Time in C

```
/*
 * Mappeinnlevering.c
 *
 * Created: 23.03.2015 12:05:45
 *  Author: michae14
 */

#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdint.h>
#include <stdlib.h>
#include <util/atomic.h>

#define F_CPU 4000000
#define tick F_CPU/64

typedef struct
{
    volatile uint8_t centiseconds;
    volatile uint16_t seconds;
} type_tid;

typedef struct
{
    volatile uint8_t loperNummer;
    volatile type_tid* startTid;
    volatile type_tid* sluttTid;
    volatile type_tid* lopeTid;
} type_loper;

typedef struct
{
    volatile uint8_t loperNummer;
    volatile type_tid tid;
    volatile uint8_t handled;
} type_event;

type_tid tid = { 0, 0 };
type_event loper_event;

uint16_t snittTid;
type_loper* besteLoper_p;
```

```
type_loper* lopere_p[255];

static void initTimers(void);
static void initPort(void);
static void simulerNyLoper(void);
static void handleEvent(type_event* ev);
static void incTime(type_tid* tid_p);
static void stopTimers(void);
static type_tid* finnLopeTid(volatile type_tid* start_p, volatile type_tid* slutt_p);
static type_loper* finnBesteLoper(type_loper* first_p, type_loper* second_p);
static void visBesteLoper(void);

static void initTimers(void)
{
    TCCR1B = ((1 << WGM12) | (1 << ICNC1));
    // Setter timeren i CTC modus (Clear Timer on Compare) og Capture på PORTD4
    TCNT1  = 0;                             // Resetter teller
    OCR1A  = tick;                          // Setter output compare register
    TIMSK  = (1 << OCIE1A | 1 << TICIE1);   // Setter interrupt til compare match

    TCCR3B = (1 << WGM12);  // Setter timeren i CTC modus (Clear Timer on Compare)
    TCNT3  = 0;             // Reetter timer-telleren
    OCR3A  = tick;          // Setter compare-registeret til v?r tick verdi
    ETIMSK = (1 << OCIE3A); // Setter (extended) interrupt flagget til compare match

    sei();                  // Enabler  global interrupt
}

/*
 * Starter timerene ved å sette prescalar til 64
 */
static void startTimers(void)
{
    TCCR1B |= ((1 << CS10) | (1 << CS11));
    TCCR3B |= ((1 << CS10) | (1 << CS11));
}

/*
 * Stopper timerene ved å fjerne prescalar mask
 */
static void stopTimers(void)
{
    TCCR1B &= ~((1 << CS10 ) | (1 << CS11)); // Stopp timer 1
    TCCR3B &= ~((1 << CS10 ) | (1 << CS11)); // Stopp timer 3
}
```

```
/*
 * PORTE leser l?pers nummer (8-bit)
 * PORTD4 leser n?r l?per passerer start/m?l
 * PORTD0 leser n?r l?pet starter (et trykk) eller stopper (et til trykk)
 * PORTA sender l?per med beste tid sitt nummer (8-bit)
 * PORTC sender l?pere til resultattavla (8-bit)
 * PORTB bit 0 og 1 er kontrollbits for overf?ring av l?pernummer til resultattavla
 */
static void initPort(void)
{
    DDRE = DDRD = 0x0;
    DDRA = DDRC = 0xFF;
    DDRB = 0x1;

    PORTA = 0x0;
}

/*
 * Interrupt p? Timer 3:
 * Dette er simuleringstimeren, som sender ut l?pere med 2 sekunders intervall
 */
ISR(TIMER3_COMPA_vect)
{
    simulerNyLoper();
}

/*
 * Interrupt p? Timer 1:
 * Dette er simuleringstimeren som blir kallet hvert sekund, og cleara (CTC)
 */
ISR(TIMER1_COMPA_vect)
{
    incTime(&tid);
}

/*
