CSCE 121

Introduction to Program Design & Concepts

Type Conversion

Dr. Tim McGuire

Grateful acknowledgment to Dr. Philip Ritchey and Dr. Michael Moore for some of the material on which these slides are based.

When You Mix Apples with Oranges: Type Conversion

Type Safety

- Every object will be used only according to its type
  - Variable is only used after it is initialized
  - Only operations defined for the variables type will be applied
  - Every operation defined for a variable results in a valid value
- IDEAL! Static type safety
  - Compiler finds all type safety violations.
- IDEAL! Dynamic type safety
  - Run-time system finds all safety violations not found by compiler

Type Safety

- Important!
  - Try hard not to violate
  - Compiler can help
- C++ not completely statically type safe
  - Most languages are not
  - Reduces ability to express ideas
- C++ is not completely dynamically type safe
  - Many languages are, but…
  - Being dynamically type safe can cause performance problems
- Most things in class will be type safe

When You Mix Apples with Oranges: Type Conversion

- Operations are performed between operands of the same type.
- If not of the same type, C++ will convert one to be the type of the other
- This can impact the results of calculations.

Type Conversion

- Implicit conversion
  - One type automatically converted to another type

- Explicit conversion
  - One type “cast” into another type (e.g. by force)

Coercion Rules

1) **`char`**, **`short`**, **`unsigned short`** automatically promoted to **`int`**
2) When operating on values of different data types, the lower one is promoted to the type of the higher one.
3) When using the = operator, the type of expression on right will be converted to type of variable on left

Mostly Safe Conversions

- “Widening” conversions
  - int x = 123456789;
  - long y = x; // ints fit in longs with plenty of room to spare

  - char b = 'k';
  - int a = b; // a is numerical representation of b, but no loss of information

  - float pi = 3.14159265;
  - double also_pi = pi;

Unsafe Conversions

- “Narrowing” conversions
  - double x = 2.7;
  - int y = x; // truncation

  - long x = 1122233445566778899;
  - double y = x; // loss of precision

  - double pi = 3.14159265358979;
  - float pi2 = pi;   // truncation to 6 digits

  - int a = 1000;
  - char b = a;

Type Casting

- Used for manual data type conversion
- Useful for floating point division using **`int`**s:

  ```
  double m;
  m = static_cast<double>(y2-y1)/(x2-x1);
  ```

- Useful to see `int` value of a `char` variable:

  ```
  char ch = 'C';
  cout << ch << " is "
       << static_cast<int>(ch);
  ```

C-Style and Prestandard Type Cast Expressions

- C-Style cast: data type name in `()`

```
   cout << ch << " is " << (int)ch;
```

- Prestandard C++ cast: value in `()`

```
  cout << ch << " is " << int(ch);
```

- Both are still supported in C++, although `static_cast` is preferred

Type Casting

```
// functional
int x = 7;
long y = long(x);

// c-like
int x = 7;
long y = (long)x;
```

```
// c++ casting operators
dynamic_cast <new_type> (expression)
reinterpret_cast <new_type> (expression)
static_cast <new_type> (expression)
const_cast <new_type> (expression)


int x = 7;
long y = static_case<long>(x)
```

http://www.cplusplus.com/doc/tutorial/typecasting/