Частное учреждение образования
«Колледж бизнеса и права»

УТВЕРЖДАЮ
Заведующий методическим кабинетом
_____ Е.В. Фалей
«_____» _____ 2017 г.

Специальность: 2-40 01 01 «Программное обеспечение информационных технологий»	Дисциплина: «Основы алгоритмизации и программирование»

Лабораторная работа № 27
Инструкционно-технологическая карта

Тема: Разработка алгоритмов и программ с использованием перегрузки арифметических и логических операций, перегрузки конструктора копирования

Цель: Научиться создавать программы с использованием перегрузки арифметических и логических операций

Время выполнения: 2 часа

1. Краткие теоретические сведения
Перегрузка операций

Пример программы с ошибочной попыткой быстрого изменения значения поля объекта, путем использования синтаксиса операторов, применимых к целочисленным переменным, для объектов класса (в этом примере класса A):

```
#include <iostream>
using namespace std;
class A
{
private:
    int a = 0;
public:
    A(int a0)
    {
        a = a0;
    }
    void Show()
    {
        cout << a << endl;
    }
};
int main()
{
    A aa(5), bb(3);//создадим с помощью конструктора с параметром два экземпляра типа класса A: у объекта aa поле a принимает значение 5, у объекта bb поле a принимает значение 3
    aa.Show();
    bb.Show();
    //допустим, мы хотим лаконичным кодом увеличить на единицу с присвоением значение поля a у экземпляра класса A. Напишем код по аналогии с целочисленной переменной:
    aa++;//ошибка: целочисленное поле находится ВНУТРИ ОБЪЕКТА класса A, а потому НЕЛЬЗЯ увеличить значение поля a на единицу с присвоением, написав это применительно к объекту aa,
    //если бы поле было открытым (public), то было бы можно написать (aa.a)++; или написать более длиннный код, объявив в классе A метод int Method(){a++;} и вызвав его в функции
    //main() у объекта aa:      aa.Method();
    aa + bb;//ошибка: мы хотим к значению поля a у объекта aa прибавить значение из поля a объекта bb, но это нельзя по вышеуказанным причинам
    system("pause");
    return 0;
}//но вышепоказанная проблема имеет решение в виде явной перегрузки операторов для классов, когда написать такой код в main'е будет можно и будет нужный результат
```

Стр: 29 Симв: 163 Табуляция CRL

4 Ошибки | 0 Предупреждения | 0 Сообщения | Сборка и IntelliSense | Поиск по списку ошибок

Код	Описание	Проект	Файл	Строка	Сост...
E0349	отсутствует оператор "++", соответствующий этим операндам	Lab7_2	Source7_2.cpp	23	
E0349	отсутствует оператор "+", соответствующий этим операндам	Lab7_2	Source7_2.cpp	26	

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

```
class AAA
{
public:
    типДанных operator # (типДанных имяПеременной)//перегрузка оператора # для класса AAA
    {
        //код тела метода перегрузки оператора # для класса AAA
```

```
        return возвращаемоеЗначение;//обычно возвращаем измененный объект класса AAA
    }
};
```

Перегрузка операторов должна соответствовать правилам:

1) хотя бы один операнд перегруженной операции должен быть типа того класса, для которого перегружается оператор;

2) синтаксис исходной операции должен сохраниться (например, если перегрузили бинарный плюс + для класса Cat, то используем его для объектов Cat cat1, cat2; cat1 + cat2; как для «обычных» целочисленных переменных);

3) перегружать можно только стандартные общеизвестные операторы (как правило, это арифметические, логические операторы, операторы присвоения, скобки и некоторые другие).

Задание 1.

