k,j]; 2) создаем матрицу Sk путем замены в матрице Sk-1 элемента S[i,j] на k. Полагаем k = k + 1 и повторяем шаг k. Таким образом, алгоритм Флойда делает n итераций, после i-й итерации матрица А будет содержать длины кратчайших путей между любыми двумя парами вершин при условии, что эти пути проходят через вершины от первой до i-й. На каждой итерации перебираются все пары вершин и путь между ними сокращается при помощи i-й.



Демонстрация алгоритма Флойда

//Описание функции алгоритма Флойда

void Floyd(int n, int \*\*Graph, int \*\*ShortestPath)

{

int i, j, k;

int Max\_Sum = 0;

for ( i = 0 ; i < n ; i++ )

for ( j = 0 ; j < n ; j++ )

Max\_Sum += ShortestPath[i][j];

for ( i = 0 ; i < n ; i++ )

for ( j = 0 ; j < n ; j++ )

if ( ShortestPath[i][j] == 0 && i != j )

ShortestPath[i][j] = Max\_Sum;

for ( k = 0 ; k < n; k++ )

for ( i = 0 ; i < n; i++ )

for ( j = 0 ; j < n ; j++ )

if ((ShortestPath[i][k] + ShortestPath[k][j]) < ShortestPath[i][j])

ShortestPath[i][j] = ShortestPath[i][k] + ShortestPath[k][j];

}

Заметим, что если граф неориентированный, то все матрицы, получаемые в результате преобразований симметричны и, следовательно, достаточно вычислять только элементы, расположенные выше главной диагонали.

1. Опишите работу с текстовыми файлами. Перечислите функции для работы с ними. Приведите пример как вывести все содержимое текстового файла на экран.

Для работы с файлами используются специальные типы данных, называемые потоками. Поток **ifstream**служит для работы с файлами в режиме чтения, а **ofstream** в режиме записи. Для работы с файлами в режиме как записи, так и чтения служит поток **fstream**.

В программах на C++ при работе с текстовыми файлами необходимо подключать библиотеки **iostream** и**fstream**.

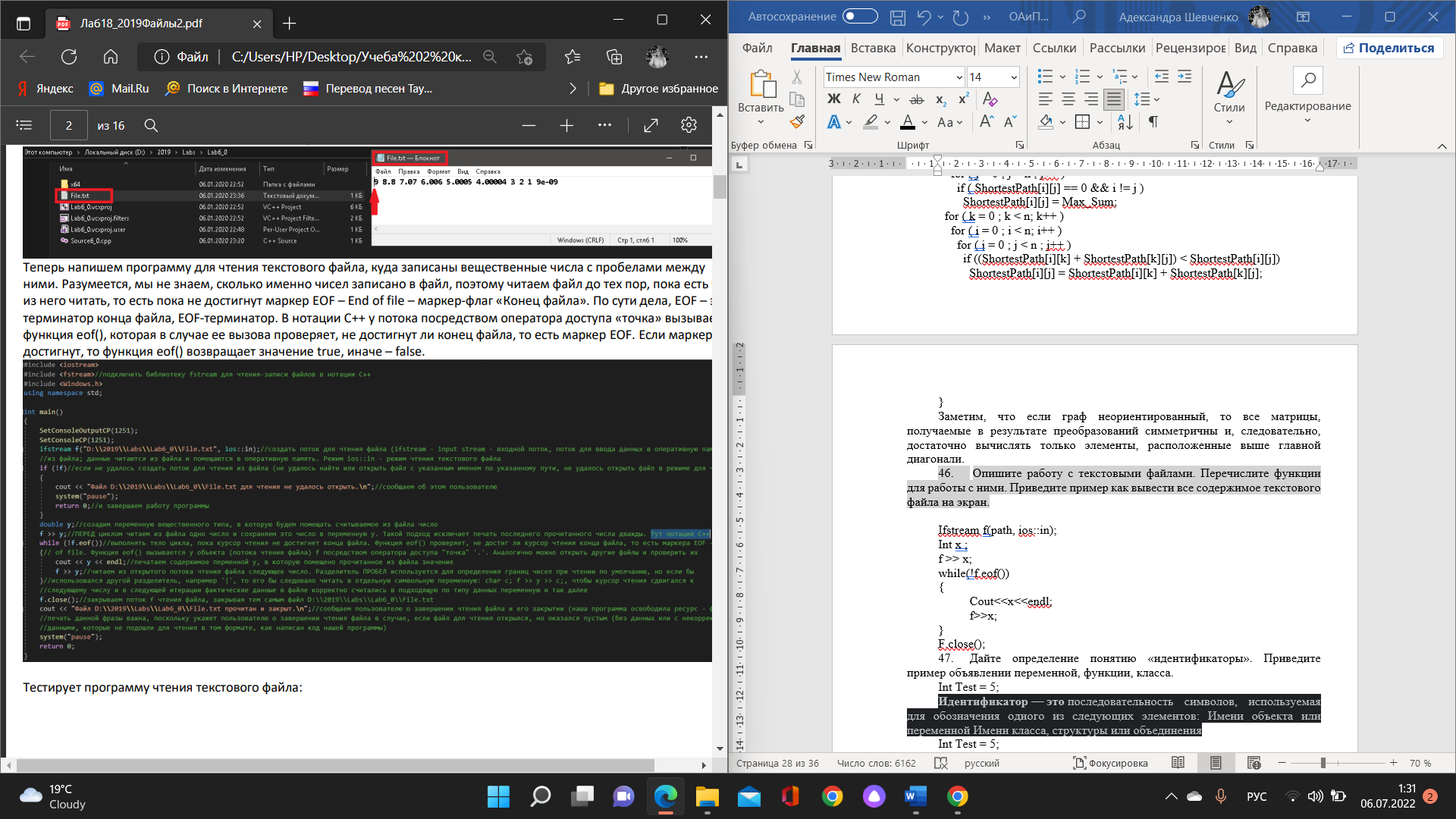
Для того чтобы записывать данные в текстовый файл, необходимо:

1. описать переменную типа **ofstream**.
2. открыть файл с помощью функции **open**.
3. вывести информацию в файл.
4. обязательно закрыть файл.

Для считывания данных из текстового файла, необходимо:

1. описать переменную типа **ifstream**.
2. открыть файл с помощью функции **open**.
3. считать информацию из файла, при считывании каждой порции данных необходимо проверять, достигнут ли конец файла.
4. закрыть файл.

* ios::in — открыть файл в режиме чтения данных; режим является режимом по умолчанию для потоков ifstream;
* ios::out — открыть файл в режиме записи данных (при этом информация о существующем файле уничтожается); режим является режимом по умолчанию для потоков ofstream;
* ios::app — открыть файл в режиме записи данных в конец файла;
* ios::ate — передвинуться в конец уже открытого файла;
* ios::trunc — очистить файл, это же происходит в режиме ios::out;
* ios::nocreate — не выполнять операцию открытия файла, если он не существует;
* ios::noreplace — не открывать существующий файл.



1. Дайте определение понятию «идентификаторы». Приведите пример объявлении переменной, функции, класса.

Int Test = 5;

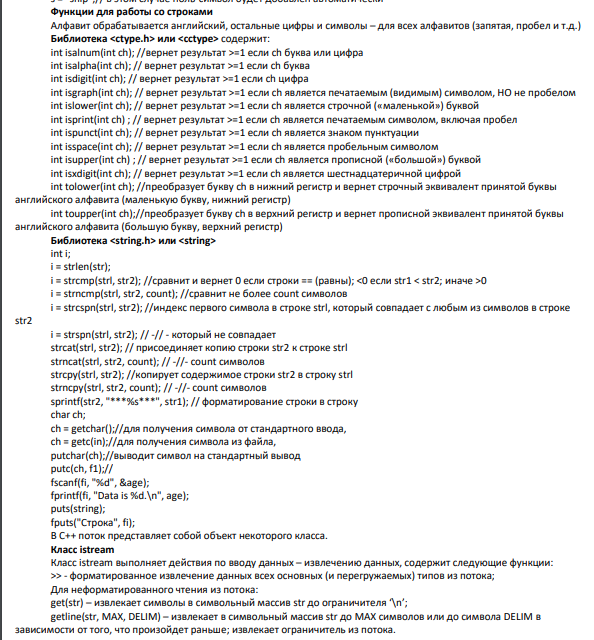
**Идентификатор** — **это** последовательность символов, используемая для обозначения одного из следующих элементов: Имени объекта или переменной Имени класса, структуры или объединения

Int Test = 5;

Void Func();

