Лабораторная работа N 4

Перегрузка операторов. Дружественные функции.

1. Введение

Программирование на C++ — процесс, чувствительный к типам и основанный на типах. Программист может использовать встроенные типы, а может определить и новые типы. Встроенные типы можно использовать с богатым набором операций C++. Операции обеспечивают программиста краткими средствами записи для выражения манипуляций с объектами встроенного типа.

Программист может также использовать операции с типами, определенными пользователем. Хотя C++ и не позволяет создавать новые операции, он все же позволяет перегружать существующие операции так, что при использовании этих операций с объектами классов они приобретают смысл, соответствующий новым типам. Это одно из наиболее мощных средств C++.

Операции перегружаются путем составления описания функции (с заголовком и телом), как это вы обычно делаете, за исключением того, что в этом случае имя функции состоит из ключевого слова operator, после которого записывается перегружаемая операция. Например, имя функции operator+ можно использовать для перегрузки операции сложения.

Чтобы использовать операцию над объектами классов, эта операция должна быть перегружена, но есть два исключения. Операция присваивания (=) может быть использована с каждым классом без явной перегрузки. По умолчанию операция присваивания сводится к побитовому копированию данных-элементов класса. Однако такое побитовое копирование опасно для классов с элементами, которые указывают на динамически выделенные области памяти; для таких классов мы будем явно перегружать операцию присваивания. Операция адресации (&) также может быть использована с объектами любых классов без перегрузки; она просто возвращает адрес объекта в памяти. Но операцию адресации можно также и перегружать.

Перегрузка больше всего подходит для математических классов. Они часто требуют перегрузки значительного набора операций, чтобы обеспечить согласованность со способами обработки этих математических классов в реальной жизни. Например, было бы странно перегружать только сложение класса комплексных чисел, потому что обычно с комплексными числами используются и другие арифметические операции.

Цель перегрузки операций состоит в том, чтобы обеспечить такие же краткие выражения для типов, определенных пользователем, какие C++ обеспечивает с помощью богатого набора операций для встроенных типов. Однако, перегрузка операций не выполняется автоматически; чтобы выполнить требуемые операции, программист должен написать функции, осуществляющие перегрузки операций. Иногда эти функции лучше сделать функциями-элементами, иногда — функциями-друзьями, а случается, что лучше не делать их ни элементами, ни друзьями.

Целью лабораторной работы является получение практических навыков при перегрузке операторов и реализации дружественных функций.

2. Общие сведения.

2.1 Дружественные функции

Дружественной функцией называется функция, которая сама не является элементом класса, но имеет права на доступ к элементам класса, описанным как *private* или *protected*. Если функция или класс объявлены как дружественные данному классу, то такие функции или функции-члены такого класса могут осуществлять непосредственный

доступ ко всем полям класса, для которого они дружественны. Дружественные функции и классы могут осуществлять прямой доступ к закрытым полям класса без использования функций-членов этого класса.

Ключевое слово **friend** - спецификатор функции, который дает функции- не члену класса доступ к скрытым членам класса. Он используется для того, чтобы выйти за строгие рамки типизации и сокрытия данных в C++.

Одна из причин их использования состоит втом, что некоторые функции нуждаются в *привилегированном* доступе более, чем к одному классу. Вторая причина - **friend**-функция передает все параметры через список параметров, и значение каждого из них подчинено преобразованию, совместимому с назначением. Такие преобразования применяются к явно переданным аргументам-классам и поэтому особенно полезны в случаях перегрузки оператора.

Объявление **friend** функции должно появляться внутри объявления класса, которому она дружественна. Имени функции предшествует ключевое слово friend, и ее объявление может находится как в **public** так и в private части класса, что не повлияет на значение. Функция-член одного класса может быть **friend**-функцией другого класса. Это происходит тогда, когда функция-член объявлена в friend классе с использованием оператора разрешения контекста для определения имени функции дружественного класса. Если все функции-члены одного класса являются **friend**-функциями другого класса, то это можно определить записью:

freind class *имя класса*

Пример 5.1

```
class t1 {
friend void a(); // friend-функция
int b(); // функция-член
};
class t2 {
friend int t1::b(); // функция-член класса t1 имеет доступ ко всем скрытым полям класса t2
};
class t3 {
friend class t1; // все функции-члены класса t1 имеют доступ ко всем полям класса t3
};
```

Рассмотрим класс **matrix** и класс **vector**. Функция умножения вектора на матрицу должна иметь доступ к **private**-членам обоих классов. Эта функция будет **friend** для обоих классов.

