INF5430
Access Types & VHDL Testbench Case

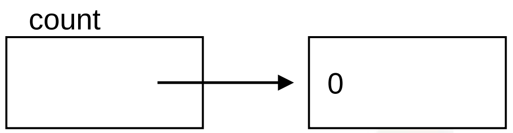
Roar Skogstrøm,
Institutt for Informatikk, Universitetet i Oslo

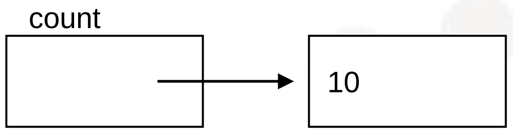
Access datatyper

- Benyttes der størrelsen av data ikke er kjent på forhånd
  - Dynamisk allokering av data
  - Definerer pekere til data
- Benyttes for å lage en kompleks sammenheng mellom datatyper
  - Scalare og composite datatyper ikke tilstrekkelig
  - F.eks. lenka lister
- Benyttes i modellering/testbenker

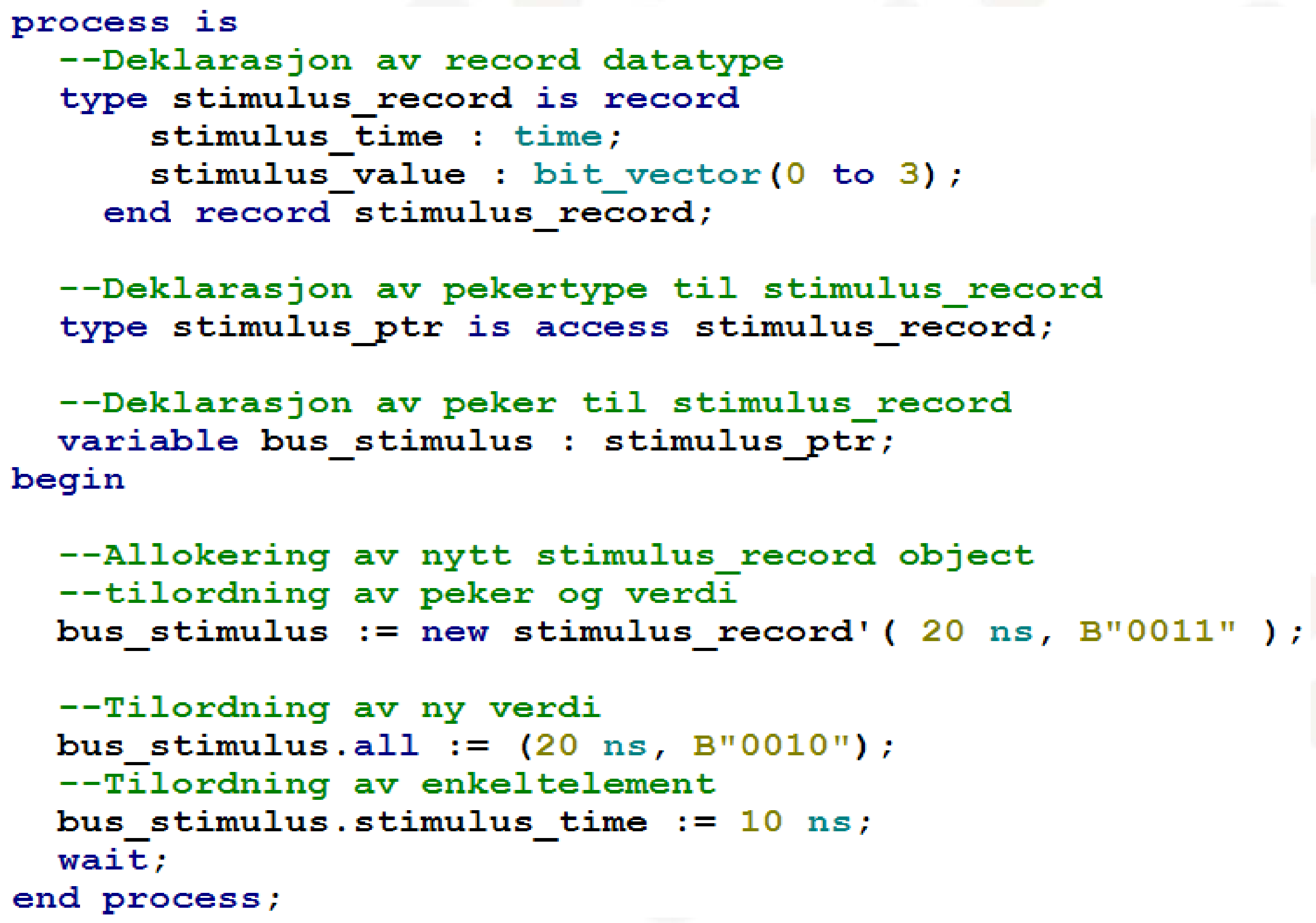
Deklarering og allokering

```
process is

   --Deklarasjon av pekertype til datatypen natural
   type natural_ptr is access natural;

   --Deklarasjon av peker
   variable count : natural_ptr;

begin
   --allokerer et nytt natural object og
   --count settes til å peke på det
   count := new natural;

   --tilordning av verdi til objectet
   count.all := 10;

   --allokering, initialisering av peker
   --og tilodning av verdi
   count := new natural'(10);
   wait;
end process;
```





