C++ Typecasting

Is it safe?

Type conversion

• Process of converting one type to another type
• Required for typed languages
• As you recall from cs270, internal representation of values differ
  • int – 2’s complement
  • float – ieee754
• Conversion occurs during assignment

Compiler Errors and Warnings

• Error – your program is not legal
• Warning – your program is legal, but may not behave as you intended
• Type conversion may generate warnings
• Type casting is a way to suppress those warnings. It tells the compiler “I know what I am doing, and accept the consequences.”
• Use with caution!

Examples of type conversion

int i = 48;
char c = ‘A’;
float f = 3.14;
double d = 6.02E+23;
i = c; // is it safe?
i = s; // is it safe?
i = f; // is it safe?
i = d; // is it safe?
c = i; // is it safe?
c = f; // is it safe?
f = i; // is it safe?
f = d; // is it safe?

When is type conversion safe?

• Type conversion is safe when the receiving type can hold all the “bits” of the source type
• Generally, if you can convert from type ‘A’ to type ‘B’ and then back to ‘A’ without losing anything, the conversion is safe.
• It is unsafe if it cannot. The compiler generates a warning.
• What you want is doable, but problematic!

Examples of type conversion

int i = 48;
char c = ‘A’;
float f = 3.14;
double d = 6.02E+23;
i = c; // is it safe? – sizeof(int) >= sizeof(char)
c = i; // is it safe? no, 32 bits may not fit in 8 bits
i = f; // is it safe? no, a floating point value may not fit in an int
f = i; // is it safe? no, 23 bit mantissa of floats may not hold 32 bits
f = d; // is it safe? no, lose precision, may be out of range
i = d; // is it safe? no, 64 bits may not fit in 32 bits
d = i; // is it safe? yes, double has more bits in mantissa than int has
Type casting – C style

```c
int i;
float f = 3.14;
float big = 6.02E+23;
i = (int) f; // I know I’m losing the fractional part
i = (int) big; // what is the result?
some = (some’s Type) other; // general form
```

Type Casting C++

- To make it explicit what you are doing, C++ uses additional “types” of casting in addition to the C style. In general, never use C style casts in C++
- The C++ define these type of casts
  - `static_cast<Type>`
  - `dynamic_cast<Type>`
  - `const_cast<Type>`
  - `reinterpret_cast<Type>`

C++ static_cast<Type>

- Conversion done at compile time
- May actually generate code

```c
int i = -1;
unsigned int u;
float f = 3.14
u = static_cast<unsigned int> (i); // any code?
i = static_cast<int> (f); // any code generated?
```

C++ const_cast<ptrType>

- Used to remove “const” from pointer or reference. Use with caution.
- Needed if using functions that should use `const`, but don’t.

```c
void func (Foo* f); // functions doesn’t actually change object
void bar (const Foo* cfp) {
  func(const_cast<Foo*>(cfp)); // warning without cast
}
```

C++ reinterpret_cast<Type>

- Leave the bits alone, but treat them differently. Rarely used in C++. Used in low level code.

```c
float f = 3.14;
int i = static_cast<int> (f); // what is i?
int j = reinterpret_cast<int> (f); // what is j?
```

C++ dynamic_cast<Type>

- Used for polymorphic pointer types (i.e. base class and derived classes)
- Only required when going down the inheritance chain
- Combines Java’s `instanceof` and casting together

```c
class Base;
class Derived : public Base;
Base* bp = new Derived();
Derived* dp = dynamic_cast<Derived*>(bp); // what if it isn’t?
```
C++ \texttt{dynamic\_cast\<type\>}

- The result of \texttt{dynamic\_cast\<type\>} returns 0 if the type of the object is not compatible with the type is the cast. This is similar to the Java’s \texttt{instanceof} operator.
- Best practice is to always check the result of the cast
- Dynamic cast relies on run time type information (RTTI).

C++ \texttt{dynamic\_cast\<type\>}

- Only appropriate for classes that have a VPTR (pointer to virtual functions table).
- VPTR only exists for classes that have virtual functions(s) in the class hierarchy
- Conceptually, simply compare object’s VPTR to that of class being cast to. If match, ok. If no match, look at parent classe(s) VPTR(s).
- Does not actually change anything in object.

C++ \texttt{narrow\_cast\<type\>}

- Proposed extension, not yet part of standard
- Verify at run time that \texttt{static\_cast\<type\>} does not lose data
- Succeeds, or throws runtime exception

```cpp
// Example C++ implementation of 'narrow_cast'<type>(v)

#include <stdexcept>

template <class Target, class Source>
Target narrow_cast(Source v) {
    auto r = static_cast<Target>(v);
    if (static_cast<Source>(r) != v)
        throw std::runtime_error("narrow_cast<>() failed");
    return r;
}

int main() {
    try {
        narrow_cast<int>(12.34);
        assert(!"Incorrect conversion from double to int should be detected by narrow_cast");
    } catch (std::runtime_error &e) {
        // OK
    }
}
```