* 1. **Перегрузка операций**

В С++ существует возможность перегрузки операции внутри класса, например, можно добиться того, что операция \* при работе с матрицами осуществляла умножение матриц, а при работе с комплексными числами – умножение комплексных чисел.

Для перегрузки операций внутри класса нужно написать специальную функцию – метод класса. При перегрузке операций следует помнить следующее:

• при перегрузке нельзя изменить приоритет операций;

• нельзя изменить тип операции (из унарной операции нельзя сделать бинарную или наоборот);

• перегруженная операция является членом класса и может использоваться только в выражениях с объектами своего класса;

• нельзя создавать новые операции;

• запрещено перегружать операции: .(доступ к членам класса), унарную операцию \*(значение по адресу указателя), ::(расширение области видимости) , ?: (операция if);

• допустима перегрузка следующих операций: +, -, \*, /, %, =, <, >, +=, -=, \*=, /=, <, >, &&, ||, ++,

--, (), [], new, delete.

Для перегрузки операции необходимо создать функцию-метод:

**тип operator операция (список параметров) { тело функции }**

здесь

● тип – тип возвращаемого операцией значения,

● operator – служебное слово,

● операция – перегружая операция,

#### Перегрузка унарных операций

Унарная функция-операция, определяемая *внутри класса*, должна быть представлена с помощью нестатического метода без параметров, при этом операндом является вызвавший ее объект.

Если функция определяется *вне класса*, она должна иметь один параметр типа класса.

Операции постфиксного инкремента и декремента должны иметь первый параметр типа **int**. Он используется только для того, чтобы отличить их от префиксной формы.

#### ЗАДАЧА 4.1. Пусть операция ++x увеличивает действительную и мнимую часть комплексного числа x на 1, а x++ увеличивает на 1 только действительную часть комплексного числа x.

#### #include <iostream>

#### using namespace std;

#### class complex {

#### float x; //Действительная часть комплексного числа.

#### float y; //Мнимая часть комплексного числа.

#### public:

#### complex(bool pr=true); //Конструктор класса

#### //Функция, перегружающая оператор ++x, в этом случае это метод без параметров.

#### complex operator++()

#### {

#### x++; y++; //Увеличиваем действительную и мнимую часть на 1.

#### return \*this;

#### }

#### //Функция, перегружающая оператор x++, в этом случае это метод с абстрактным параметром

#### //целого типа. Наличие целого типа в скобках говорит только о том, что что перегружается //оператор x++, а не ++x.

#### complex operator++(int)

#### {

#### x++; //Увеличиваем действительную часть на 1.

#### return \*this;

#### }

#### show\_complex();

#### };

#### Перегрузка бинарных операций

Бинарная функция-операция, определяемая *внутри класса*, должна быть представлена с помощью нестатического метода с параметрами, при этом вызвавший ее объект считается первым операндом, а параметр метода – вторым операндом.

#### ЗАДАЧА 4.2. Создать класс для работы с комплексными числами, в котором перегрузить операции сложения и вычитания.

#### Определение методов, перегружающих операции, внутри класса:

#### #include <iostream>

#### using namespace std;

#### class complex {

#### float x; //Действительная часть комплексного числа.

#### float y; //Мнимая часть комплексного числа.

#### public:

#### complex(bool pr=true); //Конструктор класса

#### complex operator+(complex M) //Метод, реализующий перегрузку операции сложения.

#### {

#### //Создаем комплексное число temp, в котором будет храниться результат

#### //сложения двух комплексных чисел.

#### complex temp(false);

#### //Действительная часть нового комплексного числа формируется, как результат сложения //действительной части первого и второго операнда, первым операндом является текущий //класс, вторым – передаваемое в функцию комплексное число M.

#### temp.x=x+M.x;

#### //Мнимая часть нового комплексного числа формируется, как результат сложения //мнимой части первого и второго операнда.