Bruk av record datatyper



```
process is
  --Deklarasjon av record datatype
  type stimulus_record is record
      stimulus_time : time;
      stimulus_value : bit_vector(0 to 3);
    end record stimulus_record;

  --Deklarasjon av pekertype til stimulus_record
  type stimulus_ptr is access stimulus_record;

  --Deklarasjon av peker til stimulus_record
  variable bus_stimulus : stimulus_ptr;
begin

  --Allokering av nytt stimulus_record object
  --tilordning av peker og verdi
  bus_stimulus := new stimulus_record'( 20 ns, B"0011" );

  --Tilordning av ny verdi
  bus_stimulus.all := (20 ns, B"0010");
  --Tilordning av enkeltelement
  bus_stimulus.stimulus_time := 10 ns;
  wait;
end process;
```

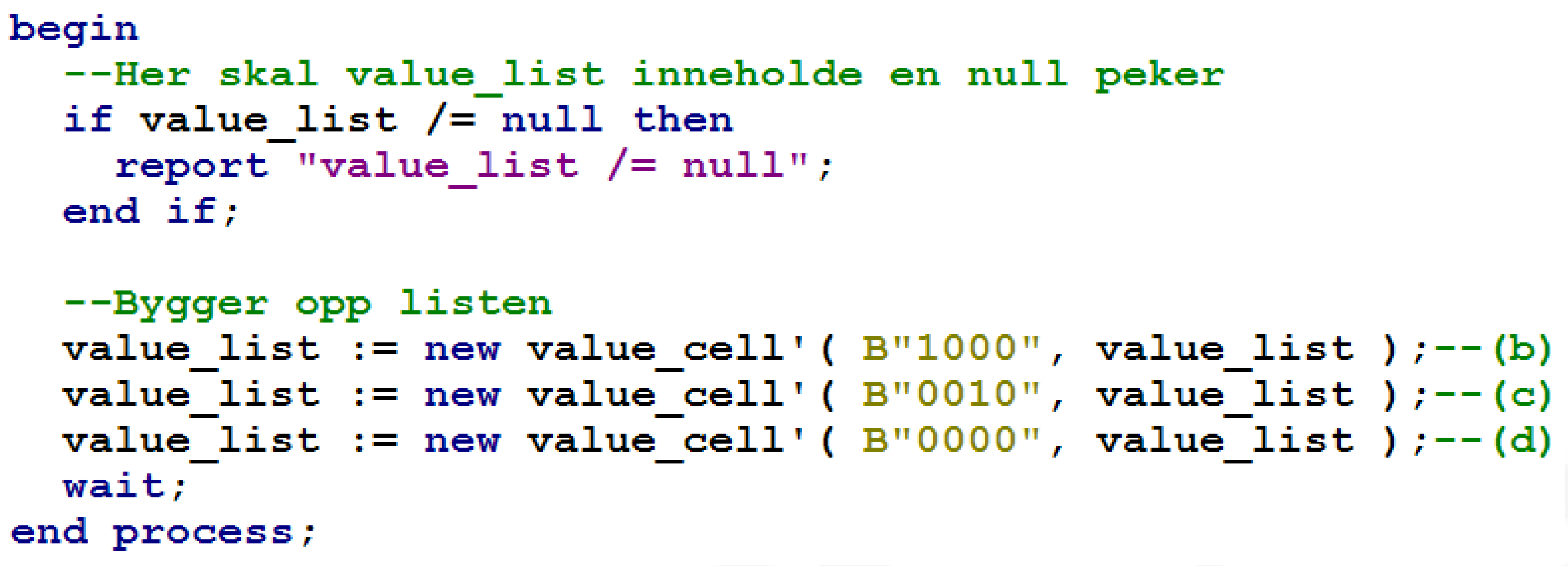
Bruk av array datatyper

```
process is

   type time_array is array (positive range <>) of time;
   type time_array_ptr is access time_array;
   variable activation_times : time_array_ptr;

begin

   --Allokerer et nytt time_array object på tre elementer
   activation_times := new time_array'(10 us, 15 us, 40 us);
   --Allokerer et nytt timearray object på to object i tillegg til det
   --eksisterer fra før
   activation_times := new time_array'( activation_times.all
                                        & time_array'(70 us, 100 us) );
   --Allokerer nytt time object med 10 elementer
   activation_times := new time_array(1 to 10);

   wait;
end process;
```

