Constants, Casting, and I/O manipulation

William Hendrix

Lecture 7

Outline

- Constants
- Using software libraries
- Variable casting
- I/O manipulation

Constants

- Unchanging values in code
- Can take the form of literal values or constant variables

- Numeric literals are `int` or `double` by default
  - Can append `F`, `U`, `L`, `UL`, `LL`, or `ULL` to change to `float`, `unsigned int`, `long`, etc.
    - Can also use lowercase letters
- Integers can also be represented in octal or hexadecimal
  - Prefix with `0` (octal) or `0x` (hexadecimal)
    - Can use `a-f` or `A-F` for hex digits past `9`
  - Useful for controlling binary bits
    - Octal digits are 3 bits, hexadecimal 4
    - `0xffffffffu` is largest 4-byte `unsigned int`
- Decimal values can be specified in exponential notation
  - E.g., `6.02E23` or `1e-13`

Other bases

- Each digit base-10 is worth 10 times as much as the digit to the right
  - 8 times (octal) or 16 times (hexadecimal)
- You use digits 0-9 in base-10 numbers
  - 0-7 (octal) or 0-f (hexadecimal)

- Examples
  - `0x30 = 3 * 16 + 0 = 48`
  - `0xff = 15 * 16 + 15 = 255`
  - `0123 = 1 * 64 + 2 * 8 + 3 = 83`
  - `324` (hex) `= 20 * 16 + 4 = 1 * 16^2 + 4 * 16 + 4 = 0x144`
  - `507` (octal) `= 63 * 8 + 3 = (7 * 8 + 7) * 8 + 3`
    `= 7 * 8^2 + 7 * 8 + 3 = 0773`

Magic numbers

- Literal values that appear in code with no explanation
  - E.g., `total = bill * 1.29;`
  - What does this number represent?
  - What if the value changed at a later date?
- **Magic numbers are to be avoided at all costs**
  - Declare named constants instead
- C++-style constants
  - `const double SALES_TAX = 0.11;`
  - Must be assigned a value when declared
  - Cannot appear on LHS of assignment statement
  - **Convention:** names of constants are all caps
- C-style constants
  - `#define SALES_TAX 0.11`
    - No `;` or `=`
  - Appears in preamble, after `using namespace std;`

Using libraries

- Libraries define helpful functions and classes you can use
- Libraries must be included before use
  - Built-in libraries: `#include <cstdlib>`
    - Nonstandard: some compilers don't require including built-in libraries
  - User-defined libraries: `#include "project.h"`
- `cstdlib`: standard functions
  - `abs(x)`: absolute value (integers)
    - Nonstandard: may also accept `double`
  - `rand()`: pseudorandom number generator
    - Pseudorandom: not actually random, but hard to predict
    - Produces `int` values between `0` and `RAND_MAX`
      - Use `rand() % n` to get values between `0` and `n - 1`
      - `rand() / (RAND_MAX / n)` can produce higher quality random numbers if randomness is really important
  - `srand(x)`: initializes random number generator for `rand()`

The `cmath` library

- All `cmath` functions accept and return `double` values
- `fabs(x)`: floating point absolute value
- `sqrt(x)`: square root
- `pow(x, y)`: exponentiation
  - `x * x` is much more efficient than `pow(x, 2.0)`
- `sin(x), cos(x), tan(x)`: trig functions
- `asin(x), acos(x), atan(x), atan2(y, x)`
  - Inverse trig functions
  - `atan2(y, x)` is `atan(y / x)`
- `exp(x), log(x), log10(x)`
  - e^x, natural log, and common log
- `floor(x), ceil(x)`
  - Rounds down or up to nearest integer

Variable casts

- Explicit instruction to convert one variable type into another
- C++ casts: `static_cast<type>(var/expression)`
  - E.g., `static_cast<double>(sum)`
- C casts: `(type) var`
  - E.g., `(double) sum`

- Casting issues
  - Converting from large integer types to small
    - Calculates the value modulo the max (least significant bytes)
  - Converting from floating point to integer
    - Rounds down to nearest integer (like `floor`)
    - Can cause undefined behavior if outside of integer range

Floating point round-off

- Due to round-off errors, floating point variables may be slightly above or below "true" value
  - Decimal example: `(1 / 3) * 3 = 0.999999...`

- Use epsilon values to account for small differences due to round-off
  - `static_cast<int>(x + EPSILON)`
  - `floor(x + EPSILON)`
  - `ceil(x - EPSILON)`
  - Equality: `fabs(x - y) < EPSILON`
  - `const double EPSILON = 1e-14;`
  - Typical values: `1e-14` or `1e-13`
    - May need larger or smaller values if using large or small double values (~14-15 places smaller)

Tonight

Lab 1 is due tonight!

Recommended reading: Sections 2.6, 6.5.1, 6.5.3