# LIST

**Definisi**

LIST adalah **sekumpulan** elemen list yang bertype sama. Jadi list adalah koleksi objek. Elemen list mempunyai keterurutan tertentu (elemen-ke…, ada pengertian suksesor, predesesor), nilai satu elemen boleh muncul lebih dari satu kali.

**Definisi rekursif**

**List** adalah sekumpulan elemen list yang bertype sama :

- Basis 0 : **list kosong** adalah sebuah **list**
- Rekurens :
  **list** terdiri dari sebuah elemen dan sublist (yang juga merupakan **list**)

List sering disebut dengan istilah lain :

- *sequence*
- *series*

Dalam kehidupan se-hari-hari, list merepresentasi :

- teks (*list of* kata)
- kata (*list of* huruf)
- *sequential file (list of record)*
- table (*list of* elemen tabel, misalnya untuk tabel integer: list of integer)
- list of atom simbolik (dalam LISP)

Dalam dunia pemrograman, list merupakan suatu type data yang banyak dipakai, untuk merepresentasi objek yang diprogram misalnya :

- Antarmuka berbasis grafis (GUI): *list of Windows, list of menu item, list of button, list of icon*
- Program editor gambar : *list of Figures*
- Program pengelola sarana presentasi : *list of Slides*
- Program pengelola *spreadsheet* : *list of Worksheet, list of cell*
- Dalam sistem operasi : *list of terminals, list of job*

Jika type dari elemen punya keterurutan **nilai** elemen(membesar, mengecil), dikatakan bahwa list terurut membesar atau mengecil. Bedakan keterurutan nilai ini dengan keterurutan elemen sebagai suksesor dan predesesor.
List L adalah **terurut menaik** jika dua elemen berturutan dari L yaitu e1 dan e2 (dengan e2 adalah suksesor dari e1), selalu mempunyai sifat bahwa nilai dari e1 $\leq$ nilai dari e2.

Dari konsep type yang pernah dipelajari, elemen list dapat berupa :

- elemen yang mempunyai type sederhana (integer, karakter,...)
- elemen yang mempunyai type komposisi, mulai dari yang sederhana sampai yang kompleks
- list (rekursif !)

List kosong adalah list yang tidak mempunyai elemen, dalam notasi fungsional dituliskan sebagai [ ] .

Dua buah list disebut sama jika semua elemen list sama urutan dan nilainya.

Jika merupakan type dasar yang disediakan oleh bahasa pemrograman, maka **konstruktor** dan **selektor** list menjadi fungsi dasar yang siap dipakai seperti halnya dengan operator dasar aritmatika, relasional dan boolean.
Beberapa bahasa juga menyediakan predikat terdefinisi untuk menentukan apakah suatu list kosong atau tidak

Pada sub bab berikutnya, akan dibahas :
- List dengan elemen sederhana
- List dengan elemen karakter (teks)
- List dengan elemen bilangan integer
- List dengan elemen type bentukan (misalnya list of Point)
- List dengan elemen list (list of list)

Pembahasan hanya akan diberikan dalam bentuk definisi, kemudian relaisasi dari beberapa primitif yang penting.

## List Dengan Elemen Sederhana

**Definisi rekursif type List L dengan elemen sederhana (type dasar, type komposisi ):**

- Basis 0: list kosong adalah sebuah list
- Rekurens : list dapat dibentuk dengan menambahkan sebuah elemen pada list (konstruktor), atau terdiri dari sebuah elemen dan sisanya adalah list (selektor)

Penambahan/pengambilan elemen dapat dilakukan pada "awal" list, atau pada "akhir" list. Kedua cara penambahan/pengambilan ini melahirkan konstruktor dan selektor sebagai berikut yang ditulis dalam definisi list sebagai berikut. Konsep konstruktor/selektor ini akan berlaku untuk semua list berelemen apapun.
Seperti halnya type bentukan, Konstruktor dan selektor list pada notasi fungsional hanya dibuat definisi dan spesifikasinya. Realisasinya akan diserahkan kepada bahasa pemrograman (pada bagian kedua). Bahasa fungsional biasanya menyediakan list sebagai type "primitif" dengan konstruktor/selektornya. Bahkan dalam bahasa LISP, *list* bahkan *list of list*, merupakan representasi paling mendasar dari semua repesentasi type lain yang mewakili koleksi.