Lenka lister

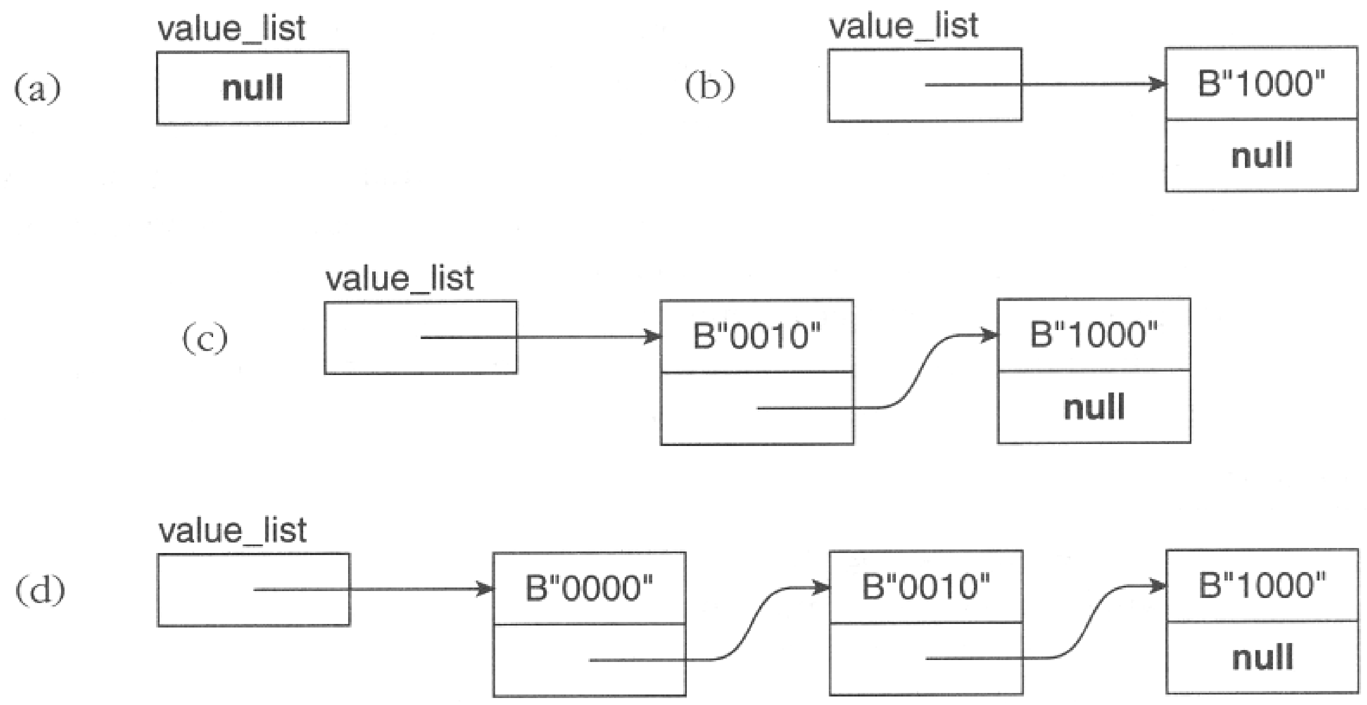
```
process is

  --ikke komplett typedefinisjon, bare navn interessant nå
  type value_cell;

  --deklarasjon av pekertype
  type value_ptr is access value_cell;

  --komplett typedefinisjon
  --med definisjon av peker til neste celle
  type value_cell is record
      value : bit_vector(0 to 3);
      next_cell : value_ptr;
    end record value_cell;

  variable value_list : value_ptr;--(a)

begin
```

