### Перегрузка операций

Язык С++ позволяет распространить перегрузку операций на пользовательские типы, то есть классы.

Для перегрузки операций используется специальная форма функции, называемая функцией операции. Функция операции имеет следующую форму:

[тип возвращаемого значения] *operator* op(аргументы),

где *ор* – символ перегруженной операции;

*operator* – ключевое слово

Например, *operator*+(…) перегружает операцию сложения, а operator\*() – операцию умножения. Операция *op* должна быть допустимой операцией в С++, а не произвольным символом. Например, объявить функцию операции *operator*@() невозможно, т.к. в С++ не существует операции «@».

Использование перегрузки операции имеет некоторые ограничения:

1. Нельзя использовать операцию в таком стиле, которая нарушает правила синтаксиса исходной операции. Например, нельзя перегрузить операцию взятия модуля так, чтобы эта операция применялась с одним операндом, например - %x.
2. Перегруженные операции должны иметь, как минимум, один операнд типа, определяемого пользователем. Это предотвращает перегрузку операций, работающих со стандартными типами. То есть переопределить операцию вычитания так, чтобы данная операция вычисляла сумму двух вещественных чисел вместо разности, не получится.
3. Невозможно определить новые символы операций.
4. Для перегрузки операций присваивания =, вызова функции (), индексации [], доступа к членам класса -> через указатель используются только функции-члены класса.
5. Операции доступа к полям . (точка), к вложенным именам доступа, к полям по указателю, *sizeof* и некоторые другие не перегружаются.
6. **Операция как глобальная функция**

В начале следует определить пользовательский тип. Пусть это будет структурный тип Point.

Определение структурного типа *Point* на алгоритмическом языке С++ (развернуть)…

struct Point // Реализация через структуру

{

int x, y; // необходимые поля

};

Теперь следует перегрузить операцию сложения для созданного типа.

Перегрузка операции сложения на алгоритмическом языке С++ (развернуть)…

Point operator + (Point A, Point B)

{

Point C;

C.x = A.x + B.x;

C.y = A.y + B.y;

return C;

}

Возвращаемое значение представляет из себя новый структурный объект *Point*, в которой хранятся координаты равные суммам соответствующих координат *Point*-операндов операции сложения.

1. **Операция – как метод класса**

Перегрузку можно организовать через метод класса.

Реализация перегрузки операции сложения через класс на алгоритмическом языке С++ (развернуть)…

class Point // Реализация через класс

{

protected:

int x, y; // свойства класса

public:

Point(int \_x, int \_y) : x(\_x), y(\_y) {};

// перегрузка операции сложения

Point operator + (Point);

};

Однако в этом случае требуется один параметр - правый операнд, поскольку левым операндом является данный объект, на который указывает *this.* Ключевое слово *this* представляет собой указатель на текущий объект данного класса. Соответственно через *this* можно обращатьсяк любым членам класса внутри него.

Point Point::operator + (Point B)

{

Point C(this->x, this->y);

C.x += B.x;

C.y += B.y;

return C;

}

Возвращаемый объект – экземпляр класса, будет хранить координаты равные суммам соответствующих координат *Point*-операндов операции сложения.

1. **Разрешение неоднозначностей перегрузок**

Рассмотрим следующий пример, иллюстрирующий неоднозначность перегрузок, на алгоритмическом языке С++ (развернуть)…

class Point // Неоднозначности перегрузок в классе

{

protected:

int x, y;

public:

Point(int \_x, int \_y) : x(\_x), y(\_y) {};

// Две эквивалентные по смыслу перегрузки

Point operator + (Point);

friend Point operator + (Point, Point);

};

Point Point::operator + (Point B) {

Point C(this->x, this->y);

C.x += B.x;

C.y += B.y;

return C;

}

Point operator + (Point A, Point B) {

Point C(A.x, A.y);

C.x += B.x;

C.y += B.y;

return C;

}

При дальнейшем вызове операции сложения с двумя операндами типа *Point* будет возникать ошибка из-за неоднозначности вызова. Таких ситуаций стоит избегать и выбирать только один способ перегрузки.

1. **Перегрузка инкрементирующих операций**

Для перегрузки префиксной и постфиксной форм инкрементирующих операций (как методов класса) существуют правила:

* Префиксный инкремент не принимает аргументов и возвращает ссылку на объект;
* Постфиксный инкремент всегда принимает неиспользуемый аргумент типа *int* и возвращает копию первоначального объекта.

Перегрузка инкрементирующих операций на алгоритмическом языке С++ (развернуть)…

class Integer // Перегрузка инкремента

{

protected:

int num;

public:

Integer() {};

Integer(int \_num) : num(\_num) {};

// Префиксный и постфиксный инкременты

Integer& operator ++ (); // Пре-

Integer operator ++ (int); // Пост-

};

Integer& Integer::operator ++ ()

{

this->num++;

return \*this;

}

Integer Integer::operator ++ (int)

{

Integer temp(this->num);

++(\*this);

return temp;

}

Теперь становится возможным инкрементировать объекты данного класса как префиксно, так и постфиксно.

1. **Перегрузка присваивающих операций**

Присваивающие операции всегда принимают объект соответствующего типа и возвращают ссылку на данный объект (\*this):

Перегрузка присваивающих операций на алгоритмическом языке С++ (развернуть)…

class Integer // Перегрузка присваивания

{

protected:

int num;

public:

Integer() {};

Integer(int \_num) : num(\_num) {};

// Простое и со сложением

Integer& operator = (const Integer&);

Integer& operator += (const Integer&);

int getNum(); // получение num

};

Integer& Integer::operator = (const Integer& b)

{

this->num = b.num;

return \*this;

}

Integer& Integer::operator += (const Integer& b)

{

this->num = this->num + b.num;

return \*this;

}

Подобным образом перегружаются операции -=, \*=, /=, %=.

1. **Перегрузка операций ввода/вывода**

Операция ввода принимает ссылку на поток ввода ostream и ссылку на объект, но возвращает ссылку на объект ввода.

Операция вывода принимает ссылку на поток вывода istream и константную ссылку на объект, возвращает ссылку на объект вывода.

Перегрузка операций ввода и вывода на алгоритмическом языке С++ (развернуть)…

class Integer // Перегрузка ввода/вывода

{

protected:

int num;

public:

Integer() {};

Integer(int \_num) : num(\_num) {};

friend istream& operator >> (istream&, Integer&); // ввод

friend ostream& operator << (ostream&, const Integer&); // вывод

};

istream& operator >> (istream& in, Integer& a) {

in >> (a.num);

return in;

}

ostream& operator << (ostream& out, const Integer& a) {

out << (a.num);

return out;

}

Теперь становится возможным вводить и выводить объекты данного класса.

Таким образом, были рассмотрены наиболее употребляемые перегрузки операторов. При необходимости перегрузки нерассмотренных операторов, всегда можно обратиться к справочным руководствам.

**Список литературы**

1. Саттер, Герб. Решение сложных задач на C++, 2017. – 400 c.

2. Страуструп, Бьерн. Программирование. Принципы и практика с использованием C++, 2018. – 1328 c.

3. Мэйерс, Скотт. Эффективное использование C++. 55 верных способов улучшить структуру и код ваших программ, 2017. – 300 c.

4. Лафоре, Роберт. Объектно-ориентированное программирование в C++, 2018. – 928 c.