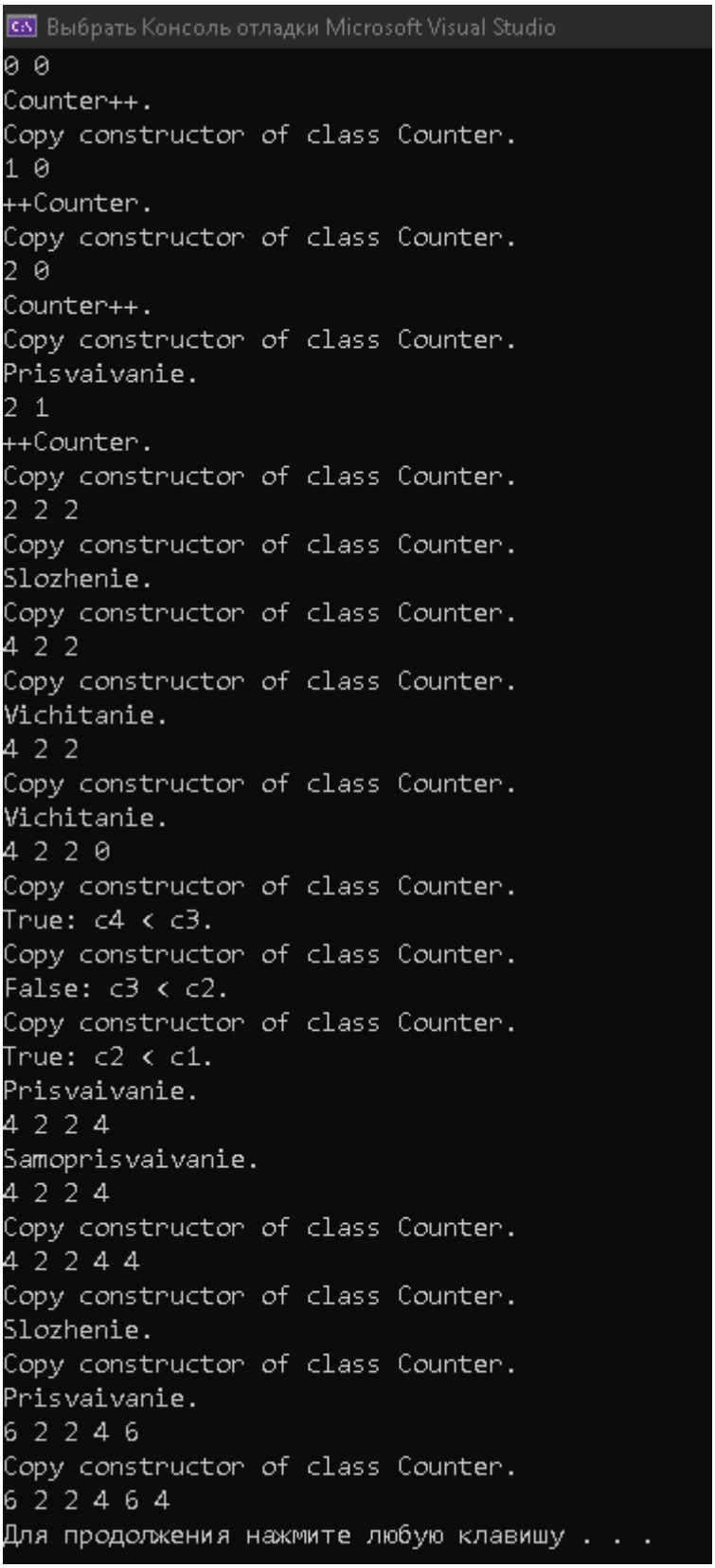
Напишите и разберите приведенную ниже программу. Изучите перегрузки операторов и перегруженный конструктор копирования:

```cpp
#include <iostream>
using namespace std;

class Counter
{
private:
    int count;
public:
    Counter()//конструктор без параметров
    {
        count = 0;
    }
    Counter(int a)//конструктор с параметрами
    {
        count = a;
    }
    Counter(const Counter& obj)//явное определение в коде конструктора копирования, который принимает объект типа данного класса и присваивает значения его полей создаваемому
    {//новому экземпляру данного класса, то есть копирует значения полей принимаемого объекта в соответсвующие поля создаваемого нового объекта
        this->count = obj.count;//в нашем классе одно поле, но их могло быть больше и тогда все их надо скопировать
        cout << "Copy constructor of class Counter.\n";//фраза-детектор, по которой мы увидим в консоли срабатывание конструктора копирования для класса Counter
    }
    int GetCount()//открытый метод класса Counter для возврата значения из поля int count;
    {
        return count;
    }
//перегрузка унарных операторов ++ и --////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    Counter operator++()//перегрузка префиксного инкремента
    {
        cout << "++Counter.\n";
        count++;// this->count++;
        return (*this);//возвращаем текущий объект (поскольку this - это указатель, то "звездочкой" берем значение объекта, на который ссылается указатель)
    }
    Counter operator++(int)//перегрузка постфиксного инкремента. В скобках обязательно пишется int, чтобы отличать от перегрузки префиксного инкремента
    {
        cout << "Counter++.\n";
        count++;// this->count++;
        return *this;
    }
    void operator--()//возможная, но НЕ универсальная, перегрузка префиксного декремента, когда ничего не возвращается, хотя поле count уменьшается на единицу с присвоением
    {//при такой перегрузке в main() НЕЛЬЗЯ будет написать  c2 = --c2;  Counter c3 = --c2;  Почему?
        this->count++;
        return;//если надо, то можно возвращать int'оновское значение (а не void) и тогда написать: return this->GetCount();
    }
    Counter operator--(int)//возможная, но НЕ универсальная, перегрузка постфиксного декремента, когда конструктором без параметров создается временный объект, ему присваивается
    {//значение из поля count текущего this объекта, далее значение поля count временного объекта уменьшается и этот временный объект возвращается
        Counter temp;//создали объект temp типа класса Counter конструктором без параметров
        temp.count = this->count;//копируем значение из поля counttекущего объекта в поле count нового объекта
        (temp.count)--;//уменьшаем значение поля count у нового объекта
        return temp;//возвращаем из метода перегрузки новый объект, поскольку по достижении закрывающей фигурной скобки } сборщик мусора очистит объекты данной локальной области
        //return Counter(--(this->count));//ускоренная запись: берется значение поля count текущего объекта this, значение этого поля уменьшается на единицу с присвоением. Теперь
        //это уменьшенное значение передается конструктору с параметром, который создает объект с полем count, куда помещается переданное уменьшенное значение. Этот новый созданный
    }//объект возвращается из перегруженного постфиксного оператора декремент
//перегрузка бинарных операторов//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
    Counter operator+(Counter object)
    {
        cout << "Slozhenie.\n";
