Преобразования базовых типов данных

При рассмотрении типов данных указывалось, какие значения может иметь тот или иной тип и сколько байт памяти он может занимать. В прошлой теме были расмотрены арифметические операции. Теперь применим операцию сложения к данным разных типов:

```
byte a = 4;
int b = a + 70;
```

Результатом операции вполне справедливо является число 74, как и ожидается.

Но теперь попробуем применить сложение к двум объектам типа **byte**:

```
byte a = 4;
byte b = a + 70;  // ошибка
```

Здесь поменялся только тип переменной, которая получает результат сложения - с int на byte. Однако при попытке скомпилировать программу мы получим ошибку на этапе компиляции. И если мы работаем в Visual Studio, среда подчеркнет вторую строку красной волнистой линией, указывая, что в ней ошибка.

При операциях мы должны учитывать диапазон значений, которые может хранить тот или иной тип. Но в данном случае число 74, которое мы ожидаем получить, вполне укладывается в диапазон значений типа byte, тем не менее мы получаем ошибку.

Дело в том, что операция сложения (да и вычитания) возвращает значение типа int, если в операции участвуют целочисленные типы данных с разрядностью меньше или равно int (то есть типы byte, short, int). Поэтому результатом операции `a + 70` будет объект, который имеет длину в памяти 4 байта. Затем этот объект мы пытаемся присвоить переменной b, которая имеет тип byte и в памяти занимает 1 байт.

И чтобы выйти из этой ситуации, необходимо применить операцию преобразования типов. **Операция преобразования** типов предполагает указание в скобках того типа, к которому надо преобразовать значение:

```
(тип_данных_в_который_надо_преобразовать)значение_для_преобразования;
```

Так, изменим предыдущий пример, применив операцию преобразования типов:

```
byte a = 4;
byte b = (byte)(a + 70);
```

Сужающие и расширяющие преобразования

Преобразования могут быть сужающие (narrowing) и расширяющие (widening). Расширяющие преобразования расширяют размер объекта в памяти. Например:

```
byte a = 4;              // 0000100
ushort b = a;    // 0000000000000100
```

В данном случае переменной типа ushort присваивается значение типа byte. Тип byte занимает 1 байт (8 бит), и значение переменной a в двоичном виде можно представить как:

```
00000100
```

Значение типа ushort занимает 2 байта (16 бит). И при присвоении переменной b значение переменной a расширяется до 2 байт

```
0000000000000100
```

То есть значение, которое занимает 8 бит, **расширяется** до 16 бит.

Сужающие преобразования, наоборот, сужают значение до типа меньшей разядности. Во втором листинге статьи мы как раз имели дело с сужающими преобразованиями:

```
ushort a = 4;
byte b = (byte) a;
```

Здесь переменной b, которая занимает 8 бит, присваивается значение ushort, которое занимает 16 бит. То есть из `0000000000000100` получаем `00000100`. Таким образом, значение сужается с 16 бит (2 байт) до 8 бит (1 байт).

Явные и неявные преобразования

Неявные преобразования

В случае с расширяющими преобразованиями компилятор за нас выполнял все преобразования данных, то есть преобразования были неявными (**implicit conversion**). Такие преобразования не вызывают каких-то затруднений. Тем не менее стоит сказать пару слов об общей механике подобных преобразований.

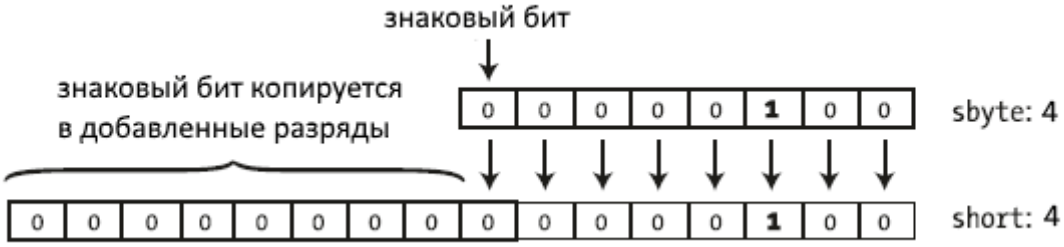
Если производится преобразование от безнакового типа меньшей разрядности к безнаковому типу большой разрядности, то добавляются дополнительные биты, которые имеют значени 0. Это называется **дополнение нулями** или zero extension.

```
byte a = 4;              // 0000100
ushort b = a;    // 0000000000000100
```

Если производится преобразование к знаковому типу, то битовое представление дополняется нулями, если число положительное, и единицами, если число отрицательное. Последний разряд числа содержит знаковый бит - 0 для положительных и 1 для отрицательных чисел. При расширении в добавленные разряды компируется знаковый бит.

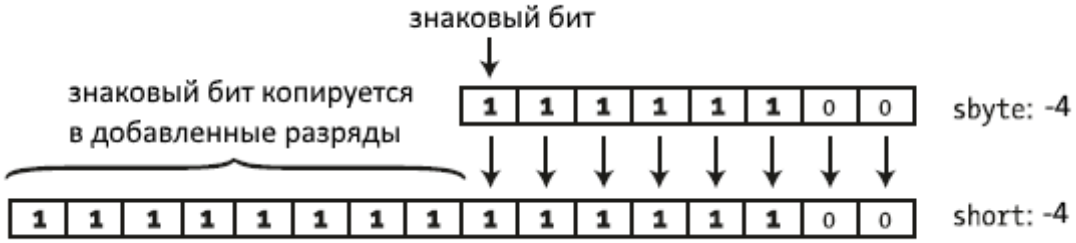
Рассмотрим преобразование положительного числа:

```
1  sbyte a = 4;              // 0000100
2  short b = a;     // 000000000000100
```



