Перегрузка операторов

- Введение

Перегрузка операторов (operator overloading) — это возможность применять встроенные операторы языка к разным типам, в том числе и пользовательским.

- Перегружаемые операторы

В C++17 стандарт разрешает перегружать следующие операторы:

+, -, *, /, %, ^, &, |, ~, !, ,, =, <, >, <=, >=, ++, –-, <<, >>, ==, !=, &&, ||, +=, -=, /=, %=, ^=, &=, |=, *=, <<=, >>=, [], (), ->, ->*, new, new[], delete, delete[].

унарные, бинарные и n-арные (n>2)

- Унарные операторы – это операторы, которые для вычислений требуют одного операнда, который может размещаться справа или слева от самого оператора.
- Бинарные операторы – это операторы, которые для вычисления требуют двух операндов.
- n-арные_ - это операторы для вычислений требуют более двух операндов

суть перегрузки операторов

- Перегрузка оператора означает использование оператора для оперирования объектами классов.
- Перегрузка оператора – способ объявления и реализации оператора таким образом, что он обрабатывает объекты конкретных классов или выполняет некоторые другие действия. При перегрузке оператора в классе вызывается соответствующая операторная функция (operator function), которая выполняет действия, которые касаются данного класса.

Способы реализации операторных функций

- внутри класса. В этом случае, операторная функция есть методом класса;
- за пределами класса. В этом случае операторная функция объявляется за пределами класса как «дружественная» (с ключевым словом friend).
- Синтаксис

Тип operator-символ (parameter-list)

Если перегружается бинарный оператор, то parameter_list содержит один аргумент. Если перегружается унарный оператор, то список аргументов пустой.

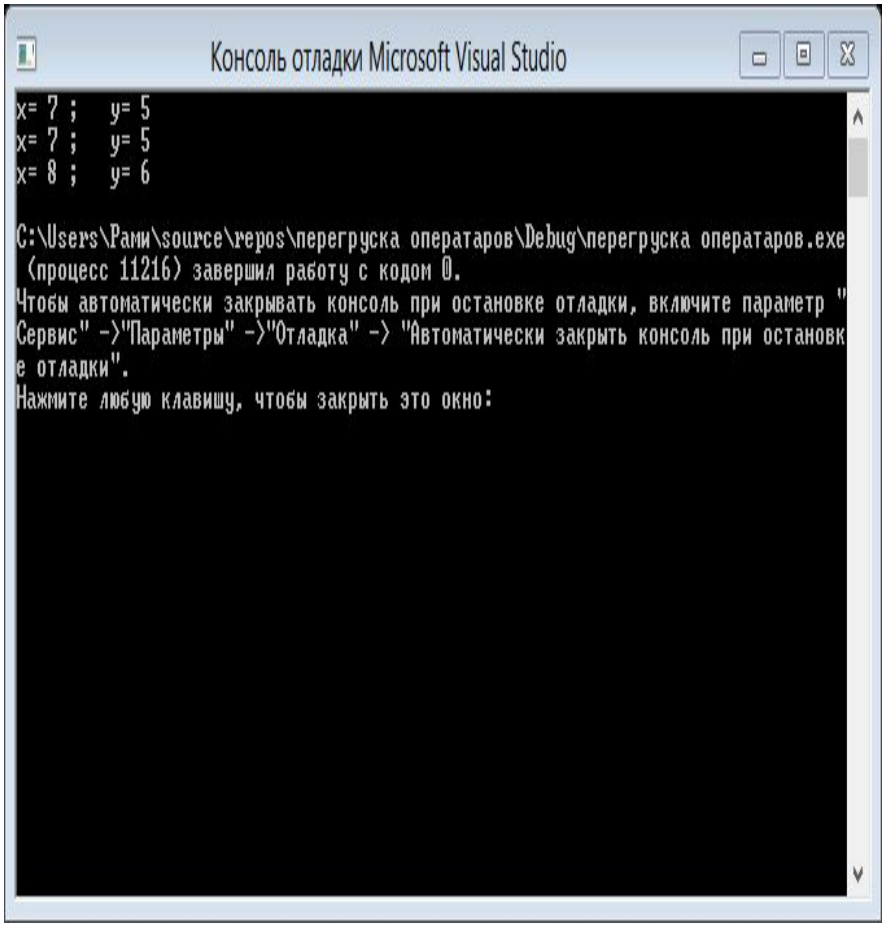
- Пример:

```
bool operator >( point& P)
    {……}
```