Пример 5.2

```
class matix;
class vect {
int *p;
int size;
friend vect mpy(const vect &,const matix &);
public:
};
class matrix {
int **base;
int row,column;
```

```
friend vect mpy(constr vect&,const matirx &);
};
vect mpy(const vect &v,const matrix &m) {
if(v.size!=m.row) { exit(1); }
vect ans(column);
//.....
return ans;
}
```

Второстепенное значение требует предварительного описания класса matrix. Оно необходимо потому, что функция mpy должна появляться в обоих классах, и использует каждый класс как тип аргумента.

Friend-функции можно рассматривать как часть общего интерфейса класса. Существует ряд ситуаций, в которых они могут быть альтернативой функциям-членам. Использование **freind**-функций спорно, потому что они нарушают инкапсуляцию, окружающую *private* члены классов. Парадигма ООП утверждает, что объекты (в С++ они - переменные класса) доступны через их *public* члены. Только функции-члены должны иметь доступ к скрытой реализации АТД. Это ясный и строгий принцип проектирования. **Friend**-функция находится на самой его границе, поскольку имеет доступ к private членам, сама не являясь функцией-членом. С ее помощью можно организовать быстрый код для доступа к подробностям реализации класса.

2.2 Перегрузка операций

Язык Си++ поддерживает средства, которые позволяют переопределять уже существующие операции. Такое переопределение называют перегрузкой (overloading) операций.

Для определения операции используется вводимая пользователем функция с ключевым словом operator, за которым следует символ операции. Далее будут приведены общие правила переопределения операций.

1. За исключением операций new и delete, операции должны быть нестатическими функциями-элементами класса или иметь минимум один аргумент типа класса. Функции-операции =, () ,[], -> должны быть нестатическими элементами класса.

4. Нельзя переопределять следующие операции: ., :: , ?:.

5. Приоритет операций при переопределении сохраняется.

6. При переопределении любой унарной операции @ выражение @x или x@ интерпретируется как x.operator@() или как operator@(x). Для любой бинарной операции @ выражение x@y интерпретируется как x.operator@(y) или как operator@(x,y).

7. Операции new и delete могут быть переопределены. Операция new должна возвращать результат типа void * и иметь первый аргумент типа size_t. Операция delete должна возвращать результат типа void и иметь первый аргумент типа void *, второй аргумент может быть типа size_t. Операции new и delete всегда являются статическими элементами класса, даже при отсутствии в их описании ключевого слова static.

2.2.1 Перегрузка унарного оператора

Одноместные операторы типа “!”, “++”, “~” “[]”.

Пример 5.3

```
class clock
{
unsigned long sec;
public:
clock(unsigned long s):sec(s) { }
```

```
void tick() { sec++; }
clock operator++() { tick(); return *this; }
};
void main() { clock t(100); ++t; }
```

Этот класс перегружает префиксный оператор приращения ++. Перегруженный оператор представляет собой функцию-член. Перегруженный operator++() также обновляет неявную переменную clock и возвращает обновленное значение.

Префиксную операцию ++ можно перегрузить, используя friend-функцию.

```
frien clock operator++(clock &s1) { s1.tick(); return s1; }
```

Так как переменная clock должна увеличиваться мы передаем ее по ссылке. Решение о выборе между представлением friend и функцией-членом обычно зависит от того, насколько необходимы и доступны операторы неявного преобразования. Явная передача аргумента, как в friend-функции, позволяет автоматическое его приведение.

Операции инкремента и декремента могут быть перегружены как для префиксной, так и для постфиксной формы записи.

Префиксная форма объявляется как операция-член класса, которая не имеет аргументов, либо как дружественная операция, которая имеет один аргумент, представляющий собой ссылку на объект данного класса.

