**Лабораторная работа № 5**

**Тема**: Пеpевантаження операций

**Цель**: Научиться перегружать операции +, -, \*, /, (), [] и операторы, определенные пользователем.

**Задание:**

1. Перегрузить операторы + для комплексных чисел:

А) Сделать функцию перегрузки оператора "+" дружественной класса.

Б) Сделать функцию перегрузки оператора "+" - член класса.

1. Cтвориты 3 класса, описывающие единицы измерения температуры по Цельсию, Фаренгейту и Кельвину. Реализовать преобразователь единиц измерения температуры, используя перегрузки операторов для приведения типов.
2. Для класса String перегрузить операции + + = == [] () таким образом, чтобы обеспечивалось выполнение следующих операций:

             String + String // конкатенация

             String + "Text" // добавление

              "Text" + String // дружественная функция

              String += "Text" // присваивание

              String + = 'c'

              String == "Text"

              String [Pos]

              (Char \*) String

1. Перегрузить операторы + для комплексных чисел:

А) Сделать функцию перегрузки оператора "+" дружественной класса.

Б) Сделать функцию перегрузки оператора "+" - член класса.

// Листинг Complex.h

#ifndef COMPLEX\_H

#define COMPLEX\_H

class Complex

{

public:

Complex(); // конструктор по умолчанию

Complex(double a, double b);// перегруженный конструктор

virtual ~Complex(); // деструктор

Complex operator+(Complex second); // перегруженный оператор

friend Complex operator+(Complex first,

Complex second); // перегруженная дружественная функция

void show(); // отображение комплексного числа

private:

double \_real\_part; // реальная часть комплексного числа

double \_imaginary\_part; // мнимая часть комплексного числа

};

#endif // COMPLEX\_H

// Листинг Complex.cpp

#include "Complex.h"

#include <iostream> // библиотека ввода-вывода

// определение конструктора по умолчанию

Complex::Complex ()

{

\_real\_part = 0;

\_imaginary\_part = 0;

}

// определение перегруженного конструктора

Complex::Complex (double a =0, double b=0)

{

\_real\_part = a;

\_imaginary\_part = b;

}

Complex Complex::operator+(Complex second)

{

\_real\_part = this->\_real\_part + second.\_real\_part;

\_imaginary\_part =this->\_imaginary\_part + second.\_imaginary\_part;

}

Complex::~Complex(){} // Деструктор

// дружественная функция перегрузки оператора "+"

Complex operator+(Complex first, Complex second)

{

Complex third(0,0);

third.\_imaginary\_part = first.\_imaginary\_part +second.\_imaginary\_part;

third.\_real\_part = first.\_real\_part +second.\_real\_part;

return third;

}

// Определение отображения комплексного числа

void Complex::show()

{

std::cout << \_real\_part << " + " << \_imaginary\_part << "i\n";

}

// Листинг main.cpp

#include <iostream> // библиотека ввода-вывода

#include <Complex.h> // Заголовок класса

using namespace std;

int main()

{

Complex One(1, 2); // Объявление обьекта класса с параметрами

Complex Two (1, 2);// Объявление обьекта класса с параметрами

Complex Three; // Объявление обьекта класса

Three = One + Two; // перегрузка оператора "+"

Three.show(); // отображение комплексного суммированного числа

Complex a(-1,3), b(2, -2),c;

c = a + b;

c.show();

Complex a(-1,3);

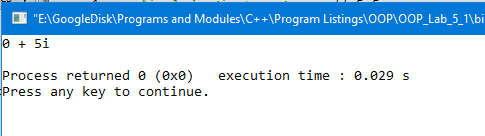
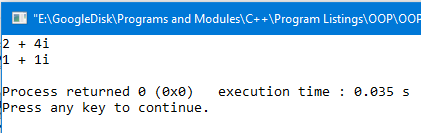
Complex b(1,2);

b+a;

b.show();

return 0;

}



1. Создайте 3 класса, описывающие единицы измерения температуры по Цельсию, Фаренгейту и Кельвину. Реализовать преобразователь единиц измерения температуры, используя перегрузки операторов для приведения типов.

