ITM 312, Fall 2013

Chapter 3 Lecture Notes

3.1. cin object

a. Standard console input

b. Need to add `#include <iostream>` for it to work

c. Reads input from keyboard

d. Use >> to set input from cin to a variable, as in:
`cin >> your_variable` (like `your_variable = ‘‘my cin input’’`)

e. Cin automatically converts input into your variable's datatype

f. If assigning multiple variables, use the notation:
`cin >> first_variable >> second_variable >> third_variable`
Your input looks like: "`first_val second_val third_val`" (separated by spaces); order is important

3.2. Mathematical Expressions

a. You can create complex expressions in C++

b. Expressions follow the order of operations

i. Negation first (e.g. `NOT / !`)

ii. Multiplication, division, modulus second (in order from left to right)

iii. Addition/Subtraction third (in order from left to right)

c. Parentheses can override order of operations

d. For exponents, use power function `pow(var, power)`

3.3. Type Conversion

a. Operations can only be performed between objects of the same type

b. C++ will try to convert one type to the other, based on hierarchy of types

i. Long double (highest)

ii. Double

iii. Float

iv. Unsigned long

v. Long

vi. Unsigned int

vii. Int (lowest)

c. Type conversion can impact the result of calculations (precision!)

d. When C++ automatically converts, this is called Type Coercion; Coercion rules:

i. Char, short, unsigned short automatically promoted to int

ii. Lower datatype gets promoted to higher datatype

iii. The result of expression will be converted to the type of the variable the result is assigned to (e.g. if you have `double var = int foo + int blah;` then `int` will be typecasted to type `double`)

e. Promotion: conversion to a higher type; Demotion: conversion to a lower type

3.4. Overflow and Underflow

a. Occurs when value cannot fit within the bounds of the type being casted

i. Overflow occurs when value too big

ii. Underflow occurs when value too small

b. Variable is wrapped around instead, resulting in incorrect value

c. Some systems may give an error when this occurs, others continue on

3.5. Type Casting

a. Manual data type conversion (e.g. double to int)

b. Good for keeping precision when doing division with ints

c. Can use to see int value of a char (ASCII)

d. Casting expressions

i. Datatype(variable) e.g. `int(foo)`

ii. Static_cast<datatype>(variable) e.g. `static_cast<int>(foo)`

3.6. Multiple Assignment and Combined Assignment

a. Multiple assignment: Use to assign a single value to multiple variables

i. `x = y = z = 5` versus `x=5; y=5; z=5;`

ii. Associates right to left `x=(y=(z=5));`

b. Combined assignment: Use to shorten your `x=x*5` -type statements

i. `sum+=1` instead of `sum = sum+1`

ii. Works with all operands +,-,*,/,%

3.7. Formatting Output

a. Used to control how numeric or string data is printed (size, position, num. of digits)

b. Use `#include <iomanip>` to use manipulator functions
c. Stream Manipulators: Used to control how an output field is displayed
    i. See Table 3-12 on slide 43 for the various manipulators

3.8. Working with characters and string objects

a. Cin can cause problems, ignores leading whitespace
b. Use `getline()` to get around this issue
c. Use `cin.get(variable)` to read next character entered (even whitespace)
d. Don't use `cin >>` and `cin.get()` together, can cause problems
e. Use `cin.ignore()` to skip characters (refer to slide 49 for params)
f. Find the length of a string with `.length()` (e.g. `mystring.length()`)
    i. When using anything`.method()` then an object is involved
g. Join (concatenate) multiple strings together with + (`string3 = string1 + string2`) – can use combined assignment operator

3.9. More Mathematical Library Functions

a. Use `#include <cmath>` for additional math functions
    i. Trig, sqrt, log, abs val
b. Takes double as input, returns double
c. Use `#include <cstdlib>` for:
    i. Rand() – random number generator (*same sequence each time*)
    ii. Srand(x) – random number generator using unsigned int x (*different sequence because it uses x as a "seed"*)

3.10. Hand Tracing a Program

a. Act if you were a computer executing the program
b. Useful for locating logic or math errors
c. DON'T GOOGLE FIRST, hand trace instead
    i. Do your own programs and your own debugging, otherwise you won't learn
d. To hand-trace:
    i. Step through each line, executing the line in your mind
    ii. Record the contents of variables after statement execution
        1. Use a table ("hand-trace chart")
    iii. Look for any mistakes in assignment, math, etc.
    iv. See slide 56 for a sample program to hand-trace

3.11. Case Study

a. Define your variables
b. Use a flowchart to map out algorithm
c. Write pseudocode to plan lines of code to write
d. Take note of formulas needed