Generating code - big picture

From declarations, you create the symbol table (ST), keep track of the base address there. From that information, you can lay out storage from computing: `base + offset*numOfBytes`

When you finish generating code, you know the address where the data will start. Traverse the ST and generate memory for the variables.

Intermediate Code Generation: 3 address code (of the form x := y op z)

Two implementations of 3-address code are triples and quadruples.

Consider the following example. Note that "--" means blank in code generation as not all items are used. Also note that this is before optimization.

```
   a := b + c*(d-e) + f * (c*(d-e))
```

Quadruples

```
   op  arg1  arg2  result
   -    d     e      t1      // keep track of temporary names in symbol
table
   *    c     t1     t2
   +    b     t2     t3
   *    f     t2     t4
   +    t3    t4     t5
 assign t5    --     a       // more generally, use word "assign"
```

Triples (has advantage that using this notation, you can avoid storing names in ST)

```
   op    arg1  arg2
0. -      d     e             // values are typically in registers
1. *      c     0.
2. +      b     1.
3. *      f     1.            // using dag (directed acyclic graph)
4. +      2.    3.
5. assign a     4.
```

Common codes:

```
Binary assignment:   x := y op z

Unary assignment:    x := op y

Copy (or assign):    x := y

Unconditional jump:  goto L      // for label
  Triples example:
   op    arg1  arg2
   goto   L    --

Conditional jump:  if true goto L
  Triples example (if x < y goto L):
      op   arg1  arg2
   0. <     x     y
   1. if    0.    L

Procedure definition:
   Example:
   proc name
   paramval x1
   paramref x2
   ...
For function:
   return  y

Procedure call: parameter x, call p with n parameters
   Example:
   param   x1
   param   x2
   ...
   param   xn
   call    p    n

Indexed statements: x := y[i] and x[i] := y
  Triples example (x := y[i]):
      op   arg1  arg2
   0. =[]   y     i
   1. :=    x     0.
  Triples example (x[i] := y):
      op   arg1  arg2
   0. []=   x     i
   1. :=    0.    y

  Quadruple example (x := y[i]):
      op   arg1  arg2  result
      =[]   y     i     t1
      :=    t1    --    x
  Quadruple example (x[i] := y):
      op   arg1  arg2  result
      []=   x     i    t1
      :=    y     --   t1

Address and pointer statements: x := &y, x := *y, *x := y
  Triples example (x = &y):
      op  arg1  arg2
   0. &    y     --
   1. :=   x     0.
  Triples example (*x = y):
      op  arg1  arg2
   0. *    x     --
   1. :=   0.    y
```