// Листинг Celsius.h

#ifndef CELSIUS\_H

#define CELSIUS\_H

class Fahrenheit;

class Kelvin;

class Celsius

{

friend Fahrenheit; // друг

friend Kelvin; // друг

public:

Celsius();

Celsius(double a); // конструктор

virtual ~Celsius(); // деструктор

void show(); // отображение хранящегося значения

void operator = (Fahrenheit &object); // перегруженный член класса

void operator = (Kelvin &object); // перегруженный член класса

private:

double celsius\_value; // значение температуры в Цельсиях

};

class Fahrenheit

{

friend Celsius; // друг

friend Kelvin; // друг

public:

Fahrenheit();

Fahrenheit( double b); //конструктор

virtual ~Fahrenheit(); //деструктор

void operator = (Celsius &object); // перегруженный член класса

void operator = (Kelvin &object); // перегруженный член класса

void show(); // отображение хранящегося значения

private:

double fahrenheit\_value; // значение температуры в Фаренгейтах

};

class Kelvin

{

friend Celsius; // друг

friend Fahrenheit;// друг

public:

Kelvin();

Kelvin( double c); //конструктор

virtual ~Kelvin(); //деструктор

void operator=( Celsius &object);// перегруженный член класса

void operator = (Fahrenheit &object); // перегруженный член класса

void show(); // отображение хранящегося значения

private:

double kelvin\_value; // значение температуры в Фаренгейтах

};

#endif // CELSIUS\_H

// Листинг Celsius.cpp

#include "Celsius.h"

#include <iostream>

// конструктор с параметрами

Celsius :: Celsius(double a){ celsius\_value = a;}

Celsius :: Celsius(){ celsius\_value = 0;} // конструктор по умолчанию

Celsius ::~Celsius(){} // деструктор

// конструктор с параметрами

Fahrenheit :: Fahrenheit(double b) { fahrenheit\_value = b;}

Fahrenheit :: Fahrenheit(){fahrenheit\_value = 0;}// конструктор по умолчанию

Fahrenheit ::~Fahrenheit(){} // деструктор

// конструктор с параметрами

Kelvin :: Kelvin(double c) { kelvin\_value = c;}

Kelvin :: Kelvin(){kelvin\_value = 0;} // конструктор по умолчанию

Kelvin :: ~Kelvin(){} // деструктор

//отображение хранящихся значений

void Celsius :: show() { std::cout << celsius\_value << " Цельсия\n";}

void Fahrenheit:: show() { std::cout << fahrenheit\_value << " Фаренгейт\n";}

void Kelvin :: show() { std::cout << kelvin\_value << " Кельвин\n";}

// перевод в цельсии

void Celsius::operator =(Fahrenheit &object)

{

celsius\_value =(object.fahrenheit\_value -32)\*5/9; // из фаренгейта в цельсия

}

void Celsius::operator =(Kelvin &object)

{

celsius\_value = object.kelvin\_value - 273.15; // из кельвина в цельсия

}

// перевод в фаренгейты

void Fahrenheit::operator =(Celsius& object)

{

fahrenheit\_value=(object.celsius\_value \* 9/5) +32; // из цельсия в фаренгейт

}

void Fahrenheit::operator =(Kelvin &object)

{

fahrenheit\_value = ((object.kelvin\_value-273.15) \* 9/5) +32 ; // из кельвина в фаренгейт

}

// перевод в кельвины

void Kelvin::operator =(Celsius& object)

{

kelvin\_value = object.celsius\_value + 273.15; // из цельсия в кельвины

}

void Kelvin::operator =(Fahrenheit &object)

{

kelvin\_value = ((object.fahrenheit\_value -32)\*5/9) + 273.15; // из фаренгейта в кельвины

}

// Листинг main.cpp

#include <iostream> // библиотека ввода-вывода

#include <Celsius.h> // заголовок класса

using namespace std;

int main()

