Latihan Soal
(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: integer
    simetris: boolean
```

```
ALGORITMA
    simestris ← true
    i ← 0
    while ((i < (l.nEff // 2)) and (simetris)) do
        if (l.contents[i] ≠ l.contents[l.nEff – i – 1]) then
            simestris ← false
        else { l.contents[i] = l.contents[l.nEff – i – 1] } then
            i = i + 1
    → simestris
```

```
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    { l3 adalah list kosong dengan nEff = 0 }
```

```
ALGORITMA
    i traversal [0..(l1.nEff – 1)]
        l3.contents[i] ← l1.contents[i] + l2.contents[i]
        l3.nEff ← l3.nEff + 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: integer
    count: integer
```

```
ALGORITMA
    i ← 0
    count ← 0
    while (i < l.nEff) do
        if (l.contents[i] = x) then
            count ← count + 1
        else { l.contents[i] ≠ x } then
            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
    equal: boolean
```

```
ALGORITMA
    i ← 0
    equal ← true
    if (l1.nEff ≠ l2.nEff) then
        equal ← false
    while ((equal) and (i < l1.nEff)) do
        if (l1.contents[i] ≠ l2.contents[i]) then
            equal ← false
        else { l1.contents[i] = l2.contents[i] } then
            i ← i + 1
    → equal
```

```
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
    i ← 0
    if (l.nEff = 0) then
        → idxUndef
    { biasanya dalam kasus skema searching tanpa boolean tidak boleh ada kasus
      kosong. Dalam kasus ini, kasus kosong di eliminasi terlebih dahulu }
    while ((i < l.nEff) and (l.contents[i] ≠ x)) do
        i ← i + 1
    if (l.contents[i] = x) then
        → i
    else { l.contents[i] ≠ x } then
        → 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
```

```
ALGORITMA
    l.contents[l.nEff] ← x  { pasang penanda sentinel }
    i ← 0
    while (l.contents[i] ≠ x) do
        i ← i + 1
    if (i < l.nEff) then
        { berhenti di index sebelum index sentinel diletakan }
        output("nilai sudah ada")
    else { i = l.nEff } then
        { berhenti di index tempat sentinel diletakan }
        l.nEff ← l.nEff + 1
```

Latihan Soal
(Soal 2)

[Tulisan di highlight]: Perbedaan antara metode implisit dan eksplisit.

```
{ Versi I : Dengan banyaknya elemen secara eksplisit, array statik }
constant kapasitas: integer = 100
constant idxUndef: integer = -1 { indeks tak terdefinisi}
{ Definisi elemen dan koleksi objek }
type ElType: integer { type elemen list }
type List: < ti: array [0..kapasitas-1] of ElType,
                { memori tempat penyimpan elemen (container) }
             nEff: integer ≥ 0 { banyaknya elemen efektif } >
```

```
{ Versi II : Dengan banyaknya elemen secara implisit, array statik }
constant kapasitas: integer = 100
constant idxUndef: integer = -1 { indeks tak terdefinisi}
constant mark: integer = -9999 { mark penanda elemen kosong }
{ Definisi elemen dan koleksi objek }
type ElType: integer { type elemen list }
type List: < ti: array [0..kapasitas-1] of ElType,
                { memori tempat penyimpan elemen (container) }
                { tidak diperlukan nEff dalam kasus implisit } >
```

Latihan Soal
(Soal 3a)

{ Versi I : Dengan banyaknya elemen secara eksplisit, array statik }

```
function getFirstIdx (l: List) → integer
{ Mengembalikan indeks pertama dari list l. Jika list kosong, mengembalikan
  idxUndef. }
{ Asumsi bahwa tipe listnya adalah rata kiri }
```

KAMUS LOKAL

```
    { Tidak ada kamus lokal }
```

ALGORITMA

```
    if (l.nEff = 0) then
        → idxUndef
    else { l.nEff ≠ 0 } then
        → 0
```

{ Versi I : Dengan banyaknya elemen secara eksplisit, array statik }

```
function getLastIdx (l: List) → integer
{ Mengembalikan indeks terakhir dari list l. Jika list kosong, mengembalikan
  idxUndef. }
{ Asumsi bahwa tipe listnya adalah rata kiri }
```

KAMUS LOKAL

```
    { Tidak ada kamus lokal }
```

ALGORITMA

```
    if (l.nEff = 0) then
        → idxUndef
    else { l.nEff ≠ 0 } then
        → l.nEff – 1
```

{ Versi II : Dengan banyaknya elemen secara implisit, array statik }

```
function getFirstIdx (l: List) → integer
{ Mengembalikan indeks pertama dari list l. Jika list kosong, mengembalikan
  idxUndef. }
{ Asumsi bahwa tipe listnya adalah rata kiri }
```

KAMUS LOKAL

```
    { Tidak ada kamus lokal }
```

ALGORITMA

```
    if (l.ti[0] = mark) then
        → idxUndef
    else { l.ti[0] ≠ mark } then
        → 0
```

```
{ Versi II : Dengan banyaknya elemen secara implisit, array statik }
```

```
function getLastIdx (l: List) → integer
{ Mengembalikan indeks terakhir dari list l. Jika list kosong, mengembalikan
  idxUndef. }
{ Asumsi bahwa tipe listnya adalah rata kiri }
```

```
KAMUS LOKAL
    { Tidak ada kamus lokal }
```

```
ALGORITMA
    i ← 0
    while ((i < N) and (l.ti[i] ≠ mark) do
        i ← i + 1
    if (l.ti[i] = mark) then
        → idxUndef
    else { l.ti[i] ≠ mark } then
        → i
```

Latihan Soal
(Soal 3b)

[Tulisan di highlight]: Perbedaan antara metode implisit dan eksplisit.

