https://github.com/IPFCE-2024/assignment-8-nicolas-anton

Exercise 1: Consider the following program for computing factorial numbers:

a) How many arithmetic operations (+, -, *, /) are required to compute fact(5)?

```
long fact(int n) {
  // precondition:
  assert(n >= 0);
  long f = 1;
  for (long i = 1; i <= n; ++i) {
    f = i * f;
  }
  return f;
}
```

In this case 10 arithmetic operations is required to compute fact(5):

- The variable "i" is incremented by 1 ($i + 1$) **5** times during the loop
- Inside the loop we have a multiplication: ($i \cdot f$) which is also calculated **5** times in this case.

Thereby: 10 arithmetic operations

b) How many arithmetic operations (+, -, *, /) are required to compute fact(n) for any positive integer n?

For any positive integer n the required amount of arithmetic operations is $n \cdot 2$ to compute fact(n)

Exercise 2: insertion sort algorithm (void isort(node* list);

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

#include "insertion_sort.h"

// Function to sort the list:
node *isort(node *list)
{
    // Creating an empty node pointer:
    node *sorted = NULL;

    // Looping through the list
    while (list != NULL)
    {
        // Saving the current node:
        node *current = list;
        // Moving to the next element in the list:
        list = list->next;

        // Ready to insert at the right spot:
        if (sorted == NULL || current->data <= sorted->data)
        {
            // Current.next points to the first element of the sorted list:
            current->next = sorted;
            // Making current the new head of the sorted list:
            sorted = current;
        }
        // If it is not ready to insert:
        else
        {
            // Make a new pointer:
            node *temp = sorted;

            // Looping until we reach the right spot to insert the node:
            while (temp->next != NULL && temp->next->data < current->data)
            {
                // Moving the temp pointer to the next node:
                temp = temp->next;
            }
            // Setting current.next to point to the node that temp.next points to:
            current->next = temp->next;
            // Setting temp.next pointer to point to the current node:
            temp->next = current;
        }
    }
    // Return the first element of the sorted list:
    return sorted;
}
```

Exercise 3:

a) **Implement a queue based on singly-linked lists as discussed in the lecture. That is, implement the five functions mentioned above:**

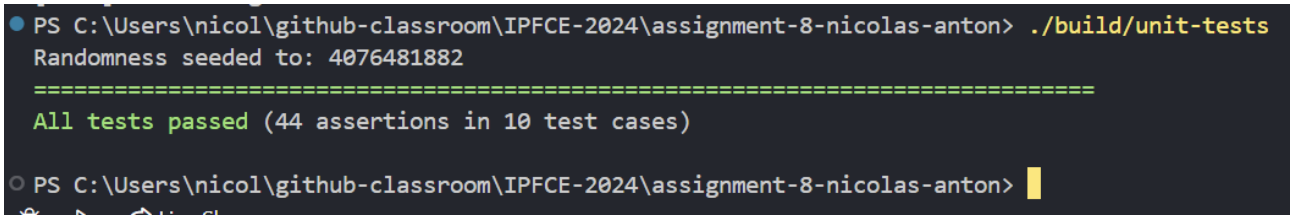
```c
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include "queue.h"

void initialize(queue *q)
{
    // Set the front and rear node to NULL:
    q->rear = NULL;
    q->front = NULL;
    // Set the size to 0
    q->size = 0;
}

bool empty(const queue *q)
{
    // Returning true if the size is 0 and the front and rear node are empty:
    return q->size == 0 && q->rear == NULL && q->front == NULL;
}

bool full(const queue *q)
{
    return false;
    // hahahahha
}

void enqueue(queue *q, int x)
{
    // Create a new node:
    node *n = (node *)malloc(sizeof(node));

    // Set the data in the node to x:
    n->data = x;

    // Make it a single node:
    n->next = NULL;

    // Checks if the list is empty:
    if (empty(q))
    {
        // Setting both the front and the back to the node n:
        q->rear = n;
        q->front = n;
    }
    // If the list is not empty:
    else
    {
        // Setting the node to next element after the rearest element:
        q->rear->next = n;
        // Setting the new node to the new rear:
        q->rear = n;
    }
    // Increment the size of the queue:
    q->size++;
}

int dequeue(queue *q)
{
    // Checks if the queue is empty:
    assert(!empty(q));

