VHDL CONSTRUCTS

Sequential Statements:

if-then-else

```
    general format:                         example:
        if (condition) then                     if (S = "00") then
            do stuff                                Z <= A;
        elsif (condition) then                  elsif (S = "11") then
            do more stuff                           Z <= B;
        else                                    else
            do other stuff                          Z <= C;
        end if;                                 end if;
```

elsif and else clauses are optional
BUT incompletely specified if-then-else (no else) implies memory element

case-when

```
    general format:                         example:
        case expression is                      case S is
            when value =>                           when "00" =>
                do stuff                                Z <= A;
            when value =>                           when "11" =>
                do more stuff                           Z <= B;
            when others =>                          when others =>
                do other stuff                          Z <= C;
        end case;                               end case;
```

for-loop

```
general format:                             example:
[label:] for identifier in range loop       init: for k in N-1 downto 0 loop
    do a bunch of junk                          Q(k) <= '0';
end loop [label];                           end loop init;
```

note: variable k implied in for-loop and does not need to be declared

while-loop

```
general format:                             example:
[label:] while condition loop               init: while (k > 0) loop
    do silly stuff                              Q(k) <= '0'
end loop [label];                               k := k – 1;
                                            end loop init;
```

note: variable k must be declared as variable in process (between
sensitivity list and begin with format:

```
variable k: integer := N-1;
```

Concurrent Statements:

logical operators with signal assignment <= example: Z <= A and B;

when-else

general format:	example:
expression when *condition* else	Z <= A when S = "00" else
expression when *condition* else	B when S = "11" else
expression when others;	C;

note: "when others" maybe redundant and incompatible with some tools

with-select-when

general format:	example:
with *selection* select	with S select
expression when *condition*,	Z <= A when "00" ,
expression when *condition*,	B when "11" ,
expression when others;	C when others;

Signal assignments in a process:

All expressions based on current value of signals
(right-hand side of <=, values at start of process execution)
Assigned signals updated at end of process execution

Example:

```
process (CK) begin
    D <= Q xor CIN;
    if (CK'event and CK = '1') then
        Q <= D;
    end if;
    COUT <= Q xor CIN;
end process;
```

Case 1: sensitivity list consists only of CK (no other implied signals)
on rising clock edge, Q gets value of D based on Q and CIN from previous
execution of process
if CIN is available prior to falling edge of CK then count works as expected
otherwise, it does not
also COUT is updated on falling edge of CK and not when Q changes

Case 2: sensitivity list consists of CK, CIN and Q (or CIN and Q implied)
D and COUT updated anytime Q or CIN changes
on rising clock edge, Q gets updated value of D & count works as expected

Signal assignments in a concurrent statement:

Like a process with implied sensitivity list (right-hand side of <=)

∴ multiple concurrent statements work like multiple 1-line processes
updates assigned signal whenever right-hand has event

Example:
```
D <= Q xor CIN;
COUT <= Q xor CIN;
process (CK) begin
    if (CK'event and CK = '1') then
        Q <= D;
    end if;
end process;
```

D and COUT updated anytime Q or CIN changes
on rising clock edge, Q gets updated value of D & count works as expected
same as if we put the 2 concurrent statements in process with Q & CIN in sensitivity list

Initialization:

All processes (and concurrent statements) evaluated once
then again and again until there are no events in sensitivity list

If explicit initialization is not defined (using := assignment operator)
then a bit is assigned '0' and a std_logic is assigned 'U'

When no events happen for a given process sensitivity list then that process is suspended

Simulation cycle:

1. Time is advanced until next entry in time queue where signals are to be updated (for example, PIs) which cause events on these signals
2. A simulation cycle starts at that point in time and processes & concurrent statements "sensitive" to events (during the current simulation time) on those signals will be executed
3. Simulation times for subsequent simulation cycles are determined and scheduled based on internal signals being updated from processes or concurrent statements

Note: we will talk about specifying delays later, right now we consider only delta (δ) delays = infinitesimal delays

4. If there are any δ delay time queues go to Step 2, else go to Step 1
   Examples:
```
Z <= X after 5ns;   -- specified delay scheduled as entry in time queue
Z <= X;             -- δ delay scheduled as entry in δ delay time queue
```