Преобразование типов

11 июня 2018 г.

int a=5, b=8;
double c=b/a
c=1.0
Избежать можно заранее приведя типы (в скобочках) или умножив на 1.0.

С операторами есть много незаконных преобразований типов.

Пример:

char a='1';
int c = *((int*) a)

В этом случае мы берем не одну ячейку, а 4, потому что инт занимает 4 бита. Но мы не можем быть уверены, что у нас есть доступ в эти ячейки.

Class A
double b;
char c;

Нам нужно: обратиться и изменить объект класса не из дружественной функции, а откуда-нибудь. (A*Q или A Q)

У нас есть указатель на начало объекта. Если мы сделаем указатель на начало объекта, присвоим ему значение int, изменим это значение и разименуем, то мы изменим значение:

a (*((int*) Q) = 2).

Чтобы перейти к значению double:

((double*) Q) + 1) =2.0

Есть знак pragma, который позволяет управлять как временем, так и памятью.

Чтобы перейти к значению чар:

(((char*) Q) + 16) = '2' – самое простое.

((char*) ((double*) Q +2)) = '2'.

О приведении типов уже говорилось ранее в разделе наследования.

Class A
virtual f / ...
Class B: A
f ()

...
B temp
A* ptr-temp = temp
ptr-temp -> f ();

Можем не задумываться, для каких типов мы вызываем функцию f.

Int a=5;
long long =a;
f (int a)
f (long a)

long a = 42;
f(a)

Вызовется функция от int, т.е. функция отработает не так, как мы ожидаем.

Вырожденное преобразование типов (касты)

static-cast
dynamic-cast
reinterpret-cast
const-cast

Все это – шаблонизируемые функции.

Статическое преобразование:
int a = 1000;
Char c = static-cast <char> (a);

Бонусы:
1. Контроль. Мы сами управляем тем, что и во что преобразовывается.
2. Компилятор сам проверяет, что у нас было получено f (static-cast <long> (146))
3. Когда код большой это можно использовать для самоконтроля и поддержки самого кода.

Динамическая типизация:
Приведение и корректность проверяется и происходит во время исполнения, в то время как у статического преобразования все это происходит во время компиляции.

Пример:
Пусть есть объект, конструктор для которого не отработал. Например наследование виртуальной функции (смотри ранее)

void foo (A a)
try
B b = dynamic$_c ast < b > (a)()$
$b.f()$;

Catch const std: bad$_c aste$

Приведение ссылок проверит, объектом какого класса является объект a.

Class A
foo ();

Class B: A
bar ();

В таком случае приведение выкинет исключение конкретного вида, которое мы можем поймать:
Catch const std: bad$_c aste$

В принципе, без кастов всегда можно обойтись. Эта некоторая заплатка, которая помогает оптимизировать код, и ее всегда видно, потому что она довольно громоздкая.

Константное преобразование.
Они помогаю подставить/убрать const там, где нам надо. (По возможности не использовать)
Интерпретированное преобразование.
reinterpret
Каламбур типизации. Когда мы делаем совершенно незаконное преобразование типов.