 * Interrupt p? PORTD4:
 * Det kommer et interrupt p? Timer1 n?r PORTD4 g?r lav
 */
ISR(TIMER1_CAPT_vect)
{
    tid.centiseconds = ((s1 * 100)/tick); // Leser hundredeler fra Input Control Register

    loper_event.loperNummer = PINE;
    loper_event.tid = tid;
    loper_event.handled = 0;
```

```
}

/*
 * Denne funksjonen sender ut ny l?per for simulering av l?p
 * Start- og m?lpasseringsdetektoren gir l?pers nummer p? PORTE og signal p? PORTD4
 * PORTD4 High = L?per passerer start eller m?l
 */
static void simulerNyLoper(void)
{
    static uint8_t teller = 0;
    static uint8_t loperId = 1;
    static uint8_t antallLopere = 15;

    if (teller < 30 && loperId == antallLopere)
    {
        teller++;
        return;
    }

    PORTE = loperId * 10;
    if (PORTD & (1 << 4))
    {
        PORTD = ~(1 << 4);
        if (teller == 30)
            loperId--;
        else
            loperId++;

        if (loperId == 0)
            PORTD = (1 << 0);
            // L?pet stoppes n?r alle l?perne har passert (simuler trykk p? PORTD0)
    }
    else
    {
        PORTD = 0xFF;
    }
}

/* Her prossesseres informasjonen fra interrupt.
 * Vi har flytta den ut av interrupt rutina for ? ikke blokkere nye interrupts
 */
static void handleEvent(type_event* ev)
{
    static uint8_t loperId = 0;

    type_loper* loper_p = NULL;
```

```
    for (uint8_t i = 0; i < 255; i++)
    {
        if (lopere_p[i] == NULL) continue;
        if (lopere_p[i]->loperNummer == ev->loperNummer)
        { // Vi kjenner allerede til l?peren, dermed er l?peren n? i m?l
            loper_p = lopere_p[i];
            loper_p->sluttTid = &ev->tid;
            loper_p->lopeTid = finnLopeTid(loper_p->startTid, loper_p->sluttTid);

            if (besteLoper_p == NULL) // Antakeligvis f?rstemann i m?l, dermed forel?pig bes
                besteLoper_p = loper_p;
            else
                besteLoper_p = finnBesteLoper(besteLoper_p, loper_p);
// Vi kaller en funksjon for ? sjekke om denne l?peren er raskere enn n?v?rende innehaver

            visBesteLoper(); // Oppdater PORTA med lopernummer til raskeste l?per

// Vi har gjort det vi kom hit for ? gj?re. Un?dvendig ? forsette loopen og NULL-sjekken.
            return;
        }
    }
    if (loper_p == NULL)
    { // L?peren var ikke i lista, dermed har han nettopp passert start og m? legges til
        loper_p = (type_loper*)malloc(sizeof(type_loper));
        loper_p->loperNummer = ev->loperNummer;
        loper_p->startTid = &ev->tid;

        lopere_p[++loperId] = loper_p;
    }
}

static void incTime(type_tid* tid_p)
{
    tid_p->seconds++;
}

/*
 * Her regnes differansen ut mellom to tider
 * Brukes til ? regne ut hvor lang tid en l?per har brukt
 */
static type_tid* finnLopeTid(volatile type_tid* start_p, volatile type_tid* slutt_p)
{
    type_tid* diff_p = (type_tid*)malloc(sizeof(type_tid));

    diff_p->seconds = slutt_p->seconds - start_p->seconds; // Trekk fra sekunder
    if (start_p->centiseconds < slutt_p->centiseconds)
```

```
    // Pass p? at start-hundredeler o stopp-hundredeler ikke ender som et minustall.