        this->count = this->count + object.GetCount();
        return *this;
    }
    Counter operator-(Counter obj)//возможная, но НЕ универсальная, перегрузка, которая возвращает новый объект, а не изменяет значения полей объектов в вычитании
    {
        cout << "Vichitanie.\n";
        return Counter(this->count - obj.GetCount());//конструктор с параметром принимает разность значений полей существующих объектов и созадает новый объект с полем, содержащим
    }//значение этой вычисленной разности
    bool operator<(Counter ob)const//перегрузка бинарного логического оператора сравнения <
    {
        return ((this->count < ob.GetCount()) ? true : false);
    }
    Counter& operator=(const Counter& ob)//перегрузка оператора присваивания =
    {
        if (this == &ob)//проверка на попытку самоприсваивания объекта самому себе же по типу: ob = ob;
        {
            cout << "Samoprisvaivanie.\n";//сообщаем пользователю
            return *this;//объект присваивается самому себе, поэтому ничего не делаем и возвращаем этот объект
```

```
        }
        cout << "Prisvaivanie.\n";//если это не самоприсваивание, то надо копировать значения из полей правого объекта соответствующим полям левого объекта
        this->count = ob.count;//в нашем классе только одно поле, поэтому одно копирование
        return *this;//возвращаем (посылаем) текущий объект
    }
};
int main()
{
    Counter c1, c2;//сработали конструкторы БЕЗ параметров
    cout << c1.GetCount() << ' ' << c2.GetCount() << endl;
    c1++;//постфиксный инкремент применяется к объекту класса Counter, в котором такой оператор ++ перегружен, поэтому код работает и увеличивает значение поля count с присвоением
    cout << c1.GetCount() << ' ' << c2.GetCount() << endl;
    ++c1;
    cout << c1.GetCount() << ' ' << c2.GetCount() << endl;
    c2 = c2++;
    cout << c1.GetCount() << ' ' << c2.GetCount() << endl;
    Counter c3 = ++c2;
    cout << c1.GetCount() << ' ' << c2.GetCount() << ' ' << c3.GetCount() << endl;
    c1 + c2;
    cout << c1.GetCount() << ' ' << c2.GetCount() << ' ' << c3.GetCount() << endl;
    c1 - c2;
    cout << c1.GetCount() << ' ' << c2.GetCount() << ' ' << c3.GetCount() << endl;
    Counter c4 = (c2 - c3);
    cout << c1.GetCount() << ' ' << c2.GetCount() << ' ' << c3.GetCount() << ' ' << c4.GetCount() << endl;
    if (c4 < c3)
    {
        cout << "True: c4 < c3.\n";
    }
    else
    {
        cout << "False: c4 < c3.\n";
    }
    if (c3 < c2)
    {
        cout << "True: c3 < c2.\n";
    }
    else
    {
        cout << "False: c3 < c2.\n";
    }
    if (c2 < c1)
    {
        cout << "True: c2 < c1.\n";
    }
    else
    {
        cout << "False: c2 < c1.\n";
    }
    c4 = c1;
    cout << c1.GetCount() << ' ' << c2.GetCount() << ' ' << c3.GetCount() << ' ' << c4.GetCount() << endl;
    c4 = c4;
    cout << c1.GetCount() << ' ' << c2.GetCount() << ' ' << c3.GetCount() << ' ' << c4.GetCount() << endl;
    Counter c5 = c1;
    cout << c1.GetCount() << ' ' << c2.GetCount() << ' ' << c3.GetCount() << ' ' << c4.GetCount() << ' ' << c5.GetCount() << endl;
    c5 = c1 + c2;//полноценно перегруженный оператор позволяет такой код и даже: c5 = c1 + c2 + c3 + c4
    cout << c1.GetCount() << ' ' << c2.GetCount() << ' ' << c3.GetCount() << ' ' << c4.GetCount() << ' ' << c5.GetCount() << endl;
    Counter c6(c4);//работает конструктор копирования
    cout << c1.GetCount() << ' ' << c2.GetCount() << ' ' << c3.GetCount() << ' ' << c4.GetCount() << ' ' << c5.GetCount() << ' ' << c6.GetCount() << endl;
    system("pause");
    return 0;
}
```

