**Лабораторная работа № 3**

**Программирование с использованием перегруженных функций и операторов. Использование вложенных классов**

**Цель работы:** Получить практические навыки работы с перегруженными операторами и методами класса, а также навыки взаимодействия классов с использованием композиции (вложенных классов)

**Теоретические сведения**

**Перегрузка функций и операторов.**

Перегрузка является статическим полиморфизмом (полиморфизм времени компиляции)

Статический полиморфизм – гибкость и комфорт при программировании.

Полиморфи́зм — возможность объектов с одинаковой спецификацией иметь различную реализацию.

Перегрузка функций – это использование одного имени для нескольких функций.

Например функции перегружаются для выполнения одинаковых по смыслу действий над объектами различных типов.

**Перегружаемые функции.**

Функции с одним и тем же названием должны различаться сигнатурой (количеством или типами параметров) Возвращаемое значение в сигнатуру не входит

Повторные объявления (или ошибки) при перегрузке функций.

1. typedef не вводит нового типа

**typedef double NALOG;**

повторное объявление функции

**NALOG calc(NALOG );**

**double calc( double );**

1. Спецификаторы const или volatile не принимаются во внимание, если параметры передаются по значению, так как не влияют на их интерпретацию.

// повторное объявлене одной и той же функции

**int f ( int );**

**void f ( const int );**

1. Eсли спецификатор const или volatile применяется к указателю или ссылке, то при сравнении объявлений они учитываются.

// функции перегружаются

**void f( int\* );**

**void f( const int\* );**

**void f( int& );**

**void f( const int& );**

**Пример :** Функции для печати значений разного типа данных.

…

// печать целого числа

**void print ( int i )**

**{ printf ("\n%d",i);**

**}**

// печать дробного числа

**void print ( double d )**

**{ printf ("\n%lf",d);**

**}**

// печать строки

**void print ( char \* s )**

**{ printf ("\n%s",s);**

**}**

В вызывающей функции компилятор сам выберет одну из функций print(), основываясь на анализе типов аргументов

**int i=5;**

**double pi=3.141592;**

**print (i);**

**print(pi);**

**print ("перегрузка функций");**

Три приведенные функции print в объектном модуле будут соответствовать трем функциям с различными именами, модифицированные транслятором имена функций содержат информацию о количестве и типах параметров, причем модификацию имен компилятор выполняет для всех функций, а не только перегружаемых. Для того, чтобы функцию С++ можно было бы вызвать из программы на С, необходимо запретить модификацию имен, для чего функцию нужно описать с описателем extern "C".

**extern "C"** // отдельная функция

**extern "C"** // несколько функций

**{ int fun2(int);**

**double fun3(double);**

**}**

Очевидно, что функции описанные как extern "C" не могут быть перегружаемыми.

**Область видимости и перегрузка.**

Блок, функция, класс задают свои области видимости.

Все перегруженные функции объявляются в одной и той же области видимости.

Локально объявленная функция не перегружает, а скрывает глобальную

**using namespace std;**

**void print( int r ) {cout<< r<<endl;};**

**void print( double r ) {cout<< r<<endl;};**

**void print( const char\* s ) {cout<< s<<endl;};**

**void f (double val)**

**{ void print (double k);** // маскируются все print

**print("строка");** // ошибка при трансляции

**print (val);**

**};**

**int \_tmain(int argc, \_TCHAR\* argv[])**

**{ print("строка");**

**print (5);**

**f (3.14);**

**system ("pause");**

**return 0;**

**}**

Каждый класс определяет собственную область видимости, поэтому функции, являющиеся членами двух разных классов, не перегружают друг друга.

**class A**

**{int q;**

**public:**

**void show(const char\* s){cout<<s<<q<<endl;};**

**};**

**class B**

**{double w;**

**public:**

**void show(){cout<<"class B"<<w<<endl;};**

**};**

**Перегрузка конструкторов:**

Создание инициированных и неинициированных объектов, конструкторов копирования.

