# Лабораторная работа №3. Перегрузка стандартных операций

**1 Цель работы**

Цель работы – изучение основных принципов расширенных действий (перегрузки) стандартных операций языка С++.

**2 Порядок выполнения работы**

* ознакомиться с описанием лабораторной работы;
* получить задание у преподавателя по вариантам;
* разработать и отладить программу;
* составить и защитить отчет по лабораторной работе у преподавателя.

**3 Содержание отчета**

* титульный лист;
* краткое теоретическое описание;
* задание на лабораторную работу, включающее математическую формулировку задачи;
* результаты выполнения работы, включающие схему алгоритма, тексты программ, результаты вычислений;

**4 Краткая теория**

**4.1 Цель перегрузки операций**

Одной из интересных особенностей языка С++ является возможность распространения действия стандартных операций на операнды, для которых эти операции первоначально в языке не предполагались. Например, если **S1** и **S2** – символьные строки, то их соединение удобно было бы обозначить как **S1+S2**. Однако бинарная операция **+** в языке С++ предназначена для арифметических операндов и не предусматривает строковых операндов. Но если **S1** и **S2** определить как объекты некоторого класса, то для них можно ввести операцию **+**, выполняемую по правилам, которые выбрал программист. Для этих целей язык С++ позволяет распространить действие любой стандартной операции на новые типы данных, вводимые пользователем. Распространить операцию на новые типы данных позволяет механизм перегрузки стандартных операций.

При перегрузке операций нельзя вводить новые лексические обозначения операций, даже формируя их из допустимых символов.

Все операции С++ могут быть перегружены, кроме операций точка (**.**), разыменование (**\***), расширение области действия (**::**), условная (**?:**) и **sizeof**. Операции **=**, **[]**, **()** и **->** могут быть перегружены только как нестатические функции-компоненты.

Все остальные операции можно перегружать, чтобы применять их к каким-то новым типам объектов, вводимым пользователем. Кроме того, многие операции уже перегружены в С++. Например, арифметические операции применяются к разным типам данных – целым числам, действительным и т.д., именно в результате того, что они перегружены.

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. Такое побитовое копирование опасно для классов с элементами, которые указывают на динамически выделенные области памяти; для таких классов следует 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 не может изменять старшинство и ассоциативность операций. Нельзя также изменить число операндов операции. Например, унарную операцию можно перегрузить только как унарную.

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 С++ 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.

**4.2 Способы перегрузки стандартных операций**

Чтобы перегрузить стандартные операции с необычными для нее данными, необходимо специальным образом определить ее новое поведение. Это возможно, если хотя бы один из операндов является объектом некоторого класса, т.е. введенного пользователем типа. Для распространения действия операции на новые данные программист определяет специальную функцию, называемую «операция-функция». Формат ее определения:

**тип\_возращаемого\_значения operator знак\_операции**

**(спецификация\_параметров\_операции-функции)**

**{ операторы\_тела\_операции-функции }**

При необходимости может добавляться и прототип операции-функции.

Например, для распространения действия бинарной операции ’**\***’ на объекты класса **Т** может быть введена функция с заголовком:

**T operator \*(Tx, Ty)**

Количество параметров у операции-функции зависит от арности операции и от способа определения функции. Чтобы явная связь с классом была обеспечена, операция-функция должна быть либо компонентом класса, либо она должна быть определена в классе как дружественная, либо у нее должен быть хотя бы один параметр типа класс (или ссылка на класс). Начнем с последнего варианта.

Если для класса **Т** введена операция-функция с приведенным выше заголовком и определены два объекта **A** и **B** класса **Т**, то выражение **A\*B** интерпретируется как вызов функции **operator \*(A,B)**.

Кроме сокращенной формы вызова такого виды операции-функции (с помощью выражения с операндами нужных типов) возможна и полная форма вызова:

**operator знак\_операции (фактические\_параметры);**

Вторую возможность перегрузки бинарной операции представляют компонентные функции классов. Любая стандартная бинарная операция ‘**@**’ может быть перегружена с помощью нестатической операции-функции, входящей в число компонентов класса. В этом случае у нее должен быть только один параметр и заголовок может выглядеть так:

**T operator @(T x )**

(здесь **Т** - определенный пользователем класс). В этом случае выражение **А@B** с объектами **А, В** класса **Т** в качестве операндов интерпретируется как вызов функции **А.operator @(B)**, при чем в теле операции-функции выполняется обработка компонентов объекта-параметра **В** и того объекта **А**, для которого осуществлен вызов. При необходимости принадлежность компонентов объекту **А** в теле операции-функции можно сделать явным образом с помощью указателя **this**.