Постфиксная форма объявляется как операция-член класса, которая имеет один аргумент типа int, либо как дружественная операция, которая имеет два аргумента: ссылку на объект данного класса и аргумент типа int. Аргумент типа int на самом деле не используется. Фактически он имеет нулевое значение.

Пример 5.4

```
class X {
public:
X & operator ++(); //Префиксная форма
X & operator ++(int); // Постфиксная форма
};
class Y {
public:
friend Y & operator ++(Y &); //Префиксная форма
friend Y & operator ++(Y &,int); // Постфиксная форма
};
```

2.2.2 Перегрузка бинарного оператора

Если бинарный оператор перегружается с помощью функции-члена, то в качестве своего первого аргумента он получает неявно переданную переменную класса, а второго - единственный из списка аргументов. При объявлении friend-функции или обычной функции в списке параметров определяют оба аргумента.

Пример 5.5

```
class clock {
friend clock operator+(const clock &c1,const clock &c2) ;
};
clock operator+(const clock &c1,const clock &c2) {
clock temp(c1.sec+c2.sec); return temp;
}
```

Оба параметра явно определяются и являются кандидатами для преобразования назначением. Используя это определение имеем:

```
int i=5; clock c(100);
c+i; // допустимо: i - преобразуется в clock
i+c; // допустимо: i - преобразуется в clock
```

В противоположность этому, перегрузим двухместный - функцией-членом.

```
class clock {
clock operator -(const clock &c) { clock temp(sec-c.sec); return temp; }
};
```

Существует первый неявный аргумент. В него попадает некоторый используемый параметр. Это может вызвать асимметричное поведение двухместных операторов.

```
int i=5; clock c(100);
c-i; // допустимо i преобразуется в clock; c.operator-(i)
i-c; // недопустимо i не относится к типу clock; i.operator-(c)
```

Определим операцию умножения: long и clock

```
clock operator *(long m, const clock &c) {
clock temp(m*c.sec); return temp;
```

Такая реализация вынуждает операцию умножения иметь фиксированный порядок выполнения, зависящий от типа. Для избежания этого обычно пишется второй перегруженный функциональный оператор.

```
clock operator *(const clock &c,long m) {
clock temp(m*c.sec); return temp;
}
```

y@x интерпретируется как y.opertor@(x), если operator @ - функция-член и operator@(y,x), если operator @ - дружественная функция.

2.2.3 Перегрузка операторов присваивания = и индексации []

```
class vect {
private:
int *p;
int size;
public:
vect(void); // создает массив из 10 элементов
vect(int mysize); // создает массив из mysize элеметов
vect(int *a,int n); // инициализируется массивом
vect(const vect &v); // инициализируется vect
~vect() { delete p; }
int & operator[](int i);
vect& operator=(const vect &v); // перегруженное присваивание
vect operator+(const vect &v); // перегруженное сложение
};
vect::vect(int n) { // преобразует обычный массив
p=new int[n]; size=n;
for(int index=0; index < size; index++) p[i]=index;
}
vect::vect(int *a,int n) { // преобразует обычный массив
p=new int[n];
size=n;
```

```
for(int index=0; index < size; index++) p[i]=a[i];
}
vect::vect(const vect &v) { // конструктор копии
p=new int[v.size]; size=v.size;
for(int index=0; index < size; index++) p[i]=v.p[i];
}
// операция индексации
int &vect::operator[](int i) {
if(i < 0 || i > =size) { cerr << “”;}
return p[i];
}
// операция присваивания
vect &vect::operator=(const vect &) {
int s=(size<v.size) ? size : v.size;
for(int i=0; i<s; i++) p[i]=v.p[i]; return *this; }
```

Перегруженный оператор индексации берет целый аргумент и проверяет, находится ли значение в пределах диапазона. Если да, он использует это значение для того, чтобы возвратить адрес индексированного элемента.

Перегруженный оператор индексации имеет возвращаемый тип и один параметр. Он должен быть нестатической функцией-членом.

