Тема: Перегрузка операторов.

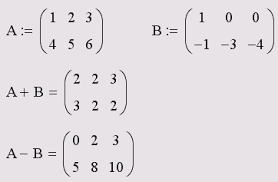
План занятия:

1. перегрузка унарных операторов,
2. бинарных операторов,
3. таблица разрешённых к перегрузке операторов,
4. r-value, l-value выражения

ПРО ПЕРЕГРУЗКУ ОПЕРАТОРОВ

Во многих языках программирования используются операторы: как минимум, присваивания (=, := или похожие) и арифметические операторы (+, -, \* и /).

Если мы определим свои классы для математических объектов, например, для матриц, мы можем реализовать метод их сложения.



Когда стоит перегружать операторы?

Запомните главное: перегружайте операторы тогда и только тогда, когда это имеет смысл. То есть если смысл перегрузки очевиден и не несёт в себе скрытых сюрпризов. Перегруженные операторы должны действовать так же, как и их базовые версии.

Приведём хороший и плохой примеры перегрузки операторов. Вышеупомянутое сложение матриц — наглядный случай. Здесь перегрузка оператора сложения интуитивно понятна и, при корректной реализации, не требует пояснений:

Matrix a, b;

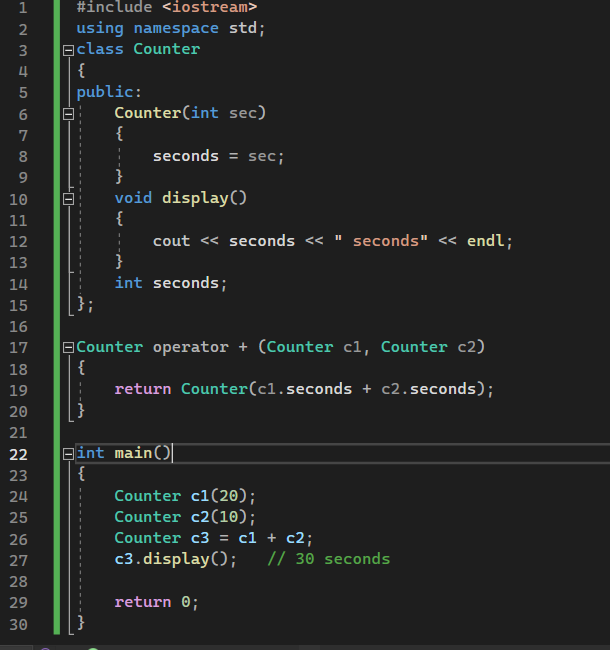
Matrix c = a + b;

Перегрузка операторов позволяет определить действия, которые будет выполнять оператор. Перегрузка подразумевает создание функции, название которой содержит слово operator и символ перегружаемого оператора. Функция оператора может быть определена как член класса, либо вне класса.

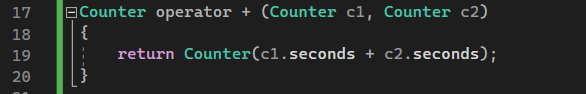
Перегрузить можно только те операторы, которые уже определены в C++. Создать новые операторы нельзя.

Если функция оператора определена как отдельная функция и не является членом класса, то количество параметров такой функции совпадает с количеством операндов оператора. Например, у функции, которая представляет унарный оператор, будет один параметр, а у функции, которая представляет бинарный оператор, - два параметра. Если оператор принимает два операнда, то первый операнд передается первому параметру функции, а второй операнд - второму параметру. При этом как минимум один из параметров должен представлять тип класса

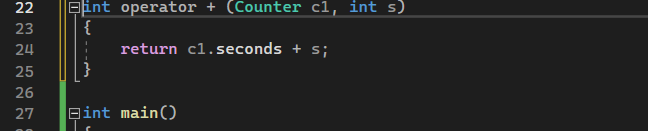
Рассмотрим пример с классом Counter, который представляет секундомер и хранит количество секунд:



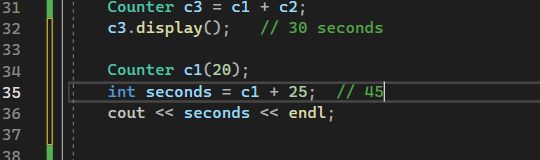
Здесь функция оператора не является частью класса Counter и определена вне его. Данная функция перегружает оператор сложения для типа Counter. Она является бинарной, поэтому принимает два параметра. В данном случае мы складываем два объекта Counter. Возвращает функция также объект Counter, который хранит общее количесто секунд. То есть по сути здесь операция сложения сводится к сложению секунд обоих объектов:



