## A.23 Calculați diferența a două mulțimi.

Completarea enunțului

O mulțime este o secvență liniară de numere reale. Să se dea o funcție care preia două mulțimi și returnează diferența dintre prima mulțime și cea de a doua.

Specificare

Diferența(prima, aDoua, rezultat)

* Date de intrare:
  + prima: o secvență de numere reale, deține un câmp denumit Lungime egal cu numărul total de elemente conținute.
  + aDoua: o secvență de numere reale, deține un câmp denumit Lungime egal cu numărul total de elemente conținute.
* Date de ieșire:
  + rezultat: o secvență de numere reale, deține un câmp denumit Lungime egal cu numărul total de elemente conținute. Secvența este dată de mulțimea {primai | i = 1.. prima.Lungime și primai ≠ aDouaj, j = 1..aDoua.Lungime} iar rezultat.Lungime este egal cu valoarea cardinalului mulțimii respective.

Dezvoltare din specificații

Specificare

Φ: A = (a)A.Lungime ∈ ℝ ^ B = (b)B.Lungime ∈ ℝ

Ψ: C = A / B, unde A / B = {(c)n ≤ A.Lungime, oricare ar fi i ∈ **N** < A.Lungime, exista ci = aj,

j ∈ **N** < A.Lungime si nu exista ci = bj, ∈ < B.Lungime, n = (i maxim + 1)

Program 0

[A = (a)A.Lungime ∈ ℝ ^ B = (b)B.Lungime ∈ ℝ; C= A/B]

Program 1 (compunere)

[A = (a)A.Lungime ∈ ℝ ^ B = (b)B.Lungime ∈ ℝ; (C U D) / S = A/B]

[(C U D) / S = A/B; C= A/B]

Program 2 (atribuire)

(C.Lungime, D, S) ← (0, A, B)

[(C U D) / S = A/B; C= A/B]

Program 3

(C.Lungime, D, S) ← (0, A, B)

[(C U D) / S = A/B; ((C U D) / S = A/B) ^ (D.Lungime = 0)]

Program 4 (iteratie)

(C.Lungime, D, S) ← (0, A, B)

while D.Lungime ≠ 0 do

[D.Lungime ≠ 0; (C U D) / S = A / B]

end

Program 5 (compunere)

(C.Lungime, D, S) ← (0, A, B)

while D.Lungime ≠ 0 do

[D.Lungime ≠ 0; index = 0]

[index = 0; (C U D) / S = A / B]

end

Program 6 (compunere)

(C.Lungime, D, S) ← (0, A, B)

while D.Lungime ≠ 0 do

[D.Lungime ≠ 0; index = 0]

[index = 0; (index = 0) ^ (index = B.Lungime) ^ (bindex ≠ a(D.Lungime -1))]

[(index = 0) ^ (index = B.Lungime) ^ (bindex ≠ a(D.Lungime -1); (C U D) / S = A / B]

end

Program 7 (atribuire)

(C.Lungime, D, S) ← (0, A, B)

while D.Lungime ≠ 0 do

index = 0

[index = 0; (index = 0) ^ (index = B.Lungime) ^ (bindex ≠ a(D.Lungime -1))]

[(index = 0) ^ (index = B.Lungime) ^ (bindex ≠ a(D.Lungime -1); (C U D) / S = A / B]

end

Program 8 (iteratie)

(C.Lungime, D, S) ← (0, A, B)

while D.Lungime ≠ 0 do

index = 0

while ¬((index = B.Lungime) ^ (bindex ≠ a(D.Lungime -1)) do

index = index + 1

end

[(index = 0) ^ (index = B.Lungime) ^ (bindex ≠ a(D.Lungime -1); (C U D) / S = A / B]

end

Program 9 (alternanta)

(C.Lungime, D, S) ← (0, A, B)

while D.Lungime ≠ 0 do

index = 0

while ¬((index = B.Lungime) ^ (bindex ≠ a(D.Lungime -1)) do

index = index + 1

end

if index = B.Lungime then

CC.Lungime ← a(D.Lungime – 1)

C.Lungime ← C.Lungime + 1

D.Lungime ← D.Lungime – 1

else

D.Lungime ← D.Lungime -1

end

end

Demonstrarea corectitudinii

Functia u = A.Lungime \* D.Lungime – index

Predicatul invariant: (C ∩ B = Ø) ^ (C ⊂ A)

## B.16 Determinați cel mai lung subșir de rădăcini ale unui polinom.

Completarea enunțului

Șirul de numere este format din numere reale iar polinomul este un șir de numere reale care conține coeficienții polinomului în ordine descrescătoare după grad. De exemplu polinomul 3x3 - x are șirul de coeficienții (3, 0, -1, 0).

Specificare

Subșir(șir, polinom, rezultat)

* Date de intrare:
  + șir: un șir de numere reale, deține un câmp denumit Lungime egal cu numărul total de elemente conținute.
  + polinom: un șir de numere reale, deține un câmp denumit Lungime egal cu numărul total de elemente conținute.
* Date de ieșire:
  + rezultat: un șir de numere reale, deține un câmp denumit Lungime egal cu numărul total de elemente conținute.

rezultat = (șiri+p, șiri+p+1, …, șiri + p +n), 0 ≤ p < șir.Lungime, 0 ≤ n ≤ șir.Lungime – i,

1 ≤ i ≤ șir.Lungime, n maxim,

când j = p + 1..p + n.

rezultat.Lungime = n.