**class Int**

**{ int i;**

**public:**

// Перегрузка конструкторов

**Int(){i=0;}** // без инициализации

**Int (int n){i=n;}** // с инициализацией

**Int (const Int& k){i=k.i;}**//конструктор копирования

**void print(){cout<<"int:"<<i<<endl;}**

**void set (int k){i=k;}**

**int get (){return i;}**

**voiv show\_m()**

**}**;//----------конец определения класса Int

**const int N=5;**

**void main()**

**{ Int a,b(10),c(b),\*pi;**

**int i;**

**a.print(); b.print(); c.print();**

**Int A[N];** // массив без инициализации

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

**{A[i].set(i+1);A[i].print();}** // инициализация

**Int B[]={10,20,30};** // массив с инициализацией

**for (i=0;i<3;i++) B[i].print();**

**pi=new Int[N];** // конструктор без инициализации

// иниициализация данными массива А

**for(i=0;pi+i<pi+N;i++)**

**{(pi+i)->set(A[i].get());**

**(pi+i)->print();**

**}**

**system ("pause");**

**return 0;**

**}**

**Указатель this**

До сих пор мы не задавали вопроса, как метод определяет с полями какого именно объекта надо работать, если объектов много, (например объекты Int a,b,c)

В тексте метода set () на этот счет нет никаких указаний!!!!

При вызове методы класса получают:

* список параметров
* неявный константный указатель this c адресом объекта, вызвавшего метод

**void set (int k){this->i=k;}**

Указатель this константный, то есть всегда указывает на один и тот же объект, адресные операции с ним запрещены.

**int a;**

**this=&a;** // ошибка!!!

**this++;** // ошибка!!!

Указатель this необходим когда необходимо работать с адресами объектов

1. Необходимо получить доступ ко всему объекту, а не к отдельным его полям
2. Объект является параметром функции или возвращаемым значением.

Напомним, что друзья класса – это внешние функции и классы, у которых доступ такой же, как у методов класса.

**class Int**

**{ int i;**

**public:**

**. . .**

**friend void show\_m(Int\*, int);**

**};** //----------конец определения класса Int

**void show\_m (Int\* pI, int n)**

**{int i;**

**cout<<"-------mass ["<<n<<"]----------"<<endl;**

**for (i=0;i<n;i++,pI++) cout<<pI->i<<'\t';**

**cout<<endl;**

**}**

**void main()**

**{ Int A[N];** // массив без инициализации

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

**{A[i].set(i+1);A[i].print();}** // инициализация

**show\_m(A,N);**

**system ("pause");**

**return 0;**

**}**

**Перегрузка операторов**

Для выполнения действий над абстрактными (пользовательскими) типами в классическом С используются функции.

В С++ можно использовать операции

то есть можно создавать свои операции над экземплярами класса (новыми, пользовательскими типами).

Об операторе, который работает с абстрактными типами (объектами классов), говорят, что он перегружен или переопределен

Например, можно определить оператор умножения для матриц или комплексных чисел.

Однако одной функции на оператор зачастую оказывается недостаточно. Например, матрицу можно умножить на другую матрицу, а можно на константу.

**При перегрузке операторов существует несколько ограничений:**

* нельзя создавать новые символы операций
* нельзя переопределять операции для стандартных типов
* Нельзя поменять правила, определенные в трансляторе, например, бинарный оператор сделать унарным и наоборот.
* нельзя переопределять операции :

:: разрешение области видимости

. выбор элемента

\* (разыменование)

?: троичный оператор

* Следующие операторы должны быть реализованы только в виде нестатических методов класса:

= [] () ->

* переопределение операций не меняет их приоритетов, а также порядок их выполнения (слева направо или справа налево);
* для параметров в операторных функциях нельзя задавать значения по умолчанию.

Например :

<< только для бинарной операции,

! только для унарной,

\* только для бинарной;

Стандартное поведение перегруженных операторов не определено, например

* коммутативный закон для сложения:

а + b равно b + a;

* комбинированная операция, например: i=i+j; эквивалентно i += j

и некоторые другие правила.

Соблюдение подобных правил для перегруженных операторов лежит на программисте. Например, необходимо явно написать одинаковое поведение для операторов + и +=

**Возвращаемые значения**

Очень немного операторов жестко требуют какого-то определенного типа возвращаемого значения. В большинстве своем вы абсолютно свободны в выборе типа результата. Однако правила языка и контекст использования операторов зачастую дают рекомендации на тип значения (так, например, желательно, чтобы оператор присваивания возвращал тот же тип, что и у своего объекта).

**Два способа реализации перегружаемых операторов**

Операторы реализуются (перегружаются):

* в виде методов класса,
* в виде дружественных функций.

