Rania Sasi Kirana

18221168

**LATIHAN SOAL 1**

function isSimetris (l: List) → boolean

{ Menghasilkan true jika List l simetrik. }

{ List disebut simetrik jika:

- elemen pertama = elemen terakhir,

- elemen kedua = elemen sebelum terakhir,

- dan seterusnya.

List kosong adalah List simetris }

KAMUS LOKAL

i,j: integer

simetris: boolean

ALGORITMA

{ inisiasi variabel }

i <- 0

j <- length(l)-1

simetris <- True

{ proses }

if (isEmpty(l)) then { Jika List kosong }

simetris <- True

else { List tidak kosong }

while (i<l.nEff and simetris) do

if (getElmt(l,i) ≠ getElmt(l,j)) then

simetris <- False

i <- i+1

j <- j-1

-> simetris

function plusTab (l1,l2: List) → List

{ Prekondisi: l1 dan l2 berukuran sama dan tidak kosong. }

{ Mengirimkan l1+l2, yaitu penjumlahan setiap elemen l1 dan l2

pada indeks yang sama (seperti penjumlahan vektor dalam matematika).}

KAMUS LOKAL

i: integer

l3: List

ALGORITMA

{ inisiasi variabel }

{ Membuat list baru untuk output }

CreateList(l3)

l3.nEff <- l1.nEff

{ proses }

i traversal [0..l1.nEff-1]

insertFirst(l3, getElmt(l1,i) + getElmt(l2,i))

-> l3

function countOccurence(l: List, x: ElType) → integer

{ Menghasilkan berapa banyak kemunculan elemen bernilai x di List l. }

{ Jika l kosong, menghasilkan 0. }

KAMUS LOKAL

i, count: integer

ALGORITMA

{ inisiasi variabel }

count <- 0

{ proses }

if isEmpty(l) then { jika list kosong }

count <- 0

else

i traversal[0..l.nEff-1]

if (getElmt(l,i) = x) then

count <- count + 1

-> count

function isEqual (l1,l2:List) → boolean

{ Mengirimkan true jika l1 setara dengan l2, yaitu jika ukuran

l1 sama dengan ukuran l2 dan semua elemen l1 dan l2 pada indeks

yang sama bernilai sama; L1 dan L2 tidak kosong. }

KAMUS LOKAL

i: integer

cek: boolean

ALGORITMA

{ inisiasi variabel }

i <- 0

cek <- True

{ proses }

{ ukuran l1 tidak sama dengan l2 }

if (l1.nEff ≠ l2.nEff) then

cek <- False

else

while (i<l1.nEff and cek) do

if(getElmt(l1,i) ≠ getElmt(l2,i)) then

cek <- False

-> cek

function indexOf (l:List, x:ElType) → IdxType

{ Mencari apakah ada elemen List l yang bernilai x. }

{ Jika ada, menghasilkan indeks i terkecil, di mana elemen l ke-i = x.

Jika tidak ada, mengirimkan indeks tak terdefinisi (idxUndef).

Jika list kosong, menghasilkan indeks tak terdefinisi (idxUndef). }

{ Memakai skema searching tanpa boolean. }

KAMUS LOKAL

i: integer

ALGORITMA

{ proses }

if isEmpty(l) then { Jika list kosong }

-> idxUndef

else { List tidak kosong }

i traversal [0..l.nEff-1]

if (getElmt(l,i)=x) then

-> i

-> idxUndef

procedure insertUnique (input/output l:List, input x:ElType)

{ Menambahkan x sebagai elemen terakhir list l,

pada list dengan elemen unik. }

{ I.S. List l boleh kosong, tetapi tidak penuh

dan semua elemennya bernilai unik, tidak terurut.

F.S. Menambahkan x sebagai elemen terakhir l

jika belum ada elemen yang bernilai x.

Jika sudah ada elemen list yang bernilai x

maka I.S. = F.S. dan dituliskan pesan

“nilai sudah ada”. }

{ Proses : Cek apakah X ada dengan sequential search

dengan sentinel, kemudian tambahkan jika belum ada. }

KAMUS LOKAL

i: integer

cek: boolean

ALGORITMA

{ inisiasi variabel }

i <- 0

cek <- False

{ proses }

while (i<l.nEff and not cek) do

if (getElmt(l,i) = x) then

cek <- True

i <- i+1

if not cek then

insertLast(l,x)

else

output("nilai sudah ada")