Преобразование отрицательного числа:

```
1  sbyte a = -4;             // 1111100
2  short b = a;     // 111111111111100
```



Явные преобразования

При явных преобразованиях (**explicit conversion**) мы сами должны применить операцию преобразования (операция `()`). Суть операции преобразования типов состоит в том, что перед значением указывается в скобках тип, к которому надо привести данное значение:

```
1  int a = 4;
2  int b = 6;
3  byte c = (byte)(a+b);
```

Расширяющие преобразования от типа с меньшей разрядностью к типу с большей разрядностью компилятор проводит неявно. Это могут быть следующие цепочки преобразований:

byte -> **short** -> **int** -> **long** -> **decimal**

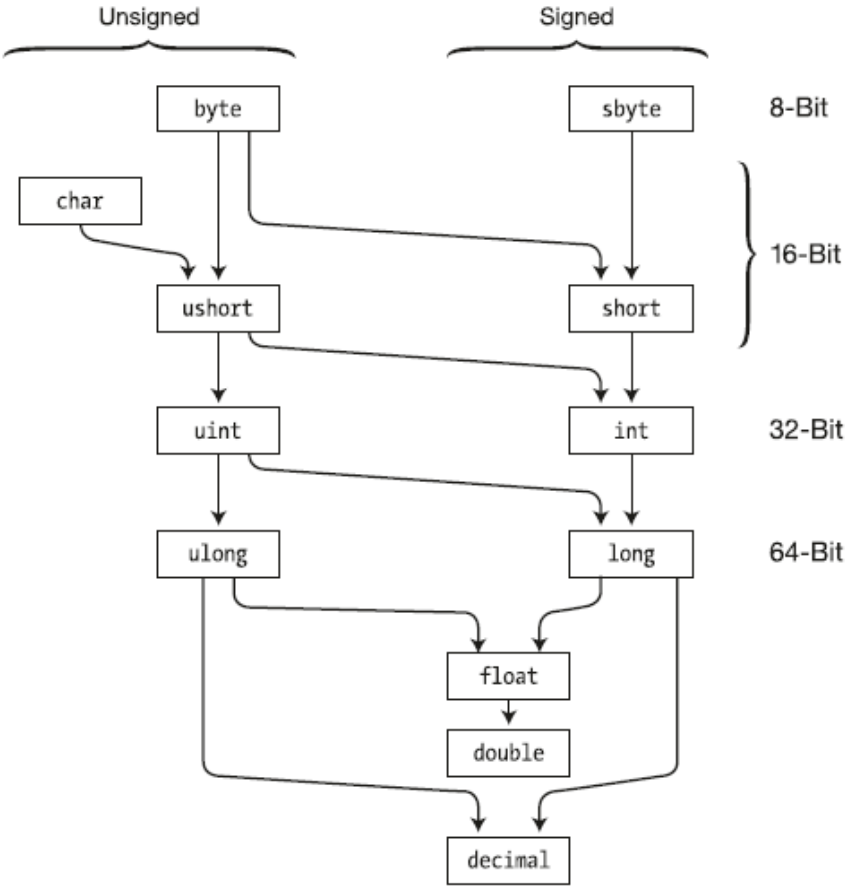
int -> **double**

short -> **float** -> **double**

char -> **int**

Все безопасные автоматические преобразования можно описать следующей таблицей:

Тип	В какие типы безопасно преобразуется
byte	short, ushort, int, uint, long, ulong, float, double, decimal
sbyte	short, int, long, float, double, decimal
short	int, long, float, double, decimal
ushort	int, uint, long, ulong, float, double, decimal
int	long, float, double, decimal
uint	long, ulong, float, double, decimal
long	float, double, decimal
ulong	float, double, decimal
float	double
char	ushort, int, uint, long, ulong, float, double, decimal



В остальных случаях следует использовать явные преобразования типов.

Также следует отметить, что несмотря на то, что и double, и decimal могут хранить дробные данные, а decimal имеет большую разрядность, чем double, но все равно значение double нужно явно приводить к типу decimal:

```
1  double a = 4.0;
2  decimal b = (decimal)a;
```

Потеря данных и ключевое слово checked

Рассмотрим другую ситуацию, что будет, например, в следующем случае:

```
int a = 33;
int b = 600;
byte c = (byte)(a+b);
Console.WriteLine(c);   // 121
```

Результатом будет число 121, так число 633 не попадает в допустимый диапазон для типа byte, и старшие биты будут усекаться. В итоге получится число 121. Поэтому при преобразованиях надо это учитывать. И мы в данном случае можем либо взять такие числа a и b, которые в сумме дадут число не больше 255, либо мы можем выбрать вместо byte другой тип данных, например, int.

Однако ситуации разные могут быть. Мы можем точно не знать, какие значения будут иметь числа a и b. И чтобы избежать подобных ситуаций, в c# имеется ключевое слово `checked`:

```
try
{
    int a = 33;
    int b = 600;
    byte c = checked((byte)(a + b));
    Console.WriteLine(c);
}
catch (OverflowException ex)
{
    Console.WriteLine(ex.Message);
}
```

При использовании ключевого слова `checked` приложение выбрасывает исключение о переполнении. Поэтому для его обработки в данном случае используется конструкция try...catch. Подробнее данную конструкцию и обработку исключений мы рассмотрим позже, а пока надо знать, что в блок try мы включаем действия, в которых может потенциально возникнуть ошибка, а в блоке catch обрабатываем ошибку.