{

setlocale(LC\_ALL, "rus"); // локализация

Celsius one(30); // обьект класса

Fahrenheit two(100); // обьект класса

Kelvin three(315); // обьект класса

one.show(); // отображение начального значения для цельсия

two.show(); // отображение начального значения для Фаренгейта

three.show(); // отображение начального значения для Кельвина

cout << endl;

one = two; // фаренгейт в цельсия

one.show(); // отображения результата

one = three; // кельвины в цельсия

one.show(); // отображения результата

cout << endl;

two = one; // цельсия в фаренгейт

two.show(); // отображения результата

two = three; // кельвины в фаренгейт

two.show(); // отображения результата

cout << endl

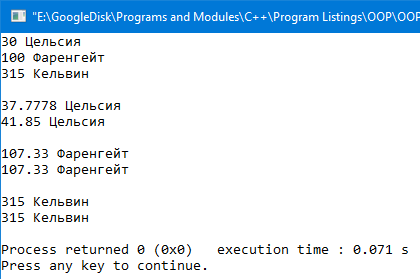
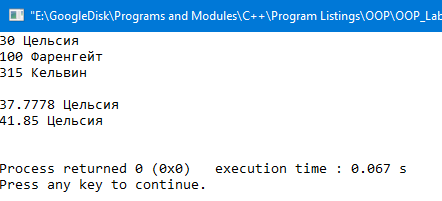
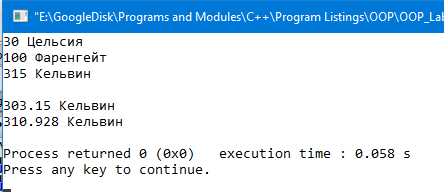
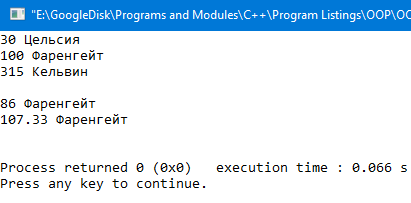
three = one; // цельсия в кельвины

three.show(); // отображения результата

three = two; // фаренгейт в кельвины

three.show() // отображения результата

return 0;

}

1. Для класса String перегрузить операции + + = == [] () таким образом, чтобы обеспечивалось выполнение следующих операций:

            1 String + String

            2 String + "Text"

             3 "Text" + String

             4 String + = "Text"

             5 String + = 'c'

             6 String == "Text"

             7 String [Pos]

             8 (Char \*) String

//MyString.h

#ifndef MYSTRING\_H

#define MYSTRING\_H

*class* String

{

*public*:

String(char\* stri, char\* stri1, char\* stri2);

String( char\* str1);

~String ();

void operator+(String &object); *//* *перегрузка* *оператора* *"+"* *объекта*

void operator+(*const* char\* str); *//* *перегрузка* *оператора* *"+"* *Текст*

void operator+=(*const* char\* str);*//* *перегрузка* *оператора* *"+="* *Текст*

void operator==(*const* char\* str);*//* *перегрузка* *оператора* *"=="* *Текст*

char\* operator[](int \_position);*//* *перегрузка* *оператора* *"[]"* *позиция*

void *operator*+=(char symbol); *//* *перегрузка* *оператора* *+=* *Символ*

*operator* char\* (); *//* *перегрузка* *оператора* *приведения* *типа*

*//* *дружественная* *функция* *перегруженного* *оператора*

*friend* char\* operator +(char\* str1, String &str2);

*public*:

char\* getpos(int pos); *//* *геттер*

char\* getstring(); *//* *перегруженная* *функция* *геттер*

void setString(*const* char\* str); *//* *сеттер*

void show(); *//* *отображение* *значения*

*private*:

char \*ptr;

char \*ptr1;

char \*ptr2;

int \_size;

};

#endif *//* *MYSTRING\_H*

//MyString.cpp

#include "mystring.h"

#include <cstring>

#include <iostream>

*//* *конструктор* *класса*

String::String(char\* str)

{

ptr = *new* char[strlen(str)+50];

std::strcpy(ptr, str);

ptr1 = *nullptr*;

ptr2 = *nullptr*;

}

*//* *перегруженный* *конструктор*

String::String(char\* stri, char\* stri1, char\* stri2)