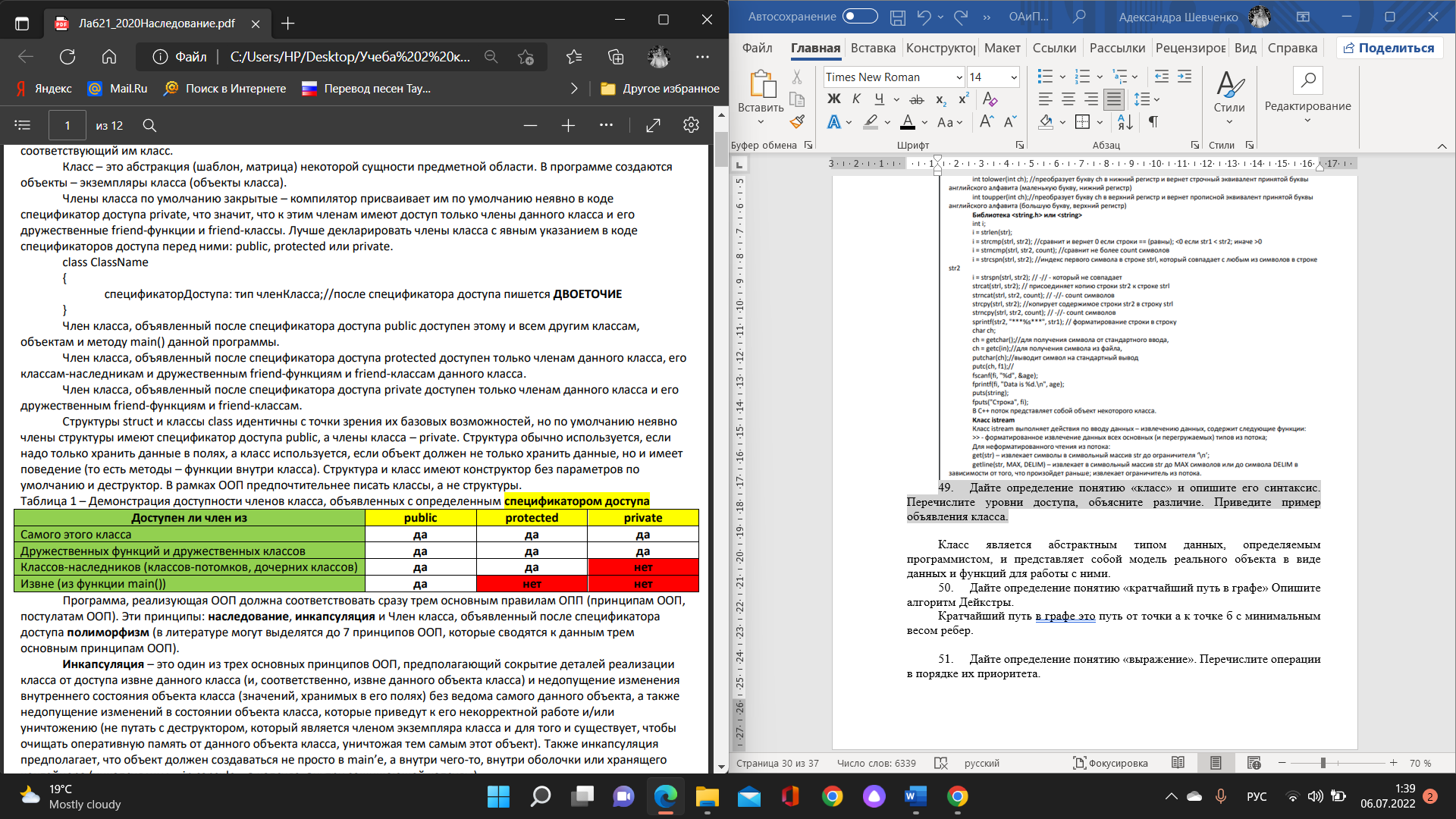
Class Stud;

1. Опишите работу с символьными переменными и строками. Перечислите функции для работы с ними. Приведите пример подсчета цифр в текстовой строке(string).



1. Дайте определение понятию «класс» и опишите его синтаксис. Перечислите уровни доступа, объясните различие. Приведите пример объявления класса.

Каждый класс может состоять из членов. Члены класса, это: 1) поле – переменная, массив, структура для хранения данных определенного типа: int number; double weight; 2) свойство – метод, предназначенный для обеспечения доступа к полю класса (то есть через свойство можно получить значение конкретного поля (get) и/или присвоить конкретному полю значение (set). Получить значение конкретного поля можно почти всегда, но если поле константное или его значение вычисляется, зависит от других значений и условий, то присвоить ему значение будет нельзя (то есть свойство set в таком случае создавать программисту нельзя). Не надо создавать свойства для присвоения значений полям, которые получают конкретные смысловые значения, а не любые (в классе Калькулятор поле Результат нет смысла делать со свойством, устанавливающим ему значение, поскольку лучше, чтобы значение поля Результат устанавливал метод, вычисляющий результат из двух чисел (см. ниже)). 3) конструктор – особый метод, который создает экземпляры своего одноименного класса (объекты своего класса, переменные типа своего класса). Класс – это общая конструкция, схема, идея, общий шаблон, «матрица» некого явления, процесса или предмета (машина, компьютер, человек). Мы описываем класс подобно структуре struct, а после можем создавать объекты данного класса (например, конкретная машина Honda Accord III 1989 года выпуска в кузове седан с двигателем В20А I4 весом 1255 кг и регистрационным номером BY6426IP-1, принадлежащая водителю Ильину И.К. – то есть конкретный автомобиль, реально существовавший, существующий сейчас или могущий быть созданным (обнаруженным) в будущем; объект обычно можно потрогать руками, а сам класс как таковой – нет). Конструктор класса Машина создает этот объект – конкретный автомобиль Honda Accord III, экземпляр класса Машина. Конструктор пишется программистом. Конструктор имеет имя, всегда совпадающее с именем его класса. Конструктор – это метод, поэтому пишется со скобками(), может принимать ни одного или несколько входных параметров и никогда ничего не возвращает (не имеет return'a). Конструктором можно создавать переменные типа своего класса (объекты), а также указатели и ссылки типа своего класса. Переменные типа некого одного класса можно объединять в массивы. То есть класс – это пользовательский (программистский) тип данных, которым можно пользоваться как базовым типом данных, но создаваемая переменная будет иметь свою внутреннюю структуру из своих полей и у нее можно вызвать ее личные методы. 4) метод – член класса, представляющий собой функцию, которая находится в данном классе (принадлежит ему и его объектам, может быть вызвана только у экземпляров данного класса). Метод, подобно функции, может принимать ни одного или несколько входных параметров и одно значение возвращать return'ом или ни одного (void). Метод видит содержимое поля своего экземпляра класса (даже закрытые от других экземпляров классов – private, поскольку метод сам находится в этом же (своем) экземпляре класса) и может изменять значения неконстантных полей своего класса. Методы обычно оформляются как внешние: в классе описывается прототип метода, а за классом с указанием пространства имен данного класса (ИмяКласса:: ) дается полное определение метода (тело метода). 5) деструктор – особый метод класса, который уничтожает объекты своего класса, когда они уже, по мнению программиста, не нужны. Деструктор очищает память от неиспользуемых объектов своего класса, аналогично коду delete. Если вызов конструктора «рождает» объект класса, то вызов для него деструктора – уничтожает этот объект, то есть означает смерть этого объекта.



Класс является абстрактным типом данных, определяемым программистом, и представляет собой модель реального объекта в виде данных и функций для работы с ними.

1. Дайте определение понятию «кратчайший путь в графе» Опишите алгоритм Дейкстры.

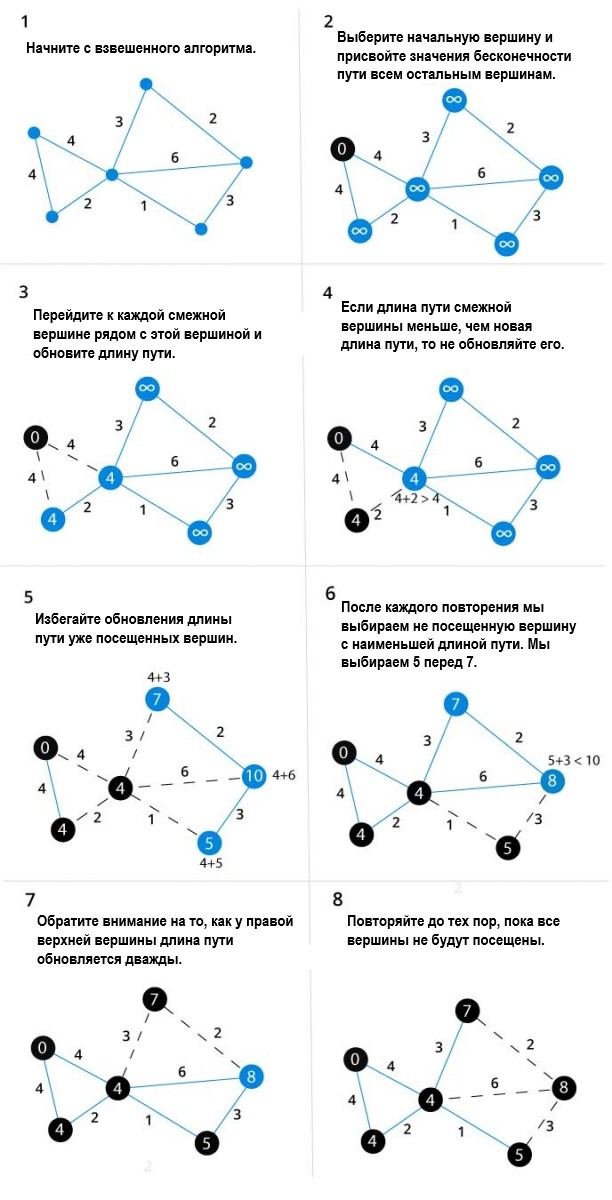
Кратчайший путь в графе это путь от точки а к точке б с минимальным весом ребер.

