Opgave 1 (20%)

a) Skriv et prædikat

$$even(x)$$

der udtrykker, at heltallet x er et ikke-negativt lige tal.

I det følgende er h ***halveringsfunktionen***, det vil sige, en funktion som tilfredsstiller specifikationen:

$$h(x) \ \underline{\textbf{sat}} \ even(x) \ \rightarrow \ x = h' + h'$$

Betragt nu algoritmen:

Algoritme: Multiplikation
Stimulans: a,b: Int, $(a \geq 0) \wedge (b \geq 0)$
Respons: z: Int, z=a*b
Metode: ≪initialiser x, y og z≫
do { x*y+z=a*b }
    x≠0 → **if** *even*(x) → ≪step 1≫
        & true → ≪step 2≫
        fi
od

b) Udfyld de ubestemte stumper, så algoritmen bliver gyldig og korrekt. Det er kun tilladt at anvende addition, subtraktion og halvering. Der ønskes et bevis for gyldigheden og korrektheden.

c) Angiv, udtrykt ved a, antallet af additioner, subtraktioner og halveringer, algoritmen udfører.

Opgave 2 (20%)

Der skal konstrueres en box Multi med følgende udseende

```
Box Multi
    Type M = ≪sæk af heltal≫
    Proc Init[m: M]
    Proc Insert[m: M](i: Int)
    Proc Remove[m: M](i: Int)
    Proc Peek[m: M](p: Int) → (List(Int))
end Multi
```

som realiserer en datastruktur, hvis værdier er ***sække*** af heltal.

Proceduren Init giver den tomme sæk. Proceduren Insert indsætter en forekomst af tallet i. Proceduren Remove fjerner en forekomst af tallet i. Proceduren Peek giver listen af de elementer i sækken m, der optræder netop p gange, For sækken:

$$\mathrm{m}: \; < 0, 3, 4, 4, 7, 8, 8, 8, 8, 9, 11, 11, 11, 13, 17, 17 >$$

vil Peek[m](2) således returnere listen $(4, 17)$, Peek[m](4) listen (8) og Peek[m](7) den tomme liste.

I det følgende angiver $||\mathrm{m}||$ antallet af ***forskellige*** elementer i sækken m.

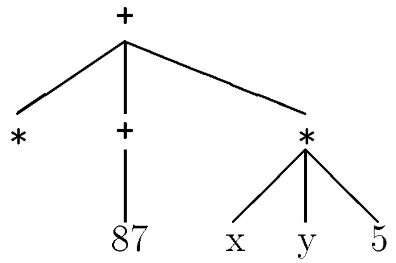
> Beskriv en realisation af typen M, så Init får tidskompleksitet $O(1)$, Insert og Remove får tidskompleksitet $O(\log ||\mathrm{m}||)$, og Peek får tidskompleksitet $O(k + \log ||\mathrm{m}||)$, hvor k er længden af den liste, der returneres.
>
> Det kræves ikke, at besvarelsen indeholder et TRINE program.

Opgave 3 (20%)

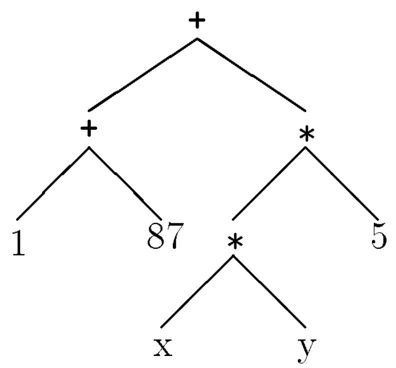
Typerne Exp1 og Exp2 kan begge repræsentere regneudtryk med talkonstanter, variabler, plus og gange.

> **Type** $\mathrm{Exp}_1 = $ **Sum**(const: Int, id: Text, plus, times: Arg_1)
> **Type** $\mathrm{Arg}_1 = $ **Prod**(left, right: Exp_1)
>
> **Type** $\mathrm{Exp}_2 = $ **Sum**(const: Int, id: Text, plus, times: Arg_2)
> **Type** $\mathrm{Arg}_2 = $ **List**(Exp_2)

I forbindelse med Exp_1 har plus og times altid to argumenter, hvorimod Exp_2 tillader et vilkårligt antal argumenter til disse operatorer. Der gælder her, at plus med nul argumenter har værdi 0, at times med nul argumenter har værdi 1, og at plus eller times med et enkelt argument har samme værdi som argumentet selv. Fx er følgende Exp_2-udtryk:



ækvivalent med dette Exp_1-udtryk:



Skriv en Trine procedure:

Proc Trans[e_2: Exp_2] $\rightarrow$ (Exp_1)

der oversætter et Exp_2-udtryk til et ækvivalent Exp_1-udtryk. Der lægges vægt på, at besvarelsen er letlæselig, detaljeret og korrekt.

Opgave 4 (15%)

Betragt følgende TRINE program.

```
(+ Type A = List(B)
   Type B = Prod(x: A, y: C, z: Bool)
   Type C = Sum(x: A, y: C, z: Bool)

   Var a: A
   Var b: B
   Var c: C

   a := A(B(a, C(x: A()), true))
   read[c]
   b := B(c.x, c.y, c.z)
+)
```

a) Angiv normalformen af typen A.

b) Vil programmet blive accepteret af TRINE oversætteren? Begrund dit svar.

c) Hvad ville der ske, hvis programmet blev udført? Begrund dit svar.

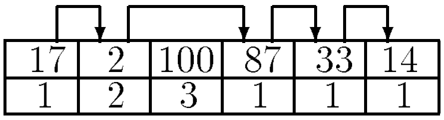
Opgave 5 (25%)

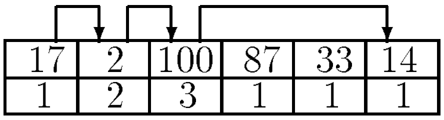
Et spillebræt består en række felter, der hver består af to tal:

17	2	100	87	33	14
1	2	3	1	1	1

Spillet går ud på at hoppe fra det første til det sidste felt i rækken. Det øverste tal angiver omkostningen ved at besøge et felt. Det nederste tal (1, 2 eller 3) angiver hvor langt til højre, man højst må hoppe fra dette felt. Den samlede omkostning ved et spil er summen at omkostningerne af de felter, der besøges.

Følgende ses to lovlige spil, med samlet omkostning henholdsvis 153 og 133:





Det ses, at "grådighed" ikke altid betaler sig.

Følgende program indlæser et spillebræt og beregner den samlede omkostning af det billigste spil. Her er min(x,y) den sædvanlige minimumsfunktion på heltal.

```
(+ Type Board = List(Prod(cost: Int, jump: Sum(one, two, three: Unit)))

   Proc Cheap[B: Board](i: Int) → (Int)
      if i ≥ |B| → return 0 fi
      (+ Var s1, s2, s3: Int
         s1 := B.(i).cost+Cheap[B](i+1)
         s2 := B.(i).cost+Cheap[B](i+2)
         s3 := B.(i).cost+Cheap[B](i+3)
         if is(B.(i).jump, one) → return s1
         & is(B.(i).jump, two) → return min(s1, s2)
         & is(B.(i).jump, three) → ≪case three≫
         fi
      +)
   end Cheap
```

```
    Var B: Board

    read[B]
    write(Cheap[B](0))
+)
```

a) Udfyld ≪case three≫, så programmet bliver færdigt.

b) Udled tidskompleksiteten af programmet.

c) Modificér programmet, så det anvender dynamisk programmering til at forbedre effektiviteten. Hvad bliver den nye tidskompleksitet?