#### temp.y=y+M.y;

#### //Возвращаем сумму двух комплексных чисел, в качестве результата.

#### return temp;

#### };

#### complex operator-(complex M) //Метод, реализующий перегрузку операции вычитания.

#### {

#### complex temp(false);

#### //Действительная часть нового комплексного числа формируется, как результат //вычитания действительной части первого и второго операнда, первым операндом //является текущий класс, вторым – передаваемое в функцию комплексное число M.

#### temp.x=x-M.x;

#### //Мнимая часть нового комплексного числа формируется, как результат вычитания //мнимой части первого и второго операнда.

#### temp.y=y-M.y;

#### //Возвращаем разность двух комплексных чисел, в качестве результата.

#### return temp;

#### };

#### void show\_complex();//Метод вывода комплексного числа на экран.

#### };

#### //Конструктор класса complex, с логическим параметром (true - по умолчанию), если параметр равен //true, то в конструкторе будет запрашиваться действительная и мнимая часть числа, если же //параметр конструктора равен 0, то будет создаваться комплексное число с нулевой действительной //и мнимой частью.

#### complex::complex(bool pr)

#### {

#### if (pr){

#### cout<<"VVedite x\t"; cin>>x; cout<<"Vvedite y\t"; cin>>y;

#### show\_complex();

#### }

#### else{x=0;y=0;}

#### }

#### void complex::show\_complex() //Метод вывода комплексного числа на экран.

#### {

#### if (y>=0) cout<<x<<"+"<<y<<"i"<<endl;

#### else cout<<x<<y<<"i"<<endl;

#### }

#### int main()

#### {

#### complex chislo1, chislo2, chislo4(false), chislo3(false);

#### //Для сложения двух комплексных чисел достаточно использовать операцию +.

#### chislo3=chislo1+chislo2;

#### cout<<"chislo3=";

#### chislo3.show\_complex();

#### //Для вычитания двух комплексных чисел достаточно использовать операцию -.

#### chislo4=chislo1-chislo2;

#### cout<<"chislo4=";

#### сhislo4.show\_complex();

#### return 1;

#### }

#### Определение методов, перегружающих операции, вне класса:

#### complex operator+(complex M1,complex M2) //Метод, реализующий перегрузку операции сложения.

#### {

#### complex temp(false);

#### temp.x= M1.x +M2.x;

#### temp.y= M1.y+M2.y;

#### return temp;

#### };

#### complex operator-(complex M1,complex M2) //Метод, реализующий перегрузку операции вычитания.

#### {

#### complex temp(false);

#### temp.x= M1.x-M2.x;

#### temp.y= M1.y –M2.y;

#### return temp;

#### };

#### Перегрузка операции индексирования

Операция индексирования [] обычно перегружается, когда тип класса представляет множество значений, для которого индексирование имеет смысл. Операция индексирования должна возвращать ссылку на элемент, содержащийся в множестве. Покажем это на примере класса **Vect**, предназначенного для хранения и работы с массивом целых чисел:

|  |  |
| --- | --- |
| #include <iostream.h>  #include <stdlib.h>  class Vect{       int\* p;       int size;  public:       Vect(int n = 10);       Vect(const int a[], int n); //инициализация  массивом       ~Vect() { delete [] p; }       int operator [] (int i);       void Print();  ...  };  Vect::Vect(int n){  size=n;       p = new int[n];  }  Vect::Vect(const int a[], int n){  size=n;       p = new int[n];       for (int i = 0; i < size ; i++) p[i] = a[i];  } | // Перегрузка операции индексирования:  int Vect::operator [] (int i){       if(i < 0 || i >= size){  cout << "Неверный индекс (i = " << i << ")" << endl;  cout << "Завершение программы" << endl;  exit(0);       }       return p[i];  }  void Vect::Print(){       for (int i = 0; i < size; i++)  cout << p[i] << " ";       cout << endl;  }  int main(){       int arr[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};       Vect a(arr, 10);       a.Print();       cout << a[5] << endl;       return 0;  } |

 Перегруженная операция индексирования получает целый аргумент и проверяет, лежит ли его значение в пределах диапазона массива. Если да, то возвращается адрес элемента, что соответствует семантике стандартной операции индексирования.