**TYPE LIST**

**DEFINISI DAN SPESIFIKASI TYPE**

*{Konstruktor menambahkan elemen di awal, notasi prefix }*

**type** List : [ ], atau [e o List]

*{Konstruktor menambahkan elemen di akhir, notasi postfix }*

**type** List : [ ] atau [List • e]

*{Definisi List : sesuai dengan definisi rekursif di atas }*

**DEFINISI DAN SPESIFIKASI KONSTRUKTOR**

**Konso** : elemen, List → List

*{Konso(e,L): menghasilkan sebuah list dari e dan L, dengan e sebagai elemen pertama e : e o L →L'}*

**Kons•** : List, elemen → List

*{ Kons(L,e): menghasilkan sebuah list dari L dan e, dengan e sebagai elemen terakhir list : L • e →L'}*

**DEFINISI DAN SPESIFIKASI SELEKTOR O**

**FirstElmt**: List tidak kosong → elemen

*{FirstElmt(L) Menghasilkan elemen pertama list L }*

**Tail** : List tidak kosong → List

*{Tail(L) : Menghasilkan list tanpa elemen pertama list L, mungkin kosong }*

**LastElmt** : List tidak kosong → elemen

*{LastElmt(L) : Menghasilkan elemen terakhir list L }*

**Head** : List tidak kosong → List

*{Head(L) : Menghasilkan list tanpa elemen terakhir list L, mungkin kosong}*

**DEFINISI DAN SPESIFIKASI PREDIKAT DASAR**

**(UNTUK BASIS ANALISA REKURENS}**

*{Basis 0 }*

**IsEmpty** : List → boolean

*{IsEmpty(L) benar jika list kosong }*

*{Basis 1 }*

**IsOneElmt**: List → boolean

*{IsOneElmt (X,L) adalah benar jika list L hanya mempunyai satu elemen }*

**DEFINISI DAN SPESIFIKASI PREDIKAT RELASIONAL**

**IsEqual** : 2 List → boloean

*{ IsEqual (L1,L2) benar jika semua elemen list L1 sama dengan L2: sama urutan dan sama nilainya }*

**BEBERAPA DEFINISI DAN SPESIFIKASI FUNGSI LAIN**

**NbElmt** : List → integer

*{NbElmt(L) : Menghasilkan banyaknya elemen list, nol jika kosong }*

**ElmtkeN** : integer ≥ 0 , List → elemen

*{ElmtkeN (N, L) : Mengirimkan elemen list yang ke N, N ≤ NbELmt(L) dan N>=0}*

**Copy** : List → List

*{ Copy(L) : Menghasilkan list yang identik dengan list asal}*

**Inverse** : List → List

*{ Inverse(L) : Menghasilkan list L yang dibalik, yaitu yang urutan elemennya adalah kebalikan dari list asal}*

**Konkat** : 2 List → List

*{Konkat(L1,L2) : Menghasilkan konkatenasi 2 buah list, dengan list L2 "sesudah" list L1}*

**BEBERAPA DEFINISI DAN SPESIFIKASI PREDIKAT**

**IsMember**: elemen, List → boolean

*{Ismember (X,L) adalah benar jika X adalah elemen list L }*

**IsFirst**: elemen, List (tidak kosong ) → boolean

*{IsFirst (X,L) adalah benar jika X adalah elemen pertama list L }*

**TYPE LIST (lanjutan)**

**IsLast** : elemen, List (tidak kosong ) → boolean

*{IsLast (X,L) adalah benar jika X adalah elemen terakhir list L }*

**IsNbElmtN (N,L)** : integer ≥ 0 dan ≤ NBElmt(L), List → integer

*{IsNbElmtN (N, L) true jika banyaknya elemen list sama dengan N }*

**IsInverse** : List , List → boolean

*{IsInverse(L1,L2) true jika L2 adalah list dengan urutan elemen terbalik dibandingkan L1}*