Перегрузка операторов в унарных операторах


```
class point
{
private:
   int x, y;
public:
   point(int _x, int _y)
   {
      x = _x;
      y = _y;
   }
   point()
   {
      x = y = 0;
   }
   void print()
   {
      cout << "x= " << x << " ;  y= " << y << endl;
   }
    void operator ++()
    {
      ++x;
      ++y;
    }
};
int main()
{
   point p(7, 5);
   p.print();
   ++p;
   p.print();
```

Пример : возвращаемое значение из функции оператора

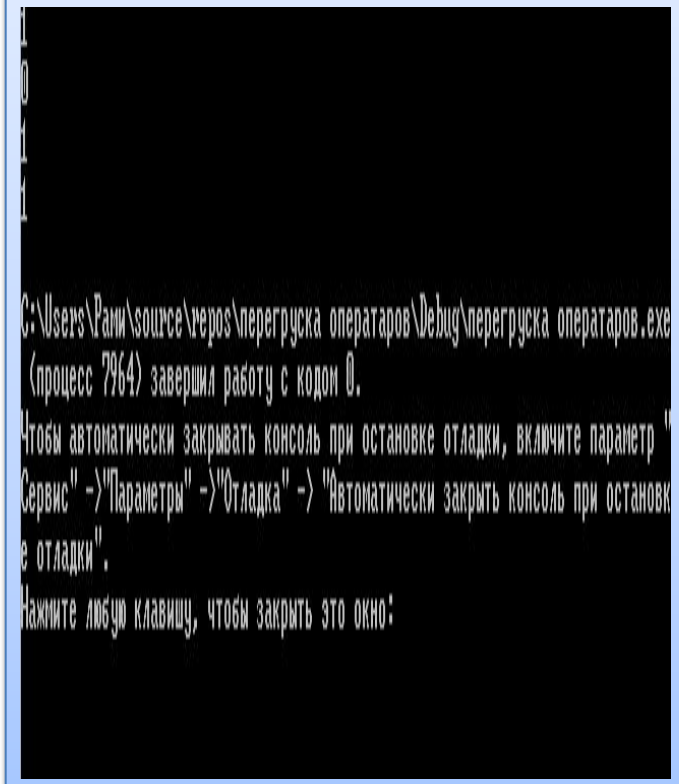



```
class point
{
private:
   int x, y;
public:
   point(int _x, int _y)
   {
      x = _x;
      y = _y;
   }
   point()
   {
      x = y = 0;
   }
   void print()
   {
      cout << "x= " << x << " ;   y= " << y << endl;
   }
    point& operator ++(int)
    {
        point pt;
       pt.x=x++;
       pt.y=y++;
       return pt;
    }
};
int main()
{
   point p(7, 5),p1;
   p.print();
   p1= p++;
   p1.print();
   p.print();
```

перегрузка двоичного оператора


```
point& operator *(const point& p)
  {
     point pt;
     pt.x = x * p.x;
     pt.y = y * p.y;
     return pt;
  }
  point& operator +(const point& p)
  {
     point pt;
    pt.x=x+p.x;
    pt.y=y+p.y;
    return pt;
  }
};
int main()
{
   point p(7, 5), p1(3, 4), p2;
   p2 = p + p1;
   p2.print();
   p2 = p*p1;
   p2.print();
```

Операторы сравнения




```
bool operator ==(const point& pt)
  {
    if (x == pt.x && y == pt.y)
      return true;
    else return false;
   }
  bool operator !=(const point pt)
  {
    if (x != pt.x || y != pt.y)
      return true;
    else
      return false;
  }
  bool operator >(const point& pt)
  {
    if ((sqrt(x) + sqrt(y)) > (sqrt(pt.x) + sqrt(pt.y)))
      return true;
    else
      return false;

  }
  bool operator <=(const point& pt)
  {
    if ((sqrt(x) + sqrt(y)) <= (sqrt(pt.x) + sqrt(pt.y)))
      return true;
    else
      return false;

  }
};
int main()
{
  point p(7, 5), p1(3, 4), p2(7,5);
  bool l = (p == p2);
  cout << l << endl;
  cout << (p <= p1) << endl;
  cout << (p > p1) << endl;
  cout << (p != p1) << endl;

}
```