Тестируем:



В языке C++ для перегрузки операций используется ключевое слово *operator*, с помощью которого определяется специальная операция-функция (*operator function*).

Формат операции-функции:

типВозвращаемогоЗначения operator знакОперации (тип1 параметр1, тип2 параметр2, …)

{

операторы_тела_функции;

}

Перегрузка унарных операций

Любая унарная операция (обозначим ее символом ⊕) может быть определена двумя способами: либо как компонентная функция без параметров, либо как глобальная (возможно дружественная) функция с одним параметром. В первом случае выражение ⊕ Z означает вызов Z.operator ⊕ (), во втором − вызов operator ⊕(Z).

Унарные операции, перегружаемые в рамках определенного класса, могут перегружаться только через нестатическую компонентную функцию без параметров. Вызываемый объект класса автоматически воспринимается как операнд (то, над чем производится операция (действие)).

Унарные операции, перегружаемые вне области класса (как глобальные функции), должны иметь один параметр типа класса. Передаваемый через этот параметр объект воспринимается как операнд.

Синтаксис:

а) в первом случае (описание в области класса):

типВозвращаемогоЗначения operator знакОперации;

б) во втором случае (описание вне области класса):

типВозвращаемогоЗначения operator знакОперации(идентификатор_типа);

Перегрузка бинарных операций

Любая бинарная операция ⊕ может быть определена двумя способами: либо как компонентная функция с одним параметром, либо как глобальная (возможно дружественная) функция с двумя параметрами. В первом случае x⊕y означает вызов x.operator⊕(y), во втором – вызов operator ⊕(x,y).

Операции, перегружаемые внутри класса, могут перегружаться только нестатическими компонентными функциями с параметрами. Вызываемый объект класса автоматически воспринимается в качестве первого операнда.