**IsXElmtkeN** : integer ≥ 0 dan ≤ NBElmt(L), elemen, List → boolean

*{IsXElmtkeN (N, X,L) adalah benar jika X adalah elemen list yang ke N }*

Beberapa catatan:

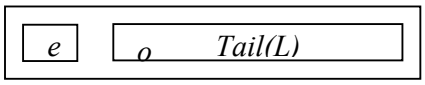
- Perhatikanlah definisi beberapa fungsi yang "mirip", dalam dua bentuk yaitu sebagai fungsi atau sebagai predikat. Misalnya :
  - LastElmt(L) dan IsLast(X,L)
  - Inverse(L) dan IsInverse(L1,L2)
  - ElmtKeN(N, L) dan IsXElmtKeN(N,X,L)

  Realisasi sebagai fungsi dan sebagai predikat tersebut dalam konteks fungsional murni sangat berlebihan, namun realisasi semacam ini dibutuhkan ketika bahasa hanya mampu menangani predikat, seperti pada program lojik. Bagian ini akan merupakan salah satu bagian yang akan dirujuk dalam kuliah pemrograman lojik.

Berikut ini adalah realisasi dari beberapa fungsi untuk list tersebut, penulisan solusi dikelompokkan menurut klasifikasi rekurens. Untuk kemudahan pembacaan, setiap fungsi yang pernah dituliskan definisi dan realisasi nya akan diulang penulisannya, namun konstruktor, selektor dan predikat dasar untuk menentukan basis tidak ditulsikan lagi.
Analisa rekurens akan dituliskan melalui ilustrasi bentuk rekursif list sebagai berikut :
Basis : List kosong atau list 1 elemen
Rekurens : sebuah list terdiri dari elemen dan sisanya adalah list
Visualisasi dari list adalah sebagai "segi empat", dan segiempat yang menggambarkan list tidak digambarkan ulang pada setiap ilustrasi

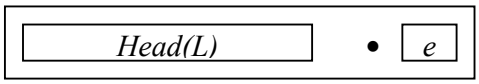
Dengan Selektor Konso, ilustrasinya adalah

*Rekurens:*




Dengan Selektor Kons• ilustrasinya adalah

*Rekurens:*



**Contoh1 : Banyaknya elemen sebuah list**

**Persoalan :** Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menghasilkan banyaknya elemen sebuah list. Contoh aplikasi dan hasilnya:

NbElmt ([ ]) = 0 ; NBElmt ([ a, b, c] = 3

**BANYAKNYA ELEMEN** **NbElmt(L)**

**DEFINISI DAN SPESIFIKASI**

**NbElmt** : List → integer

*{NbElmt(L) : Menghasilkan banyaknya elemen list, nol jika kosong }*

**DEFINISI DAN SPESIFIKASI**

**NElmt (L)** : List → integer

*{NbElmt (L) menghasilkan banyaknya elemen list L }*

**REALISASI: DENGAN KONSO**

*{ Basis 0: List kosong: tidak mengandung elemen, nbElmt ([]) = 0*

*Rekurens:*

*e* o *Tail(L)*

*1 + NbElmt ( Tail(L))*

```
NbElmt(L) :
        if IsEmpty(L) then {Basis 0}
              0
        else {Rekurens}
             1 + NbElmt(Tail (L))
```

**Contoh 2: keanggotaan elemen**

**Persoalan :** Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah predikat yang memeriksa apakah sebuah elemen x merupakan anggota dari list. Contoh aplikasi dan hasilnya:

