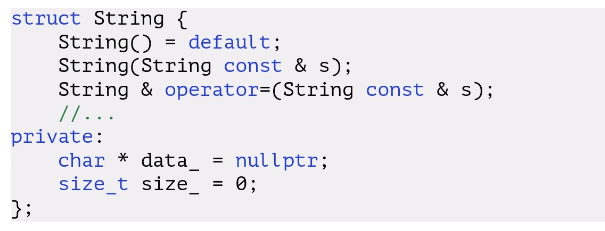
РЕФЕРАТ

Семантика перемещения. Перемещающий конструктор. Перемещающее присваивание

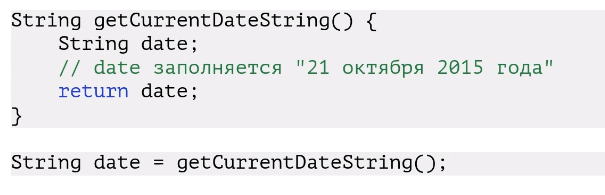
1. Проблема: излишнее копирование

Копирование объекта часто является дорогой операцией. В качестве демонстрации можно привести простой класс string (Прим.1.1). Чтобы скопировать объект этого типа необходимо выделить буфер нужного размера в новом объекте и посимвольно скопировать саму строку. Аналогично для оператора присваивания, только буфер сначала придётся очистить. Т.е. копирование – дорогая операция (пропорциональна длине строки).



Прим.1.1

Пусть теперь есть функция, которая возращает строчку с датой. (Прим. 1.2)



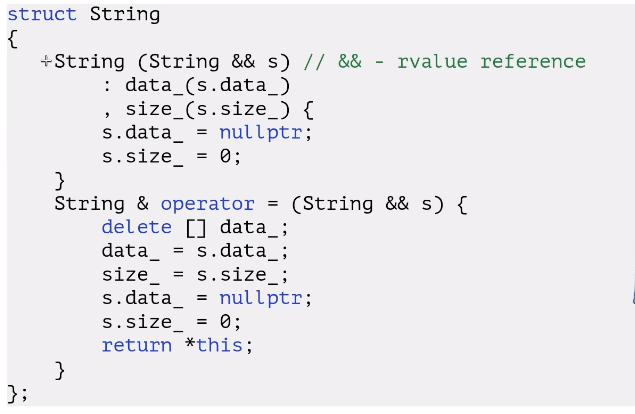
Прим. 1.2

При вызове функции, после заполнения переменная date будут скопирована в место зарезервированное для возращаемого значения. Далее, чтобы сохранить полученное значение, создадим (уже вне функции) переменную date и проинициализируем её значением возращаемым из функции. Т.е. ещё один вызов конструктора копирования. Итого, два ненужных копирования.

В стандарте C++11 есть специальный механизм, позволяющий избежать такого лишнего копирования.

1. Перемещающий конструктор и перемещающий оператор присваивания

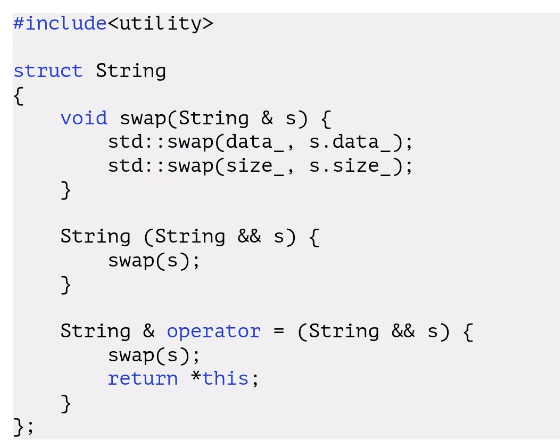
В стандарте появились два новых метода: перемещающий конструктор и перемещающий оператор присваивания. Их сигнатуры можно посмотреть на примере нашей строчки (прим. 2).



Прим. 2.1

В чём отличие от контруктора копирования? В контруктора копирования объект передаётся по константной ссылке, т.е. его изменять нельзя. В перемещающем конструкторе же – по неконстантной ссылке нового типа (rvalue reference), которая задаётся &&. Получается мы можем менять исходный объект, но его необходимо оставить в согласованном состаянии. Т.е. в перемещающем конструкторе не обязательно копировать данные, а можно просто “оотбрать” их у исходного объекта. В случае в нашей строчкой, это делатся легко: нужно скорировать указатель на буфер и скопировать размер буфера. А исходный объект остаётся в состоянии пустой строки. Выгода, по сравнению с коструктором копирования, очевидна – дорогое посимвольное копирования заменилось на копирования указателя. В перемещающем операторе присваивания ситуация аналогичная: опять же, сначала нужно удалить текущий буфер, затем отобрать данные.

Ниже приведёт более лаконичный пример реализации перемещающего конструктора и оператора присваивания с использованием метода swap. Причём эта реализации немного отличается в работе. В предыдущем примере после перемещения исходный объект остаётся в состоянии пустой строки. А данном случае объекты обмениваются данными, а исходная строка после перемещение может не быть пустой и будет висеть в какое-то время в памяти. Тут важно, чтобы при объявлении строки у неё было какое-то значение по умолчанию.

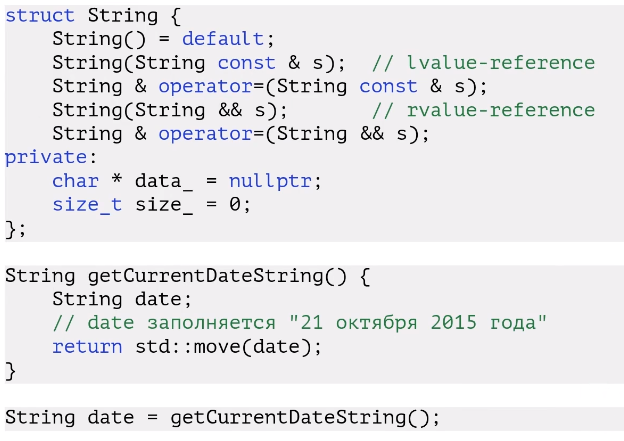


Прим.2.2

1. Использование перемещения

Дополним пример 1 перемещающим конструктором и перемещающим оператором присваивания (прим.3). Теперь вместо двух вызовов конструктора копирования будет вызываться перемещающий конструктор.

Итого, дорогое посимвольное копирование (линейная сложность) заменилось на дешёвое копирование указателя и размера буфера (константная сложность).



Прим. 3

Стоить заметить, что в данном примере не обязательно явно прописывать std::move(), т.к. в функции по значению возвращается локальный объект data.

Когда используется перемещающие методы:

* если передавать объект при помощи std::move()
* если передавать временный объект
* если из функции по значению возвращается локальный объект функции