Csc 301 Lab 7
Baheem Ferrell

# GIML: Tutorial Five

## More Recursive Functions

1. Type in and test the following functions, be sure that you understand what each does:

```
        fun     index(0, h::t)    = h
|       index(n, h::t)    = index(n-1, t);
fun     takeN(0, h::t)  = nil
|       takeN(n, h::t)  = h :: takeN(n-1, t);
fun     dropN(0, x)     = x
|       dropN(n, h::t)  = dropN(n-1,t);
```

2. Sorting. The insert function inserts an integer into an ordered list:

```
fun     insert (n:int) nil = [n]
|       insert n (h::t) = if (n&lt;h) then ...
                                    else …
```

Complete the definition and test insert. To sort a list we proceed recursively. Sorting the empty list is trivial, sorting a list (h::t) is a matter of inserting h into the sort t

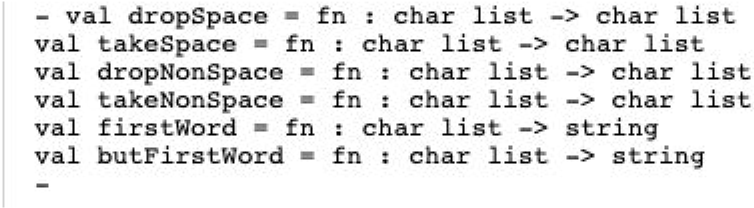
```
fun     sort nil = nil
|       sort (h::t)         = …
```

3. Define the function upto: upto 5 8 = [5,6,7,8]

4. The following functions are required in the next diversion a)The function dropSpace returns a list with leading spaces removed. The function takeSpace returns just the leading spaces.

```
fun dropSpace nil = nil
| dropSpace(h::t) = if h=" " then dropSpace t else h::t;
fun takeSpace nil = nil
| takeSpace (h::t)= if h=" " then h::takeSpace(t)
   else nil;
```

Test these on exploded strings which start with spaces. Define the function dropNonSpace and takeNonSpace and use them to define firstWord and butFirstWord such that: firstWord(explode "One fine day") = "One" implode(butFirstWord(explode "One fine day")) = "fine day"



```
- val dropSpace = fn : char list -> char list
val takeSpace = fn : char list -> char list
val dropNonSpace = fn : char list -> char list
val takeNonSpace = fn : char list -> char list
val firstWord = fn : char list -> string
val butFirstWord = fn : char list -> string
-
```

# GIML: Tutorial Six

## Some standard functions

The following functions will be used in further work without comment.

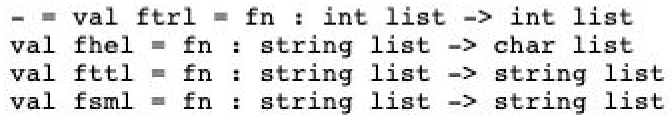
```
fun     map f nil               = nil (* pre-defined anyhow *)
|       map f (h::t)            = (f h)::map f t;
fun     reduce f b nil          = b
|       reduce f b (h::t)       = f(h,reduce f b t);
fun     filter f nil            = nil
|       filter f (h::t)   = if f h then h::filter f t
else filter f t;
fun     member x nil            = false |
member x (h::t)           = x=h orelse member x t;
fun     zip f nil nil           = nil
|       zip f (h::t) (i::s)= f(h,i)::zip f t s;
fun     fst(a,_)          = a;    (* Also try #1 *)
fun     snd(_,b)                = b;    (* Try #2 *)
```

1. Consider each of the following expressions:

```
map(fn s => s^"io") ["pat", "stud", "rat"];
map(fn i => [i]) [4, 2, 1];
map hd [[2, 3], [7, 3, 2], [8, 6, 7]];
map(hd o rev o explode)["final","omega","previous","persist"];
```

2. Define each of the following functions using map

```
ftrl([1, 7, 5, 3])=[3, 21, 15, 9]
fhel(["tom", "dot", "harriet"])=["t", "d", "h"]
fttl(["strange", "shout", "think"])=["range", "out", "ink"]
fsml(["war", "la", "tea", "per"])= ["swarm", "slam",...]
```



```
- = val ftrl = fn : int list -> int list
val fhel = fn : string list -> char list
val fttl = fn : string list -> string list
val fsml = fn : string list -> string list
```

