4.30 Проблемы, приводящие к идее автоматического вывода типов (auto) при объявлениях. Примеры, когда auto необходимо и когда неуместно. Ключевое слово decltype. Особенности поведения auto и decltype от ссылочных типов. Особенности auto в возвращаемом типе функции. Особенности поведения decltype от lvalue, xvalue и prvalue. Конструкция decltype(auto) и пример, когда она нужна.

Мотивировка auto

Прописывать тип целиком неудобно, код становится нечитаемым.

Решение этой проблемы - ключевое слово auto, которое сообщает компилятору, что тип переменной должен быть установлен исходя из типа инциализируемого значения.

Оно работает почти так же как вывод шаблонного типа, компилятор смотрит на тип выражения и подставляет нужный на этапе компиляции. auto работает по тем же правилам, то есть если исходный тип должен быть ссылкой, то и auto надо писать с ссылкой, аналогично с константностью. auto следует понимать аналогично T - это универсальная ссылка.

Примеры, когда auto уместно:

```
int main() {
    unordered_mao<string, int, hash<string>, equal_to<string>, FastAllocator
   <pair<string, int>> m;
    auto mm = m;
    return 0;
}
```

Забыв const, которое закомичено, мы получим копирование на каждом шаге, auto позволяет этого избежать:

```
#include <iostream>

int main() {
    for(const pair</*const*/string, int>& item: mm) {}
    for(auto it = begin(); it != end(); ++it) {}
    for(const auto& item: mm) {}
    //или
    return 0;
}
```

В данном случае это проблема, потому что map по реализации хранит const ключ.

Примеры, когда auto неуместно:

Когда вы пишете auto, вы предполагаете, что следующие строчки эквивалентны:

```
std::vector<bool> v(5, true);
bool bit = v[2]; // явно достаем значение из вектора в bool
auto bit = v[2]; // UB - получим тип std::vector<bool>::bit_reference
```

Но это не так. В коде, использующем auto, тип bit больше не является bool. Хотя концептуально vector<bool> хранит значения bool, operator [] у std::vector<bool> не возвращает ссылку на элемент контейнера (то, что std::vector::operator[] возвращает для всех типов за исключением bool). Вместо этого возвращается объект типа std::vector<bool>::reference (класса, вложенного в std::vector<bool>).

Ключевое слово decltype

В компайл-тайм возвращает тип выражения. Хорош тем, что не отбрасывает &&, &, *, const/volatile(ключевое слово, которое говорит компилятору не оптимизировать), как это делается при принятии аргументов в функцию. С помощью decltype можно отличить ссылку на объект от исходного объекта. На decltype тоже можно навесить модификаторы типа, при навешивании ссылок на decltype, в которм уже есть ссылки произойдет сворачивание ссылок.

Выражение внутри decltype никогда не выполняются, только оценивается их тип, так как это все происходит на этапе компиляции.

```
int x = 7;
decltype(x++) u = x;
```

Если внутри decltype выражение типа prvalue типа T, то тип declype(expression) - T
Если внутри decltype выражение типа xvalue типа T, то тип declype(expression) - T&&
Если внутри decltype выражение типа lvalue типа T, то тип declype(expression) - T&

Type deduction for return type

1. auto не может использоваться в аргументах функции
2. auto не может использоваться в качестве возвращаемого значения функции, если в зависимости от работы функции возвращаются вещи разных типов

```
auto f(int& x){
    if (x > 5) return x;
    else return 0.0;
} // CE, inconsisent deduction
```

Другой пример: пусть функция возвращает функцию. Функция $f(x)$ может возвращать как по ссылке так и по значению, и мы не знаем этого заранее. Написать auto(f(x)) нельзя, так как на данном этапе x еще не определен. Решение проблемы: trailing return type:

```
auto g(int& x) -> decltype(f(x)){
    return f(x);
}
```

С C++14 возможен вот такой синакисис, который говорит компилятору: выведи тип самостоятельно, но не по правилам auto, а по правилам decltype:

```
template <typename Container>
decltype(auto) g(const Container& cont, size_t index)){
    std::cout << "...";
    return cont[index];
}
```