    // Making a temporary pointer to point to the front node:
    node *temp = q->front;
    // Saving the data at the front node in an integer variable:
    int dequeued = temp->data;

    // Moving the front element to the second front position:
    q->front = temp->next;

    // Checks if the queue only had one node:
    if (q->front == NULL)
    {
        // Set the rear to NULL also:
        q->rear = NULL;
    }

    // Freeing the temporary note:
    free(temp);
    q->size--;

    // Returning the data of the dequeued node:
    return dequeued;
}
```

b) Test file:

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

// Making a queue pointer
queue *q;

// Test function to test if we have initialized an empty queue after calling initialize(q);
void initialize_test()
{
    // Allocating memory for the queue:
    q = (queue *)malloc(sizeof(queue));
    // Calling initialize function:
    initialize(q);
    // Checking if the queue is empty:
    assert(empty(q));
}

// Function to test for b.2:
void test_number2()
{
    // Declaring variables:
    int x = 5;
    int y;

    // Making a test pointer to a queue pointing to the queue that q points to:
    queue *test = q;

    // Calling the functions enqueue and dequeue:
    enqueue(q, x);
    y = dequeue(q);

    // Checking if the queue is the same as it was before calling the two functions:
    assert(test == q);

    // Checking if x equals y:
    assert(x == y);
}

// Function to test for b.3:
void test_number3()
{
    // Declaring variables:
    int x0 = 5;
    int x1 = 4;
    int y1;
    int y0;

    // Making a test pointer to a queue pointing to the queue that q points to:
    queue *test = q;

    // Calling the functions enqueue and dequeue with different variables:
    enqueue(q, x0);
    enqueue(q, x1);
    y0 = dequeue(q);
    y1 = dequeue(q);

    // Checking if the queue is the same is it was before calling the functions:
    assert(test == q);
    // Checking if x0 equals y0 and x1 equals y1:
    assert(x0 == y0 && x1 == y1);
}

// Main function:
int main()
{
    // Calling the test functions:
    initialize_test();
    test_number2();
    test_number3();
    // Printing succes message if every law holds:
    printf("Every law holds\n");

    // Freeing the q pointer:
    free(q);

    // Returning from the body and main:
    return 0;
}
```

Exercise 4:

```c
// Opgave 4:
// The function takes a double pointer because we have to modify the head of the function as we are inserting a node at
void push(int element, node **head)
{
    // Allocate memory for new node:
    node *n = (node *)malloc(sizeof(node));

    // Setting the data in the node to the element:
    n->data = element;
    // Inserts the node at the beginning of the list with its next pointer to the current head:
    n->next = *head;
    // Updating the head to point to the new node (the new node is now the new start of the list):
    *head = n;
}

// Double pointer allowing to change the head of the list:
int pop(node **head)
{
    // Temporary node pointer pointing to the head of the list:
    node *temp = *head;
    // Node pointer prev pointing to an empty node:
    node *prev = NULL;

    // Looping until reaching the last node:
    while (temp->next != NULL)
    {
        // Previous pointer pointing to the node just before the node that temp points to:
        prev = temp;
        // Moving through the list:
        temp = temp->next;
    }

    // Storing the data of the last node before popping it:
    int element = temp->data;

    // Checking if there is only one node in the list:
    if (prev != NULL)
    {
        // If there is more than one node then the next node after the prev node is set to NULL. Meaning that is being
        prev->next = NULL;
    }
    // If there was only one node in the list:
    else
    {
        // Setting the head to NULL meaning the queue is now empty:
        *head = NULL;
    }
    // Freeing the removed node from memory:
    free(temp);
    // Returning the data of the popped node:
    return element;
}

void enqueueStack(queue *q, int x)
{
    // Enqueueing stack using push function:
    push(x, &q->front);
    // Incrementing queue size:
    q->size++;
}

int dequeueStack(queue *q)
{
    // Dequeueing stack using pop function saving the returned element in an integer variable:
    int element = pop(&q->front);
    // Decrementing the queue size:
    q->size--;
    // Returning the element:
    return element;
}
```


PS C:\Users\nicol\github-classroom\IPFCE-2024\assignment-8-nicolas-anton> ./build/unit-tests
Randomness seeded to: 4076481882
All tests passed (44 assertions in 10 test cases)
PS C:\Users\nicol\github-classroom\IPFCE-2024\assignment-8-nicolas-anton>