Forritunarmál

Benjamín

11.27.2024

TÖL304G Lokapróf

Árið 2018

Hluti I - Bálkmótun o.fl.

Svarið að minnsta kosti tveimur spurningum í þessum hluti - Munið að svara a.m.k. 10 spurningum í heild

1.

Hverjar eftirfarandi fullyrðinga um lokanir eru sannar? Tvö röng svör gefa núll punkta.

a) Lokanir innihalda fallsbendi.

b) Lokanir eru til í C.

c) Lokanir eru til í Scheme.

d) Lokanir eru til í CAML.

e) Lokanir eru til í Morpho.

f) Lokanir eru aðeins mögulegar ef vakningarfærslur eru í kös.

g) Lokanir má nota til að utfæra strauma í Scheme.

h) Lokanir innihalda tengihlekk(aðgangshlekk).

i) Lokanir innihalda stýrihlekk.

j) Lokanir innihalda straum.

k) Lokanir eru nauðsynlegar til að skila staðværu falli sem skilagildi falls í bálkmótuðum forritunarmálum.

l) Lokanir eru nauðsynlegat til að senda staðvær föll sem viðföng í föll í bálkmótuðum forritunarmálum.

Svar:

Förum í gegnum alla svarmöguleikana til að æfa okkur:

a. Lokanir innihalda fallsbendi?: Já Lokanir innihalda kóðann fyrir fallið (fallsbendil) og Umhverfið.

b. Lokanir eru til í C? Nei C styður ekki lokanir

c. Lokanir eru til í Scheme? Já Scheme styður lokanir og notar þær mikið!

d. Lokanir eru til í CAML? Já CAML styður lokanir

e. Lokanir eru til í Morpho? Já Morpho styður lokanir.

f. Lokanir eru einungis mögulegar ef vakningarfærslur eru í kös? Nei ekki satt.

g. Lokanir má nota til að utfæra strauma í Scheme? Já Straumar krefjast þess að við geymum útreikninga sem eru gerðir síðar, lokanir eru fullkomnar til þess.

h. Lokanir innihalda tengihlekk? Já, lokanir innihalda tengihlekk sem bendir á umhverfið þar sem fallið var skilgreint

i. Lokanir innihalda stýrihlekk? Neibb, Lokanir innihalda ekki stýrihlekk, þar sem stýrihlekkur tengist kallaröðinni, ekki umhverfinu.

j. Lokanir innihalda straum? Nei, lokanir sjálfar innihalda ekki straum.

k. Lokanir eru nauðsynlegar til að skila staðværu falli sem skilagildi falls í bálkmótuðum forritunarmálum? Já, til að skila staðværu falli og tryggja að það hafi aðganga að upprunalegu umhverfi sínu, þarf lokun.

l. Lokanir eru nauðsynlegar til að senda staðvær föll sem viðföng í föll í bálkmótuðum forritunarmálum? Sama röksemd og í k. til að fallið hafi aðgang að umhverfi sínu þegar það er notað sem viðfang, þarf lokun.

a	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**
x		**x**	**x**	**x**		**x**	**x**			**x**	**x**

2.

Vakningarfærsla falls í bálkmótuðu forritunarmáli eins og Scheme inniheldur sum eftirfarandi atriða. Hver? Eitt rangt svar gefur núll stig.

a) Staðværar breytur fallsins.

b) Staðværar breytur fallsins sem kallaði á fallið.

c) Víðværar breytur sem eru aðgengilegar í fallinu.

d) Skráarkerfi tölvunnar.

e) Viðföng fallsins.

f) Aðgangshlekk (tengihlekk).

g) Stýrihlekk.

h) Vendivistfang.

i) Benda á öll föll sem hægt er að kalla á úr fallinu.

j) Benda á allar lifandi vakningarfærslur.

k) Alla hluti sem eru í kerfinu.

l) Vakningarfærslur allra falla sem hægt er að kalla á.

m) Nöfn allra falla sem hægt er að kalla á.

Svar:

a	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**
x				**x**	**x**	**x**	**x**					

3.