IsMember (x, [ ]) = false; IsMember (x, [a, b, c] ) = false

IsMember (b, [a, b, c] ) = true

**KEANGGOTAAN** **IsMember(x,L)**

**DEFINISI DAN SPESIFIKASI**

**IsMember (x,L)** : elemen, List → boolean

*{ IsMember (x,L) true jika x adalah elemen dari list L }*

**REALISASI VERSI-2 : DENGAN KONSO**

*{ Basis 0 : List kosong: tidak mengandung apapun, → false*

*Rekurens:*

*e* o *Tail(L)*

*? x*

*Kasus: e=x → true*

*e ≠ x → x adalah anggota Tail(L) }*

```
IsMember(x,L) :
        if IsEmpty(x) then {Basis 0}
             false
        else {Rekurens : analisa kasus}
            if FirstElmt(L)=x then true
```

```
                else IsMember(x,Tail (L))
IsMember(x,L) :
          if IsEmpty(x) then {Basis 0}
               false
          else {Rekurens: ekspresi boolean }
            FirstElmt(L)=x or else IsMember(x,Tail (L))
```

**REALISASI VERSI-2 : DENGAN KONS•**

*{ Basis 0: list kosong tidak mengandung apapun, → false*

*Rekurens:* *Head(L)* • *e*

*x*

*e=x → true*

*e≠x → elemen Head(L) }*

```
IsMember (x,L) :
  if IsEmpty(x) then {Basis 0}
                     false
  else {Rekurens: analisa kasus }
     if LastElmt(L)=x then true
      else IsMember (x,Head (L))
```

**Contoh 3 : menyalin sebuah list**

**Persoalan :**

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menghasilkan salinan (copy) dari sebuah list, yaitu semua elemennya sama**.** Contoh aplikasi dan hasilnya adalah :

Copy ( [ ]) = [ ]; Copy ([a, b, c] ) = [a, b, c]

**MENYALIN LIST** **Copy(L)**

**DEFINISI DAN SPESIFIKASI**

**Copy** : List → List

*{Copy (L) menghasilkan salinan list L , artinya list lain yang identik dengan L}*

**REALISASI: DENGAN KONSO**

*{ Basis 0 : list kosong: hasilnya list kosong*

*Rekurens:* *e* *o* *Tail(L)*

*e o Copy( Tail(L)) }*

```
Copy(L) :
       if IsEmpty(L) then {Basis 0}
              []
      else {Rekurens}
            Konso(FirstElmt(L), Copy(Tail (L))
```

**Contoh 4: Membalik sebuah list**

**Persoalan :** Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menerima sebuah list, dan menghasiilkan list yang urutan elemennya terbalik dibandingkan urutan elemen pada list masukan. Contoh aplikasi dan hasilnya:

Inverse ( [ ]) = [ ] ; Inverse ([a, b, c] ) = [c, b, a]

| **MEMBALIK LIST** | **Inverse(L)** |
|---|---|

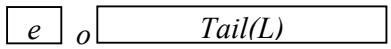
**DEFINISI DAN SPESIFIKASI**

**Inverse (L)** : List → List

*{Inverse (L) menghasilkan salinan list L dengan urutan elemen terbalik}*

**REALISASI: DENGAN KONSO**

*{ Basis 0: list kosong: hasilnya list kosong*
*Rekurens:*




*Hasil pembalikan adalah Tail(L) • e }*

```
Inverse(L) :
        if IsEmpty(L) then {Basis 0}
            []
        else {Rekurens}
            Kons•( Inverse(Tail (L), FirstElmt(L))
```

**Contoh 5: Kesamaan dua buah list**

**Persoalan :** Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah predikat yang memeriksa apakahdua buah elemen list sama. Dua buah list sama jika semua elemennya sama, dan urutan kemunculannya sama**..** Contoh aplikasi dan hasilnya:

IsEqual ([], [ ]) = true ; IsEqual ([], [a ]) = false;

IsEqual([a,b,c], [a, b, c] ) