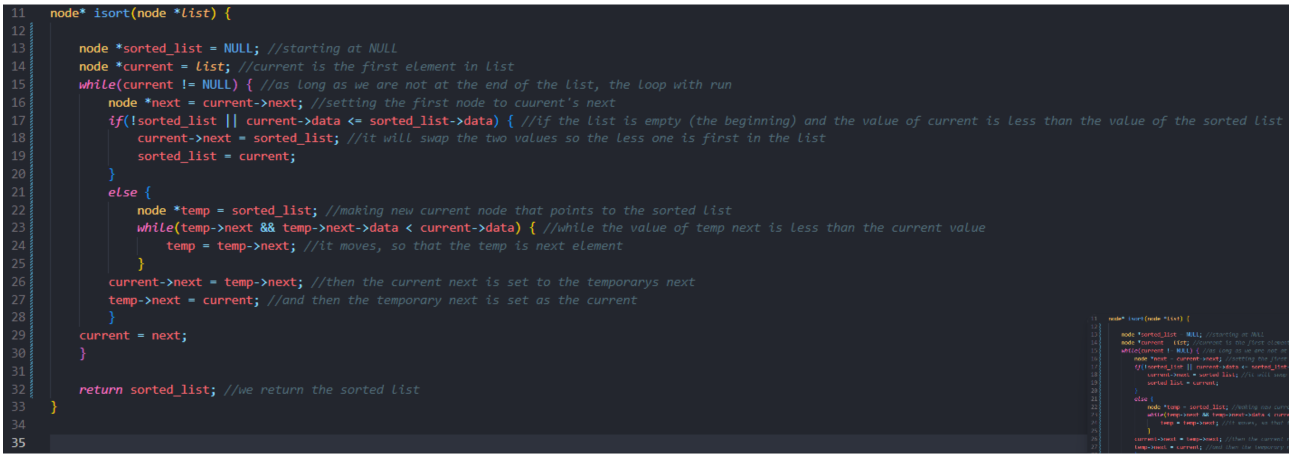
Opgave 1

a) Tæller i ++ (en addition)
Gange i*f (en multiplikation)
5*2 operationer = 10 operationer, fordi i starter på 1 og skal tælle op til og med 5

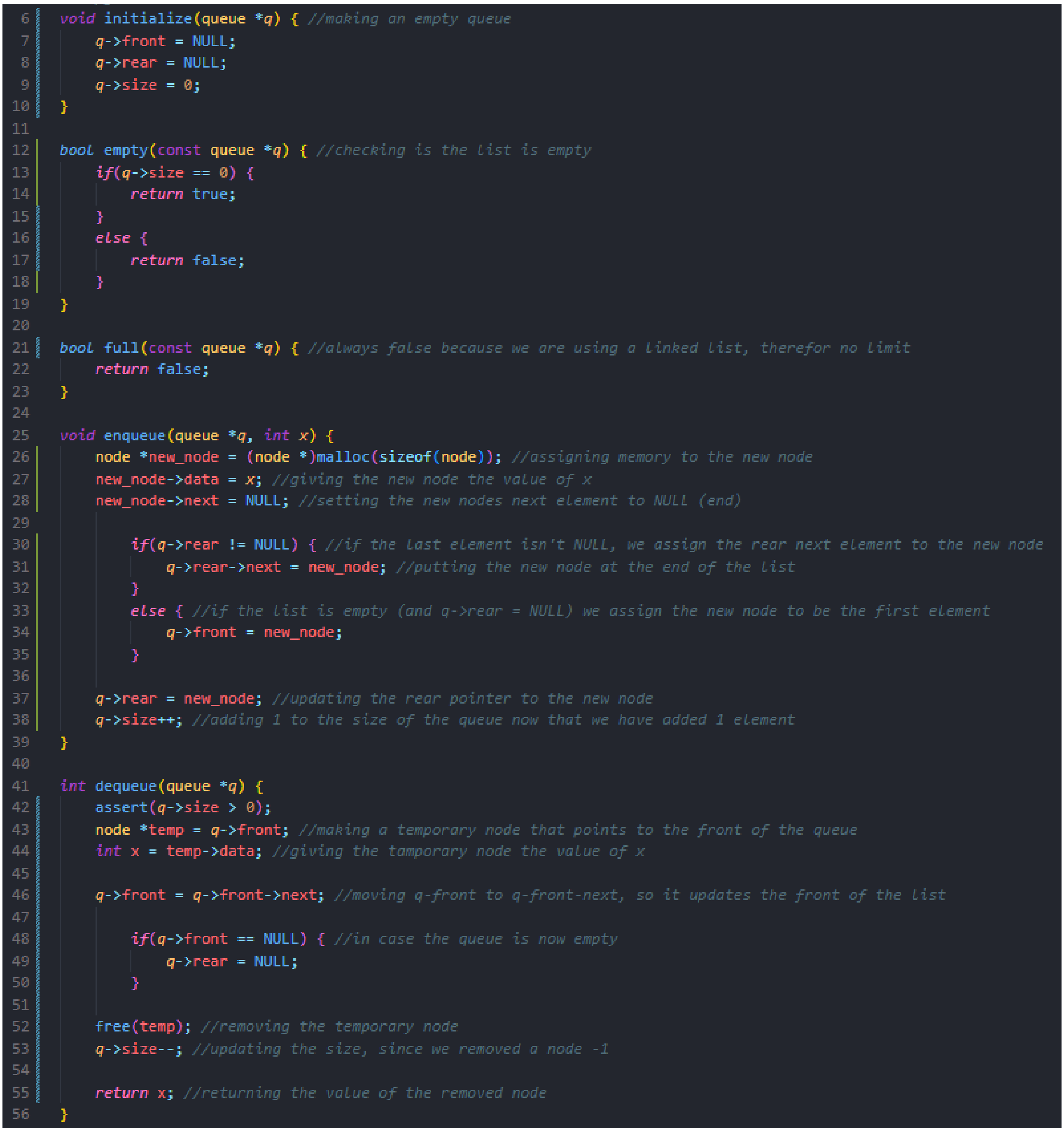
b) n*2, da i skal være mindre eller lig med n og vi har 2 operationer hver gang loopet kører