Vakningarfærsla falls í bálkmótuðu forritunarmáli eins og Scheme inniheldur sum eftirfarandi atriða. Hver? Eitt rangt svar gefur núll sitg.

a) Staðværar breytur fallsins.

b) Staðværar breytur fallsins sem kallaði á fallið

c) Víðvlrar breytur sem eru aðgengilegar i fallinu.

d) Skraarkerfi tölvunnar.

e) viðföng fallsins.

f) Aðgangshlekk (tengihlekk).

g) Stýrihlekk.

h) Vendivistfang.

i) Benda á öll föll sem hægt er að kalla á úr fallinu.

j) Benda á allar lifandi vakningarfærslur.

k) Alla hlutina sem til eru í kerfinu.

l) Vakningarfærslur allra falla sem hægt er að kalla á.

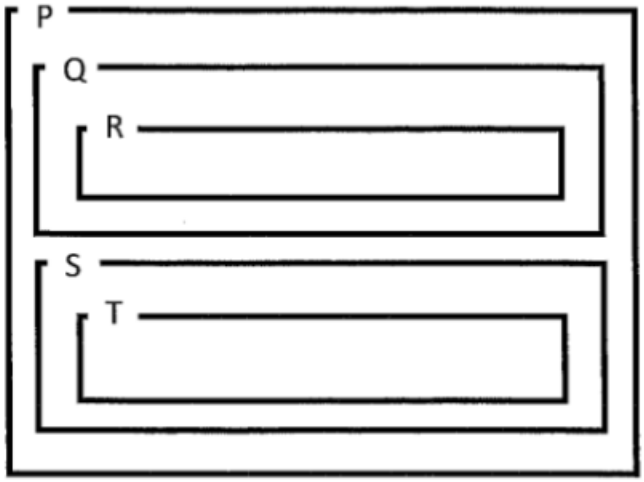
m) Nöfn allra falla sem hægt er að kalla á.

Svar:

a	**b**	**c**	**d**	**e**	**f**	**g**	**h**	**i**	**j**	**k**	**l**	**m**

4.

Íhugið myndina sem sýnir földun falla P, Q, R, S og T.



Samsvarandi Scheme forritstexti er einnig sýndur í tveimur jafngildum útgáfum hlið við hlið.

```
(define ( P...)
 (define (Q ...)
  (define (R ...)
   ...[stifn R/body of R]
  )
  ...[stofn Q/body of Q]
 )
 (define (S ...)
  (define (T...)
   ...[stofn T/body of T]
  )
  ...[stofn S/body of S]
 )
 ...[stofn P/body of P]
)
```

```
(define(P ...)
 (define(S ...)
  (define (T...)
    ...[stofn T/body of T]
  )
  ...[stofn S/body of S]
 )
 (define (Q ...)
  (define (R ...)
   ...[stofn R/body of R]
  )
  ...[stofn Q/body of Q]
 )
 ...[stofn P/body of P]
 )
```

Fyllið út eftirfarandi töflur með því að setja krossa við sannar fullyrðingar. Eitt rangt svar gefur núll í einkunn fyrir dæmið.

Svar:

Í þessu dæmi gildir að Það má alltaf kalla á ytra fall og beint barn. ekki barnabarn það skiptir ekki máli með kallanir hvort það

kalla má á P úr:

P	**Q**	**R**	**S**	**T**
x	**x**	**x**	**x**	**x**

kalla má á Q úr:

P	**Q**	**R**	**S**	**T**
x	**x**	**x**	**x**	**x**

kalla má á R úr:

P	Q	R	S	T
	x	x	x	x

kalla má á S úr:

P	Q	R	S	T
	x	x	x	x

kalla má á T úr:

P	Q	R	S	T
	x	x	x	x

Staðværar breytur í P má nota í:

P	Q	R	S	T
x	x	x	x	x

Staðværar breytur í Q má nota í:

P	Q	R	S	T
	x	x		

Staðværar breytur í R má nota í:

P	Q	R	S	T
		x		

Staðværar breytur í S má nota í:

P	Q	R	S	T
			x	x

5.

Eftirfarandi forritstexti er í einhverju ímynduðu forritunarmáli.

```
void f(x,y)
{
    y = 3;
    print x,y;
    x = 2;
}
int i,a[10];
for(int i= 0; i != 10; i++) a[i] = i +1;
f(a[a[0]],a[0]);
print a[0], a[1], a[2], a[3];
```

Hvað skrifar þetta forrit (sex gildi í hvert skipti) ef viðföngin eru:

a)

Gildisviðföng

Svar: Viðföngin eru metin og gildin þeirra eru afrituð inn í fallið. Breytingar á viðföngum innan fallsins hafa engin áhrif á upprunalegu breyturnar.

