1. (2 p)

Lösning:

```
member(L, [[1,2],[3,4,5],[],[6]]), member(4, L).
```

Rättning:

- 1p om två anrop till member, med rätta argument, fast kanske inte i rätt ordning.
- 1p om dessutom ordningen på predikaten och dess parametrar är rätt.

2. (8 p)

Lösning:

Fråga: `length(L, 2).`:

(1) Misslyckas med första regeln, därför att 0 och 2 inte kan unifieras;

(2) Skapar instans av andra regeln:
`length([A1|T1], N1) :- length(T1, NT1), N1 is NT1+1.`
unifierar: `L=[A1|T1]`, `N1=2`

(3) Delfråga: `length(T1, NT1)`:

(3.1) lyckas med första regeln; unifierar: `T1=[]`, `NT1=0`

(4) `2 is 0+1.`

(4.1) evaluerar 0+1 till 1;

(4.2) misslyckas, därför att 2 och 1 inte kan unifieras;

(4.3) backtrackar; unifieringen av `T1` och `NT1` tas bort

(5) Tillbaka till delfrågan `length(T1, NT1)`

(5.1) Skapar instans av andra regeln:
`length([A2|T2], N2) :- length(T2, NT2), N2 is NT2+1.`
unifierar: `T1=[A2|T2]`, `N2=NT1`

(5.2) Delfråga: `length(T2, NT2).`

(5.2.1) Lyckas med första regeln; unifierar: `T2=[]`, `NT2=0`;

(5.3) `NT1 is 0+1.`

(5.3.1) evaluerar 0+1 till 1;

(5.3.2) unifierar: `NT1=1`;

(6) `2 is 1+1.`

(6.1) evaluerar 1+1 till 2;

(6.2) lyckas, därför att 2 kan unifieras med 2.

Svar: `L=[A1|[A2|[]]]` som presenteras som `L=[A1, A2]`.

Rättning:

- 3p för huvuddragen i kontrollflödet: först försöker vi med en lista av längd 0, sedan försöker vi med en lista av längd 1, och till sist med en lista av längd 2. Ett poängs avdrag från detta om försöket med lista av längd 0 saknas.
- 1p om exakt angivit vart backtrackingen uppstår (att det är i evalueringen av `is` i steg (4))
- 1p om beskrivit instanserna av regler med temporära variabler som skapas.
- 2p om rätt unifieringar vid rätt tillfällen. Det finns två typer av unifieringar: de som sker vid regelmatchning i (2), (3.1), (5.1), (5.2.1), och de som sker vid `is`-instruktioner i (4.2), (5.3.2), och (6.2).
  Inget avdrag om den senare typen saknas.
  Ett poängs avdrag om två av (2), (3.1), (5.1), (5.2.1) saknas.
  Inga poäng om tre eller fyra av (2), (3.1), (5.1), (5.2.1) saknas.
- 1p om rätt svar

3. (10 p)

Lösning:

(a) Sammansättningar är lättast att representera som listor. I princip handlar det här om träd där varje nod har ett variabelt antal barn (och inte bara två som är fallet med binära träd). Trädets löv är systemets funktioner i denna representation.

Exempel 1: `[main, [min, max, avg]]`
Exempel 2: `[[foo, bar], [[open, close], [read, eval, print]]]`

(b) Möjlig lösning för den föreslagna representationen:

```
funcs(S, [S]) :- atom(S).
funcs([], []).
funcs([S | SS], FF) :-
   funcs(S, FFS),
   funcs(SS, FFSS),
   append(FFS, FFSS, FF).
```

Om man ska vara noggrann så bör den första regeln egentligen vara
`funcs(S, [S]) :- atom(S), \+ (S = []).`
eftersom `[]` också betraktas som en atom i Prolog.

Rättning:

(a) Totalt 4 poäng möjliga

- 2p om representationen motsvarar träd där varje nod har ett variabelt antal barn. Ett poängs avdrag om löven är någon annan datatyp än atomer.
- 1p om första exemplet är icke-trivialt och korrekt gentemot definitionen.
- 1p om andra exemplet är icke-trivialt och korrekt gentemot definitionen.

(b) Totalt 6 poäng möjliga

- 1p för korrekt första basfall (atom).
- 1p för korrekt andra basfall (tom lista).
- 2p för korrekt rekursion i den tredje regeln (i valfri ordning).
- 1p dessutom om append (eller liknande) i tredje regeln.
- 1p dessutom om append kommer efter funcs i tredje regeln.
- inget avdrag om man räknar tom lista som en funktion.

Andra korrekta lösningar accepteras också. Använd då ett liknande sätt att dela ut poäng som för lösningen ovan efter bästa förmåga, och sätt ett frågetecken bredvid poängen i uppgiftsprotokollet så att vi vet att vi ska dubbelkolla rättningen.