В C++ версии 2.0 имеется операция присваивания по умолчанию, которая выполняет рекурсивное по членное копирование одного объекта в другой. В более ранних версиях использовался механизм побитового копирования.

Когда назначение не перегруженно, оно определяется по умолчанию с семантикой, представляющей собой присвоения значения. Это называется “семантикой поверхностного копирования” и не всегда допустимо. Перегрузка назначения желательна, а иногда и необходима, особенно, если класс содержит указатели динамически распределяемой памяти:

3. Дополнительная литература

1. Герберт Шилдт. Самоучитель C++: Пер. с англ. – 3-е изд. – СПб.: БХВ-Петербург, 2003 (глава 6)

2. Харви Дейтел, Пол Дейтел Как программировать на C++: пер. с англ. -М: ЗАО "Издательство Бином", 1999. (главы 7-8)

2. Бьярн Страуструп. Язык программирование Си++. – М.: Бином, 2005.(глава 7)

4. Порядок выполнения работы

1. Создайте новый проект в среде MS Visual C++.

2. Для полученного варианта задания, руководствуясь описанием настоящей лабораторной работы, напишите программы на языке C++. Откомпилируйте, отладьте и выполните свою программу.

3. Подготовьте отчет о выполнении лабораторной работы

Для успешной сдачи лабораторной работы необходимо:

– представить преподавателю работающую программу для указанного варианта задания;

– ответить на вопросы по тексту программы.

5. Порядок оформления отчета

Отчет о выполнении лабораторной работы должен содержать:

1) титульный лист;
2) вариант задания;
3) текст программ;
4) результаты работы программ.

6. Варианты заданий

Для допуска к экзамену достаточно выполнить любое одно задание. Для студентов, претендующих на освобождение от экзаменов необходимо выполнить оба задания.

Вариант 1.

1. Создать класс целых чисел(int). Определить унарный оператор «++», как функцию-член и «--» как дружественную функцию. Определить бинарный оператор «+», как функцию-член и «-» как дружественную функцию.

2. Создать класс вектор, содержащий ссылку на int, размерность вектора и переменную ошибки. Класс имеет конструкторы по умолчанию, конструктор с одним и двумя параметрами, конструктор копирования и деструктор. Определить оператор +, - , * , - как дружественные функции, =, +=, -=, *=,[] - как функции-члены. Определить операторы =, +, - , *, +=, -=, *= с целым числом операторы ++ и --. Определить функцию печати. Сравнить время работы созданного класса и встроенного массива типа int. Перегрузить операторы вывода и ввода в поток.

Вариант 2.

1. Создать класс координат (int). Определить унарный оператор ++ как дружественную функцию.Определить бинарный оператор +, как функцию-член. Сложить и вычесть координаты с друг другом и с числом. Присвоить координаты(=), сравнить координаты (==, !=).

2. Создать класс вектор, содержащий ссылку на long, размерность вектора и переменную ошибки. Класс имеет конструкторы по умолчанию, конструктор с одним и двумя параметрами, конструктор копирования и деструктор. Определить оператор +, - , * , - как дружественные функции, =, +=, -=, *=,[] - как функции-члены. Определить операторы =, +, - , *, +=, -=, *= с числом типа long, операторы ++ и --. Определить функцию печати. Сравнить время работы созданного класса и встроенного массива типа long. Перегрузить операторы вывода и ввода в поток.

Вариант 3.

1. Создать класс вещественных чисел(float). Определить унарный оператор ++, как функцию-член и -- как дружественную функцию. . Определить бинарный оператор -, как функцию-член и + как дружественную функцию.

2. Создать класс вектор, содержащий ссылку на float, размерность вектора и переменную ошибки. Класс имеет конструкторы по умолчанию, конструктор с одним и двумя параметрами, конструктор копирования и деструктор. Определить оператор +, - , * , - как дружественные функции, =, +=, -=, *=,[] - как функции-члены. Определить операторы =, +, - , *, +=, -=, *= с целым числом операторы ++ и --. Определить операторы =, +, - , *, +=, -=, *= с вещественным числом, операторы ++ и --. Определить функцию печати. Сравнить время работы созданного класса и встроенного массива типа float. Перегрузить операторы вывода и ввода в поток.

