Перегрузка операторов — конспект темы

Перегрузка операторов ввода-вывода

Перегрузка операторов позволяет нескольким вариантам использования оператора одновременно существовать в одной области видимости. Имя у вариантов одинаковое, а типы параметров, к которым они применяются, — разные.

Перегрузка предназначена для пользовательских типов и классов.

Использовать перегрузку нужно только если она не усложнит чтение кода. Пример: перегрузка операций ввода и вывода для структуры `Point`:

```
struct Point {
    int x;
    int y;
};

// вывод
ostream& operator<<(ostream& output, Point point) {
    output << point.x << ","s << point.y;
    return output;
}

// ввод
istream& operator>>(istream& input, Point& point) {
    int x, y;
    char comma; // переменная для считывания запятой
    input >> x >> comma >> y;
    point = Point{x, y};
    return input;
}
```

Перегрузка арифметических операций

Пользовательские типы — классы, структуры, перечислимые типы и объединения `union`. При объявлении новых типов данных в коде появляются новые сущности из предметной области, где работает программа.

Предметная область — часть реального или нереального мира, которую программа моделирует.

В предметной области над вводимыми сущностями можно выполнять сложение векторов, умножение матриц, арифметические операции над дробями. Перегрузка позволит объявить в программе собственные операции над пользовательскими типами. Например, сложить дроби можно операцией сложения `number1 + number2` вместо функции `AddRationals(number1, number2)`.

Перегрузка операций в C++ не изменяет размерность операций и их приоритет. У операций умножения и деления будет приоритет над сложением и вычитанием. Бинарные операции останутся бинарными, унарные — унарными.

Перегрузка арифметических операций +, -, *, / позволит выполнять арифметические операции над дробями естественно, как над целыми и вещественными числами:

```
int main() {
    Rational r1{1, 6};
    Rational r2{1, 3};
    Rational sum = (r1 + r2) * r1;
    cout << sum << endl; // Выведет 1/12
}
```

Типы аргументов бинарной операции не обязательно должны быть одинаковыми. Это зависит от предметной области. Может понадобиться реализовать для них две перегрузки в порядке операндов или только одну.

Операция деления вектора на скаляр существует, а обратная операция деления скаляра на вектор — нет.

При умножении двухмерного вектора и скаляра получается новый отмасштабированный двухмерный вектор. При этом умножать можно не только вектор на скаляр, но и скаляр на вектор.

Перегрузка операций присваивания

Краткая форма операций присваивания:

`a += b` — то же что `a = a + b`

`a -= b` — то же что `a = a - b`

`a *= b` — то же что `a = a * b`

`a /= b` — то же что `a = a / b`

Операции присваивания объявляются внутри класса. Левый аргумент — текущий объект, а правый передаётся как единственный параметр операции. Пример: перегрузка операции `+=` в структуре `Vector2D`.

```
struct Vector2D {
    Vector2D() = default;

    Vector2D(double x0, double y0)
        : x(x0), y(y0) {
    }

    // Левый аргумент операции += — это текущий экземпляр класса,
    // а правый передаётся в виде параметра операции
    Vector2D& operator+=(Vector2D right) {
        // Результат операции сохраняется в текущем экземпляре класса
        x += right.x;
        y += right.y;

        // return *this позволяет вернуть ссылку на текущий объект
        return *this;
    }

    double x = 0.0;
    double y = 0.0;
};
```

На основе операций присваивания можно компактно реализовать соответствующие бинарные операции вроде `+`, `-` , `*` и `/`.

Перегрузка операций сравнения

Операции сравнения `==`, `!=`, `<`, `>`, `<=`, `>=` для пользовательских типов данных в C++ можно задать так же, как арифметические.

Программист решает, перегружать все операции сравнения или только их часть, исходя из предметной области.

Операции сравнения возвращают результат типа bool и не изменяют значения своих аргументов. Чтобы избежать глубокого копирования тяжеловесных типов, их передают в операцию сравнения по константной ссылке.

```
int main() {
    cout << "Введите две обыкновенные дроби в формате x/y:"s << endl;
    Rational a, b;
    cin >> a >> b;

    // Аналогично if (b != Rational{0})
    if (b != 0) {
        cout << "Их частное равно "s << a / b << endl;
    } else {
        cout << "Невозможно найти частное, так как делитель равен 0"s << endl;
    }
}
```