Если операция-функция определена как принадлежащая классу, то вызывать ее явно можно с использованием имени объекта или указателя на объект и операции выбора компонентов (‘**->**’, ‘**.**’). Другими словами в этом случае вызов операции-функции подобен вызову обычной компонентной функции класса.

Наконец, последнюю возможность перегрузки стандартных операций предоставляет оформление операции-функции в виде дружественной функции класса.

Любая унарная операция определяется для объектов некоторого класса двумя способами: либо как компонентная функция без параметров, либо как глобальная (возможно дружественная) функция с одним параметром.

**4.3 Перегрузка префиксных и постфиксных операций**

В отличие от всех других унарных операций операции **++** и **--** имеют, кроме префиксной формы еще и постфиксную. Это привело к особенностям при их перегрузке. В начальных версиях языка С++ при перегрузках операций **++** и **--** не делалось различия между постфиксной и префиксной формами.

В современной версии языка С++ принято соглашение, что перегрузка префиксных операций **++** и **--** ничем не отличается от перегрузки других унарных операций, т.е. глобальные и, возможно, дружественные функции **operator ++()** и **operator --()** с одним параметром некоторого класса определяют префиксные операции **++** и **--.** Компонентные операции-функции без параметров определяют те же префиксные операции. При расширении действия постфиксных операций **++** и **--** операции-функции должны иметь еще один дополнительный параметр типа **int**. Если для перегрузки используется компонентная операция-функция, то она должна иметь один параметр тип **int**. Если операция-функция определена как глобальная (не компонентная), то ее первый параметр должен иметь тип класса, а второй – тип **int**.

Когда в программе используется соответствующее постфиксное выражение, то операция-функция вызывается с нулевым целым параметром.

Пример возможностей применения разных операций-функций для постфиксной и префиксной операций **++** и **--** приведен в разделе 5 п. 2.

**4.4 Ввод-вывод для типов, определенных пользователем**

Чтобы использовать операции обмена с потоком **>>** и **<<** для данных производных типов, определяемых пользователями, необходимо расширить действие указанных операций, введя новые операции-функции. Каждая из операций обмена **<<** и **>>** бинарная, причем левым операндом служит объект, связанный с потоком, а правый операнд должен иметь желаемый тип. Этот бинарный характер операций обмена отражает спецификация параметров соответствующих операций-функций. Первый параметр – ссылка на объект потокового класса (тип **istream&** либо **ostream&**), второй параметр – ссылка или объект желаемого типа. Тип возвращаемого значения должен быть ссылкой на тот поток, для которого предназначена операция. Таким образом, формат операции-функции для перегрузки операций таков:

**ostream& operator <<(ostream& out, новый\_тип имя)**

**{ ...**

**out<<... //вывод значений нового\_типа**

**return out; //возврат ссылки**

**}**

Здесь **новый\_тип** – тип, определенный пользователем. Таким образом, если определить структурный тип (класс):

**struct point //точка трехмерного евклидова пространства**

**{**

**float x; //декартовы координаты точки**

**float y;**

**float z;**

**};**

то для типа **point** можно определить правила вывода (по умолчанию на экран) значений, например, с помощью такой операции-функции:

**ostream& operator <<(ostream& t, point d)**

**{**

**return t<<”\nx=”<<d.x<<” y=”<<d.y<<” z=”<<d.z;**

**}**

Напомним, что класс **ostream**, поток (объект) **cout** и «стандартные» режимы выполнения операции вывода (для базовых типов) определены в заголовочном файле **iostream**, который необходимо поместить в начале программы до текста операции-функции **operator <<()**.

Для перегрузки (расширения действия) операции ввода **>>** необходимо определить операцию-функцию вида:

**istream& operator >>(istream& in, новый\_тип& имя)**

**{ ...**

**in>>... //ввод значений нового\_типа**

**return in; //возврат ссылки**

**}**

Здесь **новый\_тип** – тип, определенный пользователем, т.е. некоторый класс или его частный случай – структурный тип. Основное отличие от перегрузки операции вывода – необходимость в качестве второго параметра использовать ссылку.

