Programmering hand-in uge 8

1) Analyse af fakulterings-program

*Antal aritmetiske operatorer (+, -, *, /)*

I programmet ses at der er en for-loop som kører n gange. Da i skal være mindre end eller lig med n kører loopen 5 gange hvis n er 5.

Når der står ++i, lægger man en til i ved hver iteration. Derudover ganges f med i ved hver iteration. Derfor er der to operatorer i en iteration, så derfor bruges 2*n operatorer for at computere funktionen. I tilfælde med fact(5) er der 2*5 operatorer.

2) *Insertion sort function* med singly linked list

https://github.com/IPFCE-2024/assignment-8-kathrine/blob/0d015930d06b6246555da28a90a90a56356c0d52/insertion_sort.c

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

#include "insertion_sort.h"
#include "node.h"

node *isort(node *list) {
    if (list == NULL) return NULL;

    node* sorted = NULL; //start med en tom liste

    //gå igennem alle knuder i den oprindelige liste
    node* current = list;
    while (current != NULL){
        node* nx = current->next; //gem den næste node

        if(sorted == NULL|| current->data <= sorted->data)
        {
            //indsæt current forrest i den sorterede liste
            current->next = sorted;
            sorted = current;
        }
        else
        {
            //find positionen efter hvor current skal indsættes
            node* temp = sorted;
            while (temp->next != NULL && temp->next->data < current->data)
            {
                temp = temp->next;
```

```
            }
            //indsæt current på rigtige plads i listen
            current->next = temp->next;
            temp->next = current;
        }

        //flyt til den næste knude
        current = nx;
    }

    list = sorted;

    return list;
}
```

3) Queue

Opgave a

```
// Opgave 3
void initialize(queue *q) {
    q->front = NULL;
    q->rear = NULL;
    q->size = 0;
}

bool empty(const queue *q) {
    return q->size == 0;
}

bool full(const queue *q) {
    return false;               //aldrig full, da det er en singly linked list
}

void enqueue(queue *q, int x) {
    node* newNode = (node*)malloc(sizeof(node));
    newNode->data = x;
    newNode->next = NULL;

    if(q->rear == NULL){
        q->front = q->rear = newNode;
    }
    else{
        q->rear->next = newNode;
        q->rear = newNode;
    }
    q->size++;
```

```
}

int dequeue(queue *q) {
    if(q->front == NULL){
        printf("Køen er tom\n");
        return -1;
    }
    node* temp = q->front;
    int data = temp->data;
    q->front = q->front->next;

    if(q->front == NULL){
        q->rear = NULL;
    }

    free(temp);
    q->size--;
    return data;
}
```

Opgave b

(ved ikke lige hvorfor koden ikke ligner tidligere)

#include <stdio.h>

#include <stdlib.h>

#include <assert.h>

#include "queue.h"

int main() {

//test: initialize

queue q;

initialize(&q);

assert(empty(&q));

printf("Test af initialize: Bestået\n");

```
//test af enqueue og dequeue
int x = 5;
enqueue(&q, x);
int y = dequeue(&q);

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

printf("Test af enqueue og dequeue: Bestået\n");

//test med 2 værdier
int x0;
int x1;

enqueue(&q, x0);
enqueue(&q, x1);
int y0 = dequeue(&q);
int y1 = dequeue(&q);

assert(x0 == y0);
assert(x1 == y1);
assert(empty(&q));

printf("Testen er bestået\n");

return 0;
}
```

4) Queue som stack

```
// Opgave 4
typedef struct {
    node *stack1;
    node *stack2;
} queue;

void push(int element, node **head) {
node *newNode = (node*)malloc(sizeof(node));
newNode->data = element;
newNode->next = *head;
*head = newNode;
}

int pop(node **head) {
    if (*head == NULL) {
        return -1;
    }
    node *temp = *head;
    int value = temp->data;
    *head = temp->next;
    free(temp);
    return value;
}

void enqueueStack(queue *q, int x) {
    push(x, &(queue->stack1));
}

int dequeueStack(queue *q) {
    if (q->stack2 == NULL) {
        while (queue->stack1 != NULL) {
            int temp = pop(&(q->stack1));
            push(temp, &(q->stack2));
        }
    }
    return pop(&(queue->stack2));
}
```