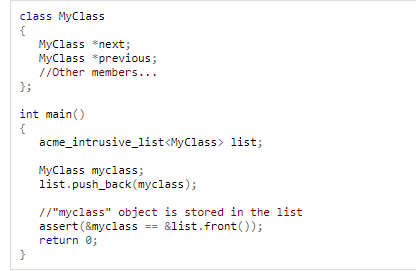
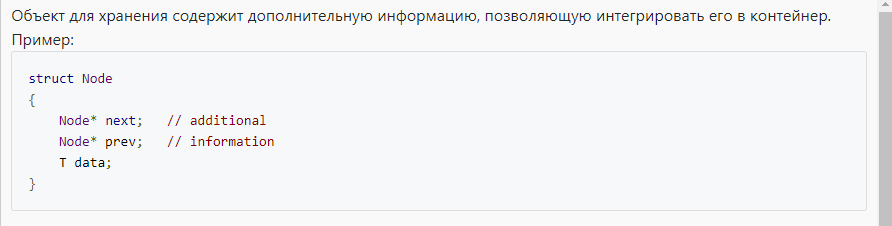
Основное различие между интрузивным контейнерами и неинтрузивными контейнерами заключается в том, что в C ++ неинтрузивные контейнеры хранят **копии** значений, переданных пользователем. Контейнеры используют Allocator параметр шаблона для размещения сохраненных значений.

Интрузивный контейнер не хранит копии переданных объектов, а хранит сами объекты. Дополнительные данные, необходимые для вставки объекта в контейнер, должны быть предоставлены самим объектом. Например, чтобы вставить MyClass интрузивный контейнер, реализующий связанный список, он MyClass должен содержать необходимые следующие и предыдущие указатели:





К примеру, интрузивный список может хранить указатель на начало и конец. А указатель на следующий/предыдущий элемент будет хранить сам "элемент".

У интрузивных контейнеров есть несколько важных преимуществ:

* Работа с интрузивными контейнерами вообще не требует никакого управления памятью. Накладные расходы времени и размера, связанные с динамической памятью, можно свести к минимуму.
* Один и тот же объект может быть вставлен в несколько контейнеров одновременно с небольшими накладными расходами в размере объекта.
* Итерация интрузивного контейнера требует меньше обращений к памяти, чем семантически эквивалентный контейнер указателей: итерация выполняется быстрее.
* Интрузивные контейнеры предлагают лучшие гарантии исключения, чем неинтрузивные контейнеры. В некоторых ситуациях интрузивные контейнеры предлагают гарантию no-throw, чего нельзя добиться с неинтрузивными контейнерами. (no-throw guarantee - данная гарантия означает, что операция, которую мы собираемся выполнить, не бросит исключение.
* Интрузивные контейнеры обеспечивают предсказуемость при вставке и стирании объектов, поскольку управление памятью не осуществляется с помощью интрузивных контейнеров. Управление памятью обычно не является предсказуемой операцией, поэтому гарантии сложности от неинтрузивных контейнеров более слабые, чем гарантии, предлагаемые интрузивными контейнерами.

У интрузивных контейнеров есть и недостатки:

* Каждый тип, хранящийся во второстепенном контейнере, требует дополнительной памяти, содержащей информацию об обслуживании, необходимую контейнеру. Следовательно, всякий раз, когда определенный тип будет храниться во второстепенном контейнере, **вы должны соответствующим образом изменить определение этого типа** .
* В интрузивных контейнерах вы не храните копию объекта, **а скорее исходный объект связан с другими объектами в контейнере** . Объекты не нуждаются в конструкторах-копиях или операторах присваивания для хранения во интрузивных контейнерах. Но вы должны позаботиться о возможных побочных эффектах всякий раз, когда вы меняете содержимое объекта (это особенно важно для ассоциативных контейнеров).
* Пользователь **должен управлять временем жизни вставленных объектов** независимо от контейнеров.
* Опять же, вы должны быть **осторожны** : в отличие от контейнеров STL **легко сделать итератор недействительным,** не касаясь напрямую интрузивного контейнера, потому что объект можно удалить до того, как он будет удален из контейнера.
* Анализировать безопасность потоков программы, использующей контейнеры, сложнее с навязчивыми контейнерами, потому что контейнер может быть изменен косвенно, без явного обращения к члену контейнера.