Svo þá munu gildin í f ekki breyta upprunulegu gildunum svo:

```
for lykkjan gerir
a[0] = 1
a[1] = 2
a[2] = 3
a[3] = 4
a[4] = 5
a[5] = 6
a[6] = 7
a[7] = 8
a[8] = 9
a[9] = 10
svo
f(a[a[0]], a[0])
gerir
f(a[1], a[0])
sem kallar á
f(2, 1)

svo prentað
2 3
1 2 3 4
```

b)

Tilvísunarviðföng Hér eru viðföngin send sem vísanir á upprunalegu breyturnar svo Breytingar á þeim innan fallsins hafa áhrif á upprunalegu breyturnar.

svo þá er a[0] til a[9] upphafsstillt í 1 til 9

en þegar f(a[a[0]], a[0]) er kallað þá breytir það stillingunni á a[0] og a[1]

svo útprentað verður

```
2 3
3 2 3 4
```

c)

Nafnviðföng

Viðföngin eru send sem óútreiknaðar segðir og eru metin í hvert

svo inni i f:

```
1. y = 3
    a[0] = verður þá 3
2. print x, y;
    x er a[a[0]], þar sem a[0] er núna 3, þannig að x er a[3] sem er 4
    y er a[0] sem er 3
    svo prentað er 4 3
3. x = 2
   a[a[0]] =2;
   þar sem a[0] er 3 þá er þetta a[3] = 2 svo a[3] er 2
```

Eftir fallið f þá prentum við

a[0], a[1], a[2], a[3];

prentar :

3 2 3 2

Saman: **4 3 3 2 3 2**

Hluti II - Listavinnsla o.fl.

Svarið að minnsta kosti tveimur spurningum i þessum hluta - Munið að svara a.m.k. 10 spurningum í heild

6.

Skrifið fall í Scheme, Caml, Morpho eða Haskell sem tekur eitt viðfang sem er listi lista af fleytitölum milli 0 og 1 og skilar tölu sem er stærsta lággildi innri listanna, þ.e. stærst af þeim tölum sem fást þegar fundinn er minnsta tala í hverjum innri lista. Þið skuluð reikna með því að hágildi í tóma menginu sé 0 og lággildi í tóma menginu sé 1. Munið fallslýsingar, eins og alltaf. Fallið þarf að skila viðeigandi gildi bæði fyrir toman lista og fyrir lista sem einungis inniheldur tóma lista.

Svar:

í Scheme: Ég ætla nota bara grunnskipanir

```
;;; Notkun: (min-i-lista lst)
;;; Fyrir: lst er listi af fleytitölum á milli 0 og 1
;;; Eftir: skilar minnsta gildi í listanum, eða 1 ef listinn er tómur
(define (min-i-lista lst)
    (if (null? lst)
        1
        (min-helper (car lst) (cdr lst))
    )
)

;;; Notkun: (min-helper current-min lst)
;;; Fyrir: current min er fleytitala, lst er listi af fleytitölum
;;; Eftir: Skilar minnsta gildi í listanum
(define (min-helper current-min lst)
    (if (null? lst)
        current-min
        (let ((next-value (car lst)))
            (if (< next-value current-min)
                (min-helper next-value (cdr lst))
                (min-helper current-min (cdr lst))
            )
        )
    )
)

;;; Notkun: (max-i-lista lst)
;;; Fyrir: lst er listi af fleytitölum
;;; Gildi: Skilar stærsta gildinu í listanum, eða 0 ef listinn er tómur
(define (max-i-lista lst)
    (if (null? lst)
        0
        (max-helper (car lst) (cdr lst))
    )
)

;;; Notkun: (max-helper current-max lst)
;;; Fyrir: lst er listi af fleytitölum current-max er fleytitala
;;; Gildi: Skilar stærsta gildinu í listanum lst
```

```
(define (max-helper current-max lst)
    (if (null? lst)
        current-max
        (let ((next-value (car lst)))
            (if (> next-value current-max)
                (max-helper next-value (cdr lst))
                (max-helper current-max (cdr lst))
            )
        )
    )
)

;;; Notkun: (list-of-mins lst-of-lsts)
;;; Fyrir: lst-of-lsts er listi af fleytitölu listum fleytitölurnar eru
 á milli 0 og 1
;;; Eftir: skilar lista af minnstu gildum í öllum listunum
(define (list-of-mins lst-of-lsts)
    (if (null? lst-of-lsts)
        '()
        (let((min-value (min-i-lista (car lst-of-lsts))))
            (cons min-value (list-of-mins (cdr lst-of-lsts)))
        )
    )
)

;;; Notkun: (largest-min lst-of-lsts)
;;; Fyrir: lst-of-lsts er listi af listum af fleytitölum á milli 0 og 1
;;; Eftir: skilar stærsta lággildi innri listanna
(define (largest-min lst-of-lsts)
    (let((mins (list-of-mins lst-of-lsts)))
        (max-i-lista mins)
    )
)
```