**LATIHAN SOAL 2**

{ representasi implisit }

constant kapasitas: integer = 100

constant idxUndef: integer = -1

{ Menambahkan MARK }

constant MARK: integer = -9999

type ElType: integer

{ Menghapus nEff (indeks efektif) }

type List: <ti: array [0..kapasitas-1] of ElType>

**LATIHAN SOAL 3**

{ Menggunakan asumsi array rata kiri }

{ implisit }

function getFirstIdx (l:List) -> integer

{ Prekondisi : l tidak kosong

Mengirimkan indeks fisik elemen pertama }

KAMUS LOKAL

ALGORITMA

-> 0

function getLastIdx (l: list) -> integer

{ Prekondisi : l tidak kosong

Mengirimkan indeks fisik elemen terakhir}

KAMUS LOKAL

i: integer

mark: constant integer = -9999

ALGORITMA

{ inisiasi variabel }

i <- 0

{ proses }

while (not mark) do

i <- i+1

-> i

{ eksplisit }

function getFirstIdx (l:List) -> integer

{ Prekondisi : l tidak kosong

Mengirimkan indeks fisik elemen pertama }

KAMUS LOKAL

ALGORITMA

-> 0

function getLastIdx (l: list) -> integer

{ Prekondisi : l tidak kosong

Mengirimkan indeks fisik elemen terakhir}

KAMUS LOKAL

ALGORITMA

-> l.nEff-1

{ Fungsi di soal 1 untuk representasi implisit }

{ Asumsi: boleh menggunakan fungsi getLastIdx }

function isSimetris (l: List) → boolean

{ Menghasilkan true jika List l simetrik. }

{ List disebut simetrik jika:

- elemen pertama = elemen terakhir,

- elemen kedua = elemen sebelum terakhir,

- dan seterusnya.

List kosong adalah List simetris }

KAMUS LOKAL

i,j: integer

simetris: boolean

ALGORITMA

{ inisiasi variabel }

i <- 0

j <- length(l)-1

simetris <- True

{ proses }

if (isEmpty(l)) then { Jika List kosong }

simetris <- True

else { List tidak kosong }

i traversal [0..(getLastIdx(l)+1)/2]

if (getElmt(l,i) ≠ getElmt(l,j)) then

simetris <- False

j <- j-1

-> simetris

function plusTab (l1,l2: List) → List

{ Prekondisi: l1 dan l2 berukuran sama dan tidak kosong. }

{ Mengirimkan l1+l2, yaitu penjumlahan setiap elemen l1 dan l2

pada indeks yang sama (seperti penjumlahan vektor dalam matematika).}

KAMUS LOKAL

i: integer

l3: array [0..getLastIdx(l1)-1] of integer

ALGORITMA

{ inisiasi variabel }

i <- 0

{ Membuat list baru untuk output }

CreateList(l3)

{ proses }

while (i<getLastIdx(l1)) do

insertFirst(l3, getElmt(l1,i) + getElmt(l2,i))

i <- i+1

-> l3

function countOccurence(l: List, x: ElType) → integer

{ Menghasilkan berapa banyak kemunculan elemen bernilai x di List l. }

{ Jika l kosong, menghasilkan 0. }

KAMUS LOKAL

i, count: integer

ALGORITMA

{ inisiasi variabel }

i <- 0

count <- 0

{ proses }

if isEmpty(l) then { jika list kosong }

count <- 0

else

while (i<getLastIdx(l)) do

if (getElmt(l,i) = x) then

count <- count + 1

i <- i+1

-> count

function isEqual (l1,l2:List) → boolean

{ Mengirimkan true jika l1 setara dengan l2, yaitu jika ukuran

l1 sama dengan ukuran l2 dan semua elemen l1 dan l2 pada indeks

yang sama bernilai sama; L1 dan L2 tidak kosong. }

KAMUS LOKAL

i: integer

cek: boolean

ALGORITMA

{ inisiasi variabel }

i <- 0

cek <- True

{ proses }

{ ukuran l1 tidak sama dengan l2 }

if (getLastIdx(l1) ≠ getLastIdx(l2)) then

cek <- False

else

while (i<getLastIdx(l1) and cek) do

if(getElmt(l1,i) ≠ getElmt(l2,i)) then

cek <- False

i <- i+1

-> cek

function indexOf (l:List, x:ElType) → IdxType

{ Mencari apakah ada elemen List l yang bernilai x. }

{ Jika ada, menghasilkan indeks i terkecil, di mana elemen l ke-i = x.

Jika tidak ada, mengirimkan indeks tak terdefinisi (idxUndef).