Реализация в виде методов класса

Оператор перегружается с помощью операторной функции

Также бывают противоположные ситуации - приходится использовать не метод, а

дружественную функцию.

**Бинарные перегруженные операторы**

(реализация через метод класса) имеют 1 параметр

Через указатель this передается левый операнд, правый операнд – через параметр

**class Int**

**{ int i;**

**public:**

// перегрузка операторов

// оператор+ не меняют значения параметров

**const Int operator + (const Int& right)**

**{cout<<"op+"<<endl;**

**return Int(this->i+right.i);**

**}**

// оператор+= меняет левый параметр

**Int& operator += (const Int& right)**

**{cout<< "op+="<<endl;**

**i+=right.i;**

**return \*this;**

**}**

**}**; //----------конец определения класса Int

**void main()**

**{ Int a(1),b(2),c(3);**

**c+=a+b; c.print();**

**}**

**На экране:**

**Ор+**

**Ор+=**

**Int: 6**

**Void main()**

**{ Int a(1),b(2),c;**

**c=a+10; c.print();** // На экране:

**}** Ор+

Int: 11

**c=10+a;** // ошибка трансляции

При перегрузке в виде методов порядок операндов фиксирован: левый операнд передается неявно через this, а правый – явно через параметр

Чтобы реализовать коммутативность оператора +, нужна реализация через дружественную функцию.

Перегрузим оператор + для работы с константой

**class Int**

**{ int i;**

**public: . . .**

**friend const Int operator + (int k, const Int& right);**

**}; /**/----------конец определения класса Int

У независимой функции нет указателя , поэтому все параметры задаются явно :

1-ый параметр – левый операнд, 2-ой – правый

**const Int operator + (int k, const Int& right)**

**{cout<<"op+"<<endl;**

**return Int(right.i+k);**

**}**

**void main()**

**{ Int a(1),b(2),c;**

**c=a+10; c.print();**

**b=1+a; b.print();**

**}**

**Перегрузка унарного оператора ++**

**class Int**

**{ int i;**

**public: . . .**

// префиксная форма

**friend const Int& operator ++ (Int& q);**

// постфиксная форма

**friend const Int operator ++ (Int& q, int);**

**};**//----------конец определения класса Int

// префиксная форма возвращает значение после инкремента

**const Int& operator ++ (Int& q)**

**{ cout<<"++op"<<endl;**

**q.i++;**

**return q;**

**}**

// постфиксная форма возвращает значение до инкремента

**const Int operator ++ (Int& q, int)**

**{cout<<"op++"<<endl;**

**Int t(q);** // сохранение значения

**q.i++;**

**return t;**

**}**

**void main()**

**{ Int a(1),b(2),c;**

**Int d=++a; d.print();**

**Int e = b++; e.print();**

**}**

**Аргументы и возвращаемые значения**

1. Аргумент только читается, но не изменяется – передаем как ссылку на const

**const Int operator + (const Int& right);**

**Int a(1),b(2),c(3);**

**Int d=a+b;(a+b).show()**

**а+b – временный объект всегда константный**

1. Изменяется левосторонний аргумент

передаем как ссылку, такой аргумент может быть изменен

**Int& operator += (const Int& right)**

**Int d+=a**

1. Если оператор должен создавать новое значение, то нужно создавать новый объект

**const Int operator + (int k, const Int& right)**

**{cout<<"op+"<<endl;**

**return Int(right.i+k);**

**}**

возвращаемое значение – новый объект, т.к. этот оператор может генерировать временные объекты.

Возвращаемое значение константное, что запрещает изменять левый операнд.

1. Все операции присваивания изменяют левое значение, чтобы было возможно множественное присваивание

