```
> #Exercise from last class: Can we eliminate the while loop in the following?
> #Q1: Write a procedure that uses recursion to print the list of prime factors of a given positive
      integer.

>
> recfac := proc(n :: posint)
   local num;
   if n = 1 then return ; end if;
   num := 2;
   while frac(n/num) > 0 do num := num + 1;
   end do;
   if num > n then print(n);
   else print(num); recfac(n/num);
   end if;
   end proc;
>
> #recursive procedure to find the smallest factor of n greater than equal to "start"
> ffac := proc(n :: posint, start :: posint)

   if start = n then return n;
   elif frac(n/start) = 0 then return start;
   else return ffac(n, start + 1);
   end if;
   end proc;
   ~
ffac := proc(n::posint, start::posint)
   if start = n then
      return n
   elif frac(n/start) = 0 then
      return start
   else
      return ffac(n, start + 1)
   end if
end proc
                                        ~                                    (1)
> ffac(17, 2);
                                       17                                    (2)
> ffac(21, 2);
                                        3                                    (3)
> recfac2 := proc(n :: posint)
   local num;
   if n = 1 then return ; end if;
   num := ffac(n, 2); #find the smallest prime factor of n
   print(num);
```

```
    recfac2(n/num);
    end proc;
recfac2 := proc(n::posint)                                                    (4)
    local num;
    if n = 1 then return  end if; num := ffac(n, 2); print(num); recfac2(n/num)
end proc
> recfac2(48);
                                    2
                                    2
                                    2
                                    2
                                    3                                         (5)
>
> #Q1.from lab 9 (Thu)
> power2 := proc(n)
    if n < 1 then return false;
    elif n = 1 then return true;
    elif type(n, integer) = false then return false;
    else return power2(n/2);
    end if;
    end proc;
power2 := proc(n)                                                             (6)
    if n < 1 then
        return false
    elif n = 1 then
        return true
    elif type(n, integer) = false then
        return false
    else
        return power2(1/2*n)
    end if
end proc
>
> power2(6);
                                  false                                       (7)
> power2(32);
                                  true                                        (8)
> power2(16);
                                  true                                        (9)
> power2(-2);
                                  false                                       (10)
>
```

```
>
>
> #Q1 from Lab 9 (Tue)
>
> rprint :=proc(n :: integer, num :: integer)
   if num = 0 then return;
   else print(n);
   rprint(n, num − 1);
   end if;
   end proc
```

$$\mathit{rprint} := \mathbf{proc}(n::\mathit{integer}, \mathit{num}::\mathit{integer}) \quad \textbf{(11)}$$

$$\mathbf{if}\ \mathit{num} = 0\ \mathbf{then\ return}\ \ \mathbf{else}\ \mathit{print}(n);\ \mathit{rprint}(n, \mathit{num} - 1)\ \mathbf{end\ if}$$

$$\mathbf{end\ proc}$$

```
>
> rprint(5, 3);
```

$$5$$
$$5$$
$$5 \quad \textbf{(12)}$$

```
> #Q2: Write a procedure using recursion that takes as input a positive integer and prints out a
      single 1, 2 2's, 3 3's,..., n n's. E.g. if the input is 4 the output should be 1 2 2 3 3 3 4 4 4 4.
>
> q2 :=proc(n :: integer)
   if n = 1 then print(1);return;
   else

   rprint(n, n); q2(n − 1);
   end if;
   end proc;
```

$$q2 := \mathbf{proc}(n::\mathit{integer}) \quad \textbf{(13)}$$

$$\mathbf{if}\ n = 1\ \mathbf{then}\ \mathit{print}(1);\ \mathbf{return}\ \ \mathbf{else}\ \mathit{rprint}(n, n);\ q2(n - 1)\ \mathbf{end\ if}$$

$$\mathbf{end\ proc}$$

```
> q2(4);
```

$$4$$
$$4$$
$$4$$
$$4$$
$$3$$
$$3$$
$$3$$
$$2$$
$$2$$
$$1 \quad \textbf{(14)}$$

```
>
> #Q3:

   notprime :=proc(L :: list, n :: posint)#find the sum of non-prmes in L[1..n]
```

```
    if n = 1 and isprime(L[n]) = true then return 0;
    elif n = 1 and isprime(L[n]) = false then return L[n];
    elif isprime(L[n]) = true then return notprime(L, n - 1);
    else
    return L[n] + notprime(L, n - 1);
    end if;
    end proc;
```

```
notprime := proc(L::list, n::posint)                                    (15)
    if n = 1 and isprime(L[n]) = true then
        return 0
    elif n = 1 and isprime(L[n]) = false then
        return L[n]
    elif isprime(L[n]) = true then
        return notprime(L, n - 1)
    else
        return L[n] + notprime(L, n - 1)
    end if
end proc
```

```
>
>
>
> L := [11, 21, 31, 42];
```

$$L := [11, 21, 31, 42] \tag{16}$$

```
> notprime(L, 4);
```

$$63 \tag{17}$$

```
>
>
> #=================================================
> #Some programs are equally easy can be written using recursion or otherwise
> #======== Recursive Fibonacci ================================
> recfib := proc (n :: posint)
    option remember;
    if n < 3 then return 1
    else return recfib(n-1) + recfib(n-2)
    end if ;
    end proc;
```

```
recfib := proc(n::posint)                                               (18)
    option remember;
    if n < 3 then return 1 else return recfib(n - 1) + recfib(n - 2) end if
end proc
```

```
> #============iterative Fibonacci============
> iterfib :=proc(n :: posint)
    local num, L;
    L := Array(1 ..n);
```

```
    #L:=[seq(0, i=1..n)];

    if n < 3 then return 1 ;
     else
    L[1] := 1; L[2] := 1;
     for num from 3 to n do
     L[num] := L[num - 1] + L[num - 2];
     end do;
     end if;
     end proc;
iterfib := proc(n::posint)                                                    (19)
     local num, L;
     L := Array(1..n);
     if n < 3 then
          return 1
     else
          L[1] := 1; L[2] := 1; for num from 3 to n do
               L[num] := L[num - 1] + L[num - 2]
          end do
     end if
end proc
> iterfib(250);
                  7896325826131730509282738943634332893686268675876375       (20)
> iterfib2 :=proc(n :: posint)
    local num, curr, last, lastlast;
    if n < 3 then return 1 ;
     else
     last := 1;
     lastlast := 1;
     for num from 3 to n do
     curr := last + lastlast;
     lastlast := last;
     last := curr;
     end do;
     return curr;
     end if;
     end proc;
iterfib2 := proc(n::posint)                                                   (21)
     local num, curr, last, lastlast;
     if n < 3 then
          return 1
     else
          last := 1;
          lastlast := 1;
          for num from 3 to n do curr := last + lastlast; lastlast := last; last := curr end do;
          return curr
```

```
    end if
end proc
> iterfib2(250);
```

$$7896325826131730509282738943634332893686268675876375 \quad \textbf{(22)}$$

```
>
>
>
>
>
```