C++: Casts

Miro Jurišić
meeroh@meeroh.org

Why cast?

- a.k.a. coercion
- Tell the compiler: "You are wrong."
- Tell the compiler: "I know something you don't know."
- Casts are difficult to maintain
- Most of the time, there are better ways to accomplish the same goal

C casts in C++

```
int x = -10;
unsigned int y = (int) x; // Arithmetic conversion

int* xp = &x;
double* yp = (double*) xp; // Pointer coercion

int z = 10;
int* zp = (int*) z; // Coercion

const int w = 10;
int* wp = (int *) &w; // Loss of constness
*w = 20;

BaseClass* bp;
DerivedClass* dp = (DerivedClass*) bp; // Casting down a hierarchy
```

etc.

Casts in C++

Distinguish four types of casts

- changes in constness (and volatileness)
- allowed compile-time conversions
- runtime conversions
- other (typically non-portable) conversions
- new cast syntax: verbose and easy to spot

```
int x = 5;
char y = static_cast <char> (x); // Arithmetic conversion

int* xp = &x;
double* yp = reinterpret_cast <double*> (xp); // Pointer coercion

int z = 10;
int* zp = reinterpret_cast <int*> (z); // Coercion
```

```
const int w = 10;
int* wp = const_cast <int*> (&w); // Loss of constness
*w = 20;

BaseClass* bp;
DerivedClass* dp = dynamic_cast <DerivedClass*> (bp); // Casting down a hierarchy
```

const_cast

Remove const or volatile qualifier from a type

```
const int w = 10;
int* wp = const_cast <int *> (&w);

const int w = 10;
const int* wp = &w;
const int** wpp = &wp;
int** wpp2 = const_cast <int**> (wpp);

const int& wr = w
int& wr2 = const_cast <int&> (wr);
```

static_cast

- Only allowed for conversions which the compiler can check
- Can't static_cast a pointer to a non-pointer
- Can't use static_cast to remove constness (or volatileness)
- Can't static_cast from a virtual base class

```
int x = 10;
int y = 20;
double d = static_cast <double> (x) / y;

int x = 10;
unsigned int y = static_cast <unsigned int> (x);
```

dynamic_cast

- Run-time checked casts in an inheritance hierarchy
- Finding the beginning of an object

```
BaseClass* b;
DerivedClass* d = dynamic_cast <DerivedClass*> (b);
if (d) {
    // cast was successful
} else {
    // handle error
}

BaseClass& b;
// Throws if unsuccesful
DerivedClass& d = dynamic_cast <DerivedClass&> (b);

Class* c;
void* v = dynamic_cast <void*> (c); // Points to the beginning (most-derived object)
void* v2 = c; // Equivalent
```

reinterpret_cast

- The most dangerous cast, implementation-dependent
- Practically any kind of conversion allowed

```
int x = 10;
char* str = reinterpret_cast <char*> (x);

Class1* c1;
Class2* c2 = reinterpret_cast <Class2*> (c1);
```

Avoiding casts

- There is usually a better way
- Understand why you are casting
- Understand the alternatives

Avoiding `const_cast`

- `const_cast` of a function argument
  - Are you really trying to modify it?
- `const_cast` of a member variable or `this`
  - Should you use a mutable member?
  - Should the method be non-const?
  - Should you overload on constness?
- `const_cast` of a local variable
- Why did you make it const to begin with?
- Majority of `const_casts` can be traced to a design error
  - Fix your design
  - Work around other people's designs

Avoiding `static_cast`

- `static_cast` in arithmetic conversions
  - Are you using the correct integer and floating point types?
  - Use a constructor instead for safe conversions:

```
int x = 1;
int y = 2;
float f1 = static_cast <float> (x) / y;
float f2 = float (x) / y;
```

- `static_cast` from void* to a different pointer

Avoiding `dynamic_cast`

- Use a common base class and virtual functions instead
- Let the compiler dispatch for you: compile-time checked and more efficient
- Use covariant return types to reduce the need for `dynamic_cast`

```
class Base {
    virtual Base* Clone ();
};

class Derived : public Base {
    virtual Derived* Clone (); // Covariant return type
};

Derived* original = new Derived ();
Derived* copy = original.Clone (); // No cast needed
```

- Avoid chains of `dynamic_cast` – usually indicative of a serious design problem

```
if (dynamic_cast <Type1*> (x)) {
    // x is a pointer to Type1
} else if (dynamic_cast <Type2*> (x)) {
    // x is a pointer to Type2
} else if ...
```

- Shorthand: declarations in if statements

```
if (Derived* d = dynamic_cast <Derived*> (b)) {
    // d is a pointer to Derived
} else {
    // d is not in scope
}
```

- Use `dynamic_cast` to `void*` to get to the most derived object

```
Derived* d; // Class with a virtual function
void* memoryBlock = dynamic_cast <void*> (d);
```

Avoiding `reinterpet_cast`

- Avoid it
- Use every other kind of cast before you resort to `reinterpet_cast`
- Review your design before you resort to `reinterpet_cast`
  - Shouldn't you be using void* instead?