**Int a(1),b(2),c(3);**

**a=b=c;**

**(a=c).print();**

**Пример :**

**class Byte**

**{unsigned char b;**

**public:**

**Byte (unsigned char bb='a'): b(bb){};**

**void print ()**

**{cout<<":"<<b<<endl;**

**}**

// перегрузка операторов

**const Byte& operator++ ()**

**{cout<<"++Byte"<<endl;**

**b++;**

**return \*this;**

**}**

**const Byte& operator-- ()**

**{cout<<"--Byte"<<endl;**

**b--;**

**return \*this;**

**}**

**const Byte operator+ (const Byte& right) const**

**{cout<<"Byte+"<<endl;**

**return Byte(b+right.b);**

**}**

**friend int operator>(const Byte& left, const Byte& right);**

**};** //-------- конец определения Byte

**int operator>(const Byte& left, const Byte& right)**

**{cout<<"Byte>"<<endl;**

**return left.b>right.b;**

**};**

**int \_tmain(int argc, \_TCHAR\* argv[])**

**{ Byte f;**

**f.print();**

**++f; f.print();**

**Byte d=f+10; d.print();**

**Byte e=(f+d+10);e.print();**

**if (f>d) f.print();**

**else d.print();**

**system("pause");**

**return 0;**

**}**

**Перегрузка присваивания**

Вновь создаваемый объект проходит инициализацию (вызывается конструктор), в других случаях вызывается **operator =**

**Byte d;**

**Byte t=d;** //работает конструктор копирования

**t=d;** //работает оператор присваивания

1. operator = может быть только функцией класса и неразравно связана с «левым» объектом
2. Глобальное переопределение operator = запрещено!

operator = обязательно должна быть функцией класса

Иными словами область видимости – класс

**class Numb**

**{ int a,b;**

**double c;**

**public:**

// конструкторы

**Numb (int aa=0, int bb=0, double cc=0):**

**a(aa),b(bb),c(cc){};**

// перегрузка присваивания

**Numb& operator= (const Numb& right)**

**{ a=right.a;**

**b=right.b;**

**c=right.c;**

**return \*this; }**

// перегрузка оператора вывода <<

**friend ostream& operator<<**

**(ostream& os, const Numb& right)**

**{return os<<"a="<<right.a<<",b="<<right.b<<",c="<<right.c;**

**}**

**};**

**void main()**

**{ Numb q1, q2(1,5,6.45);**

**cout<<"q1:"<<q1<<endl;**

**cout<<"q2:"<<q2<<endl;**

**system("pause");**

**return 0;**

**}**

. **Контрольные вопросы**

1. Какие классы и функции называются дружественными?
2. Как осуществляется перегрузка операций?
3. Сколько аргументов требуется для определения перегруженной унарной (бинарной) операции?
4. Чем отличается действие перегруженной операции ++ при ее использовании в префиксной форме от использовании в постфиксной форме?
5. Какие ограничения существуют при перегрузке операторов?
6. Какую роль играет указатель this при перегрузке операторов?
7. Какую роль играет указатель this при вызове метода класса?
8. Приведите примеры повторных объявлений функций (ошибок перегрузки).

**Общие требования**

1. Для подготовки к лабораторной работе используйте данные из папок «Справочная информация» и «Теория»
2. Начать работу над новым проектом
3. В каждой базе должна быть статическая переменная для учета общего числа элементов в базе данных.
4. Данные размещаются в динамической памяти.
5. Для выполнения функций, указанных в задании, написать диалоговый интерфейс, позволяющий выполнять функции в произвольном порядке многократно
6. При выполнении функции «выход из программы» нужно сохранить базу на диске
7. Первичное создание базы – ввод данных с клавиатуры
8. Если программа уже запускалась, то данные загружаются из файла перед выходом на диалог. Иными словами вносятся изменения и дополнения в уже существующую базу данных.
9. Обязательные функции для всех вариантов:

* добавить новый элемент в базу
* распечатка данных в табличном виде
* выход из программы

1. Остальные функции для работы с базой указаны в задании индивидуально.
2. Примеры диалогового интерфейса и табличного вывода смотрите в лабораторной работе №1
3. Перегруженные операторы реализовывать как с помощью дружественной функции (ДФ), так и с помощью метода класса (МК). Если в задании не указан метод реализации – решаете по своему усмотрению.
4. В классе должны быть все необходимые элементы для работы тестовой программы.

**Использование вложенных классов (композиции)**

В данном примере рассмотрим взаимодействие классов с использованием вложенного класса (композиции) . Следующий пример показывает синтаксические особенности композиции.

**Пример:**

1. Создать два класса.

Класс **Point** **(точка)** : два целых числа (координаты).

Класс **Circle(окружность)**: точка – центр окружности (вложенный объект) и целое число (радиус).

