UiT
NORGES
ARKTISKE
UNIVERSITET

SENSORVEILEDNING

For eksamen i: **INF-1100**

Dato: **Tirsdag 5. desember 2017**

Sensorveiledningen er på 8 sider inklusiv forside

Fagperson/intern sensor: Robert Pettersen

Telefon: 47247552

Eksamen INF-1100
Innføring i programmering og datamaskiners virkemåte
Høst 2017

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.

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

Interaksjon mellom minne, med write enabled signalet.

Oppgave 2 - 25%

Gitt følgende funksjon:

```
int ukjent(int tall)
{
    int x = 0;

    while (tall != 0) {
        tall = tall >> 1;
        x++;
    }

    return x;
}
```

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

(1) *ukjent(3)*

(2) *ukjent(11)*

b) Hva er det funksjonen beregner?

Løsningsforslag

a) (1) 2

(2) 4

b) Funksjonen beregner antall signifikante bits i tallet, bit-posisjon til most significant bit (msb), log2 av tallet, eller lignende forklaring.

Oppgave 3 - 25%

Gitt et array *A* med heltall (*int* i C) mellom 0 og 99.

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 lengdeA, int k);
```

b) Skriv en funksjon `frekvens` som teller hvor mange ganger tallene mellom 0 og 99 forekommer i A. Funksjonen skal skrive antall forekomster inn i et array B, slik at $B[0]$ inneholder antall forekomster av verdien 0; $B[1]$ inneholder antall forekomster av verdien 1, osv. Funksjonen skal ta som inn-parametre en peker til A, en angivelse av antall elementer i A, samt en peker til B:

```
void frekvens(int *A, int lengdeA, int *B);
```

Løsningsforslag

a) An excellent student will write a compact working implementation using a for loop, and without redundant variables and steps. Please note if the candidate use correct array bounds.

```
int forekommer(int *A, int lengdeA, int k){
  int count = 0;

  for (int x = 0; x < lengdeA; x++) {
    if (A[x] == k) count++;
  }
  return count;
}
```

b) An excellent student will make note of his interpretation of the phrase "mellom 0 og 99". Below we assume this implies the 100 element range $[0, 99]$. An excellent student will also define the range in a macro. Students may assume the B array exists. An important point of this assignment is for the candidate to show reuse of the `forekommer` function above to solve this assignment.

```
#define CMAX 100;
```

```
void frekvens(int *A, int lengdeA, int *B) {
  for (int x = 0; x < CMAX; x++) {
    B[x] = forekommer(A, lengdeA, x);
  }
}
```

Oppgave 4 - 25%

I denne oppgaven skal vi håndtere lister med ukjent antall elementer. Det eksisterer en funksjon `sammenlign` som tar to elementer som argument og returnerer en verdi som indikerer forholdet mellom dem:

```
int sammenlign(void *e1, void *e2);
```

Funksjonen returnerer verdien -1 dersom e1 er mindre enn e2, 0 dersom e1 er lik e2, og 1 dersom e1 er større enn e2.

a) Gitt to lister a og b, begge sortert og med minste element først. Lag funksjonen `merge` som tar a og b som argument og returnerer en ny liste med elementene fra a og b sortert i rekkefølge, fra minst til størst:

```
list_t *merge(list_t *a, list_t *b);
```

b) Gitt to lister a og b, begge sortert og med minste element først i listen, lag en funksjon `mergenum` som tar a og b som argument og returnerer antall unike elementer.

```
int mergenum(list_t *a, list_t *b);
```

Du kan anta at følgende listefunksjoner er tilgengelige:

```
// Lag en ny liste
list_t *list_create(void);

// Fjern og returner første element i en liste
void *list_removefirst(list_t *list);

// Sett inn et element sist i listen
void list_addlast(list_t *list, void *item);

// 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

a) The assignment can be solved using the destructive `removefirst` function, or by not modifying a or b, as in the suggestion below. A good candidate will handle last element and NULL arguments.

```
list_t *merge(list_t *a, list_t *b) {

   list_iterator_t ia, ib;
   void *va, *vb;
   list_t *c = list_create();

   ia = list_createiterator(a);
   ib = list_createiterator(b);
   va = list_next(ia);
   vb = list_next(ib);

   while (va != NULL || vb != NULL) {
     if (va == NULL || sammenlign(va, vb) == 1) {
        list_addlast(c, vb);
        vb = list_next(ib);
        continue;
     }

     if (vb == NULL || sammenlign(va, vb) < 1) {
        list_addlast(c, va);
        va = list_next(ia);
        continue;
     }
   }
    list_destroyiterator(ia);
    list_destroyiterator(ib);
    return c;
}
```

b) One possible solution is to iterate over the merged and sorted list and count 1 each time we find an element unequal to the previous element. A good candidate will handle special cases, like having an empty lists.

```
int mergenum(list_t *a, list_t *b){

   int count = 0;
   void *prev = NULL;
   void *next = NULL;

   list_iterator_t i = list_createiterator(merge(a, b));
```

```
    next = list_next(i);

    while (next != NULL) {

      /* If first element (prev == NULL) or different from
       * the previous one, count 1. Assumes list is sorted */

     if (prev == NULL || sammenlign(next, prev) != 0 ) {
         count++;
      }
      prev = next;
      next = list_next(i);
    }

    list_destroyiterator(i);
    return count;
}
```