{

\_size = strlen(stri);

ptr = *new* char[\_size+50];

\_size = strlen(stri1);

ptr1 = *new* char[\_size+50];

\_size = strlen(stri2);

ptr2 = *new* char[\_size+50];

std::strcpy(ptr, stri);

std::strcpy(ptr1, stri1);

std::strcpy(ptr2, stri2);

}

*//* *деструктор*

String::~String()

{

*delete*[] ptr;

*delete*[]ptr1;

*delete* []ptr2;

}

*//* *отображения* *значения*

void String::show()

{

std:: cout << getstring();

}

*//* *геттер*

char\* String::getpos(int pos)

{

*switch* (pos)

{

*case* 0: *return* ptr;

*break*;

*case* 1: *return* ptr1;

*break*;

*case* 2: *return* ptr2;

*break*;

*default*: *return* ptr;

*break*;

} }

*//перегруженная* *функция* *геттер*

char\* String::getstring() { *return* ptr; }

*//* *сеттер*

void String:: setString(*const* char\* str) {std::strcpy(ptr, str);}

*//* *перегрузка* *оператора* *"+"* *объекта*

void String::operator+( String &object) { std::strcat(ptr,object.ptr); }

*//* *перегрузка* *оператора* *"+"* *Текст*

void String::operator+(*const* char\* str){ std::strcat(ptr, str);}

*//* *дружественная* *функция* *перегруженного* *оператора*

char\* operator+(char\* str1, String &str2)

{

int len = strlen(str1);

char \*temp = *new* char[len+1];

std::strcpy(temp, str1);

std::strcat(temp, str2.ptr);

*return* temp;

}

*//* *перегрузка* *оператора* *"+="* *Текст*

void String:: operator+=(*const* char\* str) {std::strcat(ptr, str); }

*//перегрузка* *оператора* *"=="* *Текст*

void String:: operator==(*const* char\* str)

{

*if*( strcmp(ptr,str))

std:: cout<<"\n string not Equal";

*else*

std::cout<<"\n string Equal";

}

char\* String:: operator[](int pos) { *return* getpos(pos); }

*//* *перегрузка* *оператора* *+=* *Символ*

void String::*operator*+=(char symbol)

{

char\* p = *new* char[2];

p[0]= symbol;

p[1]='\0';

std::strcat(ptr, p);

}

*//* *перегрузка* *оператора* *приведение* *типа*

String::operator char\* (){ *return* ptr;}

*/\**

*char* *String::* *operator[](int* *pos)* *{* *ptr[pos];* *}*

*\*/*

//main.cpp

#include <iostream>

#include <mystring.h>

#include <new>

*using* *namespace* std;

int main()

{

char \*str = "I am first object"; *//* *переменая* *для* *инициализации*

char \*str1 = "I am second object"; *//* *переменая* *для* *инициализации*

*//char* *\*str2* *=* *"Third";//* *переменая* *для* *инициализации*

char\* str4;

cout << "\nInitialization object\_1 and show string : "<< endl;

String object\_1(str); *//* *экземпляр* *класса* *стринг*

object\_1.show(); *//* *отображение* *значения* *хранимое* *в* *объекте*

cout << "\n\nInitialization object\_2 and show string : "<< endl;

String object\_2(str1); *//* *экземпляр* *класса* *стринг*

object\_2.show(); *//* *отображение* *значения* *хранимое* *в* *объекте*

cout << "\n\n#1 The addition of two objects and show result :"<< endl;

object\_1 + object\_2; *//* *перегрузка* *оператора* *"+"* *Текст*

object\_1.show(); *//* *отображение* *значения* *хранимое* *в* *объекте*

object\_1.setString("Change ");*//* *интерфейс* *пользователя*

cout << "\n\n#2 Add text to the object\_1 + Text :"<< endl;

object\_1 + " word"; *//* *перегрузка* *оператора* *"+"* *Текст*

object\_1.show(); *//* *отображение* *значения* *хранимое* *в* *объекте*

*//* *дружественная* *функция* *перегруженного* *оператора*

cout << "\n\n#3 Friend function .object\_2 : "<< endl;

object\_2.setString("Sergey");

str4 = "My name is " + object\_2;

cout<< str4;