1. Перегрузить операторы для класса Сircle:

**>>** (ввод c клавиатуры)

**!** (унарная операция, проверка) лежит ли центр окружности на координатных осях

1. Создать независимую функцию для вычисления площади окружности.

**Тестовая программа:**

Создать исходный массив С для N окружностей (данные ввести с клавиатуры), исходный массив распечатать.

Определить окружности с центрами лежащими на координатных осях, результат распечатать.

Все данные в классах –private. Кроме элементов явно перечисленных в задании, определите все необходимые элементы в классах для выполнения тестовой программы.

**#include <iostream>**

**#define n 3**

**using namespace std;**

// Область определения классов

**class Point**

**{ int x;**

**int y;**

**public:**

**Point() { x = y = 0;}** // конструктор по-умолчанию

**void show();** // печать данных

**void input();** // ввод данных с клавиатуры

// методы доступа к закрытым данным

**void setx(int \_x){x=\_x;}**

**void sety(int \_y){y=\_y;}**

**int getx (){return x;}**

**int gety (){return y;}**

**};** //--------------------------------- end of class Point

**class Circle**

**{ Point center;** //встроенный объект класса Point

**int radius;**

**public:**

**Circle() { radius = 1;}** // конструктор по-умолчанию

// методы класса Circle

**void show();**

**void input();**

// перегруженные операторы

**friend void operator>>(istream& is,Circle& c);**

**bool operator! ();**

**};** //---------------------------------- end of class Circle

//\*\*\*\*\*\*\*\*\* реализация методов класса Point

**void Point::show()**

**{cout<<"Point::show()\t";**

**cout<<"x="<<x<<"\ty="<<y;**

**}**

**void Point:: input ()**

**{cout<<"Point:: input ()";**

**cout<<"x="; cin>>x;**

**cout<<"\ty="; cin>>y;**

**}**

//\*\*\*\*\*\*\*\*\* реализация методов класса Circle

**void Circle::show()** // печать объектов Circle

**{cout<<"Circle::show()\t";**

**center.show();** // печать координат центра (вызов метода класса Point)

**cout<<"\tr="<<radius<<endl;**

**}**

**void Circle:: input()**

**{cout<<"\nCircle:: input()";**

**center.input();**  // ввод точки

**cout<<"\tr="; cin>>radius;** // ввод радиуса (вызов метода класса Point)

**}**

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

**bool Circle::operator! ()** // проверка центра окружности

**{if (center.getx()==0 || center.gety()==0) return true;**

**else return false;**

**}**

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

// реализация перегруженных операторов (друж.функция)

**void operator>>(istream& is, Circle& c)** // ввод объекта Circle с клавиатуры

**{int x1,y1;**

**cout<<"\nx="; is>>x1;** // ввод с клавиатуры в переменную

**c.center.setx(x1);** // запись в поле встроенного объекта (класса Point)

**cout<<"y="; is>>y1;**

**c.center.sety(y1);**

**cout<<"r="; is>>c.radius;** // запись в "своё" поле объекта класса Circle

**}**

// тестовая программа с использованием методов класса Circle

**int main()**

**{ int i;**

**Circle C[n];** // создать массив окружностей (работают конструкторы по-умолчанию)

// Задаем значения с клавиатуры

**cout<<"Ввод данных для "<<n<<" окружностей\t";**

**for (int i=0; i<n;i++) C[i].input();**

// Печатаем исходные данные

**for (i=0; i<n; i++) C[i].show();**

**return 0;**

**}**

// Тестовая программа с использованием перегруженных операторов

**int main()**

**{ Circle C[n];** // создать массив окружностей (работает конструктор по-умолчанию)

// Вводим и Печатаем исходные данные

**cout<<"Ввод данных, "<<n<<" окружностей\t";**

**for (int i=0; i<n;i++) cin>>C[i];** // используем перегруженный оператор >>

**for (int i=0; i<n; i++) C[i].show();**

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

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

**{ cout<<"Окружность "<<i;**

// используем перегруженный оператор !