Lenka lister

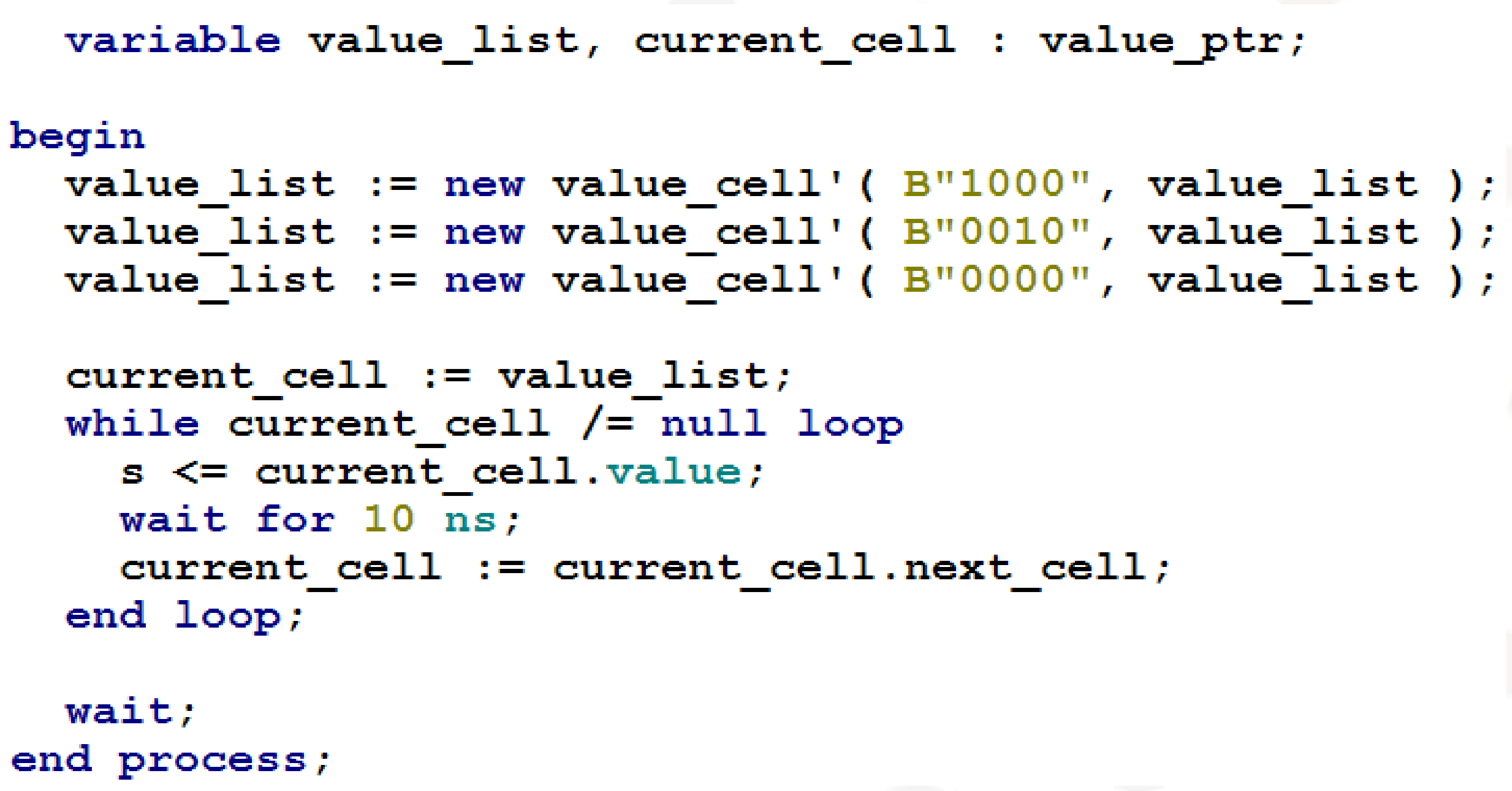


```
begin
  --Her skal value_list inneholde en null peker
  if value_list /= null then
    report "value_list /= null";
  end if;

  --Bygger opp listen
  value_list := new value_cell'( B"1000", value_list );--(b)
  value_list := new value_cell'( B"0010", value_list );--(c)
  value_list := new value_cell'( B"0000", value_list );--(d)
  wait;
end process;
```