7.

Skrifið fall Zip2 i Sheme, CAML, Morpho eða Haskell sem tekur tvíundaraðgerð (fall) og tvo jafnlanga lista sem viðföng og skilar lista þeirra útkomna sem fást þegar tvíundaraðgerðinni er beitt á gildin í listunum, par fyrir par. Til dæmis, í Scheme þá ætti segðin (zip2 + '(1 2 3) '(4 5 6)) að skila listanum (5 7 9). Notið einungis einfaldar aðgerðir svo sem car, cdr, cons, null?

Svar:

Skrifum í scheme:

```
;;; Notkun: (zip2 bin lst1 lst2)
;;; Fyrir: bin er tvíundaraðgerð, lst1 og lst2 eru jafnlangir listar.
;;; Eftir: skilar niðurstöðu tvíundaraðgerðinni á lst1 og lst2 í lista
(define (zip2 bin lst1 lst2)
   (if (or (null? lst1) (null? lst2))
    '()
     (cons (bin (car lst1) (car lst2))
           (zip2 bin (cdr lst1) (cdr lst2))
     )
   )
)
```

8.

Skrifið ykkar eigin útgáfur af föllunum tveimur sem í CAML Light eru kölluð list_it og list_it. Í Haskell eru þau kölluð foldl og foldr. Þið megið skrifa þessi föll í Scheme, CAML, Morpho eða Haskell. Notið ekki lykkjur í Morpho. Kallið föllin myLeft og myRight. þið megið nota aðra röð viðfanga en í it_list og it_list. Sjáið til þess að a.m.k. annað fallið sé Halaendurkvæmt og tiltakið hvort það er. Notið aðeins einföld innbygð föll svo sem car, cdr og null?.

Halaendurkvæma fallið er:

Forritstexti (með lýsingum):

```
 ;;; Notkun: (myLeft bin init lst)
 ;;; Fyrir:
 ;;;        - bin er tvíundaraðgerð
 ;;;        - init er upphafsgildi
 ;;;        - lst er listi af gildum
 ;;; Eftir: skilar niðurstöðu þess að fella lst frá vinstri með f og
init
 (define (myLeft bin init lst)
    (if (null? lst)
    init
    (myLeft bin (bin (car lst)) (cdr lst))
    )
 )
 ;;; Notkun: (myRight bin init lst)
 ;;; Fyrir:
 ;;;        - bin er tvíundaraðgerð
 ;;;        - init er upphafsgildi
 ;;;        - lst er listi af gildum
 ;;; Eftir: Skilar niðurstöðu þessa að fella lst frá hægri með f og
init
 (define (myRight bin init lst)
    (if (null? lst)
        init
        (bin (car lst) (myRight bin init (cdr lst)))
    )
 )
```

9.

Skrifið halaendurkvæmt fall í Scheme, CAML, MORPHO eða Haskell sem tekur sem viðföng einn lista talna, x, auk tveggja talna a og b, og skilar lista þeirra talna z innan x þar sem $a \leq z \leq$. Þið munuð vilja nota hjálparfall.