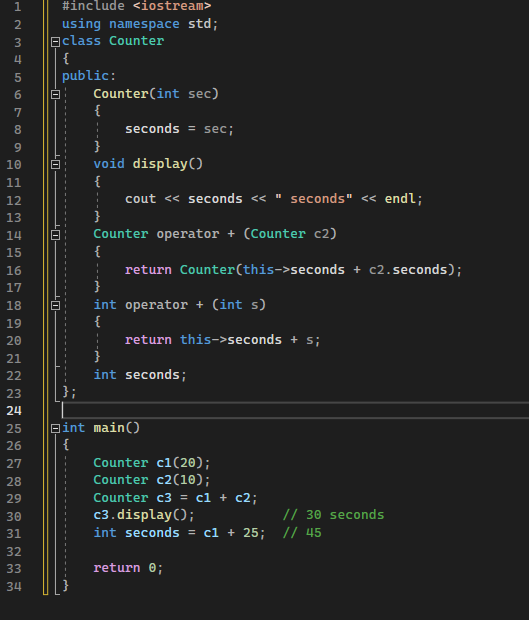
При этом необязательно возвращать объект класса. Это может быть и объект встроенного примитивного типа. И также мы можем определять дополнительные перегруженные функции операторов:



Данная версия складывает объект Counter с числом и возвращает также число. Поэтому левый операнд операции должен представлять тип Counter, а правый операнд - тип int. И, к примеру, мы можем применить данную версию оператора следующим образом:

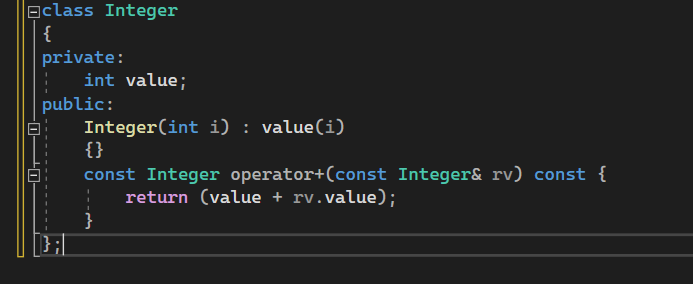


Также функции операторов могут быть определены как члены классов. Если функция оператора определена как член класса, то левый операнд доступен через указатель this и представляет текущий объект, а правый операнд передается в подобную функцию в качестве единственного параметра:



В данном случае к левому операнду в функциях операторов мы обращаемся через указатель this.

СИНТАКСИС ПЕРЕГРУЗКИ



В данном случае, оператор оформлен как член класса, аргумент определяет значение, находящееся в правой части оператора. Вообще, существует два основных способа перегрузки операторов: глобальные функции, дружественные для класса, или подставляемые функции самого класса. Какой способ, для какого оператора лучше, рассмотрим в конце топика.

В большинстве случаев, операторы (кроме условных) возвращают объект, или ссылку на тип, к которому относятся его аргументы (если типы разные, то вы сами решаете как интерпретировать результат вычисления оператора).

