Преобразование типов

13 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.
  Преобразование между связанными значениями.
  Связанность проверяется на этапе компиляции, поэтому и называется static.
  Типы, к котрым применим static_cast:
  - числовые
  `static_cast<int>(5.5);`
  - типы, связанные наследованием
  `B * b = ...;A * a = static_cast<A *>(b);b = static_cast<B *>(a);`
  - пользовательские преобразования
  `static_cast<string>("Hello");`
  - к void *
  `struct A; struct B; struct D; A*a=...; B*b=static_cast<B*>(a); B* b=(B*)a;`

- dynamic_cast/
  Осуществляет безопасное преобразование указатели или ссылки на базовый класс в указатель или сссылку на произвольный класс.
  Используется для:
  - Приведения типов
  `Circle*c=dynamic_cast<Circle>(a)`
  - Преобразования указателей
  `A* a=newC(); B*b=dynamic_cast<B*>(a);`
  - Преобразованиея ссылок
  `Bb=dynamic_cast<B>(*a);`

- reinterpret_cast
  Приводит друг к другу указатели, которые друг от друга не зависят, не меняя константности

Оператор reinterpret_cast является жестко машинно-зависимым. Чтобы безопасно ипользовать оператор reinterpret_cast, следует хорошо понимать, как именно реализованы используемые типы, а также то, как компилятор осуществляет приведение

Пример:
int* p=...; double*d=reinterpret_cast<int*>(p);

- const_cast
  Позволяет убрать или добавить константность.
  A const* ac=...; A* a=const_cast<A*>(ac);

  const_cast не применим:
  - Для тех объектов, которые объявлены как const, нельзы отменять константность. Будет undefined behaviour.

  - Если в функцию, например, передана константная ссылка, то внутри мы можем снять константность, хотя это не хорошо.

  - Для приведения констант применяется лишь оператор const_cast. Применение любого другого оператора приведения типов в данном случае привело бы к ошибке при компиляции. Аналогично, ошибка при компиляции произойдет в случае использования оператора const_cast в записи, которая осуществляет любые другие преобразования типов, отличные от создания или удаления константности

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 на-

следование 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.