3. Determine what each of the following do

val r = reduce (fn(a,b)=&gt;b@[a]) nil;

This command reverses a list

val p = reduce (op ::);

This command appends the values to a list and then reverses it

fun m x = reduce (fn(a,b)=&gt;(a=x) orelse b) false;

This function returns true if x is the list, if not then it will return false

fun n x = reduce (fn(a,b)=&gt;(a=x) andalso b) true;

This function returns true if all of the elements of the list are equal to x

val im = reduce (op ^) "";

This function is going to implode

val ts = reduce (fn(a,b)=&gt;if a=" " then nil else a::b) nil;

This function returns the elements in the list upto over to the first space

```
val r = reduce (fn(a:int,b)=>b @ [a]) nil;
 val p = reduce (fn(a:int, b)=>a::b);
 val dr = reduce (fn(a,b)=>a+10*b) 0;
 fun m x = reduce (fn(a,b)=>(a=x) orelse b) false;
 fun n x = reduce (fn(a,b)=>(a=x) andalso b) true;
 val im = reduce (op ^) "";
 val ts = reduce (fn(a,b)=>if a=" " then nil else a::b) nil;
```

4. Define each of the following using reduce

prodlist [4,2,5,1] = 40

flatten [[4,2,5],[],[1]] = [4,2,5,1]

count [3,2,5,1] = 4

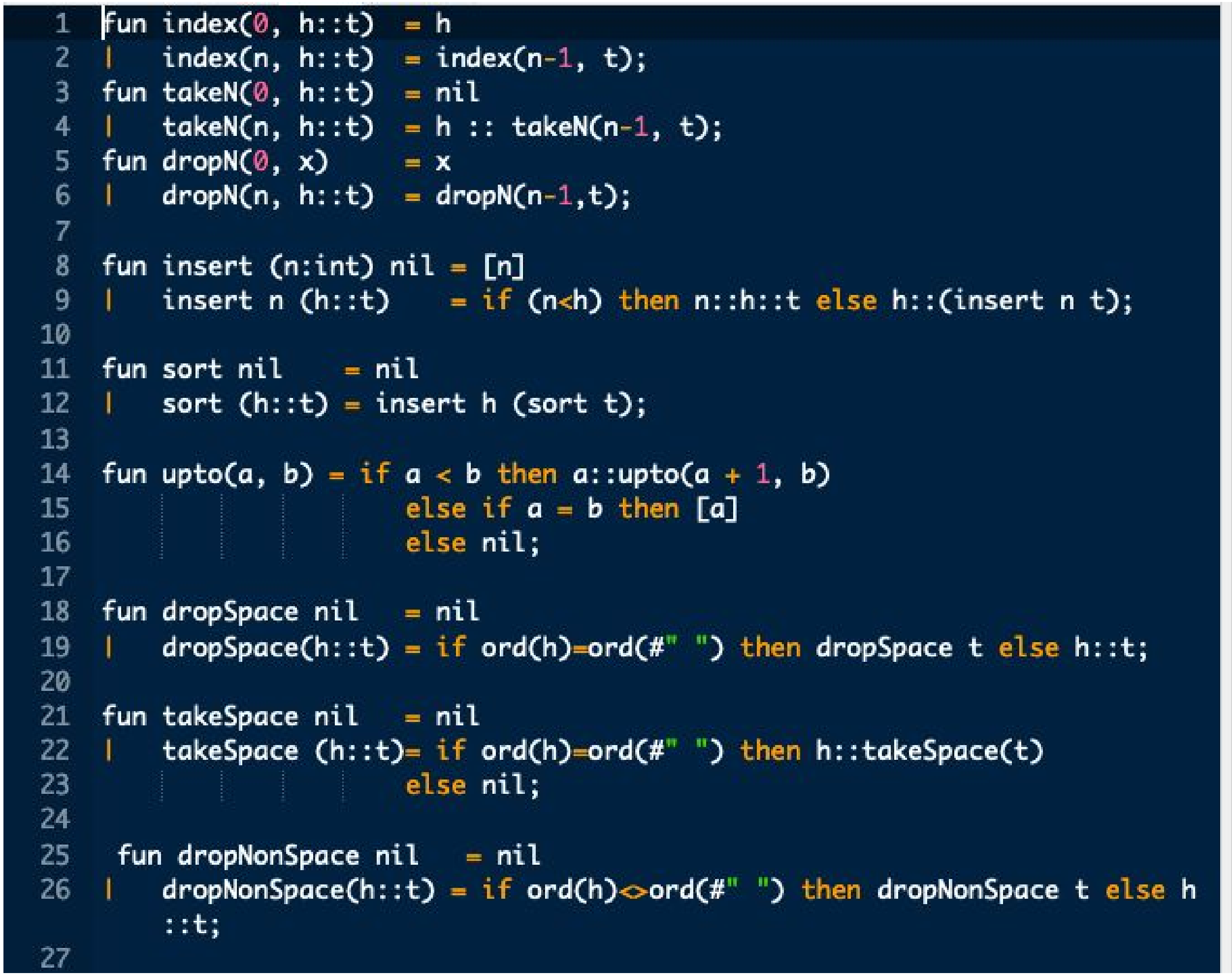
duplist [4,2,5,1] = [4,4,2,2,5,5,1,1]

```
val prodList = reduce (fn(a:int, b:int) => a * b) 1;
val flatten = reduce (fn(a:int list, b:int list) => a @ b) nil;
val count = reduce (fn(a:int, b:int) => b + 1) 0;
val duplist = reduce (fn(a:int, b:int list) => [a, a] @ b) nil;
```

