(1) (Text answer) (old exam question) Consider the following program fragment:

```
1   int x;
2   int y;
3   int z;
4   int* w;
5   int* q;
6   x = 0;
7   y = 1;
8   z = 2;
9   w = &x;
10  q = &y;
11 *w = y;
12 *q = z;
13 *w = x + y + z + *q;
14 *q = x + y + z + *w;
15 printf("x=%d, y=%d, z=%d",x,y,z);
```

What does the program print when it is executed?

Answer: _____, _____, _____

$x = 7, y = 18, z = 2$

(2) (code answer) Write a function `int max(int* numbers, int size)` that, given an array of numbers (and its size), finds the maximum value in the array. You may assume that the array is not empty (i.e. size > 0). Include assertions in the implementation of `max` to ensure that the precondition is fulfiled when executing the function

(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 }
```

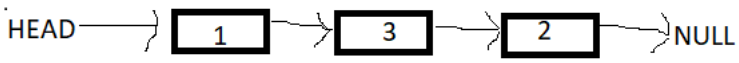
(a) Draw two diagrams that shows `list` at `/*show list here*/` in `main`. **Note**: The first element is empty and holds no data. I.e. if I have a list with two elements, it has three nodes (the first, empty one and then two nodes holding data). The same definition is used in all functions.

(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. E.g. if `size(list)` was printed out at the first `/*show list here*/`) in `main`, the result would be 3.

(c) What does the following code do when executed? (i.e. do the code fulfil the post condition? If not, what happens?)

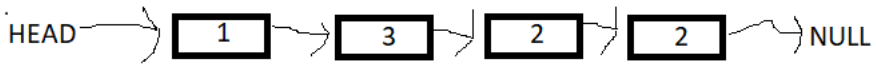
a)

1.



2.

2.



b)

Done

c)

It does not fulfill the postcondition, instead it just prints the first non-empty element of the list infinitely.

d)

```
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");
}
```

(d) Correct the function above so that the post condition is fulfilled
(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*/
```

d)

Done

e)

Done

Link to GitHub

GitHub

(4) **Challenge:** (PC-1.6.6) *Interpreter.* A certain computer has ten registers and 1,000 words of RAM. Each register or RAM location holds a three-digit integer between 0 and 999. Instructions are encoded as three-digit integers and stored in RAM. The encodings are as follows

100	means *halt*
2*dn*	means *set register d to n (between 0 and 9)*
3*dn*	means *add n to register d*
4*dn*	means *multiply register d by n*
5*ds*	means *set register d to the value of register s*
6*ds*	means *add the value of register s to register d*
7*ds*	means *multiply register d by the value of register s*
8*da*	means *set register d to the value in RAM whose address is in register a*
9*sa*	means *set the value in RAM whose address is in register a to that of register s*
0*ds*	means *goto the location in register d unless register s contains 0*

All registers initially contain 000. The initial content of the RAM is read from standard input. The first instruction to be executed is at RAM address 0. All results are reduced modulo 1.000.