Операции, перегружаемые вне области класса, должны иметь два операнда, один из которых должен иметь тип класса.

Перегрузка операции присваивания

Операция отличается тремя особенностями:

- операция не наследуется;
- операция определена по умолчанию для каждого класса в качестве операции поразрядного копирования объекта, стоящего справа от знака операции, в объект, стоящий слева;
- операция может перегружаться только в области определения класса. Это гарантирует, что первым операндом всегда будет леводопустимое выражение.

Формат перегруженной операции присваивания:

имяКласса& operator=(имяКласса&);

Отметим две важные особенности функции *operanor=*. Во-первых, в ней используется параметр-ссылка. Это необходимо для предотвращения создания копии объекта, передаваемого через параметр по значению. В случаи создания копии, она удаляется вызовом деструктора при завершении работы функции. Но деструктор освобождает распределенную память, еще необходимую объекту, который является аргументом. Параметр-ссылка помогает решить эту проблему. Во-вторых, функция

operator=() возвращает не объект, а ссылку на него. Смысл этого тот же, что и при использовании параметра-ссылки.

Функция возвращает временный объект, который удаляется после завершения ее работы. Это означает, что для временной переменной будет вызван деструктор, который освобождает распределенную память. Но она необходима для присваивания значения объекту. Поэтому, чтобы избежать создания временного объекта, в качестве возвращаемого значения используется ссылка.

2. Пример выполнения программы

1) Перегрузка функции инкремента (без параметров)

```
class Person
{
private:
	int age;
	...
public:
	...
	void operator++()
	{
		++age;
	}
};
int main()
{
	Person jon;
	++jon;
}
```

2) Перегрузка функции инкремента (с параметром)

```
class Person
{
private:
	int age;
	...
public:
	...
	void operator++(person&);
};
void Person::operator++(person& ob)
{
	++ob.age;
}
int main()
{
	Person jon;
	++jon;
}
```

3.Порядок выполнения работы

1. Изучить теоретические сведения к лабораторной работе.

Задание 2.

Реализовать алгоритм решения задачи. Ввести класс для работы с объектом «рациональная дробь» (вида *m*/*n*). Реализовать: приведение дроби к несократимому виду (оператор **!**).

2. Разработать на языке C++ программу вывода на экран решения задачи в соответствии с вариантом индивидуального задания, указанным преподавателем.

3. Отлаженную, работающую программу сдать преподавателю. Работу программы показать с помощью самостоятельно разработанных тестов.

Ответить на контрольные вопросы.

Задание 3:

Дописать один ваш класс, заданный в лабораторной работе № 23, перегрузив в нем арифметический оператор, оператор декремента (инкремента) и логический оператор сравнения.

При, например, сложении двух объектов класса все их числовые поля попарно складываются и возвращается экземпляр класса со значениями этих просуммированных полей. Если складывается объект класса и число, то к одному (или нескольким) из подходящих полей этого объекта нужно прибавить данное число и возвратить экземпляр класса с полем, увеличенным на значение данного числа.

При перегрузке оператора декремента или инкремента учитывать префиксный или постфиксный его характер, заданный по заданию вашего варианта (см. ниже). Например, при инкременте следует увеличить на единицу значения всех числовых полей вашего объекта и возвратить объект с измененными значениями полей.

По заданию № 3 следует перегрузить логический оператор сравнения, смысл которого определяется его обозначением (> - «больше ли объект?», < - «меньше ли объект?», >= - «объект больше либо равен?» и <= - «объект меньше либо равен?»). Если сравниваются 2 объекта на >=, то нужно вернуть булевское значение (true или false), указывающее, что первый объект больше или равен второму, хотя реально вы сравниваете конкретные поля данных объектов. Если сравнивается объект и число на <=, то нужно сравнить значение подходящего по типу поля объекта со значением этого числа («поле объекта меньше либо равно данному числу?») и вернуть соответствующее булевское значение (true или false).

Если полей не хватает для обеспечения демонстрации перегрузки операторов, можно дополнить класс новыми полями.

Продемонстрировать работу перегруженных операторов в функции main() с несколькими объектами, числами.