**5 Примеры программ**

1. Класс «комплексное число». Перегружены операции: бинарный + (сложение двух комплексных чисел), –= (из действительной и мнимой частей комплексного числа вычесть какое-то значение), > (сравнение двух комплексных чисел по их модулям). Все операции-функции описаны как дружественные функции класса.

**#include <iostream>**

**#include <math.h>**

**using namespace std;**

**class complex**

**{**

**double real;**

**double imag;**

**public:**

**complex(double = 1.0, double = 0.0);**

**void display();**

**//прототипы дружественных операций-функций**

**friend complex& operator +(complex, complex);**

**friend complex& operator -=(complex&, double);**

**friend bool operator >(complex, complex);**

**};**

**//определение конструктора**

**complex::complex(double re, double im)**

**{**

**real = re;**

**imag = im;**

**}**

**//определение компонентной функции печати**

**void complex::display()**

**{**

**cout<<"real = "<<real;**

**cout<<", imag = "<<imag<<'\n';**

**}**

**//перегрузка операции +. Результат - комплексное число**

**complex& operator +(complex ob1, complex ob2)**

**{**

**//создаем новое комплексное число с нужными параметрами**

**complex ob(ob1.real + ob2.real, ob1.imag + ob2.imag);**

**return ob;**

**}**

**//перегрузка операции -=. Результат - комплексное число**

**complex& operator -=(complex& ob, double d)**

**{**

**//уменьшаем значения действительной и мнимой частей на d**

**ob.real -= d;**

**ob.imag -= d;**

**return ob;**

**}**

**//перегрузка операции >. Результат - true или false**

**bool operator >(complex ob1, complex ob2)**

**{**

**//считаем модули обоих комплексных чисел**

**double mod1 = sqrt(pow(ob1.real,2) + pow(ob1.imag,2));**

**double mod2 = sqrt(pow(ob2.real,2) + pow(ob2.imag,2));**

**//и сравниваем их**

**return (mod1 > mod2);**

**}**

**void main()**

**{**

**//создание объектов класса**

**complex x1,x2(3.0, 5.0);**

**//вызов перегруженного оператора +**

**complex x3 = x1 + x2;**

**x3.display();**

**//вызов перегруженного оператора -=**

**x3 -= 5;**

**x3.display();**

**//вызов перегруженного оператора >**

**if (x3 > x2)**

**cout<<"x3 is greater!\n";**

**else**

**cout<<"x2 is greater or equal!\n";**

**}**

Результат работы программы:

**real = 4, imag = 5**

**real = -1, imag = 0**

**x2 is greater or equal!**

2. Программа иллюстрации возможности применения разных операций-функций для постфиксной и префиксной операций **++** и **--**. Для демонстрации полной независимости смысла перегруженной операции от ее традиционного (стандартного) значения в операциях-функциях для префиксных операций **++** соответствуют увеличению в 10 раз, а **--** уменьшению в 10 раз. Для постфиксных операций **++** определили как увеличение в 2 раза, а **--** как уменьшение в 2 раза. Попытка использовать в постфиксных операциях-функциях значение дополнительного параметра **int k** подтверждает его равенство 0.

**#include <iostream>**

**using namespace std;**

**//класс «пара чисел»**

**class para**

**{**

**int N; //целое число**

**double x; //вещественное число**

**//дружественная функция для префиксной операции**

**friend para& operator ++(para&);**

**//дружественная функция для постфиксной операции**

**friend para& operator ++(para&,int);**

**public:**

**para (int n,double xn) //конструктор**

**{**

**N=n;**

**x=xn;**

**}**

**void display()**

**{**

**cout<<"N = "<<N<<" x = "<<x<<endl;**

**}**

**//компонентная функция (префиксная --)**

**para& operator --()**

**{**

**N/=10;**

**x/=10;**

**return \*this;**

**}**

**//компонентная функция (постфиксная --)**

**para& operator --(int k)**

**{**

**N/=2;**

**x/=2.0;**

**return \*this;**

**}**

**};**

**para& operator ++(para& P) //префиксная ++**

**{**

**P.N\*=10;**

**P.x\*=10;**

**return P;**

**}**

**para& operator ++(para& P,int k) //постфиксная ++**

**{**

**P.N=P.N\*2+k;**

**P.x=P.x\*2+k;**

**return P;**

**}**

**void main()**

**{**

**para Z(10,20.0);**

**Z.display();**

**++Z;**

**Z.display();**

**--Z;**

**Z.display();**

**Z++;**

**Z.display();**

**Z--;**

**Z.display();**

**}**

Результаты выполнения программы:

**N = 10 x = 20**

**N = 100 x = 200**

**N = 10 x = 20**

**N = 20 x = 40**

**N = 10 x = 20**

3. Программа, иллюстрирующая пример перегрузки операций ввода-вывода. В ней определен класс **employee**, предназначенный для хранения сведений о некоторых служащих.