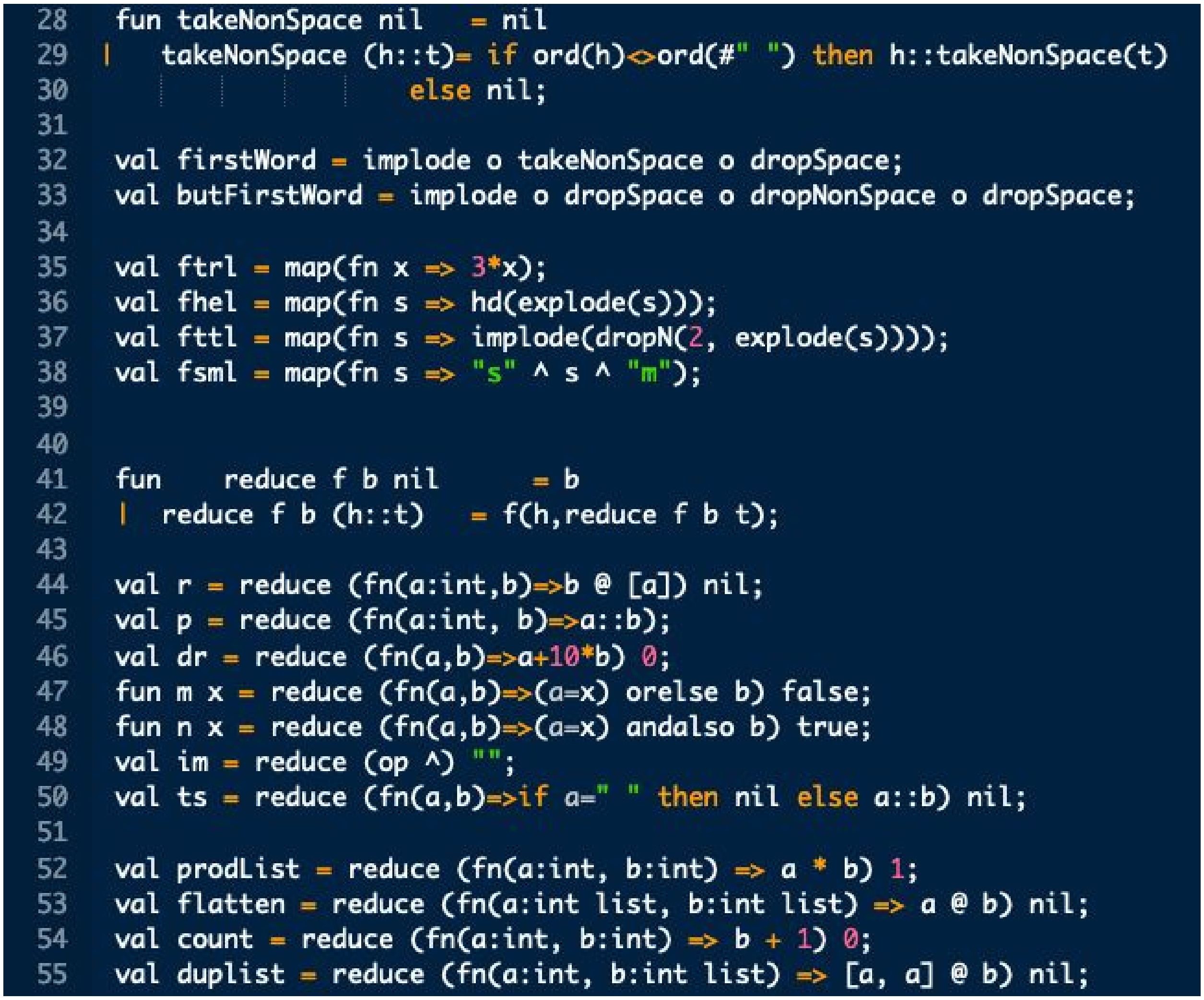
5. Determine what each of the following do

```
fun rm x = filter (fn a=&gt; a&lt;&gt;x);
val mx = reduce max ~1000000;
fun sq (x:int list) = zip (op * ) x x;
fun rprime x = filter (fn i =&gt; i mod x &lt;&gt;0);
fun sieve nil = nil
```

