Лекция 3

Перегрузка операций

Перегрузка операций

Перегрузка постфиксной унарной операции

Для постфиксной операции характерно то, что в первую очередь операнд участвует в выражении, после чего над ним выполняется сама операция.

Для отличия постфиксной операции от префиксной, в нее вводится дополнительный аргумент целого типа, который нигде не используется.

Перегрузка операций

Пример перегрузки постфиксного инкремента.

```
class Test
{
    // …
    Test operator ++(int)
    {
    Test Temp(*this);
    test++;
    return Temp;
    }
    // …
};
```

Перегрузка операций

В данном примере перегрузка осуществлена как составляющая функция класса. Перегрузку через дружественную и внешнюю функции реализуйте самостоятельно.

Перегрузка операций

Перегрузка бинарных операций

Бинарная операция имеет два аргумента. Если она перегружается как составляющая функция класса, то должна быть представлена нестатическим методом класса, при этом вызвавший ее объект считается первым (левым) операндом.

Перегрузка операций

Рассмотрим пример бинарной операции сравнения на больше.

```
class Test
{
    int test;
    // …
public:
    bool operator >(const Test &t)
    {
      if(this->test > t.test) return true;
      cout << “ Наша операция > “ << endl;
      return false;
    }
    // …
};
```

Перегрузка операций

После такой перегрузки можно сравнивать два объекта типа Test:

```
Test tst_1(10), tst_2(2);
//…
if(tst_1 > tst_2) // …
// …
```

Перегрузка операций

Если бинарная операция перегружается как внешняя функция, ей необходимо передать два параметра.

```
bool operator >(const Test &t_1, const Test &t_2)
{
 if(t_1.test > t_2.test) return true;
 return false;
}
```

Важное замечание: поле test должно быть объявлено с ключом доступа public.

Перегрузка операций

Перегрузку через дружественную функцию попробуйте сделать самостоятельно.

Перегрузка бинарных арифметических операций

Бинарные арифметические операций: +, -, *, / должны возвращать объект класса (именно объект), для которого они используются.

Перегрузка операций

Если левый операнд перегружаемой бинарной операции представляет собой не пользовательский тип, а один из стандартных, то такая операция не может быть перегружена как составляющая функция класса. В этом случае операция должна быть перегружена как внешняя функция, у которой первый аргумент представляет один из встроенных, а второй – пользовательский тип.

Перегрузка операций

Например,

```
class Test
{
public:
    double test;
    Test(){};
    Test(double t):test(t){};
    friend ostream &operator <<(ostream &, const
    Test &);
};
```

Перегрузка операций

```
ostream &operator <<(ostream &out, const Test &t)
{
    out << t.test;
    return out;
}
Test operator +(double d, Test &t)
{
    return d+t.test;
}
```

Заметим, что и в данном случае в качестве результата возвращается объект типа Test.

Перегрузка операций

Пример использования:

```
int main()
{
   double d=3.22;
   Test t(5.46);
   cout << d+t << endl;
   return 0;
}
```

Перегрузка операций

(2 сем)Перегрузка операции присваивания и инициализации

Говоря о перегрузке операций, нельзя не отметить, что две операции: присваивания (=) и взятие адреса(&) предопределены языком для любого пользовательского типа. Это означает, что если пользователь не перегрузил эти операции, они все равно действительны.

Перегрузка операций

Операция присваивания определяется как операция поэлементного копирования полей одного объекта в поля другого (метод «поле за полем»). Эта операция вызывается каждый раз, когда одному объекту присваивается значение другого. Операцию присваивания следует переопределять в тех случаях, если класс содержит поля, память под которые выделяется динамически.

Перегрузка операций

Чтобы сохранить семантику присваивания, операция-функция должна возвращать ссылку на объект, для которого она вызвана, и содержать единственный параметр – ссылку на присваиваемый объект.

Операция присваивания может быть переопределена только как составляющая функция класса (!).