**#include <iostream>**

**using namespace std;**

**//определение класса «служащий»**

**class employee**

**{**

**char name[50]; //фамилия**

**long salary; //оклад**

**int age; //возраст**

**//прототипы дружественных операций-функций**

**friend istream& operator >>(istream&, employee&);**

**friend ostream& operator <<(ostream&,employee);**

**};**

**//перегрузка операции >> на операнд типа employee**

**istream& operator >>(istream& input,employee& em)**

**{**

**cout<<"Input data:\nName: ";**

**input.getline(em.name, 50);;**

**cout<<"Salary: ";**

**input>>em.salary;**

**cout<<"Age: ";**

**input>>em.age;**

**input.get();**

**return input;**

**}**

**//перегрузка операции << на операнд типа employee**

**ostream& operator <<(ostream& out,employee e)**

**{**

**out<<"\nData:";**

**out<<"\nName: "<<e.name;**

**out<<", salary: "<<e.salary;**

**out<<", age: "<<e.age<<endl;**

**return out;**

**}**

**void main()**

**{**

**employee E; //определен объект класса employee**

**cin>>E;**

**cout<<E;**

**}**

Возможный результат выполнения программы:

**Input data:**

**Name: Соколов <Enter>**

**Salary: 15000 <Enter>**

**Age: 25 <Enter>**

**Data:**

**Name: Соколов, salary: 15000, age: 25**

**6 Контрольные вопросы**

* 1. С какой целью применяется перегрузка стандартных операций?
  2. Все ли операции языка С++ можно перегружать? Перечислите операции, не подлежащие перегрузке.
  3. Что такое операция-функция? Назовите известные вам способы ее определения.
  4. Можно ли с помощью перегрузки ввести новые лексические обозначения операций?
  5. В чем отличие перегрузки префиксных операций от перегрузки постфиксных операций?
  6. С помощью каких объектов осуществляется перегрузка операций ввода-вывода?

**7 Варианты заданий для самостоятельного решения**

1. Описать класс «матрица чисел» с компонентными данными: размерности матрицы, указатель на элементы. Перегрузить операции: **<<** (вывод матрицы на экран), **+** (сложение матриц), унарный **­–** (изменить знак каждого элемента), **/=** (каждый элемент разделить на число).

2. Описать класс «матрица чисел» с компонентными данными: размерности матрицы, указатель на элементы. Перегрузить операции: **>>**, **<<**, **–** (разность двух матриц), постфиксный **++** (инкремент каждого элемента).

3. Создать класс «вектор чисел». Компонентные данные: размерность вектора, указатель на элементы. Перегрузить операции: **>>**, **<<**, **+** (сложение векторов), префиксный **++** (инкремент каждого элемента).

4. Создать класс «вектор чисел». Компонентные данные: размерность вектора, указатель на элементы. Перегрузить операции: **>>**, **<<**, **==** (проверка на равенство всех элементов векторов), **\*=** (каждый элемент вектора умножить на число).

5. Определить класс «дата» с компонентными данными: число, месяц, год. Перегрузить операции: **>>**, **<<** (вывод в формате «чч.мм.гг»), **-** (количество дней между двумя датами), **==** (равенство двух дат).

6. Определить класс «дата» с компонентными данными: число, месяц, год. Перегрузить операции: **>>**, **<<** (вывод в формате «чч/мм/гггг»), **>=** (проверка на более позднюю дату или на их равенство), **+=** (прибавить *n* дней).

7. Описать класс «время». Компонентные данные: часы, минуты, секунды. Перегрузить операции: **>>**, **<<** (вывод в формате «чч.мм.сс»), **+=** (прибавить *n* секунд), **>** (проверка на более позднее время).

8. Описать класс «время». Компонентные данные: часы, минуты, секунды. Перегрузить операции: **>>**, **<<** (вывод в формате «чч/мм/сс»), **!=** (проверка на неравенство), **–** (промежуток между двумя событиями в часах, минутах, секундах).