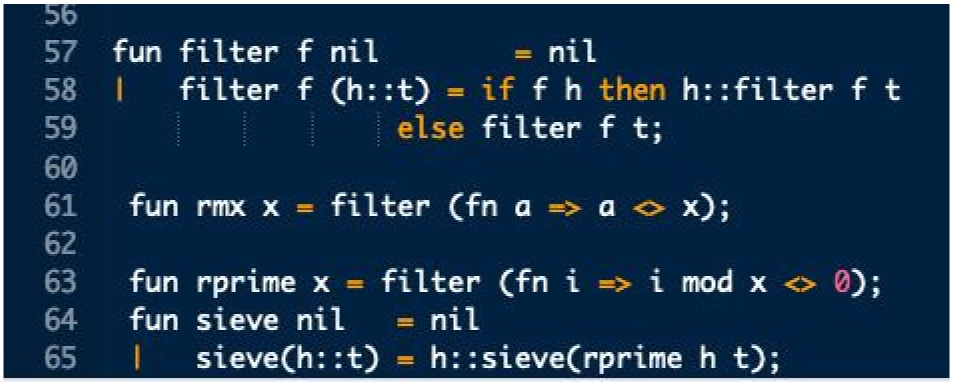
| sieve(h::t) = h::sieve(rprime h t);



```
fun index(0, h::t)  = h
|   index(n, h::t)  = index(n-1, t);
fun takeN(0, h::t)  = nil
|   takeN(n, h::t)  = h :: takeN(n-1, t);
fun dropN(0, x)     = x
|   dropN(n, h::t)  = dropN(n-1,t);

fun insert (n:int) nil = [n]
|   insert n (h::t)    = if (n<h) then n::h::t else h::(insert n t);

fun sort nil    = nil
|   sort (h::t) = insert h (sort t);

fun upto(a, b) = if a < b then a::upto(a + 1, b)
                    else if a = b then [a]
                    else nil;

fun dropSpace nil   = nil
|   dropSpace(h::t) = if ord(h)=ord(#" ") then dropSpace t else h::t;

fun takeSpace nil   = nil
|   takeSpace (h::t)= if ord(h)=ord(#" ") then h::takeSpace(t)
                    else nil;

 fun dropNonSpace nil   = nil
|   dropNonSpace(h::t) = if ord(h)<>ord(#" ") then dropNonSpace t else h::t;
```



```
fun takeNonSpace nil    = nil
|   takeNonSpace (h::t)= if ord(h)<>ord(#" ") then h::takeNonSpace(t)
                       else nil;

val firstWord = implode o takeNonSpace o dropSpace;
val butFirstWord = implode o dropSpace o dropNonSpace o dropSpace;

val ftrl = map(fn x => 3*x);
val fhel = map(fn s => hd(explode(s)));
val fttl = map(fn s => implode(dropN(2, explode(s))));
val fsml = map(fn s => "s" ^ s ^ "m");


fun    reduce f b nil      = b
|  reduce f b (h::t)    = f(h,reduce f b t);

val r = reduce (fn(a:int,b)=>b @ [a]) nil;
val p = reduce (fn(a:int, b)=>a::b);
val dr = reduce (fn(a,b)=>a+10*b) 0;
fun m x = reduce (fn(a,b)=>(a=x) orelse b) false;
fun n x = reduce (fn(a,b)=>(a=x) andalso b) true;
val im = reduce (op ^) "";
val ts = reduce (fn(a,b)=>if a=" " then nil else a::b) nil;

val prodList = reduce (fn(a:int, b:int) => a * b) 1;
val flatten = reduce (fn(a:int list, b:int list) => a @ b) nil;
val count = reduce (fn(a:int, b:int) => b + 1) 0;
val duplist = reduce (fn(a:int, b:int list) => [a, a] @ b) nil;
```



```
fun filter f nil         = nil
