**2.1 Variable and Assignments**

- **Incrementing** a variable --> x = x +1

- **Variable declaration** – statement that declares a new variable (ex. int userAge)

- **Allocation** – process of determing suitable memory location to store data.

- **Assignment Statement –** assigns the variable on the left-side of the = with current value of the right-side expression.

**2.3 Identifiers**

**Identifier –** name created by programmer for an item like a variable or function.

* must be a sequence of letters, underscores, and digits (no other characters)
* **start with a letter or underscore**
* case sensitive

**Reserved word –** a word that is a part of the coding language. (int, double, str) also know as keywords. Cannot be used as identifiers.

* List of reserved words
  + http://en.cppreference.com/w/c/keyword

**Naming Conventions for Identifiers –** two common conventions

* Camel case: capitalize each word except the first (numDogs, numCars)
* Underscore seperated: words are lowercase and serparated by underscore. (num\_dogs, num\_cars)
* Be consistent
* Good practice
  + Use meaningful identifier names
  + Avoid the use of abbreviations except for common words
  + At least two or more words

**2.4 Arithmetic Expressions**

**Expression:** any individual item (variables, literals, operators and parenthesis) or combination of items that evaluates to a value. Usually on right side of assignment statement.

**Literal:** specific value in code like a 2.

**Operator:** symbol that performs a built-in calculation. Like +

* Operators in C:
  + +: addition
  + -: subtraction also negation
  + \*: multiplication
  + /: division

**Precedence rules:** expression is evaluated using the order of standard mathematics.

* (): items within parenthesis first
* Unary -: negation is computed next ex. 2 \* -x, the –x is computed first.
* \*, /, %: multiplication, division, and modulus are evaluated next all of equal precedence
* +, -: addition and subtraction also have equal precedence below multiplication and division
* Left to right: if more than one operator of equal precedence could be evaluated, evaluation occurs left to right
* Can use parenthesis to indicate what should get evaluated first if otherwise not obvious and make it explicit.
* Usually want a space around each operator with exception of **unary-** or negating something.

**2.5 Arithmetic Expresssions (INT)**

**Compound Operator:** shorthand way to update a variable. Ex. UserAge += 1 is the same as UserAge = UserAge + 1

**2.6 Incremental Development**

- write a little code, test, iteratively

**2.7 Floating Point Numbers (double)**

**Floating-point numbers –** real number containing a decimal point that can appear anywhere (float) in the number.

**Double variable –** variable that stores a floating-point number

**Floating point literal –** number with a fractional part explicitly stated.

**Floating point division by zero –** normally dividing a non-zero floating-point number by zero is undefined in regular arithmetic. It will also produce an error in most programming languages. In C it will evaluate to infinity or –infinity this will output inf or –inf.

* If both the dividend and divisor are 0 the division results in not a number (NAN)

**Manipulating floating-point output –** printf(“%.2lf”,myFloat);

- C will round the last output digit but the floating point value is still the same in memory.

**2.8 Scientific Notation for floating point literals**

**Scientific Notation in C:** written using an e preceding the power of 10 exponent.

Ex. 6.02 e23

* For floating point literal its good practice to make the leading difit non-zero
* For converting between scientific notation and normal number just move the decimal place to the right or left the number of times necessary.
  + Ex. 4.507e -6 = 0.000004507 (moved decimal six spots to the left)
  + Ex. 37,100 = 3.71e 4 (moved decimal 4 spots to the left)

**2.9 Constant Variables**

**Constant Variable:** an initialized variable whose value cannot change; this can be done by preceding the variable initilizaton with const. The compiler wiil then raise an error if a later statement tries to change that variables value.

* Good practice is to name constant variable using all caps and words separated by underscores. (to make them more visible)

**2.10 Using Math Functions in C**

**Standard math library** – has about 20 math operations, known as functions included in the library.

**Function –** list of statements executed by invoking the functions name, invoking is called a function call.

* **Arguments –** appear with the () seperated by commas if more than one.

**2.11 Integer Division and Modulo**

**Integer rounding:** when operands of / are integers, equation does not generate any fraction. The extra digits are just dropped off of the integer number. At least one of the operands must be a floating point type to do floating division.

* Ex. 2.5 = 2, 0.75 = 0
* **Divide by 0:** In integer division the second operand of / or % must never be 0, this would cause a divide by zero error which would occur at runtime.

**Runtime error:** error occurs at runtime and cause program to terminate early.

**Modulo (%):** evaluates to the remainder of the divison between the two integer operands.

* **Both** operands must be integers
* The remainder as in number of integers left over. Ex. 596 % 10 = **6** (596 = 59 \* 10 + **6**)
* Can create a random number in range generator using modulo (see below)
* Ex. Rand(x) % 9 – this will generate a random # between 0 – 8
  + Because: 9 % 9 = 0 (no remainder); 10 % 9 = 1 (one left over after dividing)
  + 11 % 9 = 2 ... etc
  + Also works for numbers smaller than 9 (see below)
  + The actual calculation c does for modulo is in **integer arithmetic/division**
    - A % B = A - (A / B) \* B
    - 1 % 9 = 1 - (1 / 9) \* 9
    - 1 % 9 = 1 – 0 \* 9
    - 1 % 9 = 1 – 0
    - This will then create the range 1 to B because as long as A < B the division will always equal 0 and A will be left over.

**Integers –** when doing arithmetic specifically **multiplication and addition** with integers, very large inputs may cause overflow, resulting in an incorrect output.