Перегрузка операций

Перегрузка операций

- ***Перегрузка операций —***

это пример полиморфизма C++

Перегрузка операций

- C++ позволяет определять несколько функций с одинаковыми именами и разной сигнатурой (списками аргументов)
- Это называется ***перегрузкой функций*** или ***функциональным полиморфизмом.***
- Цель такой перегрузки — позволить использовать одно и то же имя функции для некоторой базовой операции, несмотря на то, что она применяется к данным разных типов.

Перегрузка операций

- ***Перегрузка операций*** расширяет концепцию перегрузки на операции, позволяя трактовать их множеством способов.
  На самом деле многие операции C++ (как и C) уже перегружены.

Перегрузка операций

- Язык C++ позволяет распространить перегрузку операций на пользовательские типы, разрешая, скажем, применять символ + для сложения двух объектов.
- Для выяснения, какое именно определение данной операции следует использовать, компилятор также использует количество и тип операндов.
- Перегруженные операции часто могут заставить код выглядеть более естественно.

Перегрузка операций

- Для перегрузки операции используется специальная форма функции, называемая ***функцией операции.***

Перегрузка операций

Функция операции имеет следующую форму, в которой ***op*** —это символ перегружаемой операции:

operator *op* (*список-аргументов)***

Перегрузка операций

Функция операции имеет следующую форму, в которой ***op*** —это символ перегружаемой операции:

operator *op* (*список аргументов)***

Перегрузка операций

Например,

operator+ ()

перегружает операцию +,

operator* ()

перегружает операцию *.

Перегрузка операций

Перегружаемая операция должна быть допустимой операцией C++, а не произвольным символом.

Например, объявить функцию operator@ ()

не получится, т.к. в C++ нет операции @

Таблица 1.
Операции, которые могут быть перегружены

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

Ограничения перегрузки

Перегруженные операции (за некоторыми исключениями) не обязательно должны быть функциями-членами.

Однако, по крайней мере, один из операндов должен иметь тип, определяемый пользователем.

Ограничения перегрузки

- **Перегруженные операции должны иметь как минимум один операнд типа, определяемого пользователем. Это предотвращает перегрузку операций, работающих со стандартными типами.**

  То есть переопределить операцию “минус” (-) так, чтобы она вычисляла сумму двух вещественных чисел вместо разности, не получится.

  Это ограничение сохраняет здравый смысл, заложенный в программу, хотя и несколько препятствует полету творчества.

Ограничения перегрузки

- **Вы не можете использовать операцию в такой манере, которая нарушает правила синтаксиса исходной операции.**

  Например, нельзя перегрузить операцию взятия модуля (%) так, чтобы она применялась с одним операндом:

```
int x;
Time shiva;
% x;          // не допускается для операции взятия модуля
% shiva;      // не допускается для перегруженной операции
```

Ограничения перегрузки

- Аналогично, не допускается изменение приоритетов операций.

  Поэтому, если вы перегрузите операцию сложения для класса, то новая операция будет иметь тот же приоритет, что и обычное сложение.

Ограничения перегрузки

- Вы не можете определять новые символы операций. Например, определить функцию operator** () для создания операции возведения в степень не получится.

Ограничения перегрузки

- **Нельзя перегружать следующие операции:**

Операция	Описание
`sizeof`	Операция `sizeof`
`.`	Операция членства
`.*`	Операция указателя на член
`::`	Операция разрешения контекста
`:?`	Условная операция
`typeid`	Операция RTTI (runtime type identification — определение типа во время выполнения)
`const_cast`	Операция приведения типа
`dynamic_cast`	Операция приведения типа
`reinterpret_cast`	Операция приведения типа
`static_cast`	Операция приведения типа

Ограничения перегрузки

- Большинство операций из табл.1 допускают перегрузку за счет использования как функций-членов, так и функций, не являющихся членами. Однако для перегрузки перечисленных ниже операций можно использовать ***только*** функции-члены:

Операция	Описание
=	Операция присваивания
()	Операция вызова функции
[]	Операция индексации
->	Операция доступа к членам класса через указатель

Ограничения перегрузки

- **В дополнение к этим формальным ограничениям при перегрузке операций вы должны использовать смысловые ограничения.**

friend

Существует три разновидности друзей:

- дружественные функции;
- дружественные классы;
- дружественные функции-члены.

friend

Объявляя функцию другом класса, вы позволяете ей иметь те же привилегии доступа, что и у функций-членов класса

friend

Функция, не являющаяся членом, не вызывается через объект.

