1 слайд

Auto в  С++11

В С++11 auto позволяет не указывать тип переменной явно, говоря компилятору, чтобы он сам определил фактический тип переменной, на основе типа инициализируемого значения.

**int** main{

auto i = 10;  *//из значения 10 выводится тип int, который паяется переменной i*

auto d = 3.14; *//из значения 3.14 выводится тип double, который паяется переменной d*

auto ld = 3.14L; *//из значения 3.14 выводится тип long double, который паяется переменной ld*

auto S = "c\_string";*//из значения c\_string выводится тип const char\*, который паяется переменной S*

}

2 слайд

Auto до C++11

Ключевое слово auto означает, что переменная находится в automatic storage и время жизни такой переменной local lifetime. Другими словами, мы указывали, что данная переменная лежит в стеке, но так как все переменные созданные в функциях как

int a = 10;

уже и так подразумевается, что они стековые - то это ключевое слово бессмысленное.

————————————

auto – это старое ключевое слово C, обозначающее “локальную область видимости”. auto a эквивалентно auto int a, а в силу того, что в данном примере область видимости ограничена блоком, в котором объявлена переменная, то это то же самое, что и просто int a.

Автоматические объекты ("объекты на стеке", впрочем, **в** **C++** нет такого понятия) - это объекты, имеющие **automatic** **storage** duration. Они создаются, инициализируются и уничтожаются при входе и выходе из блока.

3 слайд

Подобно обычному объявлению, где тип указывается один раз для некоторого множества однотипных переменных, можно указывать auto, с тем условием, что типы должны быть совместимы друг с другом:

int main(){

auto i = 10, \*p = &i; // для i выводится int;    для \*p выводится int

auto x = 2, y = 2.; //  ошибка, типы не совместимы: выводится int и double

}

4 слайд

Auto гораздо более удобен в типах, которые или сложно определить самостоятельно, или долго писать, как например итератор вектора.

Здесь, в коде, для переменной pv используется простой тип, но писать такое бывает утомительно, использование auto делает код более лаконичным и приятным для глаз.

std::vector<double> scores;

std::vector<double>::iterator pv = scores.begin(); //утомительная часть

std::vector<double> scores;

auto pv = scores.begin ();   //cтрочка стала понятнее

5 слайд

Тип выводимый auto не всегда точно совпадает с типом инициализатора, иногда компилятор подправляет тип так, чтобы тип соответствовал обычным правилам инициализации. Вот, например, пример для ссылки:

#include <iostream>

using namespace std;

int main(){

int x = 100;

int y = 200;

int &rx = x;       //rx псевдоним x

auto a = rx;       //из rx выводится int, хотя сам rx - ссылка на int

    a = y;

    cout << x <<       '\n';  //Пошло различие

    cout << a <<        '\n';

}

Этот пример достаточно простой, поэтому понять его можно относительно быстро. Можно ожидать, что с помощью auto выведется ссылочный тип, но выводится совсем не ссылочный тип, а обычный тип.

6 слайд

Одна из запутанных тем — это указатели на константы и константные указатели. Суть в том, что в указателе на константу можно менять сам указатель, но не значение, на которое он указывает, а в константном указателе можно менять значение, но не сам указатель. Так вот, к чему я это говорю, const разделяют на const верхнего уровня и на const нижнего уровня. Указатель — это самостоятельный объект, а то, на что он указывает, это его косвенная составляющая. Если квалификатор const применяется к самому указателю, то такой const называют const верхнего уровня, если квалификатор const применяется для косвенной составляющей, то такой const называют const нижнего уровня.

Возвращаемся к auto .Auto обычно откидывает const верхнего уровня.

#include <iostream>

using namespace std;

int main(){

    int i = 500;

const int ci = i, &cr = ci;

auto b = ci; // b - int (const верхнего уровня в ci отброшен)

b++;  //b свободно меняется

// ci++; //ci менять нельзя

auto c = cr; //с - int (cr - псевдоним для ci c const верхнего уровня)

// cr++;   //нельзя изменить const int

    auto d = &i; //d - int\* (присваивание адреса целочисленного, поэтому d - указатель на int)

    cout << \*d << '\n';

    auto e = &ci;   //e - указатель на константный int (const нижнего уровня не отбрасывается)

    e++; //Сам указатель свободно меняется

    (\*e)++; //Но его косвенная сущность неизменяема

}

Спецификатор auto отбрасывает такой const, который применяется к самому указателю. Вот и весь секрет примера.

7 слайд

Auto допускает применение к себе ключевого слова const. Поэтому, чтобы заявить компилятору о своих претензиях на неизменяемость переменной, тип которой выводится с помощью auto, перед auto нужно поставить const квалификатор:

#include <iostream>

using namespace std;

int main(){

    int i = 300;

    int \*p = &i;

    const auto ptr = p;

    ptr++; //указатель, выведенный спецификатором auto приобрёл маркер неизминеяемости

}

 Так же если нужно чтобы выводимый тип был ссылочным типом пишем амперсант после auto:

#include <iostream>

using namespace std;

int main(){

    int i = 300;

    int &p = i;

    auto &a = p;

    a = 500;

    cout << i << '\n'; //В i 500

}

8  слайд

В некоторых случаях использование auto может помочь избежать прячущейся проблемы. Трудно привести пример, но я нашел один.

#include <vector>

int main(){

  vector<int> v;

   //вектор заполняется каким-либо образом

  unsigned sz = v.size();        //запоминание размера вектора в переменную sz типа unsigned int

}

Изучая документацию, можно видеть, что size() возвращает беззнаковое целое:

Что-нибудь эдакое: Member type size\_type is an unsigned integral type. — в двух словах здесь говориться об unsigned int. Поэтому многие разработчики, опираясь на подобную информацию, считают, что код, написанный в показанном листинге вполне нормальный. Этот код нормальный до тех пор, пока программа работает в операционной системе, где разные типы совпадают свойствами вместимости. При переносе программы на другую систему возможно начало проблемы. В 32-х разрядных Windows тип v.size() и unsigned int эквивалентны, но на 64-х разрядных windows типы различаются: v.size() возвращает тип, который способен хранить 64 бита, а unsigned int способен хранить 32 бита. Двумя словами — рассогласования вместимости.

9 слайд

**В 14 стандарте С++ аргументами  лямбда функции могут быть auto**

auto result = [](**const** auto& p1, **const** auto &p2){ **return** \*p1 < \*p2; };

Стоить отметить, что возвращаемое значение не может быть auto. Однако, вы можете использовать auto вместо типа возвращаемого значения функции. В таком случае, 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>**

**auto compose(T a, E b) -> decltype(a+b)       // decltype - позволяет определить тип на основе входного параметра**

**{**

**return a+b;**

**}**

**auto c = compose(2, 3.14);                    // c - double**

**Конец**

Переменная auto должна быть объязательно проинициализирована

**Cторонники**: есть типы данных в С++, которые портят читаемость когда своей длинной (про итераторы std::vector, к примеру) и хотелось бы писать меньше. Для modern C++ в условиях метапрограммирования возможность возврата типа auto делает шаблон гибким.

**Противники**: тип auto бьет по читабельности кода. Приходится гадать, что за переменная и делать лишнее действие в IDE наводя мышкой, чтобы понять что за тип. Такого рода "динамический тип" встает в разрез определению, что С++ строготипизированный язык.

Я же за использование auto в меру. Не надо впадаться в крайности.