auto

Позволяет не указывать тип переменной явно, говоря компилятору, чтобы он сам определил фактический тип переменной, на основе типа инициализируемого значения.

```
auto i = 42;          // i - int
auto f = 42.f;        // f - float
auto p = new foo(); // p - foo*
```

auto

```
auto x1 = 27;
auto x2(27);

auto x3 = {27};
auto x4{27};
```

```
auto x5 = {1, 2, 3.0};
```

Ошибка. Невозможно вывести T для initializer_list

```
// тип int, значение 27
auto x1 = 27;

// тип int, значение 27
auto x2(27);

//std::initializer_list<int>, значение 27
auto x3 = {27};

//std::initializer_list<int>, значение 27
auto x4{27};
```

initializer_list

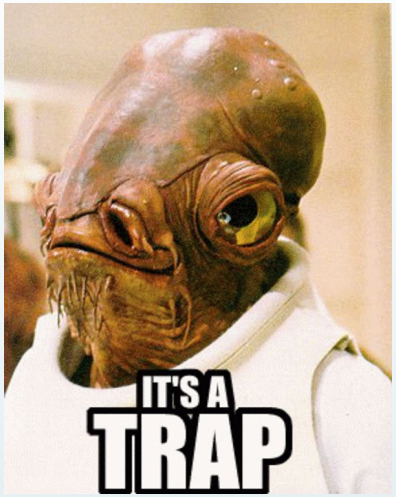
Контейнер предоставляющий доступ к массиву объектов типа T.

```
template< class T >
class initializer_list;
```

Что произойдет

```
template<typename T>
void f(T param)
{
    //реализация
    return;
}

int main()
{
    f(27, 4, 3);
    return 0;
}
```

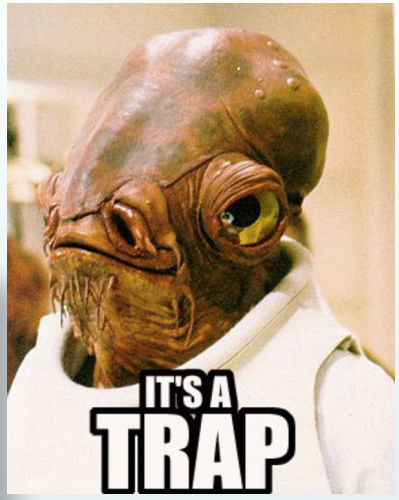


ошибка вывода типа для T

Что произойдет

```
template<typename T>
void f(std::initializer_list<T> param)
{
    //реализация
    return;
}

int main()
{
    f({27, 4, 3});
    return 0;
}
```



auto

- Возвращаемое значение не может быть auto.
- Можно использовать auto вместо типа возвращаемого значения функции
  - auto не говорит компилятору, что он должен определить тип, он только дает ему команду искать возвращаемый тип в конце функции.

В примере ниже, возвращаемый тип функции compose — это возвращаемый тип оператора +, который суммирует значения типа T и E.

```
template <typename T, typename E>
// decltype - позволяет определить тип на
основе входного параметра
auto compose(T a, E b) -> decltype(a+b)
{
   return a+b;
}
auto c = compose(2, 3.14); // c - double
```

range-based циклы

В C++11 была добавлена поддержка парадигмы foreach для итерации по набору. В новой форме возможно выполнять итерации в случае, если для объекта итерации перегружены методы begin() и end().

Это полезно, когда вы просто хотите получить элементы массива/контейнера или сделать с ними что-то, не заботясь об индексах, итераторах или количестве элементов.

range-based циклы

```
vector<int> v1 = {1, 2 ,3};
for (vector<int>::iterator it = v1.begin(); it != v1.end(); it++)
    cout << *it << " ";

cout << endl;
```

```
for (auto it = v1.begin(); it != v1.end(); it++)
    cout << *it << " ";

cout << endl;
```

```
   for (auto &it:v1)
       cout << it << " ";

    cout << endl;
```