opgave 2

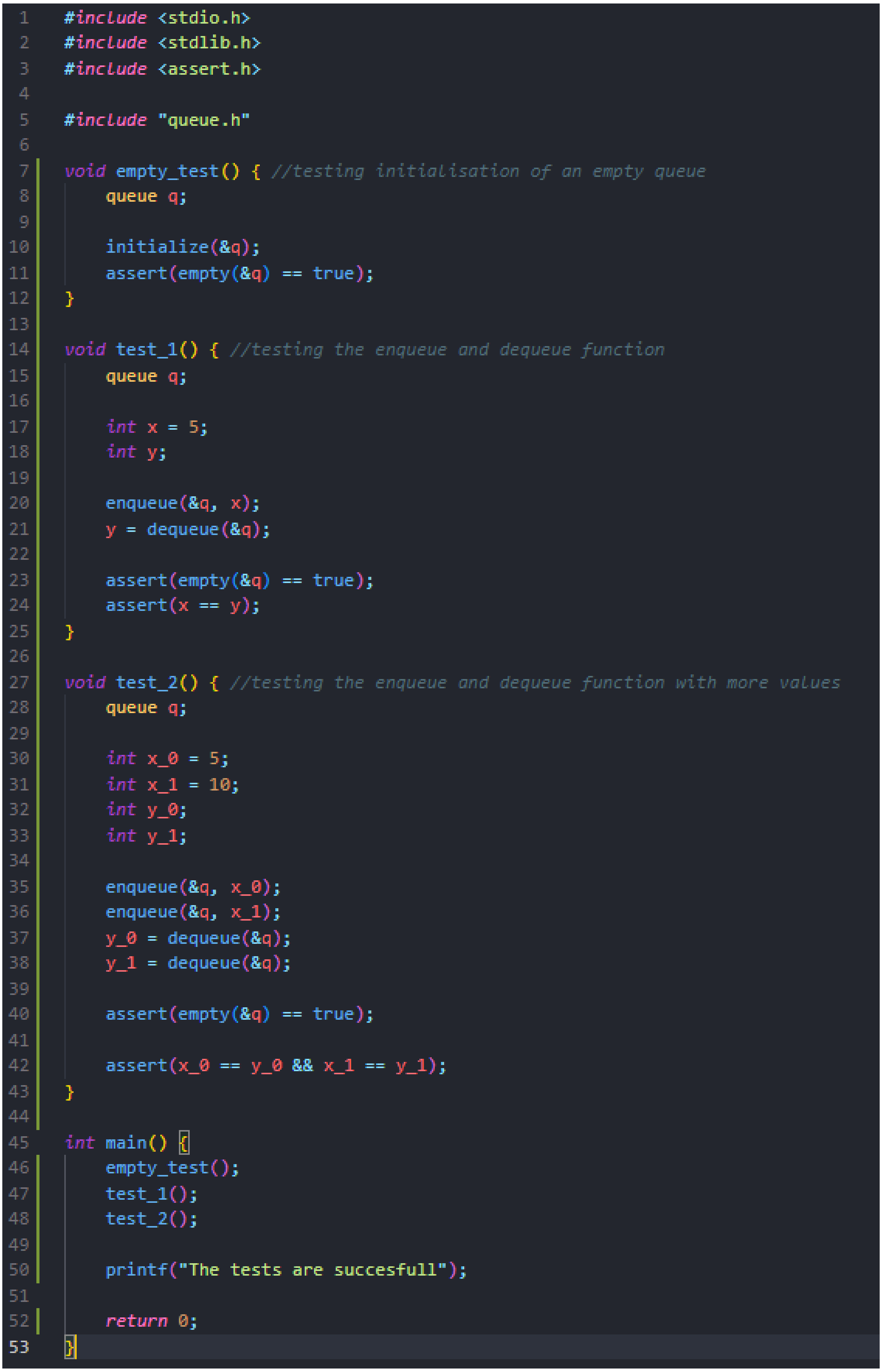


```
node* isort(node *list) {

    node *sorted_list = NULL; //starting at NULL
    node *current = list; //current is the first element in list
    while(current != NULL) { //as long as we are not at the end of the list, the loop with run
        node *next = current->next; //setting the first node to cuurent's next
        if(!sorted_list || current->data <= sorted_list->data) { //if the list is empty (the beginning) and the value of current is less than the value of the sorted list
            current->next = sorted_list; //it will swap the two values so the less one is first in the list
            sorted_list = current;
        }
        else {
            node *temp = sorted_list; //making new current node that points to the sorted list
            while(temp->next && temp->next->data < current->data) { //while the value of temp next is less than the current value
                temp = temp->next; //it moves, so that the temp is next element
            }
        current->next = temp->next; //then the current next is set to the temporarys next
        temp->next = current; //and then the temporary next is set as the current
        }
    current = next;
    }

    return sorted_list; //we return the sorted list
}
```