Вариант 4.

1. Создать класс целых чисел (long). Определить унарные операторы --, как дружественную функцию и ++ как функцию-член. Определить бинарные операторы -, как функцию-член и + как дружественную функцию.

2. Создать класс - двунаправленный список, в котором определены операции, + - добавляет в конец списка, += добавляет в этот же список в конец списка. - удаляет указанный элемент из списка (номер элемента через параметр), = - присвоение списков, сравнение списков ==, !=, >, <, >=, <=, [] получение элемента списка, ++ - устанавливает указатель на следующий элемент, -- устанавливает указатель на предыдущий элемент. () выдать подсписок от первого до второго элемента.

Вариант 5.

1. Создать класс вещественных чисел (double). Определить оператор ++, как функцию-член и -- как дружественную функцию. Определить бинарные операторы +, как функцию-член и - как дружественную функцию.

2. Создать класс - однонаправленный список, в котором определены операции, + - добавляет в конец списка, += добавляет в этот же список в конец списка. - удаляет указанный элемент из списка (номер элемента через параметр), = - присвоение списков, сравнение списков ==, !=, >, <, >=, <=, [] получение элемента списка, ++ - устанавливает указатель на следующий элемент. () выдать подсписок от первого до второго элемента.

Вариант 6.

1. Создать класс координат (double). Определить унарный оператор ++ как дружественную функцию.Определить бинарный оператор +, как функцию-член. Сложить координаты с друг другом и с числом. Присвоить координаты(=).

2. Создать класс типа очередь, которая хранит вещественные числа. Перегрузить оператор ++ как функцию член и -- как дружественную функцию. ++ добавляет элемент в очередь, -- вытаскивает элемент из очереди. Перегрузить бинарный оператор + как функцию-член и * как дружественную функцию. + складывает элемент в очереди с числом, * умножает элемент в очереди на число

Вариант 7.

1. Создать класс – координаты(float) с унарным операторами ++ и --, (постфиксная и префиксная формы). ++ как дружественная функция, -- как функция-член. Определить бинарный оператор +, как функцию-член. Сложить координаты с друг другом и с числом. Сравнить координаты (==, !=).

2. Создать класс типа стек который хранит целые числа. Перегрузить оператор ++ как функцию член и -- как дружественную функцию. ++ добавляет новый элемент в стек, -- удаляет верхушку стека. Перегрузить бинарный оператор + как функцию член и * как дружественную функцию. + складывает элемент в стеке с числом, * умножает элемент в стеке на число

Вариант 8.

1. Создать класс – координаты(float) с унарным операторами ++ и --, (постфиксная и префиксная формы). ++ как функция-член, -- как дружественная функция. Определить бинарные операторы -, как функцию-член и + как дружественную функцию. Сложить и вычесть координаты с друг другом и с числом.

2. Создать класс вектор, содержащий ссылку на double, размерность вектора и переменную ошибки. Класс имеет конструкторы по умолчанию, конструктор с одним и двумя параметрами, конструктор копирования и деструктор. Определить оператор +, - , * , - как дружественные функции, =, +=, -=, *=,[] - как функции-члены. Определить функцию печати. Сравнить время работы созданного класса и встроенного массива типа double. Перегрузить операторы вывода и ввода в поток.

Вариант 9.

1. Создать класс вещественных чисел (double). Определить оператор ++, как функцию-член и -- как дружественную функцию. Определить бинарные операторы +, как функцию-член и - как дружественную функцию.

2. Создать класс вектор, содержащий ссылку на unsigned , размерность вектора и переменную ошибки. Класс имеет конструкторы по умолчанию, конструктор с одним и двумя параметрами, конструктор копирования и деструктор. Определить оператор +, - , * , - как дружественные функции, =, +=, -=, *=,[] - как функции-члены. Определить операторы =, +, - , *, +=, -=, *= с целым без знаковым числом, операторы ++ и --. Определить функцию печати. Сравнить время работы созданного класса и встроенного массива типа unsigned. Перегрузить операторы вывода и ввода в поток.