Jika list kosong, menghasilkan indeks tak terdefinisi (idxUndef). }

{ Memakai skema searching tanpa boolean. }

KAMUS LOKAL

i: integer

ALGORITMA

{ inisiasi variabel }

i <- 0

{ proses }

if isEmpty(l) then { Jika list kosong }

-> idxUndef

else { List tidak kosong }

while (i<getLastIdx(l)) do

if (getElmt(l,i)=x) then

-> i

i <- i+1

-> idxUndef

procedure insertUnique (input/output l:List, input x:ElType)

{ Menambahkan x sebagai elemen terakhir list l,

pada list dengan elemen unik. }

{ I.S. List l boleh kosong, tetapi tidak penuh

dan semua elemennya bernilai unik, tidak terurut.

F.S. Menambahkan x sebagai elemen terakhir l

jika belum ada elemen yang bernilai x.

Jika sudah ada elemen list yang bernilai x

maka I.S. = F.S. dan dituliskan pesan

“nilai sudah ada”. }

{ Proses : Cek apakah X ada dengan sequential search

dengan sentinel, kemudian tambahkan jika belum ada. }

KAMUS LOKAL

i: integer

cek: boolean

ALGORITMA

{ inisiasi variabel }

i <- 0

cek <- False

{ proses }

while (i<getLastIdx(l) and not cek) do

if (getElmt(l,i) = x) then

cek <- True

i <- i+1

if not cek then

insertLast(l,x)

else

output("nilai sudah ada")

**LATIHAN SOAL 4A: ADT ARRAY EKSPLISIT-STATIK**

procedure closestPair (input l: List; output p1,p2: ElType)

{ I.S.: l terdefinisi, mungkin kosong, p1 dan p2 sembarang. }

{ F.S.:

Jika l tidak kosong, p1 dan p2 berisi 2 elemen l pada posisi

berurutan yang memiliki selisih (selalu positif) terkecil.

Jika kedua elemen nilainya berbeda, maka p1 adalah elemen yang

bernilai lebih kecil.

Jika ada beberapa pasang elemen yang memiliki selisih terkecil,

maka diambil pasangan elemen yang muncul pertama kali.

Jika l kosong atau hanya terdiri atas 1 elemen, p1 dan p2

berisi elemen tidak terdefinisi yaitu -999 }

{ Contoh:

l.ti = [5,3,10,11,20,19]; maka p1=10 dan p2=11

l.ti = [-2,10,7,30,40,43,9]; maka p1=7 dan p2=10

l.ti = [-2,10,10,40,40]; maka p1=10 dan p2=10 }

KAMUS LOKAL

p1, p2: ElType

selisih, simpan, i: integer

ALGORITMA

if l.nEff <= 1 then { Jika list kosong atau berisi 1 elemen }

p1 <- -999

p2 <- -999

else { List memiliki elemen lebih dari 1 }

{ inisiasi variabel selisih, p1, dan p2 }

{ karena nilai selisih selalu positif sehingga dilakukan pengecekan nilai elemen mana yang lebih besar }

if l.contents[0] >= l.contents[1] then

selisih <- l.contents[0] - l.contents[1]

p1 <- l.contents[1]

p2 <- l.contents[0]

else

selisih <- l.contents[1] - l.contents[0]

p1 <- l.contents[1]

p2 <- l.contents[0]

{ menyimpan nilai selisih di variabel simpan }

simpan <- selisih

{ proses }

i traversal [1..l.nEff-1]

if l.contents[i] <= l.contents[i+1] then

selisih <- l.contents[i+1] - l.contents[i]

else

selisih <- l.contents[i] - l.contents[i+1]

{ jika selisih baru lebih kecil dari selisih sebelumnya yang tersimpan dalam variabel simpan }

if simpan > selisih then

simpan <- selisih

if l.contents[i] <= l.contents[i+1] then

p1 <- l.contents[i]

p2 <- l.contents[i+1]

else

p1 <- l.contents[i+1]

p2 <- l.contents[i]

**LATIHAN SOAL 4B: ADT ARRAY EKSPLISIT-STATIK**

function isFront (l1,l2 : List) → boolean

{ Mengembalikan true jika elemen-elemen l1 merupakan bagian awal dari l2 }

{ Contoh: }

{ isFront ([2,3,4], [2,3,4,5,6]) = true }

{ isFront ([2,3,4], [3,4,5,6]) = false }

{ isFront ([], [2,3,4,5,6]) = true }

{ isFront ([2,3,4], [2,3]) = false }

{ isFront ([2,3,4], []) = false }

KAMUS LOKAL

i: integer

cek: boolean

ALGORITMA

{ inisiasi variabel }

cek <- True

{ proses }

if l1.nEff = 0 then

cek <- True

else

{ yang dicek adalah seluruh elemen list l1 sehingga menggunakan nEff l1}

i traversal [0..l1.nEff-1]

if cek then

if l1.contents[i] ≠ l2.contents[i] then

cek <- False

-> cek