Рассмотрим пример нахождение кратчайшего пути. Дана сеть автомобильных дорог, соединяющих города области. Некоторые дороги односторонние. Найти кратчайшие пути от центра города до каждого города области. Для решения указанной задачи можно использовать алгоритм Дейкстры – алгоритм на графах, изобретённый нидерландским ученым Э. Дейкстрой в 1959 году. Находит кратчайшее расстояние от одной из вершин графа до всех остальных. Работает только для графов без рёбер отрицательного веса. Пусть требуется найти кратчайшие расстояния от 1-й вершины до всех остальных. Ниже приведены шаги. Входные данные: граф и начальная вершина src. Выходные данные: кратчайшее расстояние до всех вершин от src. Если попадается цикл отрицательного веса, то самые короткие расстояния не вычисляются и выводится сообщение о наличии такого цикла.

1) На этом шаге инициализируются расстояния от исходной вершины до всех остальных вершин как бесконечные, а расстояние до самого src принимается равным 0. Создается массив dist[] размера |V| со всеми значениями, равными бесконечности, за исключением элемента dist[src], где src – исходная вершина.

2) Вторым шагом вычисляются самые короткие расстояния. Следующие шаги нужно выполнять |V|-1 раз, где |V| – число вершин в данном графе. Произведите следующее действие для каждого ребра u-v: если dist[v] > dist[u] + вес ребра uv, то обновите dist[v]: dist [v] = dist [u] + вес ребра uv.

3) На этом шаге сообщается, присутствует ли в графе цикл отрицательного веса. Для каждого ребра u-v необходимо выполнить следующее: если dist[v] > dist[u] + вес ребра uv, то в графе присутствует цикл отрицательного веса. Идея шага 3 заключается в том, что шаг 2 гарантирует кратчайшее расстояние только если граф не содержит цикла отрицательного веса. Если мы снова переберем все ребра и получим более короткий путь для любой из вершин, это будет сигналом присутствия цикла отрицательного веса.

Как это работает? Как и в других задачах динамического программирования, алгоритм вычисляет кратчайшие пути снизу вверх. Сначала он вычисляет самые короткие расстояния, то есть пути длиной не более, чем в одно ребро. Затем он вычисляет кратчайшие пути длиной не более двух ребер и так далее. После *i*-й итерации внешнего цикла вычисляются кратчайшие пути длиной не более*i*ребер. В любом простом пути может быть максимум *|V|-1* ребер, поэтому внешний цикл выполняется именно *|V|-1* раз. Идея заключается в том, что если мы вычислили кратчайший путь с не более чем *i* ребрами, то итерация по всем ребрам гарантирует получение кратчайшего пути с не более чем *i + 1* ребрами.

1. Дайте определение понятию «выражение». Перечислите операции в порядке их приоритета.

**Выражения** — **это** последовательности операторов и операндов, используемые в следующих целях. Вычисление значения из операндов. Назначение объектов или функций. Создание побочных эффектов.

Скорее всего + - / \*

1. Опишите объявление, описание и вызов функции. Объясните различие формальных и фактических параметров функций. Приведите пример объявления и определения функции.

Вызов функции kek которая принимает 3 значения

X = kek(1,2,3);



1. Дайте определение понятию «константа» и приведите пример объявления константы.

**Константа** — это ограниченная последовательность символов алфавита языка, представляющая собой изображение фиксированного (неизменяемого) объекта.

Const int z = 5;

1. Опишите принцип работы перегрузки функций. Приведите пример перегрузки функции нахождения максимального числа функции от 1, 2 и 3 аргументов.
2. Дайте определение понятию «деструктор». В каких случаях вызывается деструктор.

Деструктор – особый метод класса, который уничтожает объекты своего класса, когда они уже, по мнению программиста, не нужны.

Деструкторы вызываются, когда происходит одно из следующих событий:

* Локальный (автоматический) объект с областью видимости блока выходит за пределы области видимости.
* Объект, выделенный с помощью **new** оператора, явно освобождается с помощью **delete**.
* Время существования временного объекта заканчивается.
* Программа заканчивается, глобальные или статические объекты продолжают существовать.
* Деструктор явно вызываться с использованием полного имени функции деструктора.

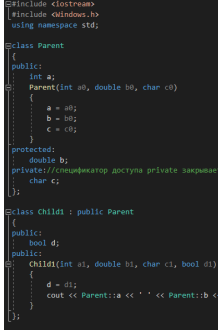
Деструкторы могут свободно вызывать функции-члена класса и осуществлять доступ к данным членов класса.

Существует два ограничения на использование деструкторов:

* Вы не можете взять его адрес.
* Производные классы не наследуют деструктор базового класса.

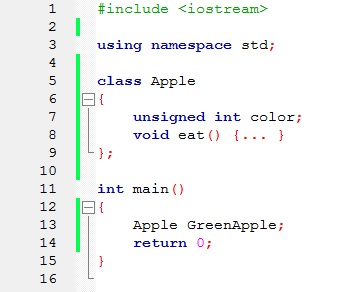
1. Объясните принцип наследования, опишите синтаксис. Приведите пример наследования.

Наследование – это один из трех основных принципов ООП, предполагающий создание иерархии классов (объектов классов), в которой объекты-потомки наследуют все члены (свойства и поведение) своих классов-предков.



1. Дайте определение понятию «объект». Как осуществляться доступ к полям и методам класса. Приведите пример объявления объекта класса.

**Объе́кт** в **программировании** — некоторая сущность в цифровом пространстве, обладающая определённым состоянием и поведением, имеющая определённые свойства (атрибуты) и операции над ними (методы).



Элементы управления доступом позволяют отделить [public](https://docs.microsoft.com/ru-ru/cpp/cpp/public-cpp?view=msvc-170) интерфейс класса от [private](https://docs.microsoft.com/ru-ru/cpp/cpp/private-cpp?view=msvc-170) сведений о реализации и [protected](https://docs.microsoft.com/ru-ru/cpp/cpp/protected-cpp?view=msvc-170) членов, которые предназначены только для использования производными классами. Спецификатор доступа действует для всех членов, объявленных после него, пока не будет объявлен следующий спецификатор доступа.

Доступ по умолчанию находится **private** в классе и **public** в структуре или объединении. Спецификаторы доступа класса могут использоваться любое количество раз и в любом порядке. Выделение хранилища для объектов типов классов зависит от реализации. Однако компиляторы должны гарантировать назначение членов последовательно более высоким адресам памяти между описателями доступа.

| **Тип доступа** | **Значение** |
| --- | --- |
| [private](https://docs.microsoft.com/ru-ru/cpp/cpp/private-cpp?view=msvc-170) | Члены класса, объявленные как **private** могут использоваться только функциями-членами и друзьями (классами или функциями) класса. |
| [protected](https://docs.microsoft.com/ru-ru/cpp/cpp/protected-cpp?view=msvc-170) | Члены класса, объявленные как **protected** могут использоваться функциями-членами и друзьями (классами или функциями) класса. Кроме того, они могут использоваться производными классами данного класса. |
| [public](https://docs.microsoft.com/ru-ru/cpp/cpp/public-cpp?view=msvc-170) | Члены класса, объявленные как **public** могут использоваться любой функцией. |

1. Опишите объявление, описание и вызов функции. В чем различие процедуры от функции. Приведите пример функции (аргументы которой три вещественных числа), которая возвращает аргументы по возрастанию.

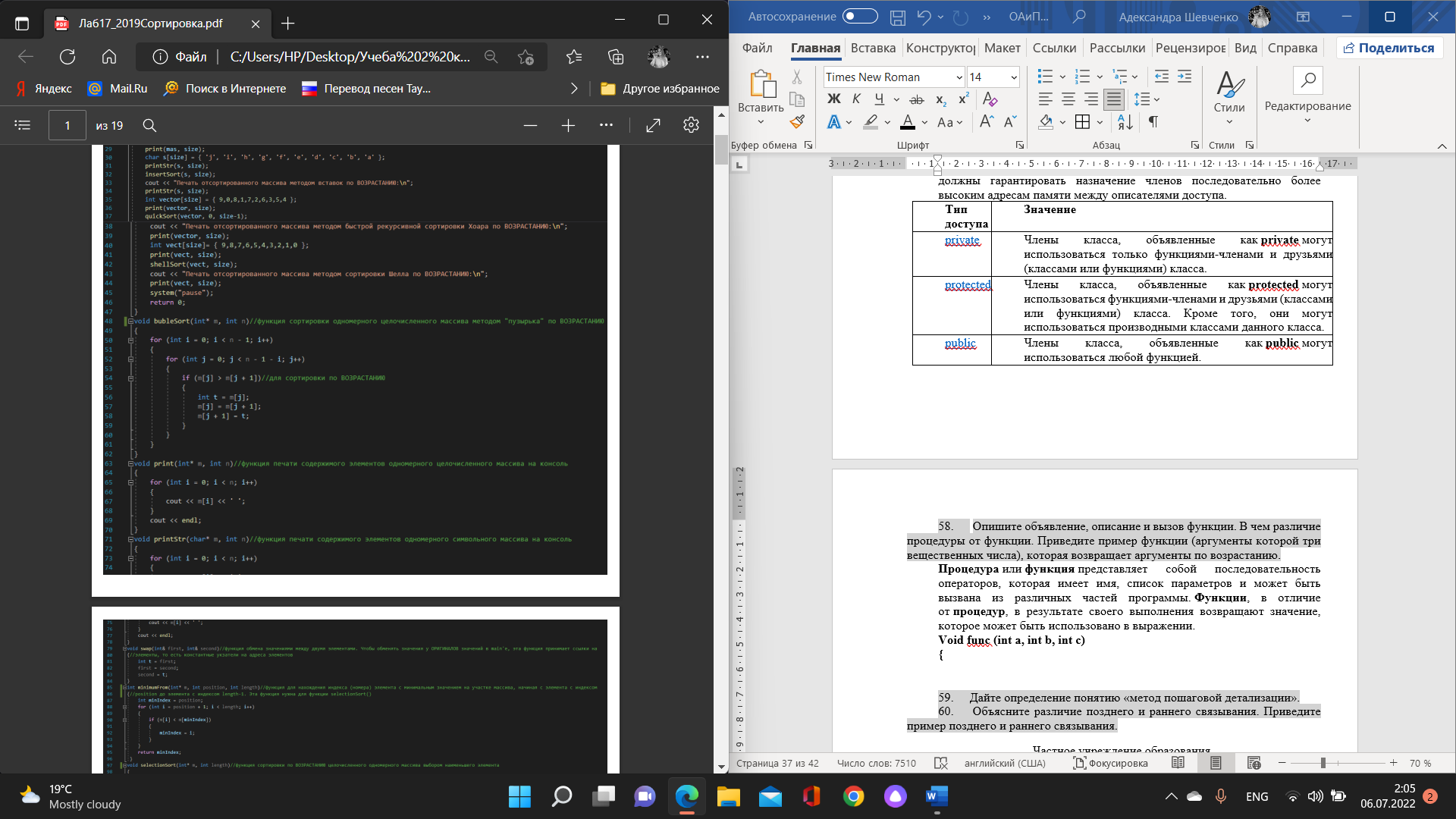
**Процедура** или **функция** представляет собой последовательность операторов, которая имеет имя, список параметров и может быть вызвана из различных частей программы. **Функции**, в отличие от **процедур**, в результате своего выполнения возвращают значение, которое может быть использовано в выражении.