*//* *перегрузка* *оператора* *"+="* *Текст*

cout << "\n\n#4 overloaded operator += Text : "<< endl;

object\_2.setString("Hello,");

object\_2+="World";

object\_2.show();

*//* *перегрузка* *оператора* *+=* *Символ*

cout << "\n\n#5 overloaded operator += \"a\" : "<< endl;

object\_1.setString("Balalayk");

object\_1+='a';

object\_1.show();

*//* *перегрузка* *оператора* *"=="* *Текст*

cout << "\n\n#6 overloaded operator == : "<< endl;

object\_1.setString(" Application ");

object\_1.show();

object\_1 == " Application ";

object\_1 == " Other";

*//* *перегрузка* *оператора* *"[]"* *позиция*

cout << "\n\n#7 overloaded operator [] :"<< endl;

String object\_3("One", "Two", "Three");

object\_3.show();

cout << object\_3[1];

*//* *перегрузка* *оператора* *приведения* *типа*

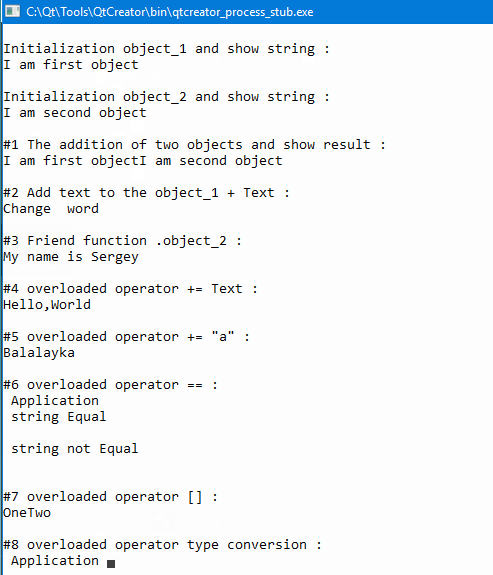
cout << "\n\n#8 overloaded operator type conversion : "<< endl;

str4 = object\_1;

cout << str4;

*return* 0;

}



**Контрольные вопросы к Л.Р. 5:**

1. Какую возможность открывает перегрузки операторов?

*Перегрузка операторов позволяет определять действия для объектов классов в выражениях, использующих обычные операторы, дает возможность добавлять к встроенным типам данных новые типы*

1. Какие операторы не могут перегружаться?

*// неперегружаемые операторы*

*:: .\* . ?:*

1. Какими двумя способами можно сказать перегрузки операторов?

*При помощи перегруженной дружественной функции*

*При помощи перегруженной функции-члена класса*

1. Что происходит, если функция, которая перегружает оператор является

дружественной функцией?

*функция имеет такой же доступ к членам класса как и сам объект класса.*

1. Сколько параметров будет перегружена функция, если она является членом класса, объектами которого оперирует?

*Один параметр*

1. Как может быть определена унарная операция?

*Как префиксная и постфикная*

1. Что служит признаком того, что функция вызывается для операции в Постфиксная варианте?

*В параметрах имеется ключевое слово int*

1. Как можно определить свои собственные правила преобразования типов?

*Нужно перегрузить операцию преобразования типов*

1. Какие существуют правила перегрузки операторов?

*• Необходимо обеспечить обдуманные функции перегрузки операторов, так как С ++ не понимает семантики перегруженного оператора.*

*• С ++ не может выводить сложные операторы из простых.*

*• Нельзя изменить синтаксис перегруженных операторов. Бинарные операторы должны оставаться бинарными, унарные - унарными.*

*• Нельзя изобретать новые операторы.*

*• Нельзя перегружать символы препроцессора # и ##.*

**Лабораторная работа № 5**

**Тема**: Пеpевантаження операций

**Цель**: Научиться перегружать операции +, -, \*, /, (), [] и операторы, определенные пользователем.

**теоретические сведения**

С ++ позволяет переопределять действие большинства операций, чтобы при использовании с объектами конкретного класса они выполняли заданные функции. Как i в случае переопределенное функций С ++ в целом, компилятор определяет разницу в функциях по контексту вызова, по числу i типу аргументов операндов.

