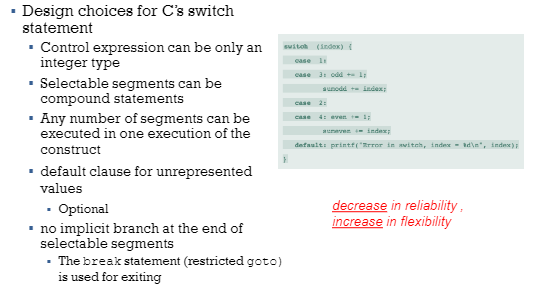
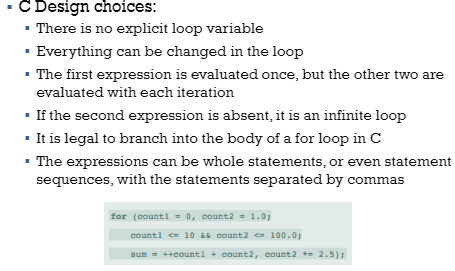
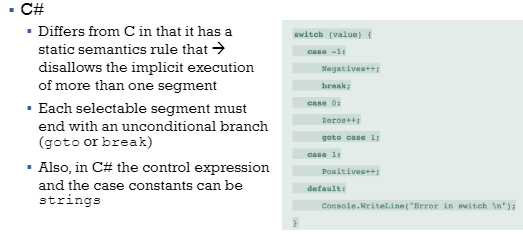
Arithmetic expressions are comprised of the parameters and functions in the forms of numbers and operations like addition and subtraction. You can also include parentheses. Unary, Binary, Ternary, 1-3. Three notations, infix, prefix, postfix. Infix; between operands. Prefix; before operands. Postfix; after operands. **Precedence** determines the order in which adjacent operators with different precedence are evaluated. **Associativity** is the rules that determine the order in which operators with the same precedence are evaluated. **Operand Evaluation Order**: Variables, fetch value from memory, Constants, fetch sometimes, sometimes constant is in machine instructions, Parenthesized expressions, Function Calls may cause side effects. Functional side effects occur when a function changes a two-way parameter or a non-local variable. This is particularly perilous when the function alters a operand within the expression in which the function is called. Remove two-way parameters in functions, remove globals, you do lose flexibility. You could write language definitions that require operands be fixed in evaluation order. However, this solution limits some optimizations. **Referential transparency**, one expression can be written differently, but still be the same value. This makes it easy to understand all parts of the program. **Overloaded operators**, operators with extended purpose for classes, must be used carefully, very powerful. They can obscure error detection, loss of readability, binding modules in a system. **Type conversions** occur in two ways, narrowing, large type to smaller type, widening, small to large type. **Implicit** conversions are conversions which occur indirectly, 3.0 + 4, 4 may convert to a double or 3.0 will convert to int. These conversions can make error detection harder. **Explicit** conversions can be used through casting, (int)name for C. Relational and Boolean Expression, compares values using <, >, ==, !=, and many others. Most relational expressions will breakdown to Boolean values, true or false. C has no Boolean types. Arithmetic expressions can be operands for relational expressions, and relational expressions can be the operands of Boolean expressions, different precedence levels. **Short-Circuit Eval.** May cause issues, by skipping certain code. Assignment statements, (=, :=, <=). Compound assignment operators (-=, +=). Unary assignment operators (=). Conditionals, if/else-if/else. Multiple assignments statements. Assignments as expressions, Sum = count = 0;

Control statements alter and control the flow of execution, providing better control to the programmer on execution. First implemented in FORTRAN, not efficient at first. Unconditional branching, (goto) transfers control to a specific location. Not supported in all languages, dangerous to use. No explicit exit for control statements, only transfer into structure, good read, no dangers. Selection statements, allow means of choosing between two or more paths of execution, two-ways and multiple-ways. **Two-Way** “if(expr)[stat]then[stat]”. **Multiple-Way**, allow a selection from any number of statements or groups of statements. **Switch statement**, control expression must be a discrete type. Switches do not always provide explicit end sections and must be quit through *exit* keyword. In C, control may only be integer.

Iterative statements, repeated execution of code until some condition is met. The body of an iterative statement is repeated, the execution is controlled by iteration statement. Pre-test, post-test of control statement in iterative statements. Counter controlled, for-loops. **Break** expression exits outermost statement, **Continue** will skip the current iteration. Iterator functions to move through data quickly.