**Void func (int a, int b, int c)**

**{**

**Int n=3;**

**Int m[3]=(a,b,c);//хз можно так или нет**



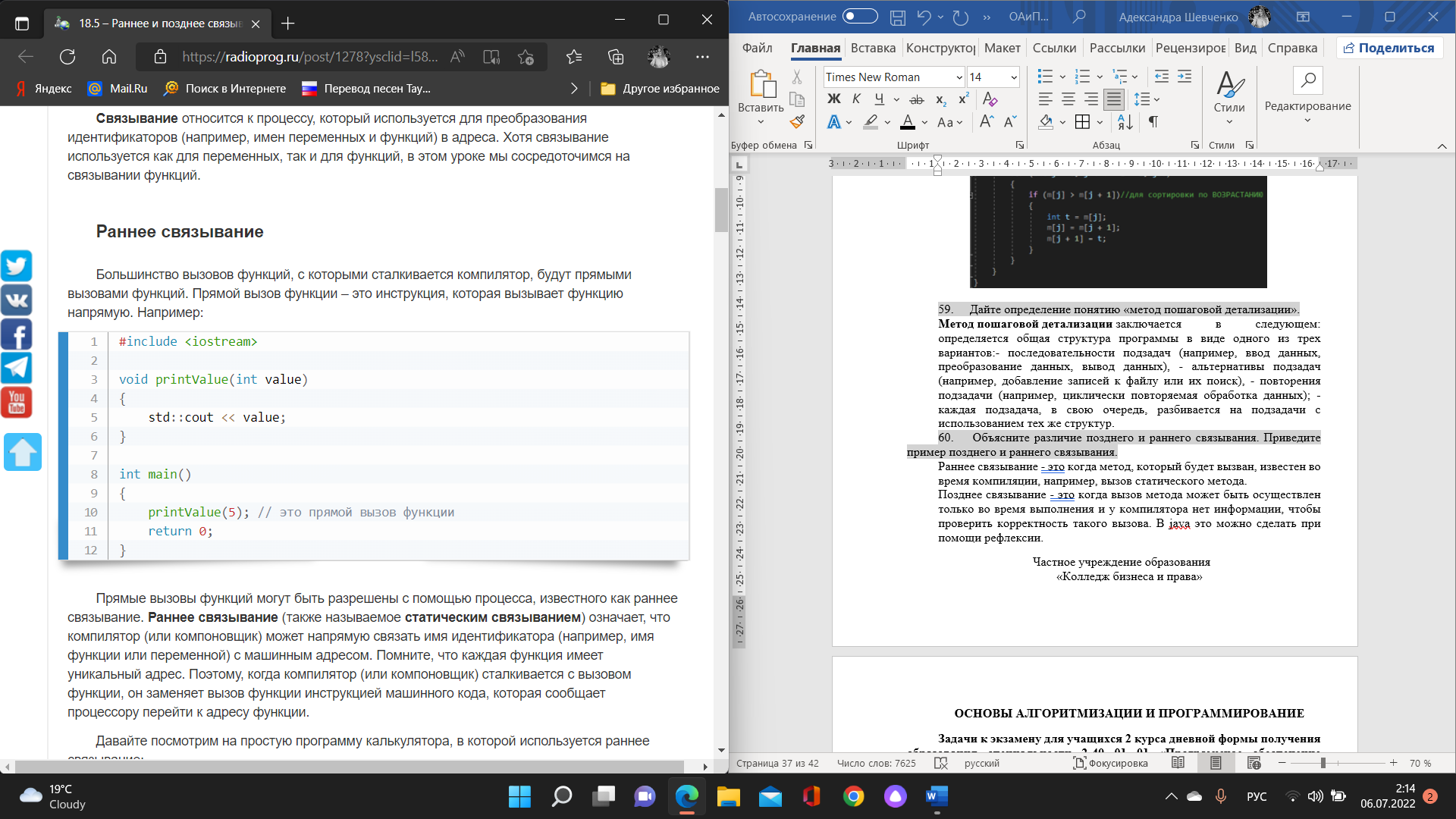
1. Дайте определение понятию «метод пошаговой детализации».

**Метод** **пошаговой** **детализации** заключается в следующем: определяется общая структура программы в виде одного из трех вариантов:- последовательности подзадач (например, ввод данных, преобразование данных, вывод данных), - альтернативы подзадач (например, добавление записей к файлу или их поиск), - повторения подзадачи (например, циклически повторяемая обработка данных); - каждая подзадача, в свою очередь, разбивается на подзадачи с использованием тех же структур.

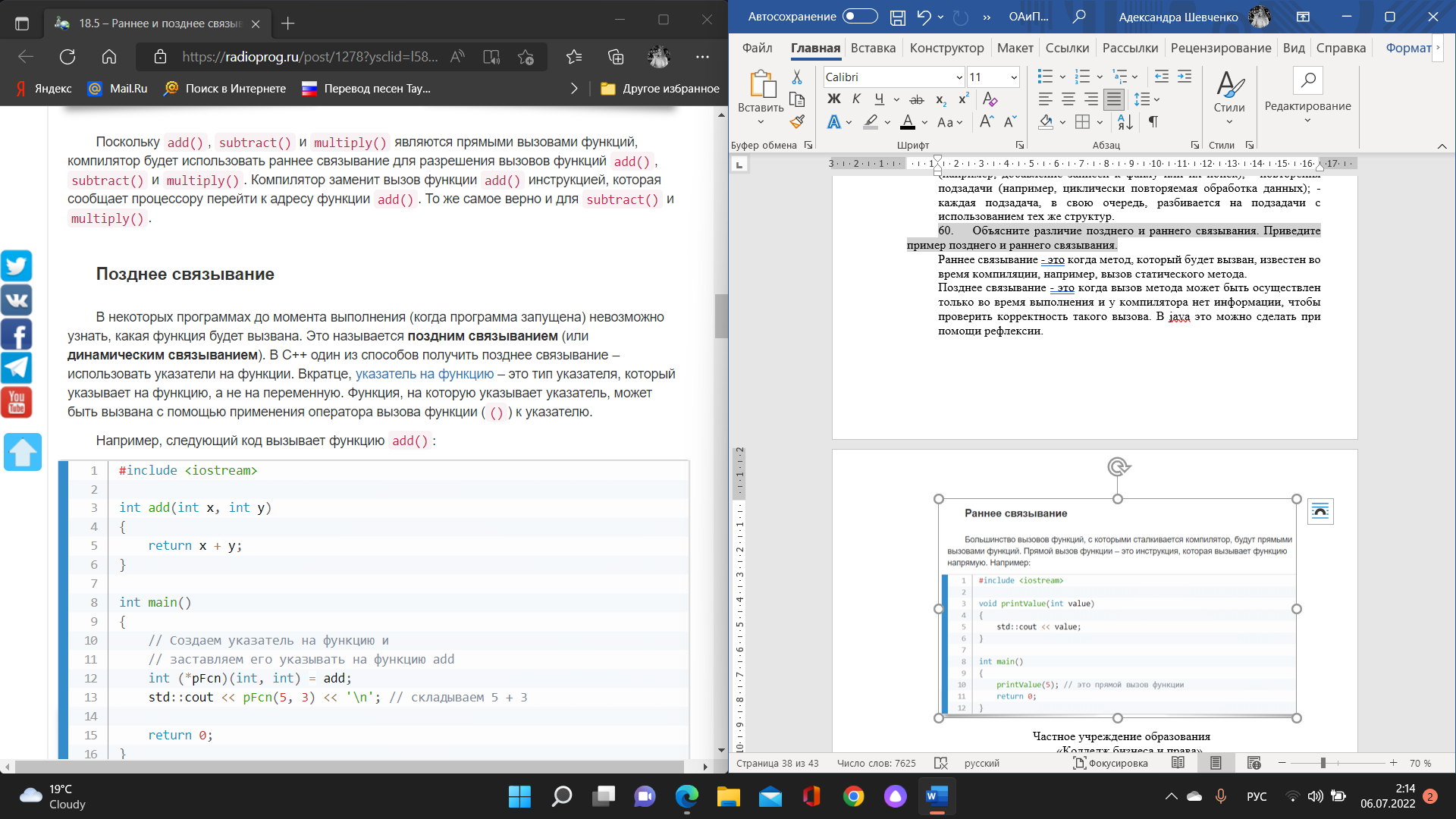
1. Объясните различие позднего и раннего связывания. Приведите пример позднего и раннего связывания.