Перегрузка операций

Рассмотрим конкретный пример:

```
class Test
{
    double *test;
public:
    Test(){};
    Test(double);
    ~Test(){ delete test; }
    Test &operator =(const Test &);
    friend ostream &operator <<(ostream &, const Test &);
};
```

Перегрузка операций

Конструктор класса:

```
Test::Test(double d)
{
   test = new double;   // поле в динамической
   области памяти
   *test = d;
}
```

Перегрузка операций

Операция присваивания:

```
Test &Test::operator=(const Test &t)
{
    cout << " Операция присваивания  !!! " << endl;
    if(&t == this) return *this;
    if(!test) delete test;
    test = new double;
    *test = *t.test;
    return *this;
}
```

Перегрузка операций

Использование перегруженной операции присваивания:

```
int main()
{
    Test tst_1(100);
    Test tst_2;
    tst_2 = tst_1;    // работает перегруженная операция присваивания
    cout << tst_1 << ' ' << tst_2 << endl;
    return 0;
}
```

Перегрузка операций

Еще раз:

tst_2 = tst_1; работает перегруженная операция присваивания, но

Test tst_3 = tst_1; работает операция инициализации, но не операция присваивания. Это разные операции (!!!).

Перегрузка операций

Операция индексирования

Операция индексирования [] перегружается когда тип класса представляет множество значений, для которого индексирование имеет смысл. Операция индексирования должна возвращать ссылку на элемент, содержащийся во множестве.

Перегрузка операций

Рассмотрим пример.

```
class Vect
{
    int *p;
    int size;
public:
    explicit Vect(int n = 10);
    Vect(const int [], int n);
    int &operator [](int);
    void Print();
};
```

Перегрузка операций

Конструкторы класса:

```
Vect::Vect(int n):size(n)
{
    p = new int[size];
}
Vect::Vect(const int a[], int n):size(n)
{
    p = new int[size];
    for(int i=0; i<=size; i++) p[i] = a[i];
}
```

Перегрузка операций

Операция индексирования:

```
int &Vect::operator [](int i)
{
     cout << “Операция индексирования “;
    if(i<0 || i>=size)
    {
      cout << " Выход за границы массива " << endl;
      exit(1);
    }
    return p[i];
}
```

Перегрузка операций

Функция вывода:

```
void Vect::Print()
{
   for(int i=0; i<=size; i++)
   cout << p[i]  << ' ';
   cout << endl;
}
```

Перегрузка операций

Использование операции индексирования:

```
int main()
{
    int arr[] = {11,22,33,44,55,66,77,88,99};
    Vect vec(arr, 8);
    vec.Print();
    cout << vec[15] << endl; // выход за границы
    return 0;
}
```

Перегрузка операций

При использовании константных объектов класса, необходимо переопределить еще одну операцию индексирования. Смысл ее тот же, но она не имеет право на изменение значения константного объекта.

```
// …
   Vect(const int [], int n);
   int &operator [](int);
   const int &operator [](int) const;  //
```

Определение операции напишите самостоятельно.

Перегрузка операций

Попытка изменения константного объекта приведет к сообщению:

const Vect vec(arr, 8);

vec[3] = 37;

\test_index: error C3892: vec: невозможно присваивать значения переменной, которая объявлена как константа

Перегрузка операций

Операция индексирования обязательно должна быть составляющей функцией класса. Она считается бинарной, поэтому может стоять как с левой стороны выражения, так и с правой.

Перегрузка операций

***Операция выбора элемента**

Операция выбора элемента ‘->’ так же может быть перегружена. Она относится к унарным операциям своего левого операнда. Этот левый операнд должен быть либо объектом класса, либо ссылкой на объект класса, для которого операция перегружается. Функция operator-> обязательно должна быть составной нестатической функцией класса.

Перегрузка операций

В качестве результата операция должна возвращать либо объект, либо указатель на объект. К этому указателю затем автоматически будет применена предопределенная операция выбора (по указателю ->).