Lenka lister

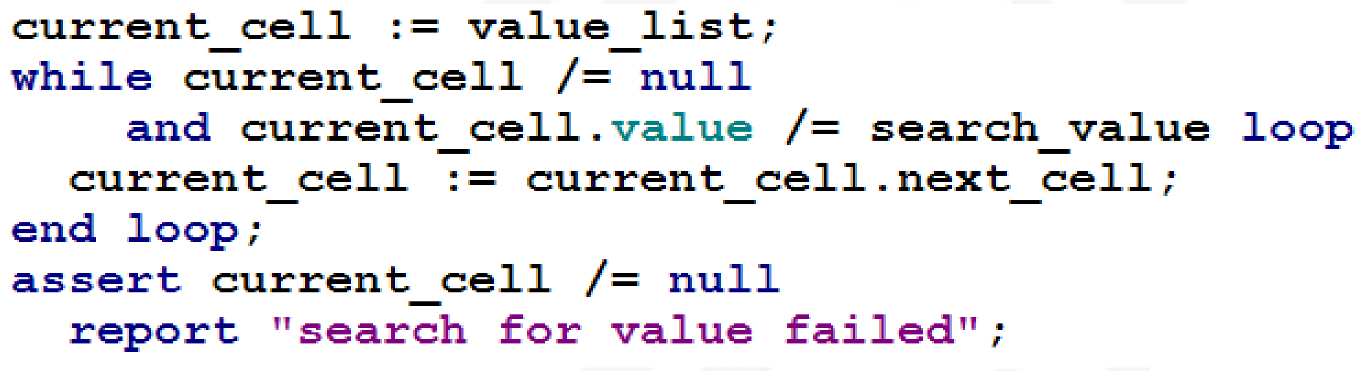


Lage stimuli ved lenka lister



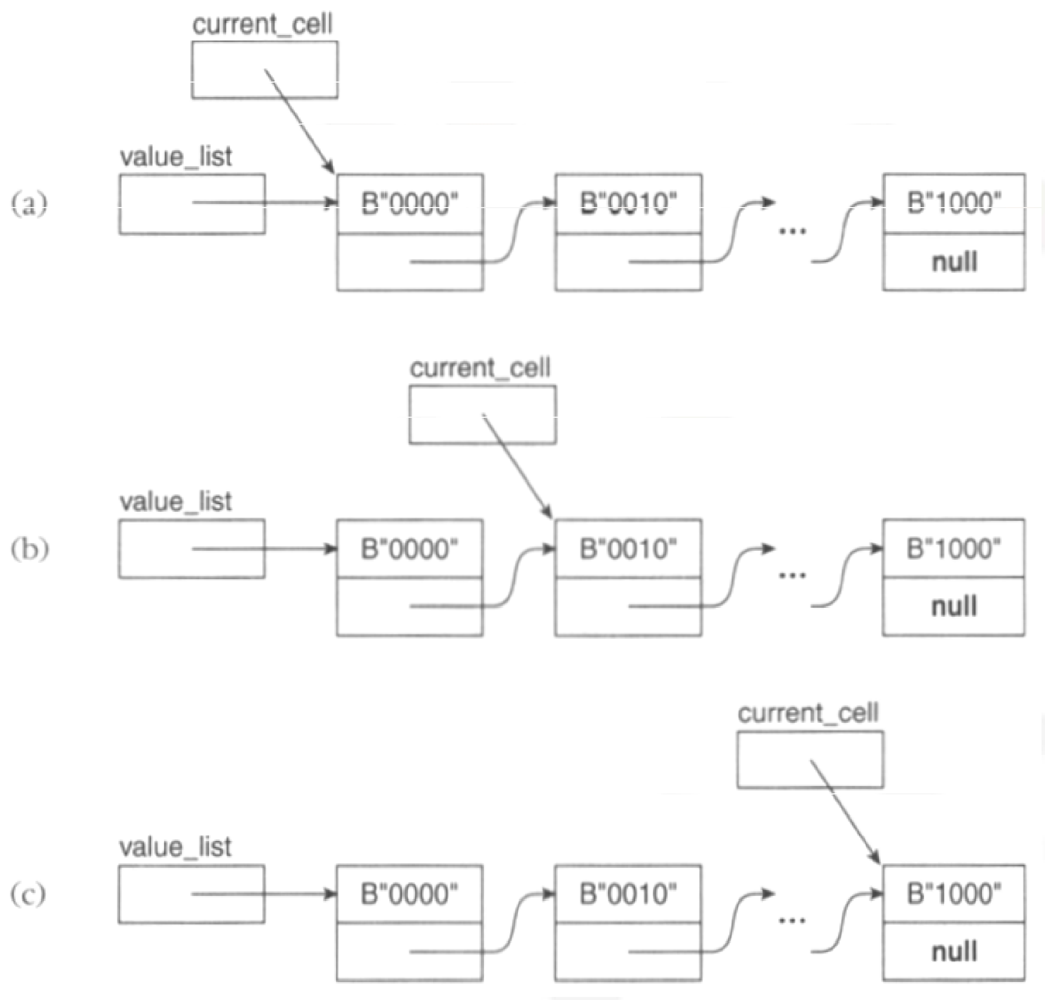
```
  variable value_list, current_cell : value_ptr;

begin
  value_list := new value_cell'( B"1000", value_list );
  value_list := new value_cell'( B"0010", value_list );
  value_list := new value_cell'( B"0000", value_list );

  current_cell := value_list;
  while current_cell /= null loop