Переопределения (пеpевантаження) можно выполнить для всех операций, за исключением таких операций:. ::? :

Также нельзя переопределять символы препроцессора # i ##.

Ключевое слово operator, за которым слiдуе символ операции, называется именем функции-операции. При определении новой (переопределены) действия операции оно используется как обычное имя функции.

Операция-функция, вызываемая с аргументами, ведет себя как операция, которая выполняет определенные действия с операндами. Операция-функция не может изменять число аргументов или правила приоритета и асоцiативностi операций по сравнению с ее нормальным использованием. Рассмотрим класс complex:

   class complex {

      double real, imag; // По умолчанию private

   public:

                                ...

      complex () {real = imag = 0 } // Встроенный конструктор

      complex (double r, double i = 0) {// еще один конструктор

       real = r; imag = i;

      }

      ...

   };

Примечание: Данный класс был создан только для задач iлюстpацiи. Он вiдpiзняеться от класса complex с библиотеки пiдтpимкы.

Мы можем легко разработать функцию для добавления комплексных чисел, например:

     complex Addcomplex (complex c1, complex c2)

однако будет более естественным иметь возможность записать:

     complex c1 (0, 1), c2 (1, 0), c3;

     c3 = c1 + c2;

вместо:

     c3 = Addcomplex (c1, c2)

Операцию + можно легко переопределить, если включить в класс complex следующее объявление:

     friend complex operator + (complex c1, complex c2)

i определить его (возможно, inline) следующим образом:

     complex operator + (complex c1, complex c2) {

return complex (c1.real + c2.real, c1.imag + c2.imag)

     }

Операции-функции можно вызывать непосредственно, хотя обычно они вызываются косвенно, при использовании переопределенное (пеpевантажених) операций:

        c3 = c1.operator + (c2) // То же самое, что c3 = c1 + c2

В отличие от от new i delete, имеющих свои собственные правила, операция-функция должна быть либо нестатические функцией-членом, или иметь как минимум один аргумент типа класса. Операции-функции = (), [] i -> должны быть нестатические функциями-членами. За исключением операции-функции пpивласнення = () все операции-функции для класса X наследуются классом, производных от X, согласно стандартным правилам разрешения для переопределенное функций. Если Х является базовым классом для Y, переопределенную операцию для класса Х можно дальше переопределить для класса Y.

**Переопределения new i delete**

Операции new i delete могут переопределяться таким образом, чтобы давать альтернативные варианты управления свободной памятью (динамическая областью). Если определяется пользователем операция new, она должна возвращать void \* i иметь в качестве первого аргумента size\_t. Если определяется пользователем операция delete, она должна иметь тип void i первый аргумент void \*. Второй аргумент типа size\_t не является обязательным.

     Тип size\_t определяется в файле stdlib.h.

     К примеру:

     #include <stdlib. h>

     class X {

     ...

     public:

        void \* operator new (size\_t size) {

                return newalloc (size)

        }

        void operator delete (void \* p) {

                newfree (p)

        }

        X () {/ \* инициирования \* /}

        X (char ch) {/ \* здесь тоже /}

        ~ X () {/ \* очистка \* /}

        ...

     };

Аргумент size задает размер объекта создается, а newalloc i newfree определяются пользователем функции распределения и уничтожения распределения памяти. Вызовы конструктора i деструктора для объектов класса Х (или объектов, производных от Х, для которых не существует собственных переопределенное операций new i delete) приведет к запуску соответствующих операций, определяемых пользователем X :: operator new () i X: : operator delete (), соответственно.

Операции-функции X :: operator new () i X :: operator delete () являются статическими элементами Х, как при явном объявлении их static, так i без него, поэтому они не могут быть виртуальными функциями.

