Week 6 Programming Assignment

Steffen Petersen — au722120

October 10th 2022

Here is the link for my repository, in which you will find all the edited code files and such.
https://github.com/Aarhus-University-ECE/assignment-6-SirQuacc

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: _____, _____, _____

The program would print: 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

Below is the function itself, it can also be found in max.c

```
1  int max(int* numbers, int size)
2  {
3      assert(size > 0);
4      int highest = numbers[0]; // Initialize the "highest" variable as the first element in
           the given array to start.
5      for(int i = 1; i < size; i++){ // Run through the array from 1 -> n-1
6        if(numbers[i] > highest) // Test whether the current slot in the array has a higher
             value than the current saved highest value.
7          highest = numbers[i]; // If above is true, highest is reassigned as the value of the
               current spot in the array.
8      }
9      return highest;
10 }
```

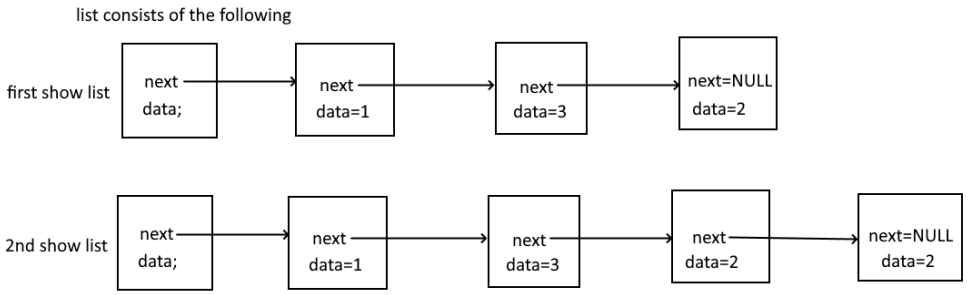
3

(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.

Here is my diagram showing the list at the two /*show list here*/ places in the code.



(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.

The code is seen below and can be found in list.c

```
int size(node *l){
  assert(l!=NULL); // Same precond, input needs to exist
  int count = 0; // Start counter for size at 0
  while(l->next!=NULL){
    l = l->next; //Progress through the list
    count++; //while the current node's "next" pointer is defined, count 1 up.
  }
  return count;
}
```

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

```
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");
}
```

No the code does not fulful the precondition, because the while condition only checks whether the current node p exists, and also it doesn't ever progress through the list.

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

The function's while condition would need to check p−>next instead of just p, and it would have to reassign p = p−>next before finishing the while loop.
The corrected function is below, and in list.c

```
void printout(node *l) {
    node *p = l->next;
    while (p->next!=NULL){ //Check if we are at the final node
      printf("%d\n, ",p->data);
      p = p->next; //Continue to next node.
    }
}
```

(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*/
```

Below is my code for this function, it can of course also be found in list.c

```
1  int largest(node *l){
2    /*Excercise 3e) Add your code below.*/
3    assert(l!=NULL); //assert l is defined
4    if (l->next == NULL) return -1; //Return -1 if nothing is in the input list
5    node *p = l->next; //Create local pointer p, that points to the same as inputs next
6    int i = 0; //Counter variable
7    int *dataElements = malloc(sizeof(int) * size(l)); //Allocate memory for data elements in
         an int array of the same size as the input's length.
8    for(int i = 0; i < size(l); i++){
9      dataElements[i] = p->data; //Input current data into data array
10     if(p->next != NULL) p = p->next; //If not at final node, continue
11   }
12   int highest = max(dataElements, size(l)); // Call our max function to get the highest value
         in the int array of data.
13   return highest;
14    /*pre: head points to the first, empty element. The last element's next is NULL. size(l>0)
15     post: returns the largest value of the list*/
16 }
```