Если операция -> перегружена для класса Name, тогда выражение name-> m, где name – объект класса Name, интерпретируется как (name.operator->())->m.

Перегрузка операций

Обычно данную операцию имеет смысл перегружать в том случае, когда иерархия классов сильно разветвлена.

Рассмотрим пример.

```
struct N
{
   int a;
};
```

Перегрузка операций

```
struct  L1
{
   N *target;
   N *operator->() const
      { return target; }
};

struct  L2
{
   L1 *target;
   L1 &operator->() const
      { return * target; }
};
```

Перегрузка операций

Использование перегруженных операций:

```
int main()
{
    N x = { 3 };
    L1 y = { & x };
    L2 z = { & y };
    cout << x.a << y->a << z->a << endl;       // print
    "333"
    return 0;
}
```

Перегрузка операций

Если изменить перегруженную операцию класса struct L2 следующим образом:

```
L1 &operator->() const
{
    if(target == 0)
    {
    cout << " Список пустой " << endl;
    exit(-1);
    }
    return * target;
}
```

Перегрузка операций

Тогда при объявлении

```
N x = { };
L1 y = { };
L2 z = { };
```

Получим сообщение о пустоте списка.

Перегрузка операций

Перегрузка операции приведения типа

Существует возможность определить функцию-операцию, которая будет осуществлять преобразование объектов класса к другому типу. Общий формат:

operator имя_типа ();

Тип возвращаемого значения и параметры указывать не требуется. Можно переопределять методы как виртуальные, которые можно использовать в иерархии классов.

Перегрузка операций

Рассмотрим несложный пример:

```
class Test
{
    int test;
public:
    Test(){};
    Test(int t):test(t){};
    operator int()
    {  return test; }
};
```

Перегрузка операций

Использование данного оператора:

```
Test t(190);
int i = int(t);
cout << i << endl;
```

Или более в «экзотическом» виде:

```
cout << t.operator int() << endl;
```

Перегрузка операций

Перегрузка операции вызова функции

Класс в котором определена операция вызова функции, называется функциональным классом. При этом, от такого класса не требуется наличия каких-либо полей и других методов.

Рассмотрим классический пример функционального класса.

Перегрузка операций

```
class if_greater
{
public:
    int operator ()(int a, int b)
    { return a>b; }
};
```

Перегрузка операций

Использование данной операции несколько необычно по сравнению с другими операциями:

```
int main()
{
    if_greater x;
    cout << x(1,5) << endl;
    cout << if_greater()(5,1) << endl;
    return 0;
}
```

Перегрузка операций

Перегруженная операция вызова функции приводит к важному понятию, используемому во многих языках программирования – *функциональный объект или функтор.*

В терминах C++ функции не являются объектами, поэтому они функторами не считаются, хотя имена функций, не функции, а именно имена функций входят в категорию функциональных объектов.

Перегрузка операций

Имя функции — это идентификатор, который неявно приводиться к указателю (как и имя массива). Поскольку указатели в терминах C++ объектами считаются, то и имя функции, приведённое к указателю, считается объектом, а поскольку оно объект, то можно говорить о функциональности объекта.

Перегрузка операций

Посмотрим еще один пример использования функтора, когда в классе перегружается операция (), а потом объект класса используется подобно функции, вбирая в себя аргументы.

Перегрузка операций

```
class Less
{
public:
    bool operator() (const int left, const int right)
    {  return left < right;  }
};

int main()
{
    int arr[] = {1,2,3,4,5};
    Less less;      //создаётся функциональный объект
     sort(begin(arr), end(arr), less);   //передаётся функтор в алгоритм
    сортировки
}
```

Перегрузка операций

Перегрузка операций

Перегрузка операций

Перегрузка операций

Перегрузка операций

Перегрузка операций

Перегрузка операций

Перегрузка операций

Перегрузка операций

Перегрузка операций