NOTE: These are double-sided pages!

**Grammars**

1. (10 pts) Consider the following grammar:
   
   $S ::= aE | bE | c$
   
   $E ::= fGf | S$
   
   $G ::= 1 | \epsilon$

   where $\epsilon$ denotes the empty string.

   Which of the following strings is in the language generated by this grammar?

   (a) abc
   (b) afcf
   (c) abbbabc
   (d) affa

2. (20 pts) Given the following grammar

   $E ::= T * E \mid T / E \mid T$
   
   $T ::= F + T \mid F - T \mid F \mid \text{neg } E$
   
   $F ::= \text{number} \mid \text{id} \mid (T)$

   where "neg" denotes unary negation.

   **Caution:** Note that this grammar looks superficially the same as grammars we have used extensively in class but is in fact rather different.

   *(Question continued on next page...)*
The grammar again:

\[ E ::= T \cdot E \mid T / E \mid T \]
\[ T ::= F + T \mid F - T \mid F \mid \text{neg } E \]
\[ F ::= \text{number} \mid \text{id} \mid (T) \]

(a) Draw a parse tree for the following expression:

```
      E
     / \n    T   *
   /   /
  F   E
 /     /
num(5) F +
   /   /
  num(3) F
```

(b) Draw an abstract syntax tree for the same expression.

```
      F
     / \\
    *   /  \n   /    3   neg
  /      /
num(5) 2
```

(c) What is the value of the expression, according to the grouping given by this grammar?

\(-35\)
Functional Programming

3. (15 pts) (a) Write a Scheme function `predcount` that takes two arguments:
   (i) a function `pred` representing a predicate
   (ii) a list of numbers `ls`

   The function should return a count of the numbers in `ls` that satisfy the predicate:
   
   ```scheme
   (predcount even? '(1 2 4 5 10 17))
   ```

   ```scheme
   (define predcount
     (lambda (pred ls)
       (if (null? ls) 0
           (if (pred (car ls))
               (+ 1 (predcount pred (cdr ls))))
               (predcount pred (cdr ls))))))
   ```

   (b) Write `predcount` in ML.

   ```ml
   fun predcount p nil = 0
     | predcount p (x::xs) =
       if p x then 1 + (predcount p xs)
       else predcount p xs;
   ```

   (c) Give an example of a list that will successfully return a value from the Scheme version of `predcount` but fail in the ML version. Explain why the Scheme version works but the ML version does not.

   Any list that mixes integers and floating point, such as [1, 1.17]. Scheme is dynamically typed — it accepts heterogeneous lists. ML is statically typed — all values in a list must be of the same type.
4. (15 pts) Given the ML datatype

```ml
datatype Exp = Number of real
              | Binary of Exp * BinOp * Exp
```

datatype BinOp = Plus | Times

(i) Write a function `eval` that given an expression returns the value of the expression:
```
- eval Binary (Number (5.0), Plus, Binary (Number (3.2), Times, Number (2.0)));
```

```ml
fun eval (Number n) = n
| eval (Binary (left, Plus, right)) = 
  eval (left) + eval (right)
| eval (Binary (left, Times, right)) = 
  eval (left) * eval (right);
```

you could also use just two function patterns and a case on the binary operator value.

(ii) What is the type ML would assign to your function?

```
Exp -> real
```
5. (20 pts) Give a type derivation for each of the following ML functions:

```
fun a nil = 0
  | a (x::xs) = if even x then 1
               else (a xs)
```

1. \(\alpha \rightarrow \beta\) \hspace{1cm} \text{(definition)}
2. \(\alpha\) \hspace{1cm} \text{(result of \(a\))}
3. \(\alpha\) \hspace{1cm} \text{(constant)}
4. \(\alpha = \text{int}\) \hspace{1cm} \text{(parameter to \(a\))}
5. \((x::xs):\alpha\) \hspace{1cm} \text{(parameter is a list)}
6. \((x::xs):\gamma \text{ list}\) \hspace{1cm} \text{(: : result is a list)}
7. \(x:\gamma\) \hspace{1cm} \text{\((x \text{ is left of } ::)\)}
8. \(\text{even: int } \rightarrow \text{ int}\) \hspace{1cm} \text{(definition of \(\text{even}\))}
9. \(x: \text{int}\) \hspace{1cm} \text{\((x \text{ parm to even)}\)}
10. \(\gamma = \text{int}\) \hspace{1cm} \text{(7,9)}
11. \(\alpha = \gamma \text{ list}\) \hspace{1cm} \text{(5,6)}
12. \(\alpha = \text{int list}\) \hspace{1cm} \text{(10,11)}
13. \(\alpha: \text{int list } \rightarrow \text{ int}\) \hspace{1cm} \text{(4,12)}

```
fun twice f x = f ( f (x))
```