9. Определить класс «комплексное число». Перегрузить операции: **>>**, **<<** (вывод в виде «*a+bi*»), **==**, префиксный **––**(декремент действительной части числа).

10. Определить класс «комплексное число». Перегрузить операции: **>>**, **<<** (вывод в виде «*ai+b*»), **!=**, унарный **–**.

11. Создать класс «точка в трехмерном пространстве». Перегрузить операции: **>>**, **-** (расстояние между точками), **==**, постфиксный **++** (инкремент каждой координаты).

12. Создать класс «точка на плоскости». Перегрузить операции: **>>**, **<<** (вывод в формате «*(x, y)*»), унарный **–** (изменить знак каждой координаты), **!=**.

13. Описать класс «окружность». Компонентные данные: координаты центра, радиус. Перегрузить операции: **>>**, **-** (расстояние между центрами окружностей), **!=** (проверка на неравенство по координатам центра и радиусу), постфиксный **--** (декремент радиуса).

14. Описать класс «окружность». Компонентные данные: координаты центра, радиус. Перегрузить операции: **>>**, **<<**, **==** (окружности равны, если равны их радиусы), **+=** (сместить координаты центра на заданное число).

15. Создать класс «дробь» с компонентными данными: целая часть, числитель, знаменатель. Перегрузить операции: **<<** (вывод в виде «*a/b*»), **+**, префиксный **--**, **>**.

16. Создать класс «дробь» с компонентными данными: целая часть, числитель, знаменатель. Перегрузить операции: **>>**, **<<**, **\*=** (умножение дроби на число), **!=**.

17. Описать класс «символ на экране». Компонентные данные: символ, координаты символа на экране. Перегрузить операции: **<<** (вывод символа в заданной позиции), префиксный **++** (инкремент горизонтальной координаты), **-=** (уменьшить вертикальную координату на заданное число), **==** (проверка на равенство символов и их координат).

18. Описать класс «отрезок». Компонентные данные: координаты начала и конца отрезка. Перегрузить операции: **>>**, **<<**, **==** (проверка на равенство длин отрезков), унарный **–** (изменить знак каждой координаты начала и конца отрезка).

19. Определить класс «строка символов» с компонентными данными: длина строки, указатель на строку. Перегрузить операции: **<<** (вывод в виде «строка – длина строки»), **!=** (проверка на неравенство двух строк), **+** (сцепление строк), **+=** (к исходной строке присоединить строку-параметр).

20. Создать класс «множество символов». Перегрузить операции: **<<** (вывод элементов множества в алфавитном порядке), **==** (проверка на равенство двух множеств), **!=** (проверка множеств на неравенство), **|** (объединение множеств).

21. Создать класс «студент». Компонентные данные: фамилия, курс. Перегрузить операции: **>>**, **<<**, **==**, постфиксный **++** (перевод студента на следующий курс).

22. Описать класс «квадрат» с компонентыми данными: координаты центра, длина стороны. Перегрузить операции: **<<**, префиксный **--** (декремент длины стороны), **!=** (проверка на неравенство длин сторон квадратов), **+=** (увеличение на *n* координат центра).

23. Определить класс «матрица целых чисел». Перегрузить операции: **<<**, **%** (каждый элемент матрицы-результата равен остатку от деления друг на друга элементов матриц-операндов), **==** (проверка на равенство элементов матриц), **+=** (каждый элемент матрицы увеличить на *n*).

24. Создать класс «линейное уравнение» с компонентными данными – коэффициентами. Перегрузить операции: **<<** (вывод в формате «*ax+b=0*»), **+**, постфиксный **--** (декремент числа *b*), **!=**.

25. Создать класс «квадратное уравнение» с компонентными данными – коэффициентами. Перегрузить операции: **<<** (вывод в формате «*ax2+bx+c=0*»), **==**, **-**, **\*=** (умножить все коэффициенты на заданное число).

26. Описать класс «вектор». Перегрузить операции: **<<**, унарный **–**, **!=**, **/=** (каждый элемент разделить на заданное число).

27. Создать класс «стек чисел». Перегрузить операции: **+=** (поместить в стек), префиксный **--** (извлечь из стека), **<<** (печать содержимого стека), **~** (вывод на экран размера стека).

28. Определить класс «очередь символов». Перегрузить операции: **+=** (поместить в очередь), префиксный **--** (извлечь из очереди), **<<** (печать содержимого очереди), **~** (вывод на экран размера очереди).