4.4a — Explicit type conversion (casting)

BY ALEX ON APRIL 16TH, 2015 | LAST MODIFIED BY ALEX ON APRIL 27TH, 2017

In the previous section **4.4 -- Implicit type conversion (coercion)**, you learned that the compiler will sometimes implicitly convert a value from one data type to another. When you want to promote a value from one data type to a larger similar data type, using the implicit type conversion system is fine.

Many new programmers try something like this: `float f = 10 / 4;`. However, because 10 and 4 are both integers, no promotion takes place. Integer division is performed on 10 / 4, resulting in the value of 2, which is then implicitly converted to 2.0 and assigned to f!

In the case where you are using literal values (such as 10, or 4), replacing one or both of the integer literal value with a floating point literal value (10.0 or 4.0) will cause both operands to be converted to floating point values, and the division will be done using floating point math.

But what if you are using variables? Consider this case:

```
1 int i1 = 10;
2 int i2 = 4;
3 float f = i1 / i2;
```

Variable f will end up with the value of 2. How do we tell the compiler that we want to use floating point division instead of integer division? The answer is by using a type casting operator (more commonly called a cast) to tell the compiler to do explicit type conversion. A **cast** represents an explicit request by the programmer to do a type conversion.

Type casting

In C++, there are 5 different types of casts: C-style casts, static casts, const casts, dynamic casts, and reinterpret casts.

We'll cover C-style casts and static casts in this lesson. Dynamic casts we'll save until after we cover pointers and inheritance.

Const casts and reinterpret casts should generally be avoided because they are only useful in rare cases and can be harmful if used incorrectly.

Rule: Avoid const casts and reinterpret casts unless you have a very good reason to use them.

C-style casts

In standard C programming, casts are done via the () operator, with the name of the type to cast to inside. For example:

```
1 int i1 = 10;
2 int i2 = 4;
3 float f = (float)i1 / i2;
```

In the above program, we use a float cast to tell the compiler to convert i1 to a floating point value. Because i1 is a floating point value, i2 will then be converted to a floating point value as well, and the division will be done using floating point division instead of integer division!

C++ will also let you use a C-style cast with a more function-call like syntax:

```
1 int i1 = 10;
2 int i2 = 4;
3 float f = float(i1) / i2;
```

Because C-style casts are not checked by the compiler at compile time, C-style casts can be inherently misused, because they will let you do things that may not make sense, such as getting rid of a const or changing a data type without changing the underlying representation (leading to garbage results).

Consequently, C-style casts should generally be avoided.

Rule: Avoid C-style casts

static_cast

C++ introduces a casting operator called **static_cast**. You've previously seen static_cast used to convert a char into an int so that std::cout prints it as an integer instead of a char:

```
char c = 'a';
std::cout << static_cast<int>(c) << std::endl; // prints 97, not 'a'
```

Static_cast is best used to convert one fundamental type into another.

```
int i1 = 10;
int i2 = 4;
float f = static_cast<float>(i1) / i2;
```

The main advantage of static_cast is that it provides compile-time type checking, making it harder to make an inadvertent error. Static_cast is also (intentionally) less powerful than C-style casts, so you can't inadvertently remove const or do other things you may not have intended to do.

Using casts to make implicit type conversions clear

Compilers will often complain when an unsafe implicit type conversion is performed. For example, consider the following program:

```
int i = 48;
char ch = i; // implicit conversion
```

Casting an int (4 bytes) to a char (1 byte) is potentially unsafe, and the compiler will typically complain. In order to announce to the compiler that you are explicitly doing something you recognize is potentially unsafe (but want to do anyway), you can use a cast:

```
int i = 48;
char ch = static_cast<char>(i);
```

In the following program, the compiler will typically complain that converting a double to an int may result in loss of data:

```
int i = 100;
i = i / 2.5;
```

To tell the compiler that we explicitly mean to do this:

```
int i = 100;
i = static_cast<int>(i / 2.5);
```

Summary

Casting should be avoided if at all possible, because any time a cast is used, there is potential for trouble. But there are many times when it can not be avoided. In most of these cases, the C++ static_cast should be used instead of the C-style cast.

Quiz

1) What's the difference between implicit and explicit type conversion?

Quiz Answers

1) **Hide Solution**

Implicit type conversion happens when the compiler is expecting a value of one type but is given a value another type.
Explicit type conversion happens when the user uses a type cast to convert a value from one type to another type.


4.4b -- An introduction to std::string


Index


4.4 -- Implicit type conversion (coercion)