Раннее связывание - это когда метод, который будет вызван, известен во время компиляции, например, вызов статического метода.

Позднее связывание - это когда вызов метода может быть осуществлен только во время выполнения и у компилятора нет информации, чтобы проверить корректность такого вызова. В java это можно сделать при помощи рефлексии.



Позднее:



Частное учреждение образования

«Колледж бизнеса и права»

**ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЕ**

**Задачи** **к экзамену для учащихся 2 курса дневной формы получения образования специальности 2-40 01 01 «Программное обеспечение информационных технологий»**

1. Создать класс «Треугольник» и класс «Точка». Реализовать конструкторы с параметрами и конструктор копирования. Объявить массив из  объектов класса Point. Написать функцию, определяющую, какая из точек лежит внутри, а какая снаружи треугольника. Написать программу, демонстрирующую все разработанные элементы.

#include <iostream>

#include <Windows.h>

#include <vector>

#include <algorithm>

#include <list>

#include <stack>

using namespace std;

class Point

{

private:

int x, y;

public:

Point()

{

}

Point(int x, int y)

{

this->x = x;

this->y = y;

}

Point(const Point& other)

{

this->x = other.x;

this->y = other.y;

}

int getX() { return this->x; }

int getY() { return this->y; }

void show()

{

cout << getX() << " " << getY() << endl;

}

};

class Triangle

{

private:

Point a, b, c;

public:

Triangle(Point a, Point b, Point c)

{

this->a = a;

this->b = b;

this->c = c;

}

void show()

{

a.show();

b.show();

c.show();

}

bool check(Point p)

{

int one, two, three;

one = (a.getX() - p.getX()) \* (b.getY() - a.getY() - (b.getX() - a.getX()) \* (a.getY() - p.getY()));

two = (b.getX() - p.getX()) \* (c.getY() - b.getY()) - (c.getX() - b.getX()) \* (b.getY() - p.getY());

three = (c.getX() - p.getX()) \* (a.getY() - c.getY()) - (a.getX() - c.getX()) \* (c.getY() - p.getY());

if (one >= 0 && two >= 0 && three >= 0 || one <= 0 && two <= 0 && three <= 0)

{

return false;

}

else

{

return true;

}

}

};

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand(time(0));

Point a(13, 10);

Point b(20, 13);

Point c(10, 20);

int size = 9;

Point \*p = new Point[size];

Triangle t(a, b, c);

for (size\_t i = 0; i < size; i++)

{

p[i] = Point(rand() % 30, rand() % 30);

if (t.check(p[i]))

{

p[i].show();

cout << "лежат вне треугольника" << endl;

}

else

{

p[i].show();

cout << "лежат внутри треугольника" << endl;

}

}

for (size\_t i = 0; i < size; i++)

{

p[i].show();

}

cout << "Triangle:" << endl;

t.show();

return 0;

}

1. Определить класс «Стек» хранящее вещественное число. Реализовать конструктор копирования. Объявить объект класса. Добавить в стек несколько элементов извлечь пару элементов и проверить пуст ли стек (т.е. реализовать методы push, pop, is\_empty). Написать программу, демонстрирующую все разработанные элементы. Продемонстрировать все разработанные функции.

#include <iostream>

#include <Windows.h>

#include <vector>

#include <algorithm>

#include <list>

#include <stack>

using namespace std;

class Stack

{

private:

vector<double>stack;

int next\_index;

public:

Stack()

{

}

void push(int &&\_val)

{

this->stack.insert(stack.begin() + this->next\_index++, \_val);

}

void pop()

{

this->stack.pop\_back();

sort(this->stack.rbegin(), this->stack.rend());

}

void show()

{

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

{

cout << this->stack[i] << ' ' << endl;

}

}

bool isEmpty()

{

if (this->stack.empty())

return true;

else

return false;

}

Stack(const Stack &other)

{

this->next\_index = other.next\_index;

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

{

this->stack.push\_back(other.stack[i]);

}

}

};

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

cout << "Стандартный стек" << endl;

stack<int> b;

b.push(1);

b.push(3);

b.push(5);

b.pop();

while (!b.empty())

{

cout << b.top() << endl;

b.pop();

}

cout << "Мой стек" << endl;

Stack a;

a.push(1);

a.push(3);

a.push(5);

a.pop();

Stack c;

c = a;

c.show();

return 0;

}

1. Определить класс «Очередь» хранящее символ. Реализовать конструктор копирования. Объявить объект класса. Добавить в очередь несколько элементов извлечь пару элементов и проверить пуста ли очередь (т.е. реализовать методы push, pop, is\_empty). Написать программу, демонстрирующую все разработанные элементы. Продемонстрировать все разработанные функции.

#include <iostream>

#include <Windows.h>

#include <vector>

#include <algorithm>

#include <list>

#include <stack>

using namespace std;

class queue

{

private:

list<char> list;

int size;

public:

queue()

{

}

queue(const queue& que)

{

this->list = que.list;

this->size = que.size;

}

void push(const char && \_val)

{

this->list.push\_back(\_val);

this->size++;

};

void pop()

{

if (!isEmpty())

{

this->list.pop\_front();

this->size--;

}

};

bool isEmpty() const

{

return !size;

};

void show()

{

for (auto i = this->list.begin(); i != list.end(); i++)

{

cout << (\*i) << " " << endl;

}

};

};

int main()

{

queue a;

a.push('d');

a.push('c');

a.pop();

if (a.isEmpty())

{

cout << "Очередь пустая";

}

a.show();

return 0;

}

1. Определить класс «Дробь» в виде пары  и использующий его класс «Смешанная дробь» в виде целой и дробной части. Реализовать конструкторы с параметрами и конструктор копирования. Перегрузить операции ввода/вывода и сложения/умножения. Объявить массив из *k* дробей, ввести. Вывести наибольшую из дробей. Написать программу, демонстрирующую все разработанные элементы.

#include <iostream>

#include <Windows.h>

#include <vector>

#include <algorithm>

#include <list>

#include <stack>

using namespace std;

class Drob

{

private:

int a, b;

public:

Drob()

{

};

Drob(int a, int b)

{

this->a = a;

this->b = b;

}

void set\_value(int a, int b)

{

this->a = a;

this->b = b;

}

int get\_a() { return this->a; }

int get\_b() { return this->b; }

Drob operator+(Drob &other)

{

Drob d1 = Drob(this->a \* other.b, this->b \* other.b);

Drob d2 = Drob(other.a \* this->b, other.b \* this->b);

return Drob(d1.a + d2.a, d1.b);

}

Drob operator-(Drob &other)

{

Drob d1 = Drob(this->a \* other.b, this->b \* other.b);

Drob d2 = Drob(other.a \* this->b, other.b \* this->b);

return Drob(d1.a - d2.a, d1.b);

}

Drob operator\*(Drob &other)

{

return Drob(this->a \* other.a, this->b \* other.b);

}

Drob(const Drob &other)

{

this->a = other.a;

this->b = other.b;

}

void show()

{

cout << this->a << "\n" << this->b << endl;

}

};

int main()

{

Drob\* arr = new Drob[5];

for (size\_t i = 0; i < 5; i++)

{

arr[i].set\_value(rand() % 15, rand() % 20);

arr[i].show();

}

int max\_value\_1 = 0, max\_value\_2 = 0;

for (size\_t i = 0; i < 5; i++)

{

max\_value\_1 = arr[i].get\_a();

max\_value\_2 = arr[i].get\_b();

if (arr[i].get\_a() >= max\_value\_1 && arr[i].get\_b() >= max\_value\_2)

{

continue;

}

}

cout << "\n Наибольшая дробь: \n" << max\_value\_1 << "\n" << max\_value\_2 << endl;

return 0;

}

1. Создать шаблон класса Matrix. Реализовать конструкторы с параметрами и конструктор копирования. Перегрузить основные операции: сложения, умножения матриц. Массив матриц упорядочить так, чтобы норма возрастала . Написать программу, демонстрирующую все разработанные элементы.
2. Создать абстрактный класс «Книга» и производные от него классы «Справочник» и «Книга по информатике». Реализовать конструкторы с параметрами и конструктор копирования. Используя виртуальную функцию ShowBook( ) реализовать полиморфизм. Все записи с книгами по информатике вывести на экран. Написать программу, демонстрирующую все разработанные элементы.

