1) (Text answer) Consider the following program fragment:

```
int x;
int y;
int z;
int* w;
int* q;
x = 0;
y = 1;
z = 2;
w = &x;
q = &y;
*w = y;
*q = z;
*w = x + y + z + *q;
*q = x + y + z + *w;
printf("x=%d, y=%d, z=%d",x,y,z);
```

What does the program print when it is executed?

Programmet printer x = 7, y = 18, z = 2.

Da w og q gemmer sine værdier i x og y, og w peger på y og q på z. Selv om q pointer på z så ændre z ikke værdi som om *q gør, det er fordi *q gemmer sine værdier i y.

2) Write a function *int max(int * numbers, int size)* that, given an array of numbers(and it's size), finds the maximum value in the array.

You may assume that the array it not empty. Include assertions in the implementation of *max* to ensure that the precondition is fulfilled when executing the function.

Programmet blev testet uden nogle fejl. Derfor kan det konkluderes at funktionen kan finde det største tal i et givet array.

3) (Text and Code answer) Consider the following program:

```
1  #include <stdio.h>  /*printf*/
2  #include <assert.h> /*assert*/
3  #include <stdlib.h> /*malloc*/
4
5  typedef struct node {
6    int data;
7    struct node *next;
8  } node;
9
10 void add(node *head, int x){
11   /*pre: head points to the first, empty element.
12              The last element's next is NULL
13     post: a new node containing x is added to the end of the list*/
14   assert(head!=NULL);
15   node *p = head;
16   while (p->next!=NULL) {
17     p = p->next;
18   } /*p points to the last element*/
19   node *element = malloc(sizeof(node));
20   element->next = NULL;
21   element->data = x;
22   p->next = element;
23 }
24
25 int main(void) {
26   node *list = malloc(sizeof(node));
27   list->next = NULL; /*create first, empty element*/
28   add(list,1);
29   add(list,3);
30   add(list,2);
31  /*show list here*/
32   add(list,2);
33  /*show list here*/
34   return 0;
35 }
```

a) Draw two diagrams that shows list at */ *show list here* */ in main.

Diagram på linje 31:

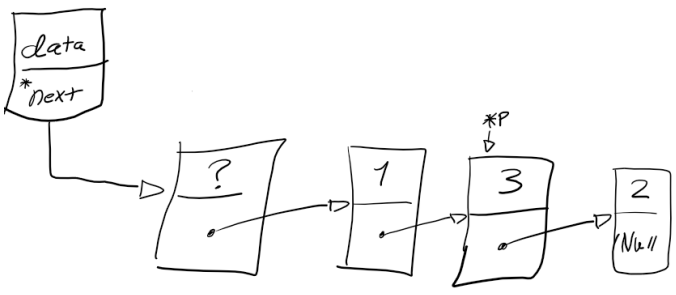

Figure 1Diagram linje 31

Diagram på linje 43:

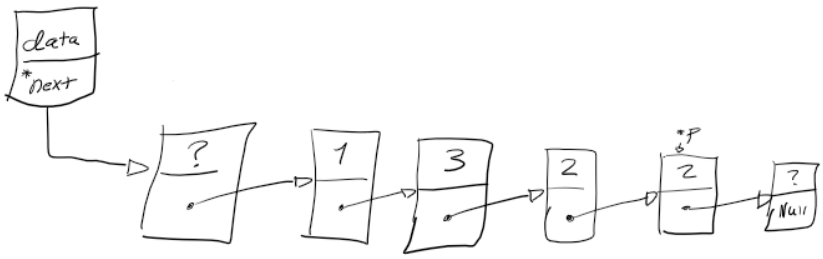

Figure 2Diagram på linje 43

b) Implement a function with the following signature *int size(node *l)*. It has the same precondition as *add* and returns the number of elements in the list.

Programmet kører uden fejl og kan finde antallet af element i list.

c) What does the following code do when executed?

```
void printout(node *l) {
  /*pre: head points to the first, empty element.
          The last element's next is NULL
    post: the values of the list are printed out*/
    node *p = l->next;
    while (p!=NULL){
      printf("\%d, ",p->data);
    }
    printf("\n");
}
```

P peger på l->next, da der ikke er noget data i det første struct. Hvis p så er null så ved vi at der ikke er nogle ekstra elementer og kan derfor ikke printe nogle dataer.

Når der kommer et element med data, går programmet ind i loopet og nu printer den p->data i nuværende element. Næste element gør den præcis det samme, bortset fra at den printer data fra det tidligere element.

Dette er fordi p ikke bliver sat til at pege på det næste element og sidder derfor stadig fast på det første element og kommer til at gøre det igennem hele programmet.

d) Correct the function above so that the post condition is fulfilled

I sidste delopgave blev det oplyst at p skal pege på p->next for at ikke side først på samme element.

Se kode for løsning.

e) Write a function *int largest(node *l)*. The pre- and post-conditions are the following.

```
/*pre: head points to the first, empty element.
           The last element's next is NULL. size(l>0)
  post: returns the largest value of the list*/
```

Alle programmer kører nu uden nogle fejl. Derfor kan det antages at alle funktioner igennem opgaven gøre lige hvad de skal.