    {
        diff_p->seconds--;
        diff_p->centiseconds = 100-(start_p->centiseconds-slutt_p->centiseconds);
    }
    else
        diff_p->centiseconds = slutt_p->centiseconds - start_p->centiseconds;

    return diff_p;
}

/*
 * Her sammenlignes to spilleres lopetider og pekeren til den raskeste returneres
 */
static type_loper* finnBesteLoper(type_loper* first_p, type_loper* second_p)
{
    if (first_p == NULL)    // Om den f?rste er NULL er den andre den beste
        return second_p;
    if (second_p == NULL)   // Om den ander er NULL en den f?rste den beste
        return first_p;

    if (first_p->lopeTid->seconds < second_p->lopeTid->seconds) // Forste loper var raskere
        return first_p;
    if (first_p->lopeTid->seconds == second_p->lopeTid->seconds)
    // Loperne lop paa like mange sekunder, sjekker hundredeler
    {
        if (first_p->lopeTid->centiseconds < second_p->lopeTid->centiseconds)
            return first_p;
        else
            return second_p;
    }

    return second_p;
}
// Ytterl?kka bestemmer ny plassering
// Raskeste løper kommer først
static void sorterLopere(void)
{
    uint8_t i;
    uint8_t j;
    uint8_t flyttFra = 0;
    type_loper* flytt = NULL;

    for (i = 0; i < 255; i++)
    {
        for (j = i; j < 255; j++)
```

```
        {
            if (lopere_p[j] == NULL) continue;

            flytt = finnBesteLoper(flytt, lopere_p[j]);
            if (flytt == lopere_p[j])
                flyttFra = j;
        }

        lopere_p[flyttFra] = lopere_p[i];
        lopere_p[i] = flytt;
        flytt = NULL;
    }
}

/*
 * Oppdaterer PORTA med nummeret til den raskeste l?peren.
 * Blir kalt hver gang en l?per kommer i m?l
 */
static void visBesteLoper(void)
{
    if (besteLoper_p == NULL)
        PORTA = 0x0;
    else
        PORTA = besteLoper_p->loperNummer;
}

/*
 * Regner ut snitt-tiden til løperne
 * Enkelt når man kun holder styr på sekunder
 */
static void regnGjennomsnittstid(void)
{
    uint16_t sumSekund = 0;
    uint8_t sumHundredeler = 0;
    uint8_t antall = 0;
    uint8_t i = 0;

    for (i = 0; i < 255; i++)
    {
        if (lopere_p[i] == NULL) continue;

        sumSekund += lopere_p[i]->lopeTid->seconds;
        sumHundredeler += lopere_p[i]->lopeTid->centiseconds;
        sumSekund += sumHundredeler / 100;
        sumHundredeler = sumHundredeler % 100;
```

```
        antall++;
    }

    sumSekund = sumSekund / antall;
    sumHundredeler = (sumHundredeler * 100) / antall;
}

int main(void)
{
    uint8_t buttonUp = 0;

    loper_event.handled = 1;

    initPort();         // Sett opp portene
    initTimers();       // Sett opp timerene
    while (!(PIND & (1 << 0))); // Vent til PIND0 har f?tt 1 trykk (starter l?pet)
    startTimers();      // Starter timerene etter første trykk
    while (1)
    {
        if (!loper_event.handled) // Vi behandler kun et event en gang
        {
            ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
// Stopper midlertidig for nye interrupts slik at den nåværende dataen ikke blir overskreve
            {
                handleEvent(&loper_event);
                loper_event.handled = 1;
            }
        }

        if (PIND & ~(1 << 0))
// Vi sjekker om det f?rste trykket er sluppet, og lagrer en tilstand for det
            buttonUp = 1;
        if (buttonUp && PIND & (1 << 0)) // Vi har f?tt et nytt trykk, stopper l?pet
            break;

    }
    // Vi kommer ut av loopen etter andre trykk (dvs. når klokka stopper)
    stopTimers();       // Stopp timerene (alle er i mål)
    sorterLopere();     // Sorter
    regnGjennomsnittstid(); // Regn snitttid
}
```