UiT
NORGES
ARKTISKE
UNIVERSITET

SENSORVEILEDNING

For eksamen i: **INF-1100**

Dato: **Onsdag 05. desember 2018**

Sensorveiledningen er på 8 sider inklusiv forside

Fagperson/intern sensor: Robert Pettersen

Telefon: 47 24 75 52

Eksamen INF-1100
Innføring i programmering og datamaskiners virkemåte
Høst 2018

Løsningsforslag

Eksamenssettet består av 4 oppgaver.

Les oppgaveteksten grundig og disponer tiden slik at du får tid til å svare på alle oppgavene. I noen oppgaver kan det være nødvendig å tolke oppgaveteksten ved å gjøre noen antagelser - gjør i så fall rede for hvilke antagelser du har gjort, men pass på å ikke gjøre antagelser som trivialiserer oppgaven.

Der du skal utvikle eller beskrive en algoritme anbefales det at du først beskriver algoritmen på et høyt abstraksjonsnivå, f.eks. med figurer, før du går videre med detaljer og eventuell pseudokode. Dersom det spørres etter en implementasjon i C kreves det ikke 100% syntaktisk korrekt kode. Dersom det spørres etter pseudokode kan du kan også skrive ren C-kode om du ønsker.

Husk også at du kan referere tilbake til funksjoner du tidligere har definert.

Oppgave 1 - 25%

Gi en kort beskrivelse av von Neumann modellen og instruksjonssyklusen. Beskrivelsen bør omfatte de ulike komponentene i modellen og hvordan disse interagerer med hverandre.

Løsningsforslag

Alle komponentene skal være kort beskrevet, helst med en tegning/figur. Interaksjon med minne via register MDR/MAR og Write Enabled signalet må være med.

Alle stegene i instruksjons syklusen (Fetch, decode, evaluate adddress, fetch operand, execute, store result) skal være skissert.

Oppgave 2 - 25%

Gitt følgende funksjon:

```
int ukjent(int a, int b)
{
    int x = 0;

    while (a >= b) {
        x += 1;
        a -= b;
    }

    return x;
}
```

a) Hvilken verdi vil funksjonen *ukjent* ovenfor returnere ved følgende kall:

(1) *ukjent(12, 5)*

(2) *ukjent(16, 3)*

b) Hva er det funksjonen beregner?

Løsningsforslag

a) (1) 2

(2) 5

b) Funksjonen beregner heltalls divisjon av a/b.

Oppgave 3 - 25%

Gitt et array *A* som inneholder n heltall (integers).

a) Skriv en funksjon `forekommer` som teller hvor mange ganger et gitt tall k forekommer i A. Funksjonen skal ta som inn-parametre: en peker til A, en angivelse av antall tall i A, samt en verdi for k:

```
int forekommer(int *A, int n, int k);
```

b) Skriv en funksjon `sorter` som flytter rundt på tallene i A slik at de blir liggende i stigende rekkefølge. Funksjonen skal ta som inn-parametre: en peker til A og en angivelse av antall tall i A. Funksjonen skal ikke returnere noe.

```
void sorter(int *A, int n);
```

c) A har ett *majoritetselement* dersom ett tall forekommer mer enn $n/2$ ganger i A. Følgende algoritme kan benyttes for å avgjøre om A har ett majoritetselement. Anta at vi først sorterer A. Dersom A har ett majoritetselement må dette tallet være i $A[n/2]$. Dersom antall forekomster av $A[n/2]$ i A er høyere enn $n/2$, er $A[n/2]$ ett marjoritetselement.

Skriv en funksjon som avgjør om A har ett majoritetselement. Funksjonen skal ta som inn-parametre: en peker til A og en angivelse av antall tall i A. Funksjonen skal returnere 1 dersom det finnes ett majoritetselement, og 0 dersom det ikke finnes.

Bruk gjerne funksjonene du har implementert tidligere.

```
int majoritetselement(int *A, int n);
```

Løsningsforslag

---

a)
```
int forekommer(int *A, int n, int k) {
    int antall = 0;

    for (int i = 0; i < n; i++) {
        if (A[i] == k) antall++;
    }
    return antall;
}
```

b)
```
void sorter(int *A, int n) {
    int min, tmp, i, j;

    for(i = 0; i < n; i++) {
        min = i;
        for(j = i + 1; j < n; j++) {
            if (A[min] > A[j])
                min = j;
        }
        if (min != i) {
            tmp = A[i];
            A[i] = A[min];
            A[min] = tmp;
        }
    }
}
```

c)
```
int majoritetselement(int *A, int n) {
    int antall;

    sorter(A);

    antall = forekommer(A, n, A[n/2]);

    if (antall > n/2)
        return 1;
    return 0;
}
```