    s <= current_cell.value;
    wait for 10 ns;
    current_cell := current_cell.next_cell;
  end loop;

  wait;
end process;
```

Søking i lenka lister



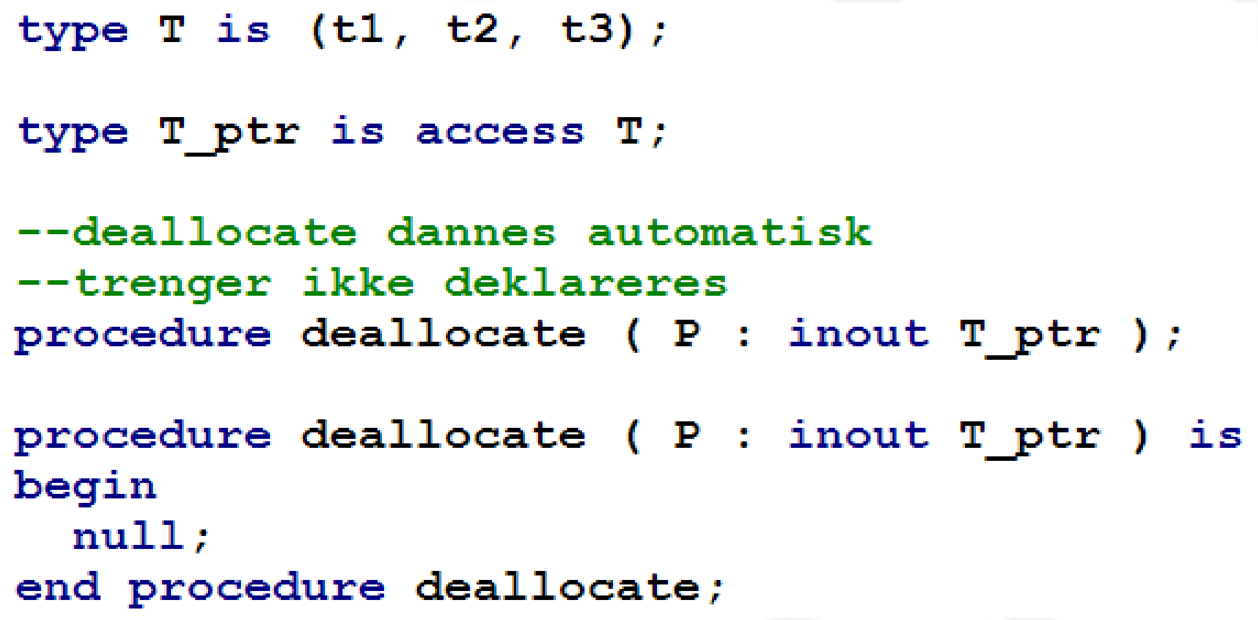
```
current_cell := value_list;
while current_cell /= null
    and current_cell.value /= search_value loop
  current_cell := current_cell.next_cell;
end loop;
assert current_cell /= null
  report "search for value failed";
```