**if (!C[i]) cout <<" лежит на оси"<<endl;**

**else cout <<" не лежит на оси"<<endl;**

**}**

**return 0;**

**}**

**Общие требования к заданиям**

1. Продолжить работу над проектом
2. Создать новый класс и встроить его объект в существующий класс, согласно заданию
3. Данные размещаются в динамической памяти.
4. Для выполнения функций, указанных в задании, написать диалоговый интерфейс, позволяющий выполнять функции в произвольном порядке многократно
5. При выполнении функции «выход из программы» нужно сохранить базу на диске
6. Первичное создание базы – ввод данных с клавиатуры
7. Если программа уже запускалась, то данные загружаются из файла перед выходом на диалог. Иными словами вносятся изменения и дополнения в уже существующую базу данных.
8. Обязательные функции для всех вариантов:

* добавить несколько новых элементов в базу
* распечатка данных в табличном виде
* выход из программы

1. Остальные функции для работы с базой указаны в задании индивидуально.
2. Примеры диалогового интерфейса и табличного вывода смотрите в лабораторной работе №1
3. Перегруженные операторы реализовывать как с помощью дружественной функции (ДФ), так и с помощью метода класса (МК). Если в задании не указан метод реализации – решаете по своему усмотрению.

|  |  |
| --- | --- |
| Номер компьютера | **Варианты заданий** |
| 1,11,  21 | **Тема: Пищевое производство (расчеты)**  **Создаем новый класс date**  **Данные-члены класса располагаются в закрытой области**  int day; // день  int month; // месяц  int year; // год  Добавьте в класс **date** все необходимые элементы для работы тестовой программы  **Модифицируем класс food, добавляем встроенный объект класса date**  **Данные-члены класса изменяются следующим образом:**  char\* name ; (pri) //Название изделия  int quant; // количество  double cost; //стоимость  date shelf\_life; // срок годности изделия (встроенный объект класса date)  static int count; // счетчик существующих объектов класса  **Внести изменения или написать заново следующие перегруженные операторы:**  **<<** вывод объекта класса **food** в выходной поток **cout** (на терминал)  **>** оператор сравнения для объекта класса **food** и объекта класса **date**  **Тестовая программа :**   1. Ввести несколько объектов класса **food** 2. Вывод данных на терминал выполнять с помощью перегруженного оператора << 3. Используя перегруженный оператор > найти все изделия с истекшим сроком годности. Текущую дату ввести с клавиатуры,результат поиска запомнить в массиве **food** **Rez** 4. Результаты выводить на экран в табличном виде |

|  |  |
| --- | --- |
| 2,12,  22 | **Тема: Поезд (заказ билетов)**  **Создаем новый класс man**  **Данные-члены класса располагаются в закрытой области (вместо С-строки можно использовать класс string)**  char\* fam; //Фамилия пассажира  char\* name; // Имя пассажира  Добавьте в класс **man** все необходимые элементы для работы тестовой программы  **Модифицируем класс ticket**  **Данные-члены класса изменяются следующим образом:**  man fname ; //Фамилия и имя пассажира (встроенный объект класса man)  char\* arrival; // пункт назначения  double price; // цена билета  static int count; // счетчик существующих объектов класса  **Внести изменения или написать заново следующие перегруженные операторы:**  **<<** ввод объекта класса **ticket** из входного потока **cin** (ввод с клавиатуры)  **=** оператор присваивания**t** для объектов класса **ticket**  **Тестовая программа :**   1. Добавить новый элемент, используя перегруженный оператор **<<** 2. Найти все объекты по заданному пункту назначения, результат поиска запомнить в массиве **ticket Rez**, используя перегруженный оператор **=** 3. Результаты выводить на экран в табличном виде |

|  |  |
| --- | --- |
| 3,13,  23 | **Тема: Почта (ценное письмо)**  **Создаем новый класс man**  **Данные-члены класса располагаются в закрытой области (вместо С-строки можно использовать класс string)**  char\* fam; //Фамилия  char\* name; // Имя  Добавьте в класс **man** все необходимые элементы для работы тестовой программы  **Модифицируем класс letter**  **Данные-члены класса изменяются следующим образом:**  char\* adr1; (pri) //адрес получателя  man fname1 ; //Фамилия и имя получателя (встроенный объект класса man)  char\* adr2; // адрес отправителя  man fname2 ; //Фамилия и имя отправителя (встроенный объект класса man  double cost; // стоимость письма  **Внести изменения или написать заново следующие перегруженные операторы:**  **==** (проверка на равенство) дляобъекта класса **letter** иобъекта класса **man**  **<<** ввод объекта класса **letter** из входного потока **cin** (ввод с клавиатуры)  **Тестовая программа (функция main):**   1. Добавить новый элемент, используя перегруженный оператор **<<** 2. Найти все письма определенного отправителя, используя перегруженный оператор **==,** результат поиска запомнить в массиве **letter Rez** 3. Результаты выводить на экран в табличном виде |