{

Автор

Название

Год

Количество страниц

Вид издания

}

#include <iostream.h>

#include <stdio.h>

#include <conio.h>

#include <string.h>

class Book

{

char Avtor [20];

char Nazvanie [20];

char Izd [20];

int god;

int kol\_str;

public:

Book();

char \*getavtor();

char \*getizd();

int getgod();

void show();

};

Book::Book()

{

cout<<"Avtor: "; cin>>Avtor;

cout<<"Nazvanie: "; cin>>Nazvanie;

cout<<"Izdat: "; cin>>Izd;

cout<<"God: "; cin>>god;

cout<<"Kol str: "; cin>>kol\_str; cout<<endl;

}

void Book::show()

{

cout<<"Avtor: "<<Avtor<<"\t||\t"<<"Nazvanie: "<<Nazvanie<<endl;

cout<<"Izdat: "<<Izd<<"\t||\t"<<"God: "<<god<<"\t||\t"<<"Kol str: "<<kol\_str<<endl; cout<<endl;

}

char \*Book::getavtor() { return Avtor; }

char \*Book::getizd() { return Izd; }

int Book::getgod() { return god; }

void spis\_book\_avtora(Book spis[],int n);

void spis\_book\_izd(Book spis[],int n);

void spis\_book\_po\_godu(Book spis[],int n);

void main()

{

clrscr();

Book \*spis;

int n,i;

cout<<"Vvedite kol-vo: "; cin>>n;

spis=new Book[n];

for (i=0;i<n;i++)

{

cout<<"============================================================="<<endl;

spis[i].show();

cout<<"============================================================="<<endl;

}

cout<<"Cpisok knig po avtoru"<<endl;

spis\_book\_avtora(spis,n);

cout<<"Cpisok knig po izdat"<<endl;

spis\_book\_izd(spis,n);

cout<<"Sposok knig posle zadannogo goda"<<endl;

spis\_book\_po\_godu(spis,n);

cout<<"\nPress any key!"<<endl;

getch();

delete[] spis;

}

void spis\_book\_avtora(Book spis[],int n)

{

char Avtor[20];

cout<<"Avtor: ";

cin>>Avtor; cout<<endl;

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

if (strcmp(spis[i].getavtor(),Avtor)==0) spis[i].show();

}

void spis\_book\_izd(Book spis[],int n)

{

char Izd[20];

cout<<"Izdat: ";

cin>>Izd; cout<<endl;

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

if (strcmp(spis[i].getizd(),Izd)==0) spis[i].show();

}

void spis\_book\_po\_godu(Book spis[],int n)

{

int g;

cout<<"God: ";

cin>>g; cout<<endl;

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

if (spis[i].getgod()>=g) spis[i].show();

}

1. Номиналы рублей могут принимать значения 10, 20, 50, 100, 500, 1000, 10000, 20000, 50000, 100000. Создать класс Money для работы с денежными суммами. Реализовать конструкторы с параметрами и конструктор копирования. Сумма должна быть представлена полями-номиналами, значениями которых должно быть количество купюр данного достоинства. Реализовать через перегрузку операций сложение сумм, вычитание сумм, и операции сравнения (==, !=, <, >). Написать программу, демонстрирующую все разработанные элементы.
2. Создать класс Fraction для работы с десятичными дробными числами. Число должно быть представлено двумя полями: int (целая часть и знак числа), дробная часть – unsigned. Реализовать конструкторы с параметрами и конструктор копирования. Перегрузить арифметические операции: сложение, вычитание, умножение и операции сравнения (==, !=, <, >). Написать программу, демонстрирующую все разработанные элементы.

public class Fraction

{

private int wholePart;

private uint fractionPart;

public double ToDouble()

{

var s = wholePart + "," + fractionPart;

return Convert.ToDouble(s);

}

public override string ToString()

{

return this.ToDouble().ToString("n2");

}

public static Fraction ToFraction(double value)

{

var arr = value.ToString().Split(new char[] {',' }, StringSplitOptions.RemoveEmptyEntries);

return new Fraction()

{

wholePart = Convert.ToInt32(arr[0]),

fractionPart = arr.Length == 2 ? Convert.ToUInt32(arr[1]) : 0

};

}

public static Fraction operator +(Fraction a, Fraction b)

{

return ToFraction(a.ToDouble() + b.ToDouble());

}

public static Fraction operator -(Fraction a, Fraction b)

{

return ToFraction(a.ToDouble() - b.ToDouble());

}

public static Fraction operator \*(Fraction a, Fraction b)

{

return ToFraction(a.ToDouble() \* b.ToDouble());

}

public static bool operator ==(Fraction a, Fraction b)

{

return a.fractionPart == b.fractionPart && a.wholePart == b.wholePart;

}

public static bool operator !=(Fraction a, Fraction b)

{

return a.fractionPart != b.fractionPart || a.wholePart != b.wholePart;

}

}

1. Создать класс Polynom для работы с многочленами. Реализовать конструкторы с параметрами и конструктор копирования. Коэффициенты должны быть представлены динамическим массивом. Младшая степень имеет меньший индекс. Перегрузить арифметические операции (+,\*) и операции сравнения (== , !=). Реализовать методы для вычисления значения полинома для заданного значения x. Написать программу, демонстрирующую все разработанные элементы.
2. Реализовать класс Triangle (треугольник) с полями-координатами вершин. Вычислить площадь и периметр треугольника. Реализовать конструкторы с параметрами и конструктор копирования. Для представления координат вершин используйте класс Point. Проверить, является ли треугольник равносторонним, равнобедренным, прямоугольным. Написать программу, демонстрирующую все разработанные элементы.

class Triangle

{

double AB;

double AC;

double BC;

double A;

double B;

double C;

double perim()

{

return AB + AC + BC;

}

double Area()

{

double p=(AB+AC+BC)/2.0;

return Math.Sqrt(p \* (p - AB) \* (p - AC) \* (p - BC));

}

string GetType()

{

if (AB == BC && AB == AC) return "равносторонний";

if (AB == BC || AB == AC || BC == AC) return "равнобедренный";

return "треугольник, как треугольник";

}

}

1. Создать шаблон класса MatVector размерности . Реализовать конструкторы с параметрами и конструктор копирования. Создать метод для вывода на экран вектора. Перегрузить операции сложения векторов, умножения на число, скалярного умножения векторов. Написать программу, демонстрирующую все разработанные элементы.

#include<iostream>

#include<math.h>

#include<windows.h>

using namespace std;

//--------------------------------------------------------------

//----------- Определение класса -------------------------------

//--------------------------------------------------------------

class vektor

{

private:

int n; //Размер вектора

float\* ptrArr; //Указатель на массив элементов

public:

vektor()

{

n = 3;

ptrArr = new float [n];

}

vektor(int num)

{

n = num;

ptrArr = new float [n];

}

~vektor()

{ delete [] ptrArr; }

void GetVektor() //Задать вектор

{

//cout << "Введите размерность вектора: "; cin >> n;

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

{

cout << "Введите элемент вектора № " << i << ": ";

cin >> \*(ptrArr + i);

}

}

float ArrModul() //Вычислить модуль

{

float modul = 0;

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

modul += \*(ptrArr + i) \* \*(ptrArr + i);

return sqrt (modul);

}

void ShowVektor() //Показать вектор

{

cout << "Вектор [";

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

cout << \*(ptrArr + i) << ", ";

cout << " ]\n";

}

vektor operator= (vektor v)

{

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

\*(ptrArr + i) = \*(v.ptrArr + i);

return \*this;

}

friend float Skalar (vektor\* , vektor\*); //Склярное произведение

friend bool Kolliniar(vektor\* , vektor\* ); //Проверка на коллинеарность

friend bool Ortogonal(vektor\* , vektor\* ); //Проверка на ортогональность

friend vektor Kommutativ(vektor\* , vektor\* );//Сложение векторов

friend vektor Netativ (vektor\* , vektor\* );//ВЫчитание векторов

};

//--------------------------------------------------------------

//----------- Определение дружественных функций ----------------

//--------------------------------------------------------------

vektor Kommutativ(vektor\* vector1, vektor\* vector2)

{

vektor temp;

for (int i=0; i < vector1->n; i++ )

\*(temp.ptrArr + i) = \*(vector1->ptrArr + i) + \*(vector2->ptrArr + i);

return temp;

}

vektor Netativ(vektor\* vector1, vektor\* vector2)

{

vektor temp;

for (int i=0; i < vector1->n; i++ )

\*(temp.ptrArr + i) = \*(vector1->ptrArr + i) - \*(vector2->ptrArr + i);

return vektor(temp);

}

float Skalar (vektor\* vector1, vektor\* vector2)

{

static float temp;

temp = 0;

for (int i=0; i < (\*vector1).n; i++ )

temp += \*(vector1->ptrArr + i) \* \*(vector2->ptrArr + i);

return temp;

}

bool Ortogonal (vektor\* vector1, vektor\* vector2)

{

if ( (Skalar (vector1, vector2)) < 0.01 )

return true;

else

return false;

}

bool Kolliniar (vektor\* vector1, vektor\* vector2)

{

if ( (Skalar (vector1, vector2) - ((\*vector1).ArrModul() \* (\*vector2).ArrModul())) < 0.01 )

return true;

else

return false;

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

vektor v1(3), v2(3), v(3);

v1.GetVektor();

v2.GetVektor();

cout << Skalar(&v1, &v2) << endl;

v = Kommutativ(&v1, &v2);

v.ShowVektor();

cout << "Ортогональность: " << Ortogonal(&v1, &v2) << endl;

cout << "Коллинеарность: " << Kolliniar(&v1, &v2) << endl;

system("pause");

return 0;

}

1. Создать класс Triad (тройка чисел); определить с помощью перегрузки операции ++ операцию увеличения всех полей одновременно на 1. Реализовать конструкторы с параметрами и конструктор копирования. Определить производный класс Time (время) с полями: часы, минуты и секунды. Перегрузить операции (++, --) увеличения (уменьшения) времени на 1 секунду и операции + и - увеличения (уменьшения) времени на n секунд. Написать программу, демонстрирующую все разработанные элементы.

class Triad

{

int x;

int y;

int z;

public:

