Eksamen INF-1100

Innføring i programmering

Høst 2009

Løsningsforslag

OPPGAVE 1 – 30%:

Se ande løsningsforslag, eksempelvis H2011.

OPPGAVE 2 – 50%:

```
typedef struct legeme legeme_t;
struct legeme{
    /*
        senter koordinat på legeme
    */
    int posx, posy;
    /*
        bredde, men for sirkel er dette omkrets
    */
    int bredde;
    /*
        hvis -1 er legeme en astroide(sirkel)
        ellers er dette et romskip(kvadrat)
    */
    int liv;
}
```

a)

```
/*
    funksjon tar inn to legemer som begge er kvadrater(romskip)
    returnerer 1 hvis det er overlapp mellom kvadratene.
    Hvis ikke, return 0
*/
int KvadratKollisjon(legeme_t *a, legeme_t *b)
    gjennomsnittBredde = (a->bredde - b->bredde)/2
    avstandX = abs(a->posx - b->posx)
    avstandY = abs(a->posy - b->posy)

    // kollisjon
    if avstandX og avstandY < gjennomsnittBredde
        return 1
    else
        //ikke kollisjon
        return 0
    end if
end func
```

```
/*
    funksjonen tar inn to legemer som begge er sirkler(astroider)
    returnerer 1 hvis det er overlapp mellom sirklene.
    Hvis ikke returneres 0
*/
int SirkelKollisjon(legeme_t *a, legeme_t *b)
    /*
        bruker formel fra oppgavetekst.
        Antar at vi har tilgjenngelig en funksjon sqrt()
        som finner kvadratrot av et tall
    */
    avstand = sqrt((b->posx -a->posx)^2 + (b->posy - a->posy)^2)

    radiusA = a->bredde/2
    radiusB = b->bredde/2


    // kollisjon
    if avstand < (radiusA + radiusB)
        return 1
    else
        // ikke kollisjon
        return 0
    end if
end func
```

b)

Funksjonen vil ikke fungere 100%, men følger oppgaveteksten!!
Hva skjer f.eks hvis en liten sirkel er midt i et stort kvadrat?
Funksjonen ville gitt kollisjonn lik 0
Her er det viktig å gå ut ifra at det som skrives i oppgaveteksten er korrekt, og vil fungere. Det er ikke din jobb å finne den optimale løsning.

```
/*

    funksjonen tar in to vilkårlige
    legemer. Sirkler eller kvadrater.
    Returnerer 1 hvis det er kollisjon
    hvis ikke returneres 0

*/

int kollisjon(legeme_t *a, legeme_t *b)

    kollisjon = 0

    // to sirkler
    if a->liv og b->liv == -1
        kollisjon = sirkelKollisjon(a, b)

    // to kvadrater
    else if a->liv og b->liv >=  0
        kollisjon = kvadratKollisjon(a, b)

    // to ulike legemer
    else
        // lat først som om at begge er sirkler
        kollisjon = sirkelKollisjon(a, b)

        // hvis det ikke er kollisjon mellom 2 sirkler:
        if kollisjon != 1


            /* ettersom vi må holde styr på hvilke av de to
                objektene som er sirkel og hvilken som er kvadrat
                laget jeg 2 tmp variabler og setter den som er sirkel lik
                "sirkel" og den som er kvadrat lik "kvadrat"
            */
            legeme_t *sirkel, kvadrat

            if a->liv lik -1
                sirkel = a
                kvadrat = b
            else
                sirkel = b
                kvadrat = a
            end if

            radius = sirkel->bredde/2


            /*
                her kan du regne ut et eller to punkter og
                si at du regner de andre på samme måte
```

```
            */
            topVenstreX = kvadrat->posx - (bredde/2)
            topVenstreY = kvadrat->posy - (bredde/2)

            bunnVenstreX = kvadrat->posx - (bredde/2)
            bunnVenstreY = kvadrat->posy + (bredde/2)

            topHøyreX = kvadrat->posx + (bredde/2)
            topHøyreY = kvadrat->posy - (bredde/2)

            bunnHøyreX = kvadrat->posx + (bredde/2)
            bunnHøyreY = kvadrat->posy + (bredde/2)


            /*
                her kan du også regne avstand for 1 eller 2 punkter
                og si at du gjør det samme for de andre
                tegn figur!
            */
            avstand1 = sqrt((topVentreX - sirkel->posx)^2 +
                (topVentreY - sirkel->posy)^2)
            avstand2 = sqrt((bunnVenstreX - sirkel->posx)^2 +
                (bunnVenstreY - sirkel->posy)^2)
            avstand3 = sqrt((topHøyreX - sirkel->posx)^2 +
                (topHøyreY - sirkel->posy)^2)
            avstand4 = sqrt((bunnHøyreX - sirkel->posx)^2 +
                (bunnHøyreY - sirkel->posy)^2)


            /*
                kollisjon
                hvis en av avstandene er mindre en radius
            */

            if avstand1 || avstand2 || avstand3 || avstand4 < radius
                kollisjon = 1
            else
                // ikke kollisjon
                kollisjon = 0
            end if
        end if
    end if

    return kollisjon
end func
```

c)

```
/*
    list struktur holder styr på antall noder i lista
    og holder peker til den første noden i lista
*/
struct list
    // antall noder i lista
    int numitems

    // peker til første node i lista
    listnode *head
end struct

/*
    struktur for noder som skal lenkes til lista
*/
struct listnode
    // peker til data i lista
    void *item
    // peker til neste node i lista
    listnode *next
end struct


/*
    iterator brukes til å gå gjennom lista
*/
struct list_iterator
    // peker til lista som ksl itereres
    list *list

    // peker til en listnode
    listnode *next
end struct
```

d)

Her går vi ut ifra at vi har tilgjengelig en full liste implementasjon og dermed kan bruke disse uten å implementere de fra bunnen av.

```
/*
    funksjonen gå igjennom en lista og fjerner
    romskip som har brukt opp sine liv. liv = 0
    Funksjonen returnerer antall romskip igjen i lista.
*/
int Renskopp(list_t *list)
    // brukers til å telle romskip
    romskip = 0

    // lager en ny iterator
    iter = create_listiterator(list)
    /*
        gå til neste node i lista og
```

```
        returnerer ut current item(legeme)
        Hvis lista er tom, return NULL
    */
    legeme = list_next(iter)

    /*
        løkke som går til lista er tom
    */
    while legeme != NULL
        // hvis ikke flere liv
        if legeme->liv == 0
            // slett fra lista
            list_remove(list, legeme)
        else
            if legeme->liv > 0
                romskip ++
            end if
        end if
        /*
            gå til neste node i lista og
            returnerer ut current item(legeme)
            Hvis lista er tom, return NULL
        */
        legeme = list_next(iter)
    end while

    // frigir minne gitt til iterator i create_iterator
    free(iter)

    // returnerer antall romskip i lista
    return romskip
end func
```

OPPGAVE 3 – 20%:

Rekursjon er ikke pensum etter forandring av kurset, dvs fra H2011