Eksamen INF-1100
Innføring i programmering
Vår 2008

Eksamenssettet består av 3 oppgaver.

Der oppgaven ber om at du skriver en funksjon kan du bruke C lignende pseudokode. Husk også at du kan referere tilbake til funksjoner du tidligere har definert.

Oppgave 1 - 30%

a) Forklar kort hva som ligger i begrepet instruksjonssettarkitektur (*instruction set architecture (ISA)*).

b) Beskriv kort fasene i en instruksjonssyklus ('instruction cycle') i en datamaskin strukturert i henhold til Von Neumann modellen.

c) Forklar hvordan I/O utføres i en Von Neumann basert datamaskin.

Oppgave 2 - 35%

a) To sirkler overlapper dersom sirkelbuene skjærer hverandre eller dersom den ene sirkelen omslutter den andre totalt. Gitt en datastruktur som spesifiserer midtpunkt og radius til en sirkel:

```
typedef struct sirkel sirkel_t;
struct sirkel {
    float x, y;
    float radius;
};
```

Skriv en funksjon *SirkelOverlapp* som avgjør om to sirkler a og b overlapper hverandre:

```
int SirkelOverlapp(sirkel_t *a, sirkel_t *b)
```

Funksjonen skal returnere **1** dersom sirklene overlapper og **0** dersom sirklene ikke overlapper. Hint: Sjekk avstanden mellom sirklenes midtpunkter. For å beregne avstanden mellom to punkter x1,y1 og x2,y2 kan du benytte Pythagoras' formel:

$$avstand = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

Løsningsforslag 2a:

```
int SirkelOverlapp(sirkel_t *a, sirkel_t *b)
{
    // Variabel for å mellomlagre avstanden mellom sirklene.
    float avstand;

    // Regner ut avstanden mellom sirklenes midtpunkter.
    avstand = sqrt(pow(b->x - a->x, 2) + pow(b->y - a->y, 2));

    // Overlapp dersom avstanden er mindre enn summen av radiene.
    if(avstand < a->radius + b->radius)
        return 1;
    else
        return 0;
}
```

b) Gitt en datastruktur som spesifiserer nedre venstre hjørne og høyde og bredde til et rektangel:

```
typedef struct rektangel rektangel_t;
struct rektangel {
    float x, y;
    float høyde, bredde;
};
```

Skriv en funksjon *RektangelOverlapp* som avgjør om to rektangel a og b overlapper hverandre:

```
int RektangelOverlapp(rektangel_t *a, rektangel_t *b)
```

Funksjonen skal returnere **1** dersom rektanglene overlapper og **0** dersom rektanglene ikke overlapper. Hint: Dersom to rektangel ikke overlapper vil det ene rektangelet i sin helhet enten ligge til høyre, venstre, over, eller under det andre rektangelet.

```
Løsningsforslag 2b:
int RektangelOverlapp(rektangel_t *a, rektangel_t *b)
{
    //  Tester om rektangel a sine sider er helt utenfor b.
    if(a->x + a->bredde < b->x || a->x > b->x + b->bredde ||
       a->y - a->høyde > b->y || a->y < b->y - b->høyde)
        return 0;
    else
        return 1;
}
```

Oppgave 3 - 35%

a) Skriv en funksjon *zenosum*:

```
double zenosum(int n)
```

Funksjonen skal beregne og returnere summen opp til n ledd av rekken:

$\frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + ... + \frac{1}{2^n}$

```
Løsningsforslag 3a:
double zenosum(int n)
{
    int i;
    double sum, mult;

    // n må være større enn 0
    if (n <= 0)
        return 0.0;

    // Setter mult til 2
    mult = 2.0;

    // Regner ut det første leddet i summen.
    sum = 1.0 / mult;

    // For hver i mindre enn n-1 dobles mult.
    // Legg også sammen 1/mult leddet i en sum.
    for (i = 0; i < n-1; i++){
        mult = mult * 2;
        sum = sum + 1.0/mult;
    }

    return sum;
}
```

b) Skriv en funksjon *uniketall* som tar som inn-parametre en peker til et integer array og en angivelse av størrelsen på arrayet:

```
int uniketall(int *array, int arraysize)
```

Funksjonen skal returnere antall unike tall i *array*. Funksjonen kan **ikke** modifisere tallene i *array*. Denne oppgaven kan løses på flere måter. Dersom du velger å benytte lister kan du anta at følgende listefunksjoner er tilgjengelige:

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

// Frigi liste. Elementer i listen blir ikke frigitt
void list_destroy(list_t *list);

// Sett inn et element først i en liste
int list_addfirst(list_t *list, void *item);

// Lag en ny listeiterator
list_iterator_t *list_createiterator(list_t *list);

// Frigi listeiterator
void list_destroyiterator(list_iterator_t *iter);

// Returner element som pekes på av iterator og la iterator peke på neste element
void *list_next(list_iterator_t *iter);

// La en iterator peke på første element i listen
void list_resetiterator(list_iterator_t *iter);
```

```
Løsningsforslag 3b:
int uniketall(int *array, int arraysize)
{
    int i, j;
    int unik, antall;

    // Tall som holder styr på om et arrayelement er unikt.
    // 1 for unikt, 0 for ikke. Begynner med å anta at første tall er unikt.
    unik = 1;

    // Starter med 0 unike tall.
    antall = 0;

    // Går gjennom hele arrayet.
    for(i = 0; i < arraysize; i++){
        // Går gjennom arrayet fra index i.
        for(j = i + 1; < arraysize; j++){
            // Sammenlikner arrayelementene.
            if(array[i] == array[j])
                // Hvis en av de er like er det ikke et unikt tall.
                unik = 0;
        }

        // Hvis unik er lik 1 vil array[i] være et unikt tall.
        if(unik == 1)
            antall++;

        // Setter unik lik 1 for neste i.
        unik = 1;
    }

    return antall;
}
```