Triad(){}

Triad(int a,int b, int c)

{

x=a;

y=b;

z=c;

}

void show(int a,int b, int c);

void srav()

{if ( Triad aa> Time bb)

cout<<"obe'ekt aa> ob'eka bb" <<endl

if (Triad aa< Time bb)

cout <<"ob'ekt aa< ob'ekta bb" <<endl

if (Triad aa==Time bb)

cout<<"ob'ekt aa= ob'ektu bb"<<endl };

void pr() {cout<<x, y, z;}

} ;

class Time: public Triad

{

public:

Time():Triad(){}

Time(int x, int y, int z):Triad(x,y,z){};

/\* void srav()

{if (Triad aa> Time bb)

cout<<"obe'ekt aa> ob'eka bb" <<endl;

if (Triad aa< Time bb)

cout <<"ob'ekt aa< ob'ekta bb" <<endl;

if (Triad aa==Time bb)

cout<<"ob'ekt aa= ob'ektu bb"<<endl; };\*/

}

void main(void)

{ Triad aa, \*pa;

Time bb;

aa.show(09,28,45);

bb.show(10,23,37);

aa.pr();

bb.pr();

bb.srav();

getchar();

}

1. Создать классы Point и Line. Реализовать конструкторы по умолчанию, с параметрами и конструктор копирования. Реализовать методы set, get. Объявить массив из n объектов класса Point и определить какая из точек находится дальше всех от прямой. Нахождение расстояния реализовать в методе. Написать программу, демонстрирующую все разработанные элементы.

14. public class Point {

15. private int x;

16. private int y;

17.

18. public Point(){}

19.

20. public Point(int x, int y) {

21. this.x = x;

22. this.y = y;

23. }

24.

25. public int getX() {

26. return x;

27. }

28.

29. public void setX(int x) {

30. this.x = x;

31. }

32.

33. public int getY() {

34. return y;

35. }

36.

37. public void setY(int y) {

38. this.y = y;

39. }

40.

41. @Override

42. public String toString() {

43. return "Point{" +

44. "x=" + x +

45. ", y=" + y +

46. '}';

47. }

48. }

1. Определить класс «Матрица» размерности . Реализовать конструкторы с параметрами и конструктор копирования. Реализовать метод для установки новой размерности матрицы и метод для установки коэффициентов матрицы. Объявить массив из  объектов. Написать функции, вычисляющие первую и вторую нормы матрицы , . Определить, какая из матриц имеет наименьшую первую и вторую нормы. Написать программу, демонстрирующую все разработанные элементы.
2. Определить классы «Треугольник». Реализовать конструкторы с параметрами и конструктор копирования. Определить, какой из m введенных «Треугольников» имеет наибольшую и наименьшею площадь. . Написать программу, демонстрирующую все разработанные элементы.
3. Создать класс Complex для работы с мнимыми числами. Реализовать конструкторы с параметрами и конструктор копирования. Используя для представления числа два числа реальная и действительная часть числа. Перегрузить арифметические операции (+, -, \*, /) и операции сравнения (==, !=, <, >). Написать программу, демонстрирующую все разработанные элементы.
4. Создать класс Complex для работы с мнимыми числами. Реализовать конструкторы с параметрами и конструктор копирования. Используя для представления числа два числа реальная и действительная часть числа. Перегрузить арифметические операции (+, -, \*, /) и операцию сравнения (==). Реализовать метод для вывода на экран числа. Написать программу, демонстрирующую все разработанные элементы.

internal class Complex

{

public Complex(int a, int b)

{

A = a;

B = b;

}

public int A { get; set; }

public int B { get; set; }

public static Complex operator +(Complex first, Complex second)

{

return new Complex(first.A + second.A, first.B + second.B);

}

public static Complex operator -(Complex first, Complex second)

{

return new Complex(first.A - second.A, first.B - second.B);

}

public static Complex operator \*(Complex first, Complex second)

{

//todo

}

public static Complex operator /(Complex first, Complex second)

{

//todo

}

public static bool operator ==(Complex first, Complex second)

{

return first.A == second.A && first.B == second.B;

}

public static bool operator !=(Complex first, Complex second)

{

return !(first == second);

}

public override string ToString()

{

return $"({A};{B})";

}

}

1. Создать класс Pair (пара чисел). Реализовать конструкторы с параметрами и конструктор копирования, определить методы изменения полей и вычисления произведения чисел. Определить производный класс RightAngled (прямоугольный треугольник) с полями-катетами. Определить методы вычисления гипотенузы и площади треугольника. Написать программу, демонстрирующую все разработанные элементы.

class Pair

{

public double FirstSide { get; set; }

public double SecondSide { get; set; }

public Pair()

{

}

public Pair(int firstSide, int secondSide)

{

this.FirstSide = firstSide;

this.SecondSide = secondSide;

}

public double Product()

{

return this.FirstSide \* this.SecondSide;

}

public override string ToString()

{

return "First side " + FirstSide + " Second side " + SecondSide;

}

}

class RightAngled : Pair

{

public RightAngled()

{

}

public RightAngled(int firstSide, int secondSide) : base (firstSide, secondSide)

{

}

public double GetHypotenuse()

{

return Math.Sqrt(Math.Pow(base.FirstSide, 2) + Math.Pow(base.SecondSide, 2));

}

public double GetArea()

{

return 0.5 \* (base.FirstSide \* base.SecondSide);

}

}

1. Создать класс Triad (тройка чисел). Реализовать конструкторы с параметрами и конструктор копирования, определить методы изменения полей и вычисления суммы чисел. Определить производный класс Triangle (треугольник) с полями-сторонами. Определить методы для определения: существует ли такой треугольник, для вычисления периметра и площади. Написать программу, демонстрирующую все разработанные элементы.
2. Создать класс Man (человек) с полями: имя, возраст, пол и вес (реализовать конструкторы с параметрами и конструктор копирования). Определить методы переназначения имени, изменения возраста и изменения веса. Создать производный класс Student (реализовать конструкторы с параметрами и конструктор копирования), имеющий поле для года обучения. Определить методы переназначения и увеличения года обучения. Перегрузить операции ++, --. Написать программу, демонстрирующую все разработанные элементы.

#include <iostream>

#include <string>

using namespace std;

class Man

{

private:

static int count; // счетчик

protected:

string n\_name;

int n\_age;

string n\_sex;

int n\_wight;

public:

static int GetCount(){return count;}

//Person();

Man(const string& name, const int& age, const string& sex, const int& wight)

: n\_name ( name ), n\_age( age ), n\_wight( wight ), n\_sex( sex )

{

count++;

}

string GetName(){ return n\_name; }

int GetAge(){ return n\_age; }

int GetWighr(){ return n\_wight; }

virtual void Print() // виртуальная функция перегружаемая в классе Student

{

cout<<n\_name<<' '<<n\_age<<' '<<n\_sex<<' '<<n\_wight<<endl;

}

};

class Student : public Man

{

private:

int n\_year;

public:

Student(int year, const string& name, const int age, const string& sex, const int wight)

:Man(name, age, sex, wight),n\_year(year)

{}

int GetYear(){ return n\_year; }

void AddYear(int addValue)

{

n\_year += addValue;

}

void AddYear()

{

n\_year++;

}

void Print()

{

cout<<n\_year<<' ';

Man::Print();

}

};

int Man::count=0; // инициализация счетчика

int main()

{

const int N=2;

Man\* pers[N];

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

{

int year,wight,age;

string name, sex;

cout<<"Enter name age sex weight age:"<<endl;

cin>> name>>age>> sex>>wight>>year;

pers[i]=new Student(year,name, age, sex, wight); // указатель на Person инициализируем классом Student

cout<<"Count="<<pers[i]->GetCount()<<endl; // вывод количества созданных экземпляров

}

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

pers[i]->Print();

Student\* s1=static\_cast<Student\*>(pers[0]); // s1 теперь имеет тип Student\*, иначе не вызвать AddYear

s1->AddYear(2);

cout<<"New age of "<<s1->GetName()<<" = "

<<s1->GetAge()<<endl;

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

delete pers[i]; // освобождение выделенной памяти

return 0;

}

1. Создать класс PairNumber (пара вещественных чисел). Реализовать конструкторы с параметрами и конструктор копирования; определить методы изменения полей и вычисления произведения чисел. Определить производный класс Rectangle (прямоугольник) (реализовать конструкторы с параметрами и конструктор копирования) с полями-сторонами. Определить методы вычисления периметра и площади прямоугольника. Написать программу, демонстрирующую все разработанные элементы.

22. #include <iostream>

23. #include <stdexcept>

24.

25. class PairNumber

26. {

27. protected:

28. double x, y;

29. public:

30. PairNumber (double x\_ = 0, double y\_ = 0) : x(x\_), y(y\_){}

31. void Set(double x\_, double y\_)

32. {

33. x = x\_;

34. y = y\_;

35. return;

36. }

37. double Prod()

38. {

39. return x \* y;

40. }

41. };

42.

43. class Rectangle : protected PairNumber

44. {

45. public:

46. Rectangle(double x\_ = 1, double y\_ = 1) : PairNumber (x\_, y\_)

47. {

48. if(x\_ <= 0 || y\_ <= 0) throw std::invalid\_argument("arguments must be greater zero\n");

49. }

50. double Perimeter()

51. {

52. return 2 \* ( x + y );

53. }

54. double Area()

55. {

56. return Prod();

57. }

58. };

59.