Svar:

```
;;; Notkun: (filter-range x a b)
;;; Fyrir:
;;;         - x er listi af tölum
;;;         - a og b eru tölur
;;; Eftur: Skilar lista af þeim tölum z í x þar sem a <= z <= b.
(define (filter-range x a b)
    (filter-range-helper x a b '())
)

;;; Notkun: (filter-range-helper lst a b acc)
;;; Fyrir:
;;;         - lst er listi af tölum
;;;         - a, b eru tölur
;;;         - acc er listi
;;; Eftir: Skilar lista af tölum í lst sem eru >= a og <= b.
(define (filter-range-helper lst a b acc)
    (if (null? lst)
        (reverse acc)
        (let ((z (car lst)))
            (if (and (>= z a) (<= z b))
                (filter-range-helper (cdr lst) a b (cons z acc))
                (filter-range-helper (cdr lst) a b acc)
            )
        )
    )
)
```

Hluti III - Einingarforritun o.fl.

Svarið að minnsta kosti tveimur spurningum í þessum hluta - Munið að svara a.m.k. 10 spurningum í heild

10.

Útfærið, að hluta, einingu fyrir fjölnota forgangsbiðröð í Morpho. Sýnið eftirfarandi.

a. Hönnunarskjal sem inniheldur lýsingar (notkun/fyrir/eftir) fyrir öll influtt og útflutt atriði einingarinnar.

b. Smíð einingarinnar, þar sem sleppa má útfærslu allra aðgerða nema þeirri sem bætir gildi í forgangsbiðröðina. Athugið að sýna þarf fastayrðingu gagna.

Unnt skal vera að nota einingaraðgerðir til að búa til afbrigði af einingunni sem gefa forgangsbiðraðir fyrir hvaða gildi sem er sem hafa viðeigandi samanburðarfall. Þið ráðið hvort forgangsbiðr-ðin er Útfærð sem hlutur eða ekki.

Svar:

```
{;;;
Hönnun
======

Útflutt atriði
--------------

Notkun: pq = makePQ();
Fyrir:  Ekkert
Eftir:  pq er ný tóm forgangsbiðröð

Innflutt atriði
---------------

Notkun: b = x <<== y;
Fyrir:  x og y eru gildi af þeirri gerð sem við notendum
        sem lykla í forgangsbiðröðum
Eftir:  b er satt ef lykillinn x má vera á undan lyklinum y

Notkun: b = pq.isEmpty() ;
Fyrir:  pq er forgangsbiðröð
Eftir:  b er satt þá og því aðeins að pq sé tóm

Notkun: g = pq.get();
Fyrir:  pq er forgangsbiðröð, ekki tóm
Eftir:  g er gildi sem var fjarlægt ú pq.
        g hafði lykil sem mátti vera fyrir framan
        lykla allra annara gilda í pq.

Notkun: pq.add(k,g);
Fyrir:  pq er forgangsbiðröð,
        k er lykilgildi þ.e. gildi sem er löglegt
Eftir:  Búið er að bæta gildinu g í forgangsbiðröðina
        undir lyklinum k.
```

```
;;;}

"pq.mmod" =
{{
makePQ =
    obj()
    {
        var list = [];
        {;;;
            Fastayrðing gagna:
             Forgangsbiðröð sem inniheldur lykla/gilda-pör
             (k1,g1),...,(kN,gN), þar sem fremri lyklar í
             rununni mega vera fyrir framan aftari lykla,
             er geymd með list == [[k1$g1],...,[kN$gN]]
        ;;;}
        msg isEmpty()
            {
                ...
            };
        msg put(k,g)
            {
                ...
            };
        msg get()
            {
                val res = tail(head(list));
                list = tail(list);
                return res;
            };
    }

}}
;
```

11.

Hverjar af eftirfarandi fullyrðingum eru í samræmi við meginregluna um upplýsingahuld? Það gætu verið núll, ein eða fleiri. TVö röng svör gefa núll stig.

a. Fastayrðing gagna einingar skal vera fullkomlega aðgengileg smiðum einingarinnar.

b. Notendur einingar geta breytt gastayrðingu gagna einingarinnar.

c. Gefa skal notendum einingar fullkomnar upplýsingar um smíð einingarinnar.

d. Fastayrðing gagna einingar skal vera fullkomlega aðgengileg notendum einingarinnar.

e. Fastayrðing gagna einingar skal ekki vera aðgengileg smiðum einingarinnar.

f. Fastayrðing gagna einingar skal ekki vera aðgengileg notendum einingarinnar.

g. Smiðir einingar geta breytt fastayrðingu gagna einingarinnar.

