**Chapter 1**

* **hardware components:** main memory, secondary memory, central processing unit (CPU)
* **main memory:** random-access memory (RAM), address
* **kinds of values stored in memory:** integers, real numbers (floats), characters (using ASCII or Unicode), strings, program instructions
* **secondary memory:** hard disk, floppy disk, CD, flash drive
* **programming languages:** machine language, assembly language, high-level languages, syntax vs. semantics

**Chapter 2**

* **components of a program (for now) in order:** explanatory comment and your name, preprocessor includes, namespace statement, main function
* use of the cout statement, including << and endl
* **variables:** programmer-defined named storage locations in memory
  + must be declared
  + must have a type
  + must have a name that follows valid *identifier* rules
  + may be initialized with a value at declaration time
* **data types:** A *data type* is a set of values and a set of operations defined on those values. Data types so far include:
  + **integer types we use in our programs:** int, unsigned
  + **other integer types we won't use:** short, long, long long, unsigned short, unsigned long, unsigned long long
    - know which of these is appropriate for various kinds of data
    - know about hex (base 16) and octal (base 8) including literals
  + **char:** character data type for single characters using single quotes for literals
  + **string:** old-fashioned C-strings that we will use for string literals and the string class that we will use later for variables
  + **floating-point type we used in our programs:** double
  + **floating-point types we won't use:** float, long double
    - literals can use scientific notation 1.4959E11 for example
    - literals can use decimal notation -123.456 for example
  + **bool:** meaning the Boolean type, with the two values true and false
* **operations:**
  + **assignment:** uses the = sign
    - distinguish left-hand side (lhs, resolves to lvalue aka address) from right-hand side (rhs, resolves to rvalue aka a value)
    - **scope:** is the regions of the program in which the variable exists and its name can be legally used
    - one form is initialization, can only be done once per variable
    - declaration should be close to where the variable is first used
  + **unary minus** negates a numeric value
  + **addition subtraction multiplication:** + - \* work the way you would expect
  + **division modulus:** division is straightforward for floats and works like on your calculator
    - with integer types, does integer division
    - division gives you the quotient
    - modulus gives you the integer remainder
    - modulus can distinguish even from odd, pick out digits, etc. . .
  + parentheses work as you would expect and are useful

**Chapter 3**

* cin reads data from standard input and converts it to the type of the variable.
* mathematical operators use precedence and associativity the way you learned in your algebra class.
* many more mathematical operations are available from the <cmath> library, including pow
* overflow and underflow can result from arithmetic operations
* mostly they just wrap around without causing errors
* mixed type arithmetic operations are possible by following coercion rules, but still discouraged
* instead use type casting: static\_cast<double>(a) casts integer a to double
* combined assignment: x += 1; y \*= 5;
* formatting output: setw(x), fixed, setprecision(x), showpoint