|  |  |
| --- | --- |
| 4,14,  24 | **Тема: Студент**  **Создаем новый класс date**  **Данные-члены класса располагаются в закрытой области**  int day; // день  int month; // месяц  int year; // год  Добавьте в класс date все необходимые элементы для работы тестовой программы  **Модифицируем класс student, добавляем встроенный объект класса date**  **Данные-члены класса изменяются следующим образом:**  char\* fam; (pri) //фамилия студента  char\* name; (pri) // имя студента  date birthday // день рождения (встроенный объект класса date)  int grup // номер группы  static int count; // счетчик существующих объектов класса  **Внести изменения или написать заново следующие перегруженные операторы:**  **<<** вывод объекта класса **student** в выходнойпоток **cout**  **==** сравнение объекта класса **student** с объектом класса **date**  **Тестовая программа (функция main):**   1. Добавить несколько новых элементов 2. Вывод данных на терминал выполнять с помощью перегруженного оператора << 3. Найти всех студентов с заданным днем рождения, используя перегруженный оператор  **==** , результат поиска запомнить в массиве **student Rez** 4. Результаты выводить на экран в табличном виде |

|  |  |
| --- | --- |
| 5,15,  25 | **Тема: Производство (поставщики)**  **Создаем новый класс date**  **Данные-члены класса располагаются в закрытой области**  int day; // день  int month; // месяц  int year; // год  Добавьте в класс date все необходимые элементы для работы тестовой программы  **Модифицируем класс supplier, добавляем встроенный объект класса date**  **Данные-члены класса изменяются следующим образом:**  char\* firma; (pri) // название фирмы поставщика  double raw\_ material // сумма поставки сырья  date delivery\_date // дата поставки сырья (встроенный объект класса date)  double payment; (pri) // оплата сырья  static int count; // счетчик существующих объектов класса  **Внести изменения или написать заново следующие перегруженные операторы:**  **>>**  ввод объекта класса **supplier** из входного потока **cin** (с клавиатуры)  **>** сравнение объекта класса **supplier** с объектом класса **date**  **Тестовая программа (функция main):**   1. Добавить новый элемент, используя перегруженный оператор **<<** 2. Отсортироватьпо уменьшению даты поставки сырья, используя перегруженный оператор > 3. Результаты выводить на экран в табличном виде |

|  |  |
| --- | --- |
| 6,16,  26 | **Тема: Поликлиника**  **Создаем новый класс time**  **Данные-члены класса располагаются в закрытой области**  int hour; // часы  int minute; // минуты  Добавьте в класс **time** все необходимые элементы для работы тестовой программы  **Модифицируем класс clinic, добавляем встроенный объект класса time**  **Данные-члены класса изменяются следующим образом:**  char\* fam; (pri) //фамилия врача  char\* name; (pri) // имя врача  time begin;// начало приема пациентов (встроенный объект класса time)  int specialty; // специальность(1-терапевт, 2-кардиолог, 3- стоматолог)  static int count; // счетчик существующих объектов класса  **Внести изменения или написать заново следующие перегруженные операторы:**  **>>**  ввод объекта класса **clinic** из входного потока **cin** (с клавиатуры)  **==** сравнение объекта класса **clinic** собъектом класса  **time**  **Тестовая программа (функция main):**   1. Добавить несколько новых элементов (ввод с клавиатуры с использованием перегруженного оператора **>>**) 2. Найти всех врачей с заданным началом приема пациентов, используя перегруженный оператор **==** , результат поиска запомнить в массиве **clinic Rez** 3. Результаты выводить на экран в табличном виде |