```
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: integer
    length: integer  { dibutuhkan, karena tidak adanya nEff }
    simetris: boolean
```

```
ALGORITMA
    simestris ← true
    length ← 0
    while (l.contents[length] ≠ mark) do
        length ← length + 1
    i ← 0
    while ((i < length // 2) and (simetris)) do
        if (l.contents[i] ≠ l.contents[length - i - 1]) then
            simestris ← false
        else { l.contents[i] = l.contents[length - i - 1] } then
            i = i + 1
    → simestris
```

```
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    { l3 adalah list kosong dengan nEff = 0 }
```

```
ALGORITMA
    i ← 0
    { l1 sebagai perwakilan kondisional, karena l1 dan l1 berukuran sama }
    while (l1.contents[i] ≠ mark) do
        l3.contents[i] ← l1.contents[i] + l2.contents[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: integer
    count: integer
```

```
ALGORITMA
    i ← 0
    count ← 0
    while (l.contents[i] ≠ mark) do
        if (l.contents[i] = x) then
            count ← count + 1
        else { l.contents[i] ≠ x } then
            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
    length1: integer  { dibutuhkan, karena tidak adanya nEff }
    length2: integer  { dibutuhkan, karena tidak adanya nEff }
    equal: boolean
```

```
ALGORITMA
    equal ← true
    length1 ← 0
    while (l1.contents[length1] ≠ mark) do
        length1 ← length1 + 1
    length2 ← 0
    while (l2.contents[length1] ≠ mark) do
        length2 ← length2 + 1
    if (length1 ≠ length2) then
        equal ← false
    i ← 0
    while ((equal) and (i < length1)) do
        if (l1.contents[i] ≠ l2.contents[i]) then
            equal ← false
        else { l1.contents[i] = l2.contents[i] } then
            i ← i + 1
    → equal
```

```
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
    i ← 0
    if (l.contents[0] = mark) then
        → idxUndef
    { biasanya dalam kasus skema searching tanpa boolean tidak boleh ada kasus
      kosong. Dalam kasus ini, kasus kosong di eliminasi terlebih dahulu }
    while ((l.contents[i] ≠ mark) and (l.contents[i] ≠ x)) do
        i ← i + 1
    if (l.contents[i] = x) then
        → i
    else { l.contents[i] ≠ x } then
        → 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
    length: integer  { dibutuhkan, karena tidak adanya nEff }
```

```
ALGORITMA
    length ← 0
    while (l.contents[length] ≠ mark) do
        length ← length + 1
    l.contents[length] ← x  { pasang penanda sentinel }
    i ← 0
    while (l.contents[i] ≠ x) do
        i ← i + 1
    if (i < length) then
        { berhenti di index sebelum index sentinel diletakan }
        l.contents[length] ← mark
        output("nilai sudah ada")
    { tidak ada else, karena saat pencarian berhenti di index sentinel, artinya
      nomor tersebut memang harus ditambahkan (sudah di awal) dan tidak perlu
       operasi tambahan seperti metode eksplisit }
```

Latihan Soal
(Soal 4a)

```
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
    i: integer
    diff: integer
    indeks: integer
```

```
ALGORITMA
    i ← 0
    diff ← 0
    indeks ← -9999
    { apabila list kosong atau list dengan 1 element, maka while tidak akan
      berjalan }
    { ketika nEff 0 atau 1, maka tidak memenuhi kondisional iterasi (iterasi
      tidak berjalan sama sekali) }
    while (i < (l.nEff – 1)) do
        { sementara diasumsikan, antara index 0 dan index 1 adalah selisih
          terkecil }
        if ((i = 0) or (abs(l.ti[i] – l.ti[i + 1]) < diff)) then
            diff ← abs(l.ti[i] – l.ti[i + 1])
            indeks ← i
        i ← i + 1
    if (index = -9999) then
        p1 ← -999
        p2 ← -999
    else if (l.ti[indeks] < l.ti[indeks + 1]) then
        p1 ← l.ti[indeks]
        p2 ← l.ti[indeks + 1]
    else { l.ti[indeks] > l.ti[indeks + 1] } then
        p1 ← l.ti[indeks + 1]
        p2 ← l.ti[indeks]
```

Latihan Soal
(Soal 4b)

```
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
    frontStatus: boolean
```

```
ALGORITMA
    i ← 0
    frontStatus ← true
    if (l1.nEff > l2.nEff) then
        frontStatus ← false
    while ((i < l1.nEff - 1) and (frontStatus)) do
        if (l1.ti[i] ≠ l2.ti[i]) then
            frontStatus ← false
        i ← i + 1
    → frontStatus
```