Opgave 3

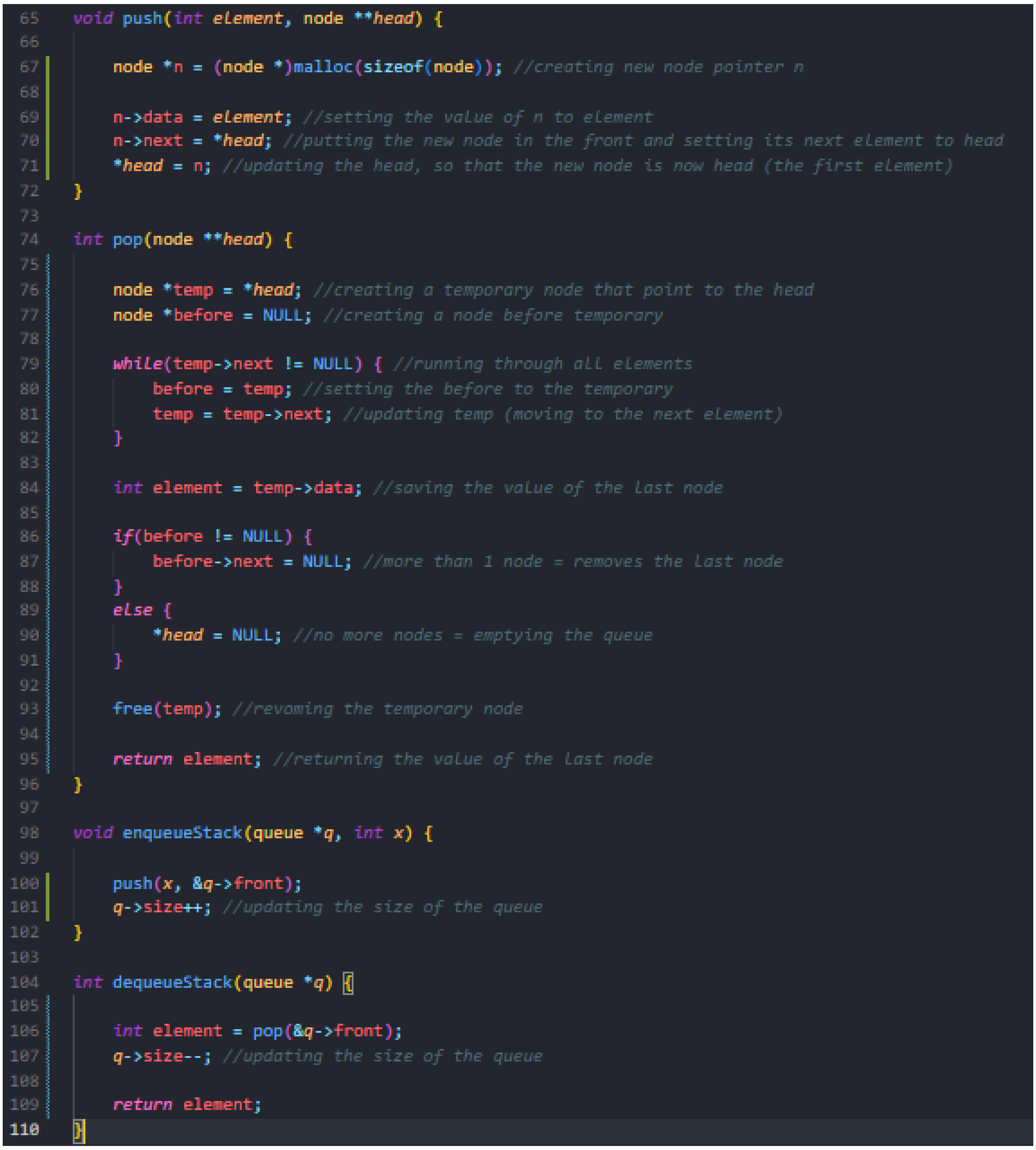


```c
void initialize(queue *q) { //making an empty queue
    q->front = NULL;
    q->rear = NULL;
    q->size = 0;
}

bool empty(const queue *q) { //checking is the list is empty
    if(q->size == 0) {
        return true;
    }
    else {
        return false;
    }
}

bool full(const queue *q) { //always false because we are using a linked list, therefor no limit
    return false;
}

void enqueue(queue *q, int x) {
    node *new_node = (node *)malloc(sizeof(node)); //assigning memory to the new node
    new_node->data = x; //giving the new node the value of x
    new_node->next = NULL; //setting the new nodes next element to NULL (end)

        if(q->rear != NULL) { //if the last element isn't NULL, we assign the rear next element to the new node
            q->rear->next = new_node; //putting the new node at the end of the list
        }
        else { //if the list is empty (and q->rear = NULL) we assign the new node to be the first element
            q->front = new_node;
        }

    q->rear = new_node; //updating the rear pointer to the new node
    q->size++; //adding 1 to the size of the queue now that we have added 1 element
}

int dequeue(queue *q) {
    assert(q->size > 0);
    node *temp = q->front; //making a temporary node that points to the front of the queue
    int x = temp->data; //giving the tamporary node the value of x

    q->front = q->front->next; //moving q-front to q-front-next, so it updates the front of the list

        if(q->front == NULL) { //in case the queue is now empty
            q->rear = NULL;
        }

    free(temp); //removing the temporary node
    q->size--; //updating the size, since we removed a node -1

    return x; //returning the value of the removed node
}
```



```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>

#include "queue.h"

void empty_test() { //testing initialisation of an empty queue
    queue q;

    initialize(&q);
    assert(empty(&q) == true);