Каждому учащемуся 3 задания по его варианту:

1	**1**	+ (сложение двух объектов класса)
	2	– – (префиксный декремент)
	3	>= (сравнение объекта и вещественного числа)
2	**1**	– (вычитание из одного объекта класса другого объекта этого класса)
	2	++ (префиксный инкремент)
	3	<= (сравнение объекта и вещественного числа)
3	**1**	* (перемножение двух объектов класса)
	2	– – (постфиксный декремент)
	3	>= (сравнение объекта и целого числа)

4	**1**	/ (деление одного объекта класса на другой объект класса)
	2	++ (постфиксный инкремент)
	3	<= (сравнение объекта и целого числа)
5	**1**	+ (сложение объекта класса и целого числа)
	2	– – (постфиксный декремент)
	3	>= (сравнение двух объектов)
6	**1**	– (вычитание их объекта класса целого числа)
	2	++ (постфиксный инкремент)
	3	<= (сравнение двух объектов)
7	**1**	* (перемножение объекта класса и целого числа)
	2	– – (префиксный декремент)
	3	>= (сравнение двух объектов)
8	**1**	/ (деление объекта класса на целое число)
	2	++ (префиксный инкремент)
	3	<= (сравнение двух объектов)
9	**1**	+ (сложение объекта класса и вещественного числа)
	2	– – (префиксный декремент)
	3	>= (сравнение двух объектов)
10	**1**	– (вычитание их объекта класса вещественного числа)
	2	++ (префиксный инкремент)
	3	<= (сравнение двух объектов)
11	**1**	* (перемножение объекта класса и вещественного числа)
	2	– – (постфиксный декремент)
	3	>= (сравнение двух объектов)
12	**1**	/ (деление объекта класса на вещественное число)
	2	++ (постфиксный инкремент)
	3	<= (сравнение двух объектов)
13	**1**	+ (сложение двух объектов класса)
	2	– – (постфиксный декремент)
	3	>= (сравнение объекта класса и целого числа)
14	**1**	– (вычитание из одного объекта класса другого объекта этого класса)
	2	++ (префиксный инкремент)
	3	<= (сравнение объекта класса и целого числа)
15	**1**	* (перемножение двух объектов класса)
	2	– – (префиксный декремент)
	3	>= (сравнение объекта класса и целого числа)
16	**1**	/ (деление одного объекта класса на другой объект класса)
	2	++ (постфиксный инкремент)
	3	<= (сравнение объекта класса и целого числа)
17	**1**	+ (сложение объекта класса и целого числа)
	2	– – (префиксный декремент)
	3	>= (сравнение двух объектов)
18	**1**	– (вычитание их объекта класса целого числа)
	2	++ (постфиксный инкремент)
	3	<= (сравнение двух объектов)
19	**1**	* (перемножение объекта класса и целого числа)
	2	– – (постфиксный декремент)
	3	>= (сравнение двух объектов)
20	**1**	/ (деление объекта класса на целое число)
	2	++ (префиксный инкремент)
	3	<= (сравнение двух объектов)
21	**1**	+ (сложение объекта класса и вещественного числа)

	2	– – (постфиксный декремент)
	3	>= (сравнение двух объектов)
22	**1**	– (вычитание их объекта класса вещественного числа)
	2	++ (префиксный инкремент)
	3	<= (сравнение двух объектов)
23	**1**	* (перемножение объекта класса и вещественного числа)
	2	– – (префиксный декремент)
	3	>= (сравнение двух объектов)
24	**1**	/ (деление объекта класса на вещественное число)
	2	++ (постфиксный инкремент)
	3	<= (сравнение двух объектов)
25	**1**	+ (сложение двух объектов класса)
	2	– – (префиксный декремент)
	3	>= (сравнение объекта класса и вещественного числа)
26	**1**	– (вычитание из одного объекта класса другого объекта этого класса)
	2	++ (постфиксный инкремент)
	3	<= (сравнение объекта класса и вещественного числа)
27	**1**	* (перемножение двух объектов класса)
	2	– – (постфиксный декремент)
	3	>= (сравнение объекта класса и вещественного числа)
28	**1**	/ (деление одного объекта класса на другой объект класса)
	2	++ (префиксный инкремент)
	3	<= (сравнение объекта класса и вещественного числа)
29	**1**	+ (сложение объекта класса и целого числа)
	2	– – (постфиксный декремент)
	3	>= (сравнение объекта класса и вещественного числа)
30	**1**	– (вычитание их объекта класса целого числа)
	2	++ (префиксный инкремент)
	3	<= (сравнение объекта класса и вещественного числа)

