

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 инициализируемого 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:

```
1 int main() {
2     unordered_map<string, int, hash<string>, equal_to<string>, FastAllocator
3     <pair<string, int>> m;
4     auto mm = m;
5     return 0;
6 }
```

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

```
1 #include <iostream>
2
3 int main() {
4     for(const pair</*const*/string, int>& item: mm) {}
5     for(const auto item: mm) {}
6     return 0;
7 }
```

Примеры, когда auto неуместно:

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

```
1 vector<bool>(const Widget& w);
2 bool bit = features(w)[5];
3 auto bit = features(w)[5];
```

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`, 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 котором уже есть ссылки произойдет сворачивание ссылок.

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

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

Если внутри `decltype` выражение типа `rvalue` типа `T`, то тип `decltype(expression)` - `T`

Если внутри `decltype` выражение типа `xvalue` типа `T`, то тип `decltype(expression)` - `T&&`

Если внутри `decltype` выражение типа `lvalue` типа `T`, то тип `decltype(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

```
1  auto f(int& x){
2      if (x > 5) return x;
3      else return 0.0;
4  } // 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:

```
1  auto g(int& x) -> decltype(f(x)){
2      return f(x);
3  }
```

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

```
1  template <typename Container>
2  decltype(auto) g(const Container& cont, size_t index){
3      std::cout << "...";
4      return cont[index];
5  }
```