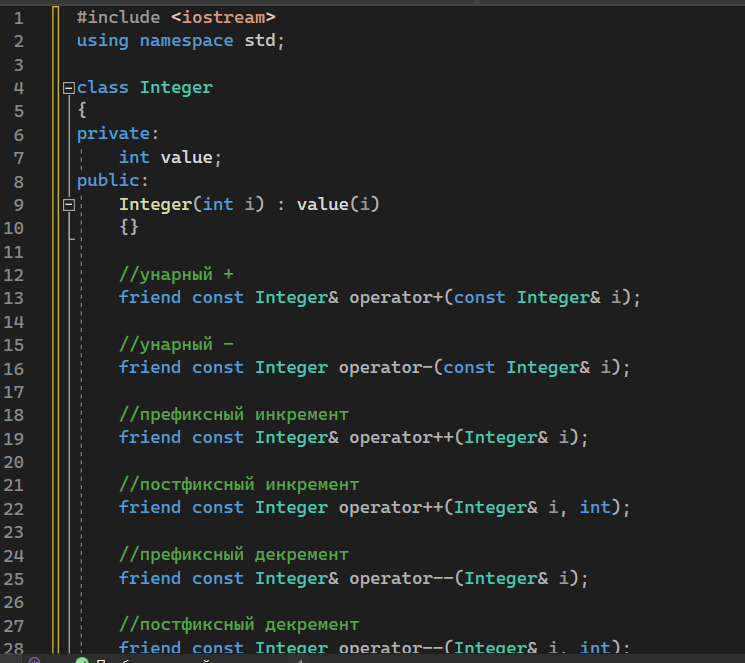
ПЕРЕГРУЗКА УНАРНЫХ ОПЕРАТОРОВ

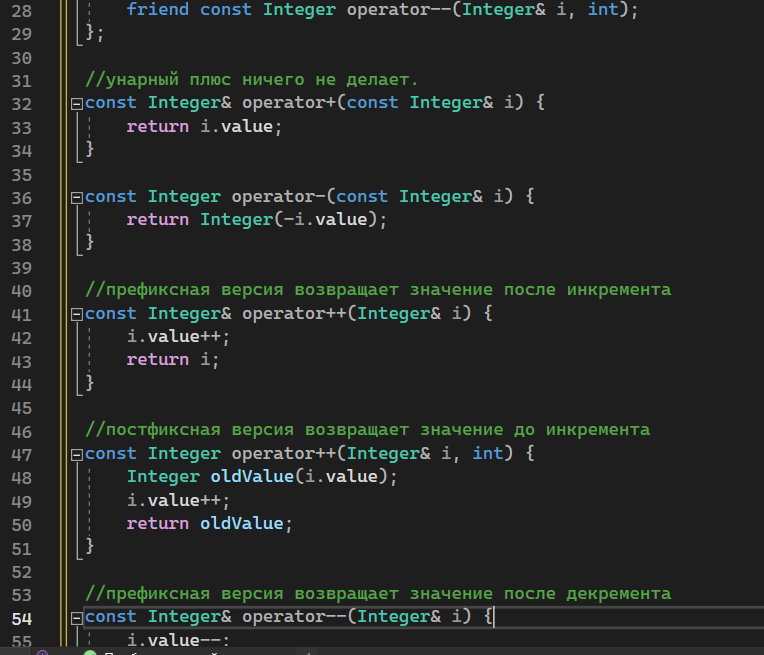
**Унарные операторы присваивания** — это операторы инкремента и декремента ( ++ и -- соответственно).

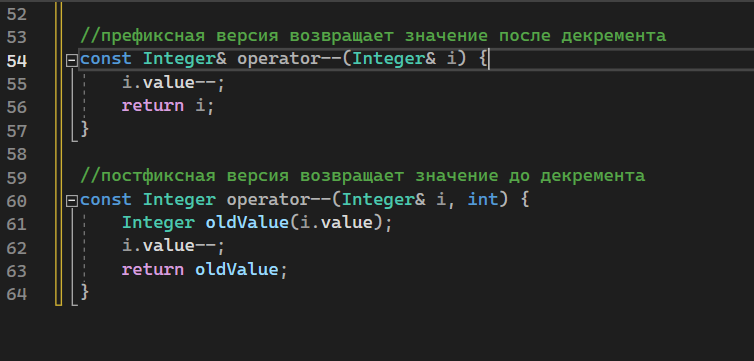
**Бинарные операторы присваивания** — это оператор простого присваивания ( = ) и составные операторы присваивания.

**Все составные** операторы присваивания состоят из другого бинарного оператора и оператора простого присваивания.

Рассмотрим примеры перегрузки унарных операторов для определенного выше класса Integer. Заодно определим их в виде дружественных функций и рассмотрим операторы декремента и инкремента:



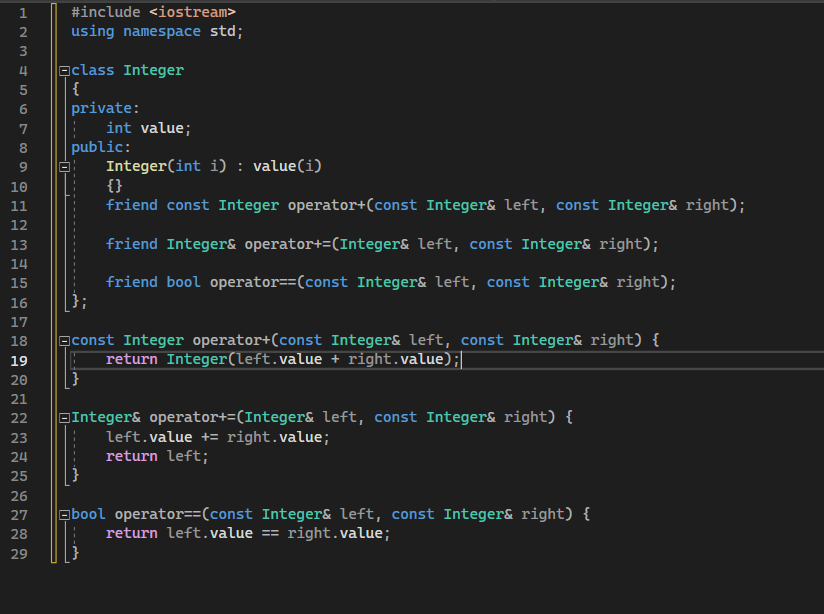




Теперь вы знаете, как компилятор различает префиксные и постфиксные версии декремента и инкремента. В случае, когда он видит выражение ++i, то вызывается функция operator++(a). Если же он видит i++, то вызывается operator++(a, int). То есть вызывается перегруженная функция operator++, и именно для этого используется фиктивный параметр int в постфиксной версии.

