F6a: Casts

When to use which cast

What do casts do?

- Primary:
  - Converting one object to another (e.g. make a bool out of a shared pointer)
  - Changing the type of a pointer (or reference)

- The exact semantics is complicated, here we give a simplified description.

Const cast

- Main use is to take away a const so we can change a const object.

```
struct A { int x; };
void FooC(const A* a) {
    const_cast<A*>(a)->x = 17;
}
```

Dynamic cast, down cast

- The main use of dynamic cast is down casts:

```
struct A {
    int x;
    virtual ~A() = default;         ///to enable dynamic typing
};
struct C : public A {
    int x;
};


void FooD(A* a) {
    a->x = 15; //A's x;
    B*  = dynamic_cast<C*>(a);
    if (c)
        c->x = 17;   // C's x
    else
        ;//*a is not a C so c==nullptr;
}
```

Dynamic cast, side cast

- If we have multiple inheritance: side cast

```
struct A {
    int x;
    virtual ~A() = default;         ///to enable dynamic typing
};
struct B { int x; };
struct C : public A {
    int x;
};
void FooS(A* a) {
    a->x = 15; //A's x
    B* b = dynamic_cast<B*>(a);
    if (b)
        b->x = 17;   // B's x
    else
        ;//a was not a B so b is nullptr
}
```

Static cast, objects

- The main use of static cast is to convert between objects
- Static _cast will try to find a way to convert the expression to the new type.

```
shared_ptr<C> ptr;
if (static_cast<bool>(ptr)) {
    //shared_ptr has an explicit conversion to bool
```

Static cast, up casts

- Casting up in a hierarchy:

```
struct A {
    int x;
};
struct C : public A {
    int x;
};

void FooU(C* c) {
    c->x = 17;                    // C's x
    static_cast<A*>(c) -> x=17; // A's x}
}
```

Static cast, down casts

- Casting down in a hierarchy:

```
struct A {
    int x;
};
struct C : public A {
    int x;
};


void FooSD(const A* a) {
    a->x = 17;                    // A's x
    static_cast<C*>(a) -> x=17; // C's x
//  BUT what happens if a is not a C?
}
```

- You should only use static downcast if you really know what you are doing!

Reinterpret cast

`reinterpret_cast<new_type>(expression);`

- Reinterpret_cast simply reinterpret the bit pattern:
- But only works if either new_type or expression is a pointer.

```
int x = 17;
float y = *reinterpret_cast<float*>(&x);
//in my computer y is now 2.382e-44 and not normalized
```

- You should NEWER use reinterpret_cast