- ocaml: no points for missing ;;
- scheme: look at indentation, mismatching () is fine
- any correct answer w/ good style is acceptable
- 50pts total

Question 1. [1+1+1+1+2]

```
Object subclass: Stack [
  |array top|
  Stack class >> new [ ^ Stack new: 16 ]
  Stack class >> new: size [ ^super new init: size ]
  init: size [
    top := 0.
    array := Array new: size.
  ]
]
```

- `Stack class >> new [ ^ super new init: 16 ]` is also fine
- `new` and `new:` are distinct and should be graded as such

Question 2. [2+1+1+1+1]

```
pop [
  |result|
  result := array at: top.
  top := top - 1.
  ^ result.
]

pop [
  top := top - 1.
  ^ array at: top + 1.
]

push: item [
  top := top + 1.
  array at: top put: item
]

empty [ ^ top = 0 ]
full [ ^ top = array size ]
count [ ^ top ]
```

- periods at the end of blocks are optional
- missing periods are fine

Question 3. [4]

```
let rec unzip list = match list with
  | [] -> ([],[])
  | (a,b)::rest -> let (al,bl) = unzip rest
                   in (a::al, b::bl);;
```

- 1pt: first line match and properly handles [] case
- 1pt: pattern for rest of match
- 1pt: recursive call
- 1pt: correct return result

Question 4. [3+3]

```
(define (reverse list)
  (define (rev in out)
     (if (null? in) out
         (rev (cdr in) (cons (car in) out))))
     (rev list '()))
```

- 1pt: uses nested inner function
- 1pt: checks for accumulator and null? check
- 1pt: proper recursive call
- assign 0 if all calls are not tail calls.

```
(define (foldl_rev list)
   (foldl (lambda (a d) (cons a d)) '() list))
```

- 1pt: general structure
- 1pt: inner lambda present
- 1pt: inner lambda correct.

Question 5. [6]

```
#!/usr/bin/perl
use strict;
use warnings;

my $lines = 0;
my $words = 0;
my $chars = 0;

while (defined (my $line = <>)) {
  ++$lines;
  $chars += length $line;
  while ($line =~ s/\S+//) { ++$words }
}

print "$lines $words $chars\n";
```

- ok to forget the `#!`
- ok to omit the strict/warnings declarations
- ok to use postfix `++` or prefix `++`
- ok to use `+=1` instead of `++`
- ok for wrong `{}` or `()`, or missing semicolons
- 1pt: declaration of lines, words, chars
- 1pt: loop, `++lines`
- 1pt: correctly incr chars
- 2pt: counting words in the line (note alternatives)
- 1pt: prints results (printf is OK)

alternates:

```
$words++ while $line =~ s/\S+/;
```

```
my @words = $line =~ m/(\S+)/g; $words += @words;
```

Question 6. [5]

```
let rec zipwith fn list1 list2 = match list1, list2 with
  | [], [] -> []
  | _, [] -> failwith "zipwith"
  | [], _ -> failwith "zipwith"
  | x::xs, y::ys -> fn x y :: zipwith fn xs ys;;
```

- 1pt: correct match
- 1pt: `[],[]` is correct and ahead of `failwith`
- 1pt: correctly matched `failwith`
- 1pt: correct recursive call
- 1pt: correct `::` in last `-> result`

Question 7. [4]

- accept any correct answer, many possible
- -1pt for calling `length`
- -1pt for non-tail-recursive
- score 0 if the entire list was copied
- name can be `drop`, `drop1`, `drop2`, `drop3` (exam typos)

```
let rec drop1 n list = match list with
  | [] -> []
  | _::xs when n > 0 -> drop (n - 1) xs
  | _ -> list ;;
```

- 1pt: `[]`
- 1pt: `_::cs`
- 1pt: uses `when` (if they choose this version)
- 1pt: `_ ->` provided it is the LAST alternative

```
let rec drop2 n list =
  if n <= 0 then list
  else match list with
    | [] -> []
    | _::xs -> drop (n - 1) xs;;
```

- 1pt: for `<= 0` test
- 1pt: `| []`
- 1pt: `_::xs` here order of matches does not matter
- 1pt: proper tail recursion

```
let rec drop3 n list = match list ith
  | [] -> []
  | _::xs -> if n <= 0 then list
             else drop3 (n - 1) xs;;
```

- 1pt: for `| []`
- 1pt: `_::xs` match
- 1pt: correct `if then else`
- 1pt: proper tail recursion

Question 8. [3]

- accept any correct answer, many possible
- -1pt for calling `length`
- -1pt for non-tail-recursive
- score 0 if the entire list was copied

```
(define (drop1 n list)
  (cond ((null? list) '())
        ((<= n 0) list)
        (else (drop1 (- n 1) (cdr list)))))
```

- 1pt: `(define` line is correct
- 1pt: check for `null?`
- 1pt: check for `n <= 0`
- see above for catchall penalties
- ok to use nested `if`s instead of `cond`.
- ok to reverse order of `null` and `<=` tests

```
(define (drop2 n list)
  (if (or (null? list) (<= n 0)) list
      (drop2 (- n 1) (cdr list))))
```

- 1pt: `(define` line is correct
- 1pt: `or` test is correct
- 1pt: for tail call
- see above for catchall penalties

Question 9. [1+1]

- 0.5 points for the `target:prereq`
- 0.5 points for the command

```
%.o : %.c
    gcc -c $<

clean :
    - rm *.o
```

Question 10. [6]

```
#!/usr/bin/perl
use strict;
use warnings;
my %hash;

while (my $line = <>) {
   while ($line =~ s/\w+//) {
      ++$hash{$&}
   }
}

for my $key (sort keys %hash) {
   print "$key $hash{$key}\n";
}
```

- first four lines are all optional
- ok for prefix or postfix `++` or `+=1`
- ok if they have strange `() {}` or missing semicolons
- ok if print statement uses separate arguments
  - `print $key, $hash{$key}, "\n";`
- ok for extra or missing semicolons
- alt 1st inner loop:
  - `++$hash{$&} while $line =~ s/\w+//;`
- alt print loop:
  - `print "$_ $hash{$_}\n" for sort keys %hash;`
- 1pt: outer while loop
- 1pt: inner while loop
- 1pt: incr hash element using `$&`
- 1pt: for loop
- 1pt: keys of hash are sorted
- 1pt: print statement (OK to use printf instead)

Question 11. [0.5+0.5+0.5+0.5]

- universal + inclusion
- universal + parametric
- ad-hoc + overloading
- ad-hoc + conversion