Оператор []

```
class point
{
private:
   int x, y;
public:
   point(int _x, int _y)
   {
      x = _x;
      y = _y;
   }
   point()
   {
      x = y = 0;
   }
   void print()
   {
      cout << "x= " << x << " ;   y= " << y << endl;
   }

};
```

```
template <class T>
class mass
{
private :
   int size;
   T* array;
public:
   mass(int  _size)
   {
      size = _size;
      array = new T[size];
   }
   T& operator[](int i)
   {
      return array[i];
   }

~mass()
   {
      delete[]array;

   }
};
int main()
{
   point p(7, 5), p1(3, 4), p2(3,9);
   mass <point> m (3);
   m[0] = p;
   m[1] = p1;
   m[2] = p2;
    point p4 = m[1];
    p4.print();
}
```

Перегрузка оператор ()

```
class point
{
private:
   int x, y;
public:
   point(int _x, int _y)
   {
      x = _x;
      y = _y;
   }
   point()
   {
      x = y = 0;
   }
   void print()
   {
      cout << "x= " << x << " ;   y= " << y << endl;
   }

};
template <class T>
```

```
template <class T>
class mass
{
private :
   int size;
   T** array;
public:
   mass(int  _size)
   {
      size = _size;
      array = new T*[size];
      for (int i = 0; i < size; i++)
         array[i] = new T[size];
   }
   T& operator()(int i, int j)
   {
      return array[i][j];
   }


   ~mass()
   {
      for (int i = 0; i < size; i++)
         delete array[i];
      delete[]array;

   }
};
int main()
{
   point p(7, 5), p1(3, 4), p2(3,9),p3(1,5);
   mass <point> m (2);
   m(0,0) = p;
   m(0,1) = p1;
   m(1,0) = p2;
   m(1, 1) = p3;
  point p4 = m(0,1);
    p4.print();
}
```

дружественная операторная функция

```
class Complex
{
private:
   float real; // вещественная часть
   float imag; // мнимая часть
public:

   Complex(float _real, float _imag)
   {
      real = _real;
      imag = _imag;
   }
   // объявление "дружественной" к классу Complex операторной функции
   friend Complex operator-(Complex c1, Complex c2);
};
// "дружественная" к классу Complex операторная функция,
// реализована за пределами класса,
// осуществляет вычитание комплексных чисел
Complex operator-(Complex c1, Complex c2)
{
   Complex c; // создать объект класса Complex

   // вычитание комплексных чисел
   c.real = c1.real - c2.real;
   c.imag = c1.imag - c2.imag;

   return c;
}
```

Перегрузка операторы ввода вывода << и >>

```
ostream& operator <<(ostream& out, point& p)
{
	out << "point_x=" <<p.x << ";   point_y=" << p.y
<< endl;
	return out;
}

istream& operator >>(istream& in, point& p)
{
	cout << "enter p.x=";
	in >> p.x;
	cout << "enter p.y=";
	in >> p.y;
	return in;
}
int main()
{
cout<<p;
cint>>p;
}
```

Перегрузка операторов

Перегрузка операторов в программировании — один из способов реализации полиморфизма, заключающийся в возможности одновременного существования в одной области видимости нескольких различных вариантов применения операторов, имеющих одно и то же имя, но различающихся типами параметров, к которым они применяются

Перегрузка операций осуществляется с помощью методов специального вида и подчиняется следующим правилам:

1- **При перегрузке сохраняются количество аргументов, приоритеты**

2- операций и правила ассоциаций (справа налево или слева направо), используемые в стандартных типах данных.

3- Для стандартных типов перегружать операции нельзя.

4- Функции-операторы не могут иметь аргументов по умолчанию.

5- Функции-операторы могут наследоваться (за исключением =).

6- Функции-операторы не могут быть статическими.

7- Для одного и того же оператора можно объявить несколько перегруженных операторов - функций. Они должны отличаться по типу и количеству аргументов.