ПЕРЕГРУЗКА БИНАРНЫХ ОПЕРАТОРОВ

Рассмотрим синтаксис перегрузки бинарных операторов. Перегрузим один оператор, который возвращает l-значение, один условный оператор и один оператор, создающий новое значение (определим их глобально):



Во всех этих примерах операторы перегружаются для одного типа, однако, это необязательно. Можно, к примеру, перегрузить сложение нашего типа Integer и определенного по его подобию Float.

КАКИЕ ОПЕРАТОРЫ МОЖНО ПЕРЕГРУЖАТЬ?

Нельзя определить новый оператор, например, operator\*\*.

**Следующие операторы перегружать нельзя:**

?: (тернарный оператор);

:: (доступ к вложенным именам);

. (доступ к полям);

.\* (доступ к полям по указателю);

sizeof, typeid и операторы каста.

**Следующие операторы можно перегрузить только в качестве методов:**

= (присваивание);

-> (доступ к полям по указателю);

() (вызов функции);

[] (доступ по индексу);

->\* (доступ к указателю-на-поле по указателю);

операторы конверсии и управления памятью.

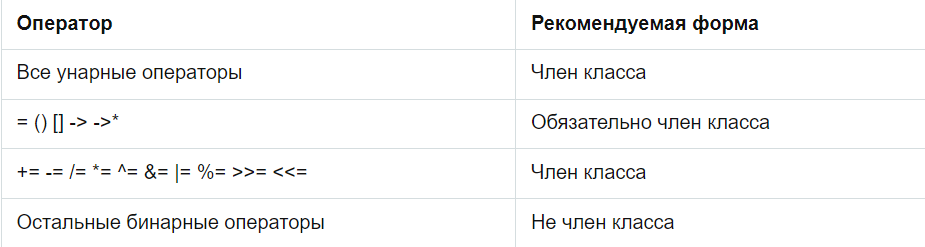
НЕПЕРЕГРУЖАЕМЫЕ ОПЕРАТОРЫ

Некоторые операторы в C++ не перегружаются в принципе. По всей видимости, это сделано из соображений безопасности.

* Оператор выбора члена класса ".".
* Оператор разыменования указателя на член класса ".\*"
* В С++ отсутствует оператор возведения в степень (как в Fortran) "\*\*".
* Запрещено определять свои операторы (возможны проблемы с определением приоритетов).
* Нельзя изменять приоритеты операторов

РЕКОМЕНДАЦИИ К ФОРМЕ ОПРЕДЕЛЕНИЯ ОПЕРАТОРОВ

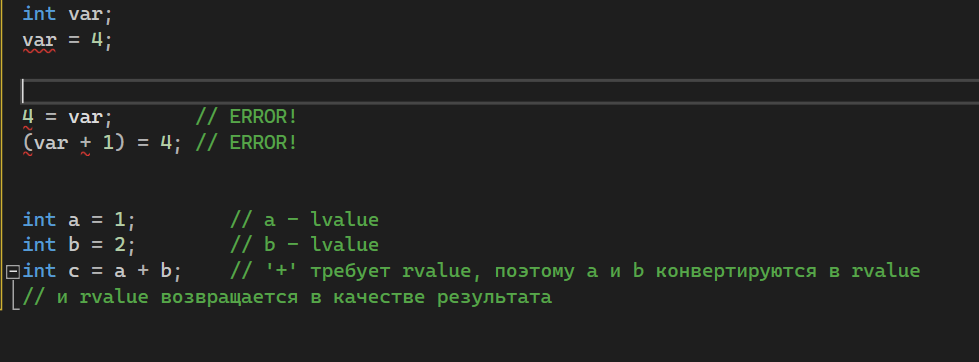
Как мы уже выяснили, существует два способа операторов — в виде функции класса и в виде дружественной глобальной функции.



R-VALUE, L-VALUE ВЫРАЖЕНИЯ

**lvalue (locator value)** представляет собой объект, который занимает идентифицируемое место в памяти (например, имеет адрес).

**rvalue** определено путём исключения, говоря, что любое выражение является либо lvalue, либо rvalue. Таким образом из определения lvalue следует, что rvalue — это выражение, которое не представляет собой объект, который занимает идентифицируемое место в памяти.



Ни константа 4, ни выражение var + 1 не являются lvalue

(что автоматически их делает rvalue). Они не lvalue, потому что оба являются временным результатом выражений, которые не имеют определённого места в памяти (то есть они могут находится в каких-нибудь временных регистрах на время вычислений). Таким образом, присваивание в данном случае не несёт в себе никакого семантического смысла. Иными словами — некуда присваивать.