Вариант 10.

1. Создать класс вещественных чисел (double). Определитьунарные операторы --, как функцию-член и ++ как дружественную функцию. Определить бинарные операторы -, как функцию-член и + как дружественную функцию.

2. Создать класс вектор, содержащий ссылку на unsigned long, размерность вектора и переменную ошибки. Класс имеет конструкторы по умолчанию, конструктор с одним и двумя параметрами, конструктор копирования и деструктор. Определить оператор +, - , * , - как дружественные функции, =, +=, -=, *=,[] - как функции-члены. Определить операторы =, +, - , *, +=, -=, *= с типа unsigned long, операторы ++ и --. Определить функцию печати. Сравнить время работы созданного класса и встроенного массива типа unsigned long. Перегрузить операторы вывода и ввода в поток.

Вариант 11.

1. Создать класс целых чисел (long). Определить унарный оператор ++, как функцию-член и -- как дружественную функцию. Определить бинарный оператор +, как функцию-член и - как дружественную функцию.

2. Создать класс вектор, содержащий ссылку на unsigned char, размерность вектора и переменную ошибки. Класс имеет конструкторы по умолчанию, конструктор с одним и двумя параметрами, конструктор копирования и деструктор. Определить оператор +, - , * , - как дружественные функции, =, +=, -=, *=,[] - как функции-члены. Определить операторы =, +, - , *, +=, -=, *= с числом типа unsigned char, операторы ++ и --. Определить функцию печати. Сравнить время работы созданного класса и встроенного массива типа unsigned char. Перегрузить операторы вывода и ввода в поток.

Вариант 12.

1. Создать класс целых чисел (long). Определить унарные операторы --, как функцию-член и ++ как дружественную функцию. Определить бинарные операторы -, как функцию-член и + как дружественную функцию.

2. Создать класс вектор, содержащий ссылку на long double, размерность вектора и переменную ошибки. Класс имеет конструкторы по умолчанию, конструктор с одним и двумя параметрами, конструктор копирования и деструктор. Определить оператор +, - , * , - как дружественные функции, =, +=, -=, *=,[] - как функции-члены. Определить операторы =, +, - , *, +=, -=, *= с целым числом операторы ++ и --. Определить функцию печати. Сравнить время работы созданного класса и встроенного массива типа long double. Перегрузить операторы вывода и ввода в поток.

Вариант 13.

1. Создать класс целых чисел(int). Определить унарный оператор «++», как функцию-член и «--» как дружественную функцию. Определить бинарный оператор «+», как функцию-член и «-» как дружественную функцию.

2. Определить класс - комплексные числа, перегрузив различные операторы, +, -, ++, --, +=, -=, *, ./, *=, /=, !, !=, ==, >,<, >=, <=. Ввода, вывода в поток. Сложение и вычитание должно производиться как с элементами данного класса так и со встроенными float.

Вариант 14.

1. Создать класс координат (float). Определить унарный оператор ++ как дружественную функцию.Определить бинарные операторы * и +, как функцию-член. Умножить координаты на число. Сложить координаты с числом..

2. Определить класс список однонаправленный с перегруженными операциями ++ вперед по списку, -- удалить элемент, на котором стоит указатель, += с другим списком и с новым элементом, - унарный удаляет с конца списка, =, ==, !=, >, <, <=, >=. Ввод, вывод в поток.

Вариант 15.

1. Создать класс – координаты(int) с унарным операторами ++ и --, (постфиксная и префиксная формы). ++ как дружественная функция, -- как функция-член. Определить операторы: присвоить координаты(=), сравнить координаты (==, !=).

2. Создать класс матриц и вектор, содержащие ссылку на long, число строк и столбцов (для вектора длину) и состояние ошибки. Определить конструкторы по умолчанию, конструктор с одним и с двумя параметрами, конструктор копирования, деструктор. Определить операторы =, +, -, +=, -=, *., *= с объектами этого класс, с float и с вектором. Определить оператор [] так, чтобы обращение [][] к элементам имело смысл, аналогичный встроенному. Перегрузить операторы вывода и ввода в поток. Для вектора должны быть определены кроме перечисленных ++, --, - унарный, [], /=, /. Операторы * и *= должны быть определены для умножения вектора и матрицы.