Svar:

a	**b**	**c**	**d**	**e**	**f**	**g**
					x	**x**

12.

íhugið klasa A og B sem er undirklasi A. Gerið ráð fyrir að klasi A innihaldi boð f með forskilyrði F_A og eftirskilyrði E_A.

```
//Notkun: x.f(...);
//Fyrir:  F_A
//Eftir:  E_A
```

Gerið ráð fyrir að í klasa B sér boðið f endurskilgreint með

```
//Notkun: x.f(...);
//Fyrir:  F_A
//Eftir:  E_A
```

Hvaða vensl þurfa þa gilda milli F_A, E_A, F_B og E_B? Svarið þarf að vera tæmandi, þ.e. tilgreina nauðsyndleg og nægjanleg vensl.

Svar:

til að tryggja rétta arfgerð og að undirklasinn B upfylli **Liskov Substitution Principle** þurfa ákveðin vensl að gilda milli forskilyrða og eftirskilyrða boðanna í f í **A** og **B**

Liskov Substitution Principle (LSP)

*Ef fyrirhugað er að **S** sé undirklasi **T**, þá ættu hlutir af gerð **T** að geta verið skipt út fyrir hluti af gerð **S** án þess að hafi áhrif á réttmæti forritsins."*

Hluti IV - Ýmislegt

Svarið að minnsta kosti einni spurningu í þessum hluta Munið að svara a.m.k. 10 spurningum í heild

13.

Lýsið ruslasöfnunaraðferðinni sem gengur undir nafninu tilvísunartalning. Koma þarf fastayrðing gagna og undir hvaða kringumstæðum minni er skilað.

Svar:

tilvísunartalning er ein af mörgum ruslasöfnunaraðferðum sem notaðar eru til að stjórna minni í forritum. Aðferðin felst í því að halda utan um fjölda virkra tilvísana á hvern hlut í forritinu. þegar tilvísunartalning hlutar verður núll, er vitað að engar tilvísanir eru lengur til á hlutinn, og því er hægt að endurheimta það minni sem hann notar.

Hvernig tilvísunartalningin virkar:

1. Uphhafsstilling - Þegar nýr hlutur er búinn til, er tilvísunartalning hans upphafsstillt a **1**, þar sem a.m.k. ein tilvísun a hann (t.d. frá breytu sem heldur utan um hann).
2. Aukning tilvísunartalningar: - Þegar ný tilvísun á hlut er búin til(t.d. þegar hluturinn er framsendur í fall, settur í gagnastrúktur eða uthlutað til annarrar breytu), er tilvísunartalningin aukin um 1.
3. Minnkun tilvísunartalningar: - Þegar tilvísun er eytt eða breyta sem hélt utan um hlutinn fer úr umfangi (scope), er tilvísunartalning hans lækkuð um 1.
4. Endurheimt minnis: - Ef tilvísunartalning hlutar verður núll eftir að hafa verið lækkuð, er hluturinn ekki lengur apgengilegur og minni hans endurheimt. - Þetta getur leitt til keðjuverkandi endurheimtar, þar sem hlutur sem er eytt getur haft tilvísanir á aðra hluti, sem síðan missa eina tilvísun og gætu prðið með tilvísunartalningu núll.

Fastayrðing gagna:

- tilvísunartalning hvers hlutar er alltaf jafngild fjölda virkra tilvisana á hann í forritinu. Þetta þýðir að á hverjum tíma keyrslu forritsins:
- tilvísunartalning hlutar endurspeglar nákvæmlega fjölda tilvisana sem eru í notkun á hlutinn,
- Þegar tilvísun er bætt við eða fjarlægð, er tilvisunartalningin uppfærð samstundis til að viðhalda fastayrðingunni.

14.

Sýnið BNF, EBNF, samhengisfrjálsa mállýsingu (CFG) eða málrit fyrir mál lölegra segða yfir stafróið {x,+,(,)} þar sem x er breytunafn og $+$ er tvíundaraðgerð

Dæmi um streng í málinu.

```
x
x + x
x + (x)
(((x)))
x + (x + x + (x + x) + x) + x + x
```

Dæmi um streng ekki í málinu.

```
\epsilon (tómi strengurinn)
(
)
+x
((x)
```