Программирование на языке C++

Шаблов Анатолий
anatoliishablov@gmail.com

ИТМО, весенний семестр 2024

Неявные преобразования типов

Последовательность неявного преобразования типов

1. 0 или 1 цепочка стандартных преобразований.
2. 0 или 1 пользовательское преобразование.
3. 0 или 1 цепочка стандартных преобразований.

Цепочка стандартных преобразований:

1. 0 или 1 преобразование категории значения, преобразование массива или функции к указателю.
2. 0 или 1 числовое расширение или числовое преобразование.
3. 0 или 1 преобразование указателя на функцию (C++17)
4. 0 или 1 повышение квалификации.

Пользовательские преобразования

- Не explicit конструктор.

- Не explicit оператор приведения типа.

Перегрузка функций

Перегрузка функций

Допустимо иметь несколько функций с одним именем, если в каждом случае вызова функции с таким именем компилятор может найти один наиболее подходящий вариант.

Этапы выбора функции

- Поиск кандидатов – поиск сущностей по имени
- Выбор подходящих к контексту вызова
- Выбор единственного наилучшего кандидата

При этом в зависимости от контекста вызова ищутся:

- Контекст вызова функции – функции с таким именем
- Контекст функционального вызова объекта – члены-операторы функционального вызова соответствующего класса и его предков, а также операторы преобразования типа к типу подходящей функции
- Контекст вызова оператора – члены-операторы (если первый или единственный операнд имеет сложный тип), свободные операторы, встроенные операторы

Правила выбора подходящих к контексту вызова

Учитываются факторы:

- Число аргументов
- Типы аргументов
- Допустимая ссылочная связываемость

Ранжирование неявных преобразований типов

Каждому из 3 возможных этапов последовательности неявного преобразования назначается ранг:

1. Точное совпадение*
2. Числовое расширение
3. Преобразование

Ранг всей последовательности = наибольшему рангу этапов.

Отсутствие необходимости преобразования считается последовательностью нулевой длины с минимальным рангом.

Ранжирование неявных преобразований типов

Стандартное преобразование лучше пользовательского.

Чем короче необходимая цепочка стандартных преобразований – тем она лучше.

Связывание rvalue с rvalue ссылкой лучше связывания rvalue с const lvalue ссылкой.

Среди связываний lvalue с lvalue ссылкой лучше то, у которого меньше cv-квалификаторов.

Из двух последовательностей, содержащих пользовательское преобразование к одному типу, лучше та, у которой лучше третий этап.

Выбор единственного наилучшего кандидата

Все подходящие кандидаты попарно сравниваются и выбирается наилучший из них.

Несколько одинаково наилучших кандидатов – ошибка компиляции.

Один кандидат лучше другого, если требуемые неявные преобразования для всех его аргументов не хуже (имеют ранг не ниже) таковых для конкурента, а кроме того:

1. Для хотя бы одного аргумента требуемое неявное преобразование лучше, чем у конкурента.
2. Либо этот кандидат – не шаблонный, а конкурент – шаблонный.
3. Либо этот кандидат более шаблонно-специализированный, чем конкурент.

Друзья классов

Друзья классов

`friend` объявление: в теле класса можно объявить другой класс или функцию “другом”, что даст такому другу доступ ко всем членам класса (игнорируя спецификаторы доступа).

Отношение дружбы – не транзитивно, не наследуемо.

Возможные формы:

- Объявление функции (оператора)
- Определение функции (оператора)
- Предварительное объявление класса
- Объявление видимого класса / `union`

Перегрузка операторов

Перегрузка операторов

Перегруженные операторы – функции со специальным именем.

`operator op` – обычный оператор

`operator type` – пользовательское приведение типа

`operator new` / `new[]` / `delete` / `delete[]` – пользовательская аллокация / деаллокация

operator user-defined-string-literal – пользовательский литерал

Если в выражении один из операндов – сложный тип (класс, enum), то производится поиск перегруженной функции, реализующей данный оператор.

Ограничения перегрузки операторов

Нельзя перегружать некоторые операторы (например, ::)

Нельзя менять приоритет, ассоциативность операторов

Нельзя менять число аргументов операторов

Нельзя создавать новые операторы

Перегруженный оператор -> должен возвращать либо указатель, либо объект, для которого перегружен ->

Перегруженные операторы && и || не имеют ленивого вычисления аргументов (но сохраняют порядок вычисления аргументов)*