}

void test_1() { //testing the enqueue and dequeue function
    queue q;

    int x = 5;
    int y;

    enqueue(&q, x);
    y = dequeue(&q);

    assert(empty(&q) == true);
    assert(x == y);
}

void test_2() { //testing the enqueue and dequeue function with more values
    queue q;

    int x_0 = 5;
    int x_1 = 10;
    int y_0;
    int y_1;

    enqueue(&q, x_0);
    enqueue(&q, x_1);
    y_0 = dequeue(&q);
    y_1 = dequeue(&q);

    assert(empty(&q) == true);

    assert(x_0 == y_0 && x_1 == y_1);
}

int main() {
    empty_test();
    test_1();
    test_2();

    printf("The tests are succesfull");

    return 0;
}
```

Opgave 4



```c
void push(int element, node **head) {

    node *n = (node *)malloc(sizeof(node)); //creating new node pointer n

    n->data = element; //setting the value of n to element
    n->next = *head; //putting the new node in the front and setting its next element to head
    *head = n; //updating the head, so that the new node is now head (the first element)
}

int pop(node **head) {

    node *temp = *head; //creating a temporary node that point to the head
    node *before = NULL; //creating a node before temporary

    while(temp->next != NULL) { //running through all elements
        before = temp; //setting the before to the temporary
        temp = temp->next; //updating temp (moving to the next element)
    }

    int element = temp->data; //saving the value of the last node

    if(before != NULL) {
        before->next = NULL; //more than 1 node = removes the last node
    }
    else {
        *head = NULL; //no more nodes = emptying the queue
    }

    free(temp); //revoming the temporary node

    return element; //returning the value of the last node
}

void enqueueStack(queue *q, int x) {

    push(x, &q->front);
    q->size++; //updating the size of the queue
}

int dequeueStack(queue *q) {

    int element = pop(&q->front);
    q->size--; //updating the size of the queue

    return element;
}
```