60. int main()

61. {

62. Rectangle r(1, 3);

63. std::cout << r.Perimeter();

64. return 0;

65. }

1. Создать класс Time для работы со временем в формате «часы:минуты:секунды». Реализовать конструкторы с параметрами и конструктор копирования. Класс должен включать в себя конструкторы инициализации: числами, строкой (например, «22:59:59»), секундами и временем. Перегрузить операции: вычисление разницы между двумя моментами времени в секундах, сложение времени и заданного количества секунд, вычитание из времени заданного количества секунд. Написать программу, демонстрирующую все разработанные элементы.

23. #include "stdafx.h"

24. #include <iostream>

25. #include <stdexcept>

26. #include <locale>

27. #include <cstdlib>

28. #include <string>

29. #include <sstream>

30.

31. using namespace std;

32.

33. class Time

34. {

35. public:

36. typedef unsigned short ushort;

37. Time(ushort hs, ushort ms, ushort ss)

38. {

39. set\_time(hs, ms, ss);

40. }

41.

42. explicit Time(size\_t ss = 0)

43. {

44. set\_time(ss);

45. }

46.

47. explicit Time(std::string const& stime)

48. {

49. set\_time(stime);

50. }

51.

52. void set\_time(ushort hs, ushort ms, ushort ss)

53. {

54. \_hs = hs;

55. \_ms = ms;

56. \_ss = ss;

57. check();

58. }

59.

60. void set\_time(size\_t ss)

61. {

62. \_hs = ss / 3600;

63. \_ms = (ss % 3600) / 60;

64. \_ss = (ss % 3600) % 60;

65. check();

66. }

67.

68. void set\_time(std::string const& stime) // "23:59:59"

69. {

70. std::stringstream sstr;

71. for(auto const& val : stime)

72. sstr << (val == ':' ? ' ' : val);

73. sstr >> \_hs >> \_ms >> \_ss;

74. if(!sstr || !sstr.eof())

75. throw std::runtime\_error("Time::set\_time: !sstr || !sstr.eof()");

76. check();

77. }

78.

79. friend std::ostream& operator<<(std::ostream& ost, Time const& rhs)

80. {

81. return ost << rhs.\_hs << ':' << rhs.\_ms << ':' << rhs.\_ss;

82. }

83.

84. operator std::string() const

85. {

86. std::stringstream sstr;

87. sstr << \_hs << ':' << \_ms << ':' << \_ss;

88. return sstr.str();

89. }

90.

91. private:

92. ushort \_hs, \_ms, \_ss;

93. void check() const

94. {

95. if(\_hs >= 24 || \_ms >= 60 || \_ss >= 60)

96. throw std::runtime\_error("Time::check: \_hs >= 24 || \_ms >= 60 || \_ss >= 60");

97. }

98. };

99.

100. int main()

101. {

102. setlocale(LC\_ALL, "rus");

103. Time hms1("23:45:40"), hms2(23 \* 3600 + 45 \* 60 + 41), hms3(23, 45, 42);

104. std::string shms3 = hms3;

105. std::cout << hms1 << '\n' << hms2 << '\n' << shms3 << "\n\n";

106. system("pause");

107. return 0;

108. }

1. Создать класс Triangle для представления треугольника. Реализовать конструкторы с параметрами и конструктор копирования. Поля данных должны включать координаты угловых точек треугольника. Требуется реализовать операции: получения и изменения полей данных, вычисления периметра, а также определения вида треугольника (равносторонний, равнобедренный или прямоугольный). Написать программу, демонстрирующую все разработанные элементы.

24. class Triangle

25. {

26. double AB;

27. double AC;

28. double BC;

29. double A;

30. double B;

31. double C;

32. double perim()

33. {

34. return AB + AC + BC;

35. }

36. double Area()

37. {

38. double p=(AB+AC+BC)/2.0;

39. return Math.Sqrt(p \* (p - AB) \* (p - AC) \* (p - BC));

40. }

41. string GetType()

42. {

43. if (AB == BC && AB == AC) return "равносторонний";

44. if (AB == BC || AB == AC || BC == AC) return "равнобедренный";

45. return "треугольник, как треугольник";

46. }

47. }

1. Создать класс Money для работы с денежными суммами. Реализовать конструкторы с параметрами и конструктор копирования. Число должно быть представлено двумя полями: типа int для рублей и типа char для копеек. Дробная часть (копейки) при выводе на экран должна быть отделена от целой части запятой. Перегрузить операции для сложения, вычитания и операции сравнения (==, !=, <, >). Написать программу, демонстрирующую все разработанные элементы.

25. #include <iostream>

26. #include <sstream>

27. using namespace std;

28.

29. class Money

30. {

31. long roubles;

32. unsigned int copecks;

33. float wholeSum;

34. void splitUp()

35. {

36. roubles = (int)wholeSum;

37. copecks = (int)((wholeSum - (int)wholeSum) \* 100);

38. };

39. public:

40. void get()

41. {

42. cin >> wholeSum;

43. splitUp();

44. };

45. string show()

46. {

47. stringstream out;

48. string str;

49. out << roubles << ",";

50. if (copecks < 10)

51. {

52. out << "0";

53. }

54. out << copecks;

55. str = out.str();

56. return str;

57. };

58. Money operator+ (Money arg)

59. {

60. Money temp;

61. temp.wholeSum = this->wholeSum + arg.wholeSum;

62. temp.splitUp();

63. return temp;

64. };

65. Money operator- (Money arg)

66. {

67. Money temp;

68. temp.wholeSum = this->wholeSum - arg.wholeSum;

69. temp.splitUp();

70. return temp;

71. };

72. float operator/ (Money arg)

73. {

74. float temp;

75. temp = this->wholeSum / arg.wholeSum;

76. return temp;

77. };

78. Money operator/ (float arg)

79. {

80. Money temp;

81. temp.wholeSum = this->wholeSum / arg;

82. temp.splitUp();

83. return temp;

84. };

85. Money operator\* (float arg)

86. {

87. Money temp;

88. temp.wholeSum = this->wholeSum \* arg;

89. temp.splitUp();

90. return temp;

91. };

92. bool operator== (Money arg)

93. { return (this->wholeSum == arg.wholeSum); };

94. bool operator!= (Money arg)

95. { return (this->wholeSum != arg.wholeSum); };

96. bool operator> (Money arg)

97. { return (this->wholeSum > arg.wholeSum); };

98. bool operator< (Money arg)

99. { return (this->wholeSum < arg.wholeSum); };

100. bool operator>= (Money arg)

101. { return (this->wholeSum >= arg.wholeSum); };

102. bool operator<= (Money arg)

103. { return (this->wholeSum <= arg.wholeSum); };

104. };

1. Определить класс «Стек» хранящее текстовую строку. Реализовать конструктор копирования. Объявить объект класса. Добавить в стек несколько элементов извлечь пару элементов и проверить пуст ли стек (т.е. реализовать методы push, pop, is\_empty). Написать программу, демонстрирующую все разработанные элементы.
2. Определить класс «Очередь» хранящее хранящее текстовую строку. Реализовать конструктор копирования. Объявить объект класса. Добавить в очередь несколько элементов извлечь пару элементов и проверить пуста ли очередь (т.е. реализовать методы push, pop, is\_empty). Написать программу, демонстрирующую все разработанные элементы.

#include <iostream>

#include <queue> // подключили библиотеку queue

using namespace std;

int main() {

setlocale(LC\_ALL,"rus");

queue <int> q; // создали очередь q

cout << "Пользователь, пожалуйста введите 7 чисел: " << endl;

for (int h = 0; h < 7; h++) {

int a;

cin >> a;

q.push(a); // добавляем в очередь элементы

}

cout << endl;

cout << "Самый первый элемент в очереди: " << q.front() << endl; // выводим первый

// элемент очереди

q.pop(); // удаляем элемент из очереди

cout << "Новый первый элемент (после удаления): " << q.front() << endl;

if (!q.empty()) cout << "Очередь не пуста!"; // проверяем пуста ли очередь (нет)

system("pause");

return 0;

}

1. Определить класс «Binary» (бинарное дерево) хранящее, текстовую строку. Реализовать конструктор копирования и конструктор создания бинарного дерева. Объявить объект класса. Реализовать методы для добавления данных в дерево, вывод всего дерева на экран, и метод удаления узла бинарного дерева. Написать программу, демонстрирующую все разработанные элементы.
2. Определить класс «Binary» (бинарное дерево), хранящее текстовую строку. Реализовать конструктор копирования и конструктор создания бинарного дерева. Объявить объект класса. Реализовать методы для добавления данных в дерево, вывод всего дерева на экран, и метод удаления узла бинарного дерева. И реализовать метод для получения количества узлов в бинарном дереве. Написать программу, демонстрирующую все разработанные элементы.
3. Определить класс «List\_1» (Циклический однонаправленный список) хранящий пару чисел (целое и вещественное). Реализовать конструктор копирования и конструктор создания списка. Объявить объект класса. Реализовать методы для добавления данных в список, вывод всего списка на экран, и метод удаления элемента из списка. И реализовать метод для получения количества элементов в однонаправленном списке. Написать программу, демонстрирующую все разработанные элементы.
4. Определить класс «List\_2» (двусвязный список) хранящий вещественное число и текстовую строку. Реализовать конструктор копирования и конструктор создания двусвязного списка. Объявить объект класса. Реализовать методы для добавления данных в список, вывод всего списка на экран, и метод удаления элемента из списка. И реализовать метод для получения количества элементов в двусвязном списке. Написать программу, демонстрирующую все разработанные элементы.