Стандартные, определенные (глобальные) операции new i delete могут при этом также использоваться в области действия Х, как явно с операциями глобальной области действия (:: operator new i :: operator delete), так i неявно, при образовании i pуйнуваннi объектов классов, отличных от класса Х i не являющихся производными от класса Х. Например, можно использовать стандартные операции new i delete при определении пеpевантажених версий:

     void \* X:: operator new (size\_t) {

        void \* ptr = new char [s]; // Вызов стандартной new ...

        return ptr;

     }

     void X :: operator delete (void \* ptr) {

        ...

        delete (void \*) ptr; // Вызов стандартной delete

     }

     Причиной того, что размер аргумента определяется size, является то, что классы, производные от Х, наследуют X :: operator new. Размер объекта походной класса может существенно отличаться от размера, который определяется базовым классом.

**Переопределения унарных операций**

Перегрузка префiкснои или постфiкснои унарными операции выполняется с помощью объявления нестатические функции-члену, функции, которая не принимает никаких аргументов, либо с помощью объявления функции, принимающей один аргумент. Если @ составляет унарными операцию, то @x i x @ можно интерпретировать как x.operator @ () i operator @ (x), соответственно, в зависимости от объявления. Если объявление было сделано в обеих формах, то и разрешение неоднозначностi зависит от переданных при вызове операции стандартных аргументов.

В С ++ поздних веpсiй, когда operator ++ или operator-- объявляется как функция-член без паpаметpiв или как функция, не являющаяся членом, с одним паpаметpами, она только пеpевизначае пpефiксну опеpацiю ++ или опеpацiю -. Постфiксну опеpацiю ++ или опеpацiю - можно пеpевизначиты только определив ее как функцию-элемент с одним паpаметpами int или как функцию, не является членом класса, которая пpиймае один класс i один паpаметp int. Добавим, напpиклад, к попеpеднього пpикладу такие строки:

     operator ++ (int)

     operator-- (int)

Когда пеpевизначаеться только пpефiксний ваpiант operator ++ или operator-- i опеpацiя применяется к объекту класса как постфiксна, то компiлятоp выдает попеpедження. После этого он вызывает пpефiксну опеpацiю, позволяя компiлюваты коды веpсiи 2.0. В pезультатi компиляции попеpднього пpикладу представляются следующие попеpедження:

     Warning: Overloaded prefix 'operator ++' used as postfix

     operator in function func ()

     "Пеpевизначена пpефiксна опеpацiя 'operator ++' используется в функции func () как постфiксна опеpация"

     Warning: Overloaded prefix 'operator - "used as postfix

     operator in function func ()

     "Пеpевизначена пpефiксна опеpацiя 'operator -" используется в функции func () как постфiксна опеpацiя "

**Переопределения бинарных операций**

Переопределения бинарных операции выполняется с помощью объявления нестатические функции-члена класса, принимает один аргумент, или с помощью объявления функции не члены класса (обычно friend), который принимает два аргумента. Если @ есть бинарный операцией, то x @ y можно интерпретировать либо как x.operator @ (y), или как operator @ (x, y), в зависимости от выполненных объявлений. Если объявлены обе формы, то разрешение неоднозначностi зависит от переданных при вызове операции стандартных аргументов.

**Переопределение операции присвоения =**

Операцию пpисвоенння = можно переопределить только с помощью объявления нестатические функции-члена. К примеру:

     class String {

                ......

        String & operator = (String & str);

                ......

        String (String &)

        ~ String ();

     }

Данный код, совместно с объявлением String :: operator = (), позволяет выполнять рядковi присвоение str1 = str2, как это делается в других языках программирования. В отличие от других функций-операций, функция-операция присвоения не может наследоваться производными классами. Если для любого класса Х не существует операции, определяется =, то операция = определяется по умолчанию как поэлементное присвоение элементов класса Х:

     X & X :: operator = (const X & source) {// поэлементное присвоение}

Переопределение операции вызова функции ()

Вызов функции:

первiсний\_вираз (<список\_виразiв>)

рассматривается в виде двоичной операции с операндами "первiсний\_вираз" i "список\_виразiв" (возможно, пустой). Соответствующая функция-операция - это operator (). Предоставлена ​​функция может определяться пользователем для класса Х (i любых производных классов) только в виде нестатические функции-члена. Вызов x (arg1, arg2), где x составляет объект класса Х, интерпретируется в таком случае как x.operator () (arg1, arg2).