Задание 4

№	Задача
1	Определить класс вектор (одномерный массив). В класс включить два конструктора для определения вектора по его размеру и путем копирования другого вектора. Определить операции над векторами: формирование нового вектора так, что каждый элемент нового вектора определяется следующим образом: c[i]=(a[i]>b[i])?a[i]:b[i];.
2	Определить класс-строку (символьный одномерный массив). В класс включить два конструктора: для определения класса строки строкой символов и путем копирования другой строки (объекта класса строки). Определить операции над строками: проверка строк на равенство.
3	Определить класс вектор (одномерный массив). В класс включить два конструктора для определения вектора по его размеру и путем копирования другого вектора. При задании вектора по его размеру предусмотреть его заполнение случайными числами. Определить операции над векторами: вставка элемента на определенное место вектора.
4	Определить класс-строку (символьный одномерный массив). В класс включить два конструктора: для определения класса строки строкой символов и путем копирования другой строки (объекта класса строки). Определить операции над

	строками: удаление символа из определенного места строки.
5	Определить класс матрицу (двумерный массив). В класс включить два конструктора для определения матрицы по количеству элементов и путем копирования другой матрицы. При задании матрицы предусмотреть ее заполнение случайными числами. Определить операции над матрицей: нахождение наименьшего значения матрицы.
6	Определить класс-строку (символьный одномерный массив). В класс включить два конструктора: для определения класса строки строкой символов и путем копирования другой строки (объекта класса строки). Определить операции над строками: конкатенация двух строк.
7	Определить класс вектор (одномерный массив). В класс включить два конструктора для определения вектора по его размеру и путем копирования другого вектора. При задании вектора по его размеру предусмотреть его заполнение случайными числами. Определить операции над векторами: добавления числа к вектору.
8	Определить класс-строку (символьный одномерный массив). В класс включить два конструктора: для определения класса строки строкой символов и путем копирования другой строки (объекта класса строки). Определить операции над строками: сравнение строк.
9	Определить класс вектор (одномерный массив). В класс включить два конструктора для определения вектора по его размеру и путем копирования другого вектора. При задании вектора по его размеру предусмотреть его заполнение случайными числами. Определить операции над векторами: удаление элемента из определенного места вектора.
10	Определить класс матрицу (двумерный массив). В класс включить два конструктора для определения матрицы по количеству элементов и путем копирования другой матрицы. При задании матрицы предусмотреть ее заполнение случайными числами. Определить операции над матрицей: получение новой матрицы, каждый элемент которой равен разности элементов двух других матриц.
11	Определить класс-строку (символьный одномерный массив). В класс включить два конструктора: для определения класса строки строкой символов и путем копирования другой строки (объекта класса строки). Определить операции над строками: преобразования символов строки в заглавные символы.
12	Определить класс вектор (одномерный массив). В класс включить два конструктора для определения вектора по его размеру и путем копирования другого вектора. При задании вектора по его размеру предусмотреть его заполнение случайными числами. Определить операции над векторами: определить наибольший элемент вектора.
13	Определить класс-строку (символьный одномерный массив). В класс включить два конструктора: для определения класса строки строкой символов и путем копирования другой строки (объекта класса строки). Определить операции над строками: удаления подстроки.
14	Определить класс вектор (одномерный массив). В класс включить два конструктора для определения вектора по его размеру и путем копирования другого вектора. При задании вектора по его размеру предусмотреть его