Вариант 16.

1. Создать класс – координаты(float) с унарным операторами ++ и --, (постфиксная и префиксная формы). ++ как дружественная функция, -- как функция-член. Определить бинарный оператор *, как функцию-член. Умножить координаты на число. Присвоить координаты(=).

2. Создать класс матриц и вектор, содержащие ссылку на int, число строк и столбцов (для вектора длину) и состояние ошибки. Определить конструкторы по умолчанию, конструктор с одним и с двумя параметрами, конструктор копирования, деструктор. Определить операторы =, +, -, +=, -=, *., *= с объектами этого класс, с float и с вектором. Определить оператор [] так, чтобы обращение [][] к элементам имело смысл, аналогичный

встроенному. Перегрузить операторы вывода и ввода в поток. Для вектора должны быть определены кроме перечисленных ++, --, - унарный, [], /=, /. Операторы * и *= должны быть определены для умножения вектора и матрицы.

Вариант 17.

1. Создать класс вещественных чисел (long double). Определить унарные операторы --, как функцию-член и ++ как дружественную функцию. Определить бинарные операторы -, как функцию-член и + как дружественную функцию.

2. Создать класс матриц и вектор, содержащие ссылку на float, число строк и столбцов (для вектора длину) и состояние ошибки. Определить конструкторы по умолчанию, конструктор с одним и с двумя параметрами, конструктор копирования, деструктор. Определить операторы =, +, -, +=, -=, *., *= с объектами этого класс, с float и с вектором. Определить оператор [] так, чтобы обращение [][] к элементам имело смысл, аналогичный встроенному. Перегрузить операторы вывода и ввода в поток. Для вектора должны быть определены кроме перечисленных ++, --, - унарный, [], /=, /.. Операторы * И *= должны быть определены для умножения вектора и матрицы.

Вариант 18.

1. Создать класс целых чисел (long). Определить унарный оператор ++, как функцию-член и -- как дружественную функцию. Определить бинарный оператор +, как функцию-член и - как дружественную функцию.

2. Создать класс матриц и вектор, содержащие ссылку на double, число строк и столбцов (для вектора длину) и состояние ошибки. Определить конструкторы по умолчанию, конструктор с одним и с двумя параметрами, конструктор копирования, деструктор. Определить операторы =, +, -, +=, -=, *., *= с объектами этого класс, с float и с вектором. Определить оператор [] так, чтобы обращение [][] к элементам имело смысл, аналогичный встроенному. Перегрузить операторы вывода и ввода в поток. Для вектора должны быть определены кроме перечисленных ++, --, - унарный, [], /=, /.. Операторы * И *= должны быть определены для умножения вектора и матрицы.

Вариант 19.

1. Создать класс координат (double). Определить унарный оператор ++ как дружественную функцию.Определить бинарный оператор +, как функцию-член. Сложить координаты с друг другом и с числом. Присвоить координаты(=).

2. Создать класс матриц и вектор, содержащие ссылку на int, число строк и столбцов (для вектора длину) и состояние ошибки. Определить конструкторы по умолчанию, конструктор с одним и с двумя параметрами, конструктор копирования, деструктор. Определить операторы =, +, -, +=, -=, *., *= с объектами этого класс, с float и с вектором. Определить оператор [] так, чтобы обращение [][] к элементам имело смысл, аналогичный встроенному. Перегрузить операторы вывода и ввода в поток. Для вектора должны быть определены кроме перечисленных ++, --, - унарный, [], /=, /.. Операторы * И *= должны быть определены для умножения вектора и матрицы.

Вариант 20.

1. Создать класс целых чисел (long). Определить унарные операторы --, как дружественную функцию и ++ как функцию-член. Определить бинарные операторы -, как функцию-член и + как дружественную функцию.