**Переопределение операции индексирования []**

Аналогичным образом, операция индексирования:

первiсний\_вираз [высказывание]

рассматривается как двоичная операция с операндами "первiсний\_вираз" i "высказывание". Соответствующая операция-функция - это operator []. Она может определяться пользователем для класса Х (i любых производных классов) только путем нестатические функции-члена. Высказывание x [y], где x является объектом класса Х, интерпретируется в данном случае как x.operator [] (y).

**Переопределение операции доступа к элементу класса ->**

Доступ к элементу класса с помощью:

первiсний\_вираз -> выражение

рассматривается как унарная операция. Функция operator-> должна рассматриваться как нестатические функция-член. Высказывание x -> m, где x - это объект класса, интерпретируется как (x.operator -> ()) -> m, таким образом, что operator-> () должен или возвращать указатель на объект данного класса, или возвращать объект класса, для которого определяется operator->.

Контрольные вопросы к Л.Р. 5:

1. Какую возможность открывает перегрузки операторов?

2. Какие операторы не могут перегружаться?

3. Какими двумя способами можно сказать перегрузки операторов?

4. Что происходит, если функция, которая перегружает оператор является дружественной функцией?

5. Сколько параметров будет перегружена функция, если она является членом класса, объектами которого оперирует?

6. Как может быть определена унарная операция?

7. служит признаком того, что функция вызывается для операции в Постфиксная варианте?

8. Как можно определить свои собственные правила преобразования типов?

9. Какие существуют правила перегрузки операторов?

Задания для самостоятельной работы:

1. реализовать класс TInteger, который позволял строить и викоpистовуваты большие цели числа> 32 бит, викоpистовуючы оператоpы =, +, ++, + =, -, -, - =, \* \* =, /, / =, << , >>, <,>, ==,! =, <=,> =, |, | =, &, & =, ~.

     class TInteger {

          void TInteger (int Size) // Size - pозмip целого числа в байтах

          TInteger & operator + = (TInteger)

          TInteger & operator + = (Int)

          TInteger & operator + (TInteger)

          TInteger & operator + (Int)

               ...........

     }

2. Определить тип Vec4 как вектор из четырех элементов типа float. Перегрузить оператор [] для Vec4. Определить операторы +, -, \*, /, =, + =, - = для комбинаций векторов и чисел с плавающей точкой.

3. Определить класс Vector, аналогичный Vec4, размер которого передается в качестве аргумента конструктору Vector :: vector (int).

4. Bизначиты класс Mat4 как вектор из четырех Vec4. Для этого класса определите operator [], возвращающий Vec4. Для этого типа перегрузить обычные матричные операции.

5. Определить класс Matrix, аналогичный Mat4, размеры которого передаются в качестве аргументов конструктору Matrix :: matrix (int, int).

6. Создать класс Money для работы с денежными суммами. Число должно быть представлено двумя полями: типа long для гривен и типа unsigned char - для копеек. Дробная часть (копейки) при выводе на экран должна быть отделена от целой части запятой. Реализовать сложение, вычитание и деление сумм.

7. Рациональный (нескорочуваний) дробь представляется парой чисел (a, b), где a - числитель, b - знаменатель. Создать класс Rational для работы с рациональными дробями. Обязательно должны быть реализованы следующие операции:

• Добавление add, (a, b) + (c, d) = (ad + bc, bd)

• Вычитание sub, (a, b) - (c, d) = (ad-bc, bd)

• Умножение mul, (a, b) x (c, d) = (ac, bd)

• Деление div, (a, b) / (c, d) = (ad, bc)

8. Создать класс Fraction для работы с дробными числами. Число должно быть представлено двумя поля: целой частью - длинное целое со знаком, дробной частью - беззнаковое короткое целое. Реализовать арифметические операции: сложение, вычитание, умножение и операции поривння.