Dezvoltare din specificații

Specificare

Φ: S = (s)S.Lungime ∈ ℝ; C = (c)C.Lungime ∈ ℝ

Ψ: R = SirDorit(S, C), unde SirDorit(S, C) = (s+p, s+p+1, …, s + p +n), 0 ≤ p < S.Lungime,

0 ≤ n ≤ S.Lungime – i, 1 ≤ i ≤ S.Lungime, n maxim,

când j = p + 1..p + n.

Program 0

[S = (s)S.Lungime ∈ ℝ; C = (c)C.Lungime ∈ ℝ; R = SirDorit(S, C)]

Program 1 (compunere)

[S = (s)S.Lungime ∈ ℝ; C = (c)C.Lungime ∈ ℝ; R = SirDorit((s)1...index, C)]

[R = SirDorit((s)1...index, C); R = SirDorit(S, C)]

Program 2 (atribuire)

(index, S.Lungime, T.Lungime) <- (0, 0, 0)

[R = SirDorit((s)1...index, C); R = SirDorit(S, C)]

Program 3

(index, S.Lungime, T.Lungime) <- (0, 0, 0)

[R = SirDorit((s)1...index, C); R = SirDorit((s)1..index, C) ^ (index = S.Lungime)]

Program 4 (iteratie)

(index, S.Lungime, T.Lungime) <- (0, 0, 0)

while index ≠ S.Lungime do

[index ≠ S.Lungime; R = SirDorit((s)1...index, C); R = SirDorit((s)1..index, C) ^ (index = S.Lungime)]

end

Program 5 (compunere)

(index, S.Lungime, T.Lungime) <- (0, 0, 0)

while index ≠ S.Lungime do

[index ≠ S.Lungime ^ R = SirDorit((s)1...index, C); suma = 0]

[suma = 0; R = SirDorit((s)1..index, C) ^ (index = S.Lungime)]

end

Program 6 (compunere)

(index, S.Lungime, T.Lungime) <- (0, 0, 0)

while index ≠ S.Lungime do

[index ≠ S.Lungime ^ R = SirDorit((s)1...index, C); suma = 0]

[suma = 0; coefIndex = 0]

[coefIndex = 0; R = SirDorit((s)1..index, C) ^ (index = S.Lungime)]

end

Program 7 (compunere)

(index, S.Lungime, T.Lungime) <- (0, 0, 0)

while index ≠ S.Lungime do

[index ≠ S.Lungime ^ R = SirDorit((s)1...index, C); suma = 0]

[suma = 0; coefIndex = 0]

[coefIndex = 0; (coefIndex =0) ^ (coefIndex = C.Lungime)]

[(coefIndex = 0) ^ (coefIndex = C.Lungime); R = SirDorit((s)1..index, C) ^ (index = S.Lungime)]

end

Program 8 (atribuire)

(index, S.Lungime, T.Lungime) <- (0, 0, 0)

while index ≠ S.Lungime do

suma <- 0

[suma = 0; coefIndex = 0]

[coefIndex = 0; (coefIndex =0) ^ (coefIndex = C.Lungime)]

[(coefIndex = 0) ^ (coefIndex = C.Lungime); R = SirDorit((s)1..index, C) ^ (index = S.Lungime)]

end

Program 9 (atribuire)

(index, S.Lungime, T.Lungime) <- (0, 0, 0)

while index ≠ S.Lungime do

suma <- 0

coefIndex <- 0

[coefIndex = 0; (coefIndex =0) ^ (coefIndex = C.Lungime)]

[(coefIndex = 0) ^ (coefIndex = C.Lungime); R = SirDorit((s)1..index, C) ^ (index = S.Lungime)]

end

Program 10 (iteratie)

(index, S.Lungime, T.Lungime) <- (0, 0, 0)

while index ≠ S.Lungime do

suma <- 0

coefIndex <- 0

while coefIndex = C.Lungime do

suma <- suma + ccoefIndex \* sindexC.Lungime – coefIndex – 1

end

[(coefIndex = 0) ^ (coefIndex = C.Lungime); R = SirDorit((s)1..index, C) ^ (index = S.Lungime)]

end

Program 11 (alternanta)

(index, S.Lungime, T.Lungime) <- (0, 0, 0)

while index ≠ S.Lungime do

suma <- 0

coefIndex <- 0

while coefIndex = C.Lungime do

suma <- suma + ccoefIndex \* sindexC.Lungime – coefIndex – 1

end

if (suma = 0 then

[(suma = 0) ^ (coefIndex = C.Lungime); R = SirDorit((s)1..index, C)]

else

[(suma ≠ 0) ^ (coefIndex = C.Lungime); R = SirDorit((s)1..index, C)]

end

end

Program 12 (alternanta)

(index, S.Lungime, T.Lungime) <- (0, 0, 0)

while index ≠ S.Lungime do

suma <- 0

coefIndex <- 0

while coefIndex = C.Lungime do

suma <- suma + ccoefIndex \* sindexC.Lungime – coefIndex – 1

end

if (suma = 0 then

TT.Lungime <- sindex

T.Lungime <- T.Lungime + 1

if T.Lungime > S.Lungime then

S <- T

end

index <- index + 1

else

T.Lungime = 0

Index = index + 1

end

end

Demonstrarea corectitudinii

Functia u = index

Predicatul invariant: S = SirDorit((s)i..index, C)