2. Создать класс матриц и вектор, содержащие ссылку на long, число строк и столбцов (для вектора длину) и состояние ошибки. Определить конструкторы по умолчанию, конструктор с одним и с двумя параметрами, конструктор копирования, деструктор. Определить операторы =, +, -, +=, -=, *., *= с объектами этого класс, с float и с вектором. Определить оператор [] так, чтобы обращение [][] к элементам имело смысл, аналогичный встроенному. Перегрузить операторы вывода и ввода в поток. Для вектора должны быть определены кроме перечисленных ++, --, - унарный, [], /=, /.. Операторы * И *= должны быть определены для умножения вектора и матрицы.

Вариант 21.

1. Создать класс координат (int). Определить унарный оператор ++ как дружественную функцию.Определить бинарный оператор +, как функцию-член. Сложить и вычесть координаты с друг другом и с числом. Присвоить координаты(=), сравнить координаты (==, !=).

2. Создать класс типа очередь, которая хранит вещественные числа. Перегрузить оператор ++ как функцию член и -- как дружественную функцию. ++ добавляет элемент в очередь, -- вытаскивает элемент из очереди. Перегрузить бинарный оператор + как функцию-член и * как дружественную функцию. + складывает элемент в очереди с числом, * умножает элемент в очереди на число

Вариант 22.

1. Создать класс целых чисел(int). Определить унарный оператор «++», как функцию-член и «--» как дружественную функцию. Определить бинарный оператор «+», как функцию-член и «-» как дружественную функцию.

2. Создать класс вектор, содержащий ссылку на unsigned , размерность вектора и переменную ошибки. Класс имеет конструкторы по умолчанию, конструктор с одним и двумя параметрами, конструктор копирования и деструктор. Определить оператор +, - , * , - как дружественные функции, =, +=, -=, *=,[] - как функции-члены. Определить операторы =, +, - , *, +=, -=, *= с целым без знаковым числом, операторы ++ и --. Определить функцию печати. Сравнить время работы созданного класса и встроенного массива типа unsigned. Перегрузить операторы вывода и ввода в поток.

Вариант 23.

1. Создать класс – координаты(float) с унарным операторами ++ и --, (постфиксная и префиксная формы). ++ как дружественная функция, -- как функция-член. Определить бинарный оператор *, как функцию-член. Умножить координаты на число. Присвоить координаты(=).

2. Создать класс вектор, содержащий ссылку на double, размерность вектора и переменную ошибки. Класс имеет конструкторы по умолчанию, конструктор с одним и двумя параметрами, конструктор копирования и деструктор. Определить оператор +, - , * , - как дружественные функции, =, +=, -=, *=,[] - как функции-члены. Определить функцию печати. Сравнить время работы созданного класса и встроенного массива типа double. Перегрузить операторы вывода и ввода в поток.

Вариант 24.

Создать класс вещественных чисел (double). Определить оператор ++, как функцию-член и -- как дружественную функцию. Определить бинарные операторы +, как функцию-член и - как дружественную функцию.

Создать класс вектор, содержащий ссылку на float, размерность вектора и переменную ошибки. Класс имеет конструкторы по умолчанию, конструктор с одним и двумя параметрами, конструктор копирования и деструктор. Определить оператор +, - , * , - как дружественные функции, =, +=, -=, *=,[] - как функции-члены. Определить операторы =, +, - , *, +=, -=, *= с целым числом операторы ++ и --. Определить операторы =, +, - , *, +=, -=, *= с вещественным числом, операторы ++ и --. Определить функцию печати. Сравнить время работы созданного класса и встроенного массива типа float. Перегрузить операторы вывода и ввода в поток.

Вариант 25.

Создать класс целых чисел (long). Определить унарный оператор ++, как функцию-член и -- как дружественную функцию. Определить бинарный оператор +, как функцию-член и - как дружественную функцию.

Создать класс - однонаправленный список, в котором определены операции, + - добавляет в конец списка, += добавляет в этот же список в конец списка. - удаляет указанный элемент из списка (номер элемента через параметр), = - присвоение списков, сравнение списков ==, !=, >, <, >=, <=, [] получение элемента списка, ++ - устанавливает указатель на следующий элемент. () выдать подсписок от первого до второго элемента.