Вместо этого все значения, которые она использует, включая объекты, передаются в виде явных аргументов.

Создание друзей

Первый шаг в создании дружественной функции предусматривает помещение прототипа в объявление класса и предварение его префиксом в виде ключевого слова **friend** :

```
friend Time operator*(double m, const Time & t); // размещается в объявлении класса
```

Создание друзей

На основе этого прототипа можно сделать два вывода:

- **Несмотря на то что функция o p e r a t o r * () присутствует в объявлении класса,**
она не является функцией-членом класса. Поэтому она не вызывается через операцию членства (.).

- **Несмотря на то что функция o p e r a t o r * () не является функцией-членом класса,**
она имеет те же права доступа, что и функции-члены.

Создание друзей

Второй шаг состоит в написании определения функции.

Поскольку она не является функцией-членом, добавлять квалификатор **Имя_класса: :** не нужно.

Кроме того, ключевое слово f r i e n d также не используется в определении

Создание друзей

Совет.

Если вы хотите перегрузить операцию для класса и применять ее с первым операндом, не являющимся объектом класса, можете использовать дружественную функцию для изменения порядка следования операндов.

Создание друзей

Дружественная функция класса — это функция, не являющаяся членом, которая имеет те же права доступа, что и функция-член.

Создание друзей

На заметку!

Ключевое слово **friend** используется только в прототипе, представленном в объявлении класса. В определении функции оно указывается, только если не присутствует в самом прототипе.

Перегруженные операции: сравнение функций-членов и функций, не являющихся членами

При реализации перегрузки многих операций имеется выбор между функциями-членами и функциями, не являющимися членами.

Как правило, версия, не являющаяся членом — это дружественная функция, в связи с чем она имеет доступ к закрытым данным класса.

Перегруженные операции: сравнение функций-членов и функций, не являющихся членами

На заметку!

Версия перегруженной операции с функцией, не являющейся членом, требует столько формальных параметров, сколько операндов есть у данной операции.

Версия с использованием функции-члена требует на один параметр меньше, поскольку один операнд передается не явно как вызывающий объект.

Перегруженные операции: сравнение функций-членов и функций, не являющихся членами

Итак, какую же форму стоит выбрать?

Как упоминалось ранее, для некоторых операций единственно правильным выбором будет функция-член.

В других случаях особой разницы между ними нет. Иногда, в зависимости от проектного решения, положенного в основу класса, вариант с использованием функции, не являющейся членом, может быть предпочтительнее, в частности, если для класса определены преобразования типов.

В C++ для классов доступны перечисленные ниже преобразования типов

- **Конструктор класса, имеющий один аргумент, служит инструкцией по преобразованию значения типа аргумента в тип класса.**

  Однако использование ключевого слова e x p l i c i t **в объявлении конструктора исключает неявные преобразования и разрешает только явные.**

В C++ для классов доступны перечисленные ниже преобразования типов

- **Специальная функция-член операции, называемая** *функцией преобразования*, **служит инструкцией для преобразования объекта класса в другой тип.** **Функция** преобразования является членом класса, не имеет типа возврата, не принимает аргументов и носит имя **operator** *имяТипа* () , где *имяТипа* —тип, в который преобразуется объект.

  Эта функция преобразования вызывается автоматически, когда вы присваиваете объект класса переменной соответствующего типа или применяете операцию приведения к этому типу.

Перегрузка операций

На заметку!

Поскольку перегрузка операций реализуется с помощью функций, одну и ту же операцию можно перегружать много раз, при условии, что каждая функция операции имеет отличную от других сигнатуру, и каждая функция операции поддерживает то же количество операндов, что и соответствующая встроенная операция C++.