Søking i lenka lister



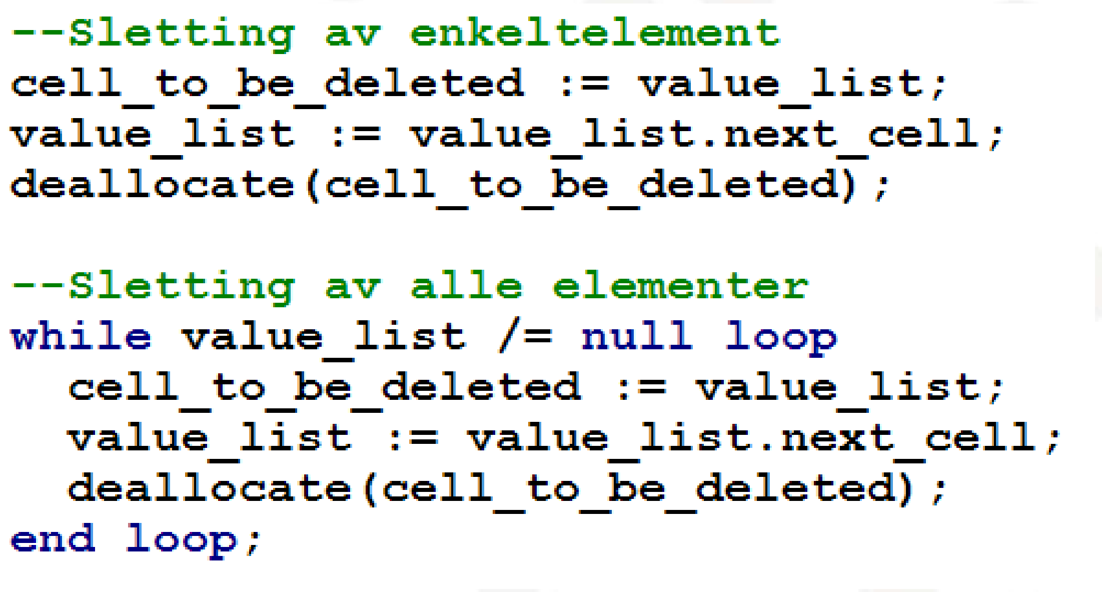
De-allokering

- Når vi definerer en accesstype får vi automatisk dannet en procedyre *deallocate*



```
type T is (t1, t2, t3);

type T_ptr is access T;

--deallocate dannes automatisk
--trenger ikke deklareres
procedure deallocate ( P : inout T_ptr );

procedure deallocate ( P : inout T_ptr ) is
begin
  null;
end procedure deallocate;
```

De-allokering



```
--Sletting av enkeltelement
cell_to_be_deleted := value_list;
value_list := value_list.next_cell;
deallocate(cell_to_be_deleted);

--Sletting av alle elementer
while value_list /= null loop
  cell_to_be_deleted := value_list;
  value_list := value_list.next_cell;
  deallocate(cell_to_be_deleted);
end loop;
```

Testbench address map file read procedures and access types:

```
type addrmap_cell;
  type addrmap_ptr is access addrmap_cell;
  type addrmap_cell is record
    name      : string(1 to 40);
    hex_value : std_logic_vector(ADDRESS_LENGTH-1 downto 0);
    next_cell : addrmap_ptr;
  end record addrmap_cell;
:
:
----------------------------------------------------------------
-- This procedure makes a list of addresses in infile and
--   and returns a pointer to the head of the list.
----------------------------------------------------------------
procedure addr_list_gen(file log              : text;
                        variable list_start  : in addrmap_ptr;
                        constant infile      : in string;
                        variable list_header : out addrmap_ptr);
----------------------------------------------------------------
-- This procedure searches the address map list for the string
--   and returns the address value.
----------------------------------------------------------------
procedure get_address(
                      variable char_address  : in string;
                      variable addrmap_list  : in addrmap_ptr;
                      variable address_value : out std_logic_vector;
                      variable found         : out boolean);
```