|  |  |
| --- | --- |
| 7,17,27 | **Тема: Склад (товары)**  **Создаем новый класс date**  **Данные-члены класса располагаются в закрытой области**  int day; // день  int month; // месяц  int year; // год  Добавьте в класс **date** все необходимые элементы для работы тестовой программы  **Модифицируем класс storage, добавляем встроенный объект класса date**  **Данные-члены класса изменяются следующим образом:**  char\* name; (pri) //Название товара  double price; (pri) // цена  int quant; // количество  date delivery\_date; // дата поступления (встроенный объект класса date)  static int count; // счетчик существующих объектов класса  **Внести изменения или написать заново следующие перегруженные операторы:**  **> (сравнение)** объекта класса **storage** с объектом класса **date**  **<<** вывод объекта класса **storage** в выходнойпоток **cout**  **Тестовая программа (функция main):**   1. Добавить несколько новых элементов 2. Вывод данных на терминал выполнять с помощью перегруженного оператора << 3. Найти все товары с датой поступления ранее заданной, используя перегруженный оператор > 4. Результаты выводить на экран в табличном виде |

|  |  |
| --- | --- |
| 8,18,  28 | **Тема: Банк (депозит)**  **Создаем новый класс man**  **Данные-члены класса располагаются в закрытой области (вместо С-строки можно использовать класс string)**  char\* fam; //Фамилия клиента  char\* name; // Имя клиента  Добавьте в класс **man** все необходимые элементы для работы тестовой программы  **Модифицируем класс bank**  **Данные-члены класса изменяются следующим образом:**  man fname // фамилия и имя клиента(встроенный объект класса man)  double sum; // Сумма на счете (сумма депозита )  static int count; // счетчик существующих объектов класса  **Внести изменения или написать заново следующие перегруженные операторы:**  **>>**  ввод объекта класса **bank** из входного потока **cin** (с клавиатуры)  **==** сравнение объекта класса **bank** собъектом класса  **man**  **Тестовая программа (функция main):**   1. Добавить несколько новых элементов (ввод с клавиатуры с использованием перегруженного оператора **>>**) 2. Найти депозит по фамилии и имени клиента и добавить число к полю «сумма на счете», использовать перегруженные операторы == и **+** 3. Результаты выводить на экран в табличном виде |

|  |  |
| --- | --- |
| 9,19,  29 | **Тема: Работник (зарплата)**  **Создаем новый класс man**  **Данные-члены класса располагаются в закрытой области (вместо С-строки можно использовать класс string)**  char\* fam; //Фамилия работника  char\* name; // Имя работника  Добавьте в класс **man** все необходимые элементы для работы тестовой программы  **Модифицируем класс worker**  **Данные-члены класса изменяются следующим образом:**  man fname // фамилия и имя работника(встроенный объект класса man)  int department; (pri)// отдел  double salary; // оклад  static int count; // счетчик существующих объектов класса  **Внести изменения или написать заново следующие перегруженные операторы:**  **==** сравнение объекта класса **worker** собъектом класса  **man**  **>>**  ввод объекта класса **worker** из входного потока **cin** (с клавиатуры)    **Тестовая программа (функция main):**   1. Добавить несколько новых элементов (использовать перегруженный оператор **>>**) 2. Найти работника с заданными фамилией и именем, используя перегруженный оператор **==** 3. Результаты выводить на экран в табличном виде |

|  |  |
| --- | --- |
| 10,20,  30 | **Тема: Преподаватель**  **Создаем новый класс date**  **Данные-члены класса располагаются в закрытой области**  int day; // день  int month; // месяц  int year; // год  Добавьте в класс **date** все необходимые элементы для работы тестовой программы  **Модифицируем класс prepod, добавляем встроенный объект класса date**  **Данные-члены класса изменяются следующим образом:**  char\* fam; (pri)//фамилия преподавателя  char\* name; // имя преподавателя  date start\_date// дата поступления на работу(встроенный объект класса date)  int status; (pri) //ученое звание( 1- преподаватель, 2- доцент, 3-доктор наук)  static int count; // счетчик существующих объектов класса  **Внести изменения или написать заново следующие перегруженные операторы:**  **=** оператор присваиваниядля объектов класса **prepod**  **>** сравнение объекта класса **prepod**  с объектом класса **date**  **>>**  ввод объекта класса **prepod** из входного потока **cin** (с клавиатуры)  **Тестовая программа (функция main):**   1. Добавить несколько новых элементов , использовать перегруженный оператор << 2. Найти всех преподавателей, которые поступили на работу ранее заданной даты, использовать перегруженный оператор > 3. Результат поиска запомнить в массиве **prepod Rez**, используя перегруженный оператор = 4. Результаты выводить на экран в табличном виде 5. Результаты выводить на экран в табличном виде |