Oppgave 4 - 25%

I ett orienteringsløp skal en deltaker besøke ett gitt antall poster i en gitt rekkefølge. På hver post vil ett kontrollelement som er unik for posten bli lagd når deltakeren er innom posten. Dette kontrollelementet inneholder en unik kode for posten (*kode*) og hvor lang tid deltakeren har brukt fra forrige post, eventuelt fra start hvis dette er første post (*strekktid*):

```
typedef struct kontrollelement kontrollelement_t;
struct kontrollelement {
    int kode;
    double strekktid;
};
```

Kontrollelementet legges til deltakerens liste over poster som er besøkt (*spor*). Mål håndteres på samme måte som alle andre poster (men skal være siste post som registreres). Når deltakerne kommer i mål vil listen av kontrollelementer kunne kontrolleres mot en fasit-liste hvor postene er plassert i rett rekkefølge (*fasit*).

a) Lag en funksjon som kontrollerer at deltakeren har besøkt alle postene, og at de er besøkt i rett rekkefølge. Funksjonen skal returnere 1 hvis deltakeren har besøkt alle postene i rett rekkefølge, 0 hvis dette ikke er tilfelle og -1 hvis noe går galt under sjekken:

```
int kontroll(list_t *spor, list_t *fasit);
```

b) Lag en funksjon som returnerer tiden deltakeren har brukt fra start til mål. Returner -1 dersom noe skulle gå galt under beregningen:

```
double tid(list_t *spor);
```

c) For alle deltakerne som har besøkt alle postene i rett rekkefølge tar vi vare på deres spor i en ny liste *godkjente* (en liste hvor elementene er lister tilsvarende *spor* ovenfor). Lag en funksjon som tar *godkjente* som argument, og finner den deltakeren med beste sammenlagt tid (minste sammenlagte tid). Funksjonen skal returnere denne bestetiden, eller -1 hvis noe går galt under beregningen:

```
double bestetid(list_t *godkjente);
```

(Hint: Du kan benytte funksjonen *tid* fra forrige deloppgave.)

Ikke gjør noen antakelser om hvordan listen er implementert. Du kan anta at følgende listefunksjoner er tilgjengelige:

```
// Lag en ny liste-iterator som peker på første element i listen
list_iterator_t *list_createiterator(list_t *list);

// Returnerer elementet som pekes på av iteratoren og
// lar iteratoren peke på neste element.
// NULL returneres når en kommer til slutten av listen
void *list_next(list_iterator_t *iterator);

// Frigir iteratoren
void list_destroyiterator(list_iterator_t *iterator);
```

Løsningsforslag

Det skal gjøres sjekk på retur fra list_createiter, for å oppdage de tilfellene det er fritt for minne.

a)
```
int kontroll(list_t *spor, list_t *fasit) {
    list_iterator_t *spor_it, *fasit_it;
    kontrollelement_t *cur_spor, *cur_fasit;

    spor_it = list_createiterator(spor);
    if (spor_it == NULL)
        return -1;

    fasit_it = list_createiterator(fasit);
    if (fasit_it == NULL)
        return -1;

    while((cur_spor = list_next(spor_it)) != NULL &&
          (cur_fasit = list_next(fasit_it)) != NULL) {
        if (cur_spor->kode != cur_fasit->kode)
            break;
    }

    list_destroyiterator(spor_it);
    list_destroyiterator(fasit_it);

    if (cur_spor != NULL || cur_fasit != NULL)
        return 0;

    return 1;
}
```

b)
```
double tid(list_t *spor) {
    list_iterator_t *iter;
    kontrollelement_t *passering;
    double total;

    total = 0.0;
    iter = list_createiterator(spor);
    if (iter == NULL)
        return -1;

    while ((passering = list_next(iter)) != NULL) {
        total += passering->strekktid;
    }

    list_destroyiterator(iter);

    return total;
}
```

c)
```
double bestetid(list_t *godkjente) {
    list_iterator_t *godkjent_it, *spor_it;
    double bestetid, sportid;
    list_t *spor;

    bestetid = -1;

    godkjent_it = list_createiterator(godkjente);
    if (godkjent_it == NULL)
        return -1;

    while ((spor = list_next(godkjent_it)) != NULL) {
        sportid = tid(spor);

        if (bestetid == -1 || bestetid > sportid)
            bestetid = sportid;
    }

    list_destroyiterator(godkjent_it);
    return bestetid;
}
```