1. \(\text{twice: } \alpha_1 \rightarrow \alpha_2 \rightarrow \beta\) \hspace{1cm} \text{(definition)}
2. \(\alpha\) \hspace{1cm} \text{(first parameter to twice)}
3. \(\alpha \rightarrow \alpha_1\) \hspace{1cm} \text{(\(\alpha\) is a function)}
4. \(\alpha_1 = (\alpha \rightarrow \alpha_1)\) \hspace{1cm} \text{(2,3)}
5. \(x: \alpha_2\) \hspace{1cm} \text{(2nd parm to twice)}
6. \(x: \alpha\) \hspace{1cm} \text{(parm to \(f\))}
7. \(\alpha = \alpha_2\) \hspace{1cm} \text{(5,6)}
8. \((f x): \alpha_1\) \hspace{1cm} \text{(result of \(f, 3\))}
9. \((f x): \alpha\) \hspace{1cm} \text{(\(f x\) is parm to \(f, 3\))}
10. \(\alpha_1 = \alpha\) \hspace{1cm} \text{(8,9)}
11. \((f (f x)): \beta\) \hspace{1cm} \text{(result of \(f, 2\))}
12. \((f (f x)): \alpha\) \hspace{1cm} \text{(result of \(f, 9\))}
13. \(\beta = \alpha\) \hspace{1cm} \text{(11,12)}
14. \(\text{twice: } (\alpha \rightarrow \alpha) \rightarrow \alpha \rightarrow \alpha\) \hspace{1cm} \text{(1,4,7,13)}
fun filter pred nil = nil
  | filter pred (x::xs) = if pred x then x::(filter pred xs) else filter pred xs

1. filter : α₁ → α₂ → β (definition)
2. pred : α₁
   (1st parm to filter)
3. pred : α → α'
   (pred is a function)
4. α₀ = α → α'
   (2, 3)
5. (x::xs) : α₂
   (second parm to filter)
6. (x::xs) : β list (=: returns a list)
7. x : β
   (x is left parm to : =, 6)
8. x : α
   (x is parm to pred)
9. α₀ = β
   (7, 8)
10. x₀ : β list (5, 6)
11. (pred x) : α'
    (result of pred, 3)
12. (pred x) : bool (if condition)
13. α₀' = bool (11, 12)
14. (x::(filter pred xs)) : β
15. (x::(filter pred xs)) : α list
16. β = α list (14, 15)
17. filter : (α → bool) → α list

(ii) Which of the above functions is polymorphic?

trunc and filter: they have a type
variable (α) in their type
6. (20 pts) Given the Java code fragment:

```java
public class parms_test
{
    int a = 3;
    int b = 4;
    int j = 0;
    int k [] = new int [5];
    void f (int c, int d)
    {
        int a = 1;
        j = c;
        d = c + j;
    }
    void g ()
    {
        for (int i = 0; i < 5; i++)
            k [i] = i;
        f (a, k[j]);
        // HERE
    }
}
```

Assume the function "g" is called. What values would each of the instance variables of `parms_test` have at the point marked "HERE", assuming call–by-value, call–by-reference, call–by-name, and macro expansion? Give your answers by filling in the following table. The first column is the initial values for each of the variables. Complete the following table, listing the values each of the variables would have at HERE for each of the parameter–passing mechanisms:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Initial value</th>
<th>By–value</th>
<th>By–reference</th>
<th>By–name</th>
<th>Macro expansion</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>j</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>k[0]</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>k[1]</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>k[2]</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>k[3]</td>
<td>3</td>
<td>3</td>
<td>16</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>k[4]</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>