VHDL testbench usage:

```
variable addrmap_list : addrmap_ptr;
variable list_start   : addrmap_ptr;
:
command_loop: while not endfile(command_file) loop

          readline(command_file, command);
          -- skipping blank lines
          rm_space(command);
          if command=null then
            next command_loop;
          end if;

          -- read command.
          readword(command, length, word);
          command_id:= new string'(word(word'low to length));
              :
          elsif command_id.all="map" then

                    readword(command, length, word);

                    -- missing file name error
                    if length=0 then
                      local_error_no:= local_error_no+1;
                      writef(log, cycle_no, "  ERROR: MAP command error. Missing file parameter");
                      deallocate(command_id);
                      next command_loop;
                    end if;

                    list_start := addrmap_list;
                    addr_list_gen(log, list_start, word(1 to length), addrmap_list);
```

Address list generation procedure:

```
begin

    file_open(file_ok, addrmap_file, infile, read_mode);

    if (file_ok = open_ok) then

      local_list := list_start;

      addrmap_loop: while not endfile(addrmap_file) loop

        readline(addrmap_file, addrmap);
        rm_space(addrmap);

        -- skipping blank lines
        if addrmap=null then
          next addrmap_loop;
        end if;

        readword(addrmap, length, word);
        if length=0 or word(1 to length)/="constant" then
          next addrmap_loop;
        end if;

        -- read address map name.
        readword(addrmap, length, word);
        if length=0 then
          name_error_cnt:= name_error_cnt+1;
          next addrmap_loop;
        else
          addrmap_name(1 to word'length):= word;
        end if;
      find_char(addrmap,'"', lastchar);  -- Searching for "
      if (lastchar='x' or lastchar='X') then
        -- read address map hex value.
        hread(addrmap, addrmap_value, read_ok);
      else
        -- read address map binary value.
        read(addrmap, addrmap_value, read_ok);
      end if;
      if not read_ok then
        value_error_cnt:= value_error_cnt+1;
        next addrmap_loop;
      end if;

      -- Add to list
       local_list := new addrmap_cell'(addrmap_name, addrmap_value,
                                      local_list);

    end loop;

    writef(log, "NOTE: Invalid addresses in address map package: ", name_error_cnt);
    writef(log, "NOTE: Invalid data in address map package: ", value_error_cnt);

    -- Address map file completed
    file_close(addrmap_file);
  else
      writef(log, "ERROR: Cannot open address map file");
  end if;

  list_header := local_list;

end procedure addr_list_gen;
```

Get address procedure returns address in hex value:

```
procedure get_address(
    variable char_address  : in string;
    variable addrmap_list  : in addrmap_ptr;
    variable address_value : out std_logic_vector;
    variable found         : out boolean) is

    variable current_cell  : addrmap_ptr;
  begin

    current_cell:= addrmap_list;
    while current_cell/=null and
      current_cell.name/=to_lower(char_address) loop
      current_cell:= current_cell.next_cell;
    end loop;

    if current_cell/=null then
      address_value:= current_cell.hex_value;
      found:= true;
    else
      found:= false;
    end if;

  end procedure get_address;
```

… and deallocate the probe_list and addrmap_list elements:

```
    :
    :
    :
        assert false report "Simulation completed" severity failure;
        writef(log, cycle_no, "  NOTE: Simulation completed");

        -- Deallocate probe list data
        while probe_list/=null loop
          probe_list_cell := probe_list;
          probe_list := probe_list.next_cell;
          deallocate(probe_list_cell);
        end loop;

        -- Deallocate addrmap list data
        while addrmap_list/=null loop
          addrmap_list_cell := addrmap_list;
          addrmap_list := addrmap_list.next_cell;
          deallocate(addrmap_list_cell);
        end loop;

        Tcycle(2);
        proc_run <= '0';
        wait; -- terminate process

    end process TB_COMMAND_0;
```