Facit och kommentarer till tentamen 2026-03-09 i DA4003

Del 1: flervalsfrågor (1p per fråga)

1. C
2. A, B, D, E
3. D
4. A, C, E
5. E
6. D
7. C
8. E

Del 2: kodfrågor

9. (a) Möjlig lösning:

```
int* filter_positive(int* p,int n) {

  int c = 0;

  // First calculate how many positive numbers there are
  for (int i = 0; i < n; i++)
    if (*(p + i) > 0)
      c++;

  // Then allocate c + 1 numbers
  int *out = malloc((c + 1) * sizeof(int));

  // Write the first part of the output
  *out = c;

  // Start counting from second index for the rest of the output
  int j = 1;

  // Write all the positive numbers to output sequence
  for (int i = 0; i < n; i++)
    if (*(p + i) > 0) {
      *(out + j) = *(p + i);
      j++;
    }

  return out;
}
```

(b) Möjlig lösning:

```
int p[4] = {1,-2,3,-4}; // Could also be done with malloc

int *q = filter_positive(p,4);
int c = q[0];

// The sum to be computed
int s = 0;

for (int i = 1; i <= c; i++)
```

```
  s += q[i];   // same as *(q + i)

printf("The sum of the positive numbers is: %d\n",s);
```

10. (a) Möjlig lösning:

```
class Item {

    private String name;
    private Double price;

    public Item(String n,Double p) {
        name = n;
        price = p;
    }

    public String getName() {
        return name;
    }

    public Double getPrice() {
        return price;
    }

    public void setPrice(Double p) {
        price = p;
    }

}
```

(b) Möjlig lösning:

```
class Produce extends Item {

    private Double price_per_kg, weight;

    public Produce(String n, Double ppk, Double w) {
        super(n,ppk*w);
        price_per_kg = ppk;
        weight = w;
    }
}
```

(c) Möjlig lösning:

```
Produce bananas = new Produce("Banana",21.0,2.0);
```

11. (a) Möjlig lösning:

```
addPositive :: [Int] -> Int
addPositive [] = 0
addPositive (x:xs) | x > 0 = x + addPositive xs
                   | otherwise = addPositive xs

-- Alternativt:
addPositive' :: [Int] -> Int
addPositive' = sum . filter (>0)
```

(b) Möjlig lösning:

```
findIndex :: Eq a => a -> [a] -> Int
findIndex x xs = findIndexHelper x xs 0
  where
```

```
    findIndexHelper :: Eq a => a -> [a] -> Int -> Int
    findIndexHelper _ [] _ = -1
    findIndexHelper x (y:ys) i | x == y    = i
                               | otherwise = findIndexHelper x ys (i+1)
-- Alternativt
findIndex' :: Eq a => a -> [a] -> Int
findIndex' x xs = maybe (-1) id $ lookup x $ zip xs [0..]
```

(c) Möjlig lösning:

```
import Prelude hiding (zip)

zip :: [a] -> [b] -> [(a,b)]
zip [] _ = []
zip _ [] = []
zip (x:xs) (y:ys) = (x,y) : zip xs ys
```

12. (a) Möjlig lösning:

```
connected(X,X) :- !.
connected(X,Y) :-
    edge(X,Z),
    connected(Z,Y), !.
```

(b) Möjlig lösning:

```
path(X,X,[X]).
path(X,Y,[X|ZS]) :-
    edge(X,Z),
    path(Z,Y,ZS).
```

(c) Möjlig lösning:

```
distance(X,Y,N) :-
    path(X,Y,XS),
    length(XS,M),
    N is M - 1.
```