	заполнение случайными числами. Определить операции над векторами: умножить вектор на число.
15	Определить класс-строку (символьный одномерный массив). В класс включить два конструктора: для определения класса строки строкой символов и путем копирования другой строки (объекта класса строки). Определить операции над строками: перевертывание строки (запись символов в обратном порядке).
16	Определить класс матрицу (двумерный массив). В класс включить два конструктора для определения матрицы по количеству элементов и путем копирования другой матрицы. При задании матрицы предусмотреть ее заполнение случайными числами. Определить операции над матрицей: получение новой матрицы, каждый элемент которой равен сумме элементов двух других матриц.
17	Определить класс-строку (символьный одномерный массив). В класс включить два конструктора: для определения класса строки строкой символов и путем копирования другой строки (объекта класса строки). Определить операции над строками: вставка символа на определенное место строки.
18	Определить класс вектор (одномерный массив). В класс включить два конструктора для определения вектора по его размеру и путем копирования другого вектора. При задании вектора по его размеру предусмотреть его заполнение случайными числами. Определить операции над векторами: вычитание числа из вектора.
19	Определить класс-строку (символьный одномерный массив). В класс включить два конструктора: для определения класса строки строкой символов и путем копирования другой строки (объекта класса строки). Определить операции над строками: формирование новой строки из двух строк по правилу c=(a>b)?a:b (a, b, c – символы строк).
20	Определить класс вектор (одномерный массив). В класс включить два конструктора для определения вектора по его размеру и путем копирования другого вектора. При задании вектора по его размеру предусмотреть его заполнение случайными числами. Определить операции над векторами: сортировка элементов вектора по возрастанию.
21	Определить класс-строку (символьный одномерный массив). В класс включить два конструктора: для определения класса строки строкой символов и путем копирования другой строки (объекта класса строки). Определить операции над строками: преобразования символов строки в маленькие символы.
22	Определить класс матрицу (двумерный массив). В класс включить два конструктора для определения матрицы по количеству элементов и путем копирования другой матрицы. При задании матрицы предусмотреть ее заполнение случайными числами. Определить операции над матрицей: получение новой матрицы, каждый элемент которой равен произведению элементов двух других матриц.
23	Определить класс-строку (символьный одномерный массив). В класс включить два конструктора: для определения класса строки строкой символов и путем копирования другой строки (объекта класса строки). Определить операции над строками: проверка строк на неравенство.

24	Определить класс вектор (одномерный массив). В класс включить два конструктора для определения вектора по его размеру и путем копирования другого вектора. При задании вектора по его размеру предусмотреть его заполнение случайными числами. Определить операции над векторами: конкатенация векторов.
25	Определить класс-строку (символьный одномерный массив). В класс включить два конструктора: для определения класса строки строкой символов и путем копирования другой строки (объекта класса строки). Определить операции над строками: нахождения самого короткого слова.
26	Определить класс вектор (одномерный массив). В класс включить два конструктора для определения вектора по его размеру и путем копирования другого вектора. При задании вектора по его размеру предусмотреть его заполнение случайными числами. Определить операции над векторами: удаление элемента из определенного места вектора.
27	Определить класс матрицу (двумерный массив). В класс включить два конструктора для определения матрицы по количеству элементов и путем копирования другой матрицы. При задании матрицы предусмотреть ее заполнение случайными числами. Определить операции над матрицей: получение новой матрицы, каждый элемент которой равен разности элементов двух других матриц.
28	Определить класс-строку (символьный одномерный массив). В класс включить два конструктора: для определения класса строки строкой символов и путем копирования другой строки (объекта класса строки). Определить операции над строками: преобразования символов строки в заглавные символы.

4. Контрольные вопросы

1. Каким образом можно перегрузить унарные операции?
2. Чем отличается перегрузка префиксных и постфиксных унарных операций?
3. Сколько операндов должна иметь бинарная функция-операция, определяемая внутри класса?

Литература

Страуструп, Б. Язык программирования C++ / Б. Страуструп. – СПб. : БИНОМ, 2011.

Павловская, Т. А. C++. Объектно-ориентированное программирование: практикум / Павловская, Т.А., Щупак. – СПб. : Питер, 2011.

Преподаватель Белокопыцкая Ю.А.

Рассмотрено на заседании цикловой
комиссии ПОИТ № 10
Протокол № ____ от «___» __ 2017 г.
Председатель ЦК_____С.В. Банцевич