Лабораторная работа № 3

**Программирование с использованием перегруженных функций и операторов.**

**Цель работы:** Получить практические навыки работы с перегруженными операторами и методами класса.

**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;

}

**Общие требования к выполнению и оформлению лабораторной работы**

1. Для подготовки к лабораторной работе используйте данные из папок «Справочная информация» и «Теория по Си», «Теория по С++»
2. В задании указаны поля и методы, которые создаются обязательно, но вы не должны этим ограничиваться. Как правило, для работы требуются и другие элементы класса, какие – решать вам.
3. Все поля класса должны находиться в закрытой области
4. Один оператор следует реализовать с помощью метода класса, другой – с помощью дружественной функции.
5. Создать несколько объектов класса с разными характеристиками, например несколько массивов различной длины и выполнить с ними заданные операции.
6. При оформлении ввода-вывода данных информация на экране должна быть отформатирована:

* на экран выводится тема задания (кратко);
* ввод данных и результат вычислений выводить с комментариями;
* выделять области ввода и вывода информации с помощью строк-разделителей.

1. При отображении объекты класса форматировать (Размещать на экране в виде таблицы), для числовой информации – в виде матрицы (строка столбец) например:

12 6 745 8

9 12 0 10

.

**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).

**Варианты заданий (выбираются по номеру компьютера)**

|  |  |
| --- | --- |
| **Номер компьютера** | **Варианты заданий** |
| 1,10,20 | Определить класс «вектор» (динамический массив целых чисел) со следующими данными-членами :   * длина массива, * указатель на начало массива.   В класс включить конструкторы:   * для создания объектов класса по размеру массива и его инициализации с помощью датчика случайных чисел. * путем копирования другого вектора   Создать метод для вывода объектов класса на экран дисплея с параметром «количество элементов в строке».  Определить операции над векторами :  ! определение максимального элемента вектора  - получение нового вектора , каждый элемент которого равен меньшему из элементов двух других векторов. |
| 2,11,21 | Определить класс «вектор» (статический массив вещественных чисел) со следующими данными-членами :   * длина массива,   В класс включить конструкторы   * для создания объектов класса по размеру массива и его инициализации с клавиатуры. * путем копирования другого вектора   Создать метод для вывода объектов класса на экран дисплея с параметрами «количество элементов в строке», «количество знаков после запятой»  Определить операции над векторами :  & «перевернуть» вектор (последний элемент становится первым, а первый - последним)  \ получение нового вектора так, что каждый элемент нового вектора определяется следующим образом: c[I]=(a[I]>b[I])?b[I]:a[I]; |
| 3,12,22 | Определить класс «динамический массив целых чисел» со следующими данными-членами :   * длина массива, * указатель на начало массива.   Предусмотреть конструкторы   * для создания объектов класса по размеру массива и его инициализации с клавиатуры. * путем копирования другого массива   Создать метод для вывода объектов класса на экран дисплея с параметром «количество элементов в строке».  Определить операции над массивами :  ~ возвести в квадрат каждый элемент массива  \* получение нового массива, каждый элемент которого равен среднему арифметическому элементов двух других массивов. |

|  |  |
| --- | --- |
| 4,13,23 | Определить класс «матрица целых чисел» со следующими данными-членами :   * Количество строк матрицы, * количество столбцов матрицы, * указатель на начало матрицы.   Предусмотреть конструкторы   * для создания объектов класса по числу строк и столбцов матрицы и её инициализации с помощью датчика случайных чисел. * путем копирования другой матрицы   Создать метод для вывода объектов класса на экран дисплея с параметром «количество элементов в строке»  Определить операции над матрицами :  ++ определение максимального элемента матрицы  % получение новой матрицы, каждый элемент которой равен среднему арифметическому элементов двух других матриц. |
| 5,14,24 | Определить класс «строка». В класс включить два конструктора:   * для определения строки с клавиатуры * путем копирования другой строки (объекта класса строки).   Определить операции над строками:  - удаление одной строки из другой (если одна строка является подстрокой другой);  -- преобразование символов строки в строчные (маленькие) символы. |
| 6,15,25 | Определить класс «матрица вещественных чисел» со следующими данными-членами :   * Количество строк матрицы, * количество столбцов матрицы,   Предусмотреть конструктор для создания объектов класса по числу строк и столбцов матрицы и её инициализации с помощью датчика случайных чисел.  Создать метод для вывода объектов класса на экран дисплея c параметрами «количество элементов в строке» , «количество знаков после запятой»..  Определить операции над матрицами :  ++ определение суммы матрицы  + получение новой матрицы, каждый элемент главной диагонали которой равны сумме соответствующих элементов двух других матриц. Остальные элементы матрицы равны нулю |

|  |  |
| --- | --- |
| 7,16,26 | Определить класс «массив целых чисел» со следующими данными-членами :   * длина массива, * указатель на начало массива.   Предусмотреть конструктор для создания объектов класса по размеру массива и его инициализации с помощью датчика случайных чисел.  Создать метод для вывода объектов класса на экран дисплея c параметром «количество элементов в строке».  Определить операции над массивами :  ++ определение суммы четных элементов массива  \* получение нового массива являющегося «пересечением» двух других массивов. |
| 8,17,27 | Определить класс-строку со следующими данными-членами :   * указатель на начало строки.   В класс включить два конструктора:   * для определения класса строки строкой символов * путем копирования другой строки (объекта класса строки).   Создать метод для вывода объектов класса на экран дисплея.  Определить операции над строками:  + конкатенация двух строк;  ++ преобразование символов строки в заглавные (большие) символы. |
| 9,18,28 | Определить класс «статическая матрица целых чисел» со следующими данными-членами :   * Количество строк матрицы, * количество столбцов матрицы,   Предусмотреть конструкторы для создания объектов класса:   * по числу строк и столбцов матрицы и её инициализации с помощью датчика случайных чисел. * путем копирования другой матрицы   Создать метод для вывода объектов класса на экран дисплея с параметром «количество элементов в строке».  Определить операции над матрицами :  ++ определение суммы главной диагонали матрицы  \* получение новой матрицы, элементы четных столбцов которой равны сумме соответствующих элементов двух других матриц, элементы нечетных столбцов равны нулю. |