|   filter f (h::t) = if f h then h::filter f t
                      else filter f t;

 fun rmx x = filter (fn a => a <> x);

 fun rprime x = filter (fn i => i mod x <> 0);
 fun sieve nil   = nil
 |   sieve(h::t) = h::sieve(rprime h t);
```

```
- stdIn:1.6-2.36 Warning: match nonexhaustive
          (0,h :: t) => ...
          (n,h :: t) => ...

val index = fn : int * 'a list -> 'a
stdIn:3.5-4.41 Warning: match nonexhaustive
          (0,h :: t) => ...
          (n,h :: t) => ...

val takeN = fn : int * 'a list -> 'a list
stdIn:5.5-6.35 Warning: match nonexhaustive
          (0,x) => ...
          (n,h :: t) => ...

val dropN = fn : int * 'a list -> 'a list
val insert = fn : int -> int list -> int list
val sort = fn : int list -> int list
val upto = fn : int * int -> int list
val dropSpace = fn : char list -> char list
val takeSpace = fn : char list -> char list
val dropNonSpace = fn : char list -> char list
val takeNonSpace = fn : char list -> char list
val firstWord = fn : char list -> string
val butFirstWord = fn : char list -> string
val ftrl = fn : int list -> int list
val fhel = fn : string list -> char list
val fttl = fn : string list -> string list
val fsml = fn : string list -> string list
val reduce = fn : ('a * 'b -> 'b) -> 'b -> 'a list -> 'b
val r = fn : int list -> int list
val p = fn : int list -> int list -> int list
val dr = fn : int list -> int
stdIn:47.31 Warning: calling polyEqual
val m = fn : ''a -> ''a list -> bool
stdIn:48.31 Warning: calling polyEqual
```

```
val n = fn : ''a -> ''a list -> bool
val im = fn : string list -> string
val ts = fn : string list -> string list
val prodList = fn : int list -> int
val flatten = fn : int list list -> int list
val count = fn : int list -> int
val duplist = fn : int list -> int list
val filter = fn : ('a -> bool) -> 'a list -> 'a list
stdIn:61.32-61.34 Warning: calling polyEqual
val rmx = fn : ''a -> ''a list -> ''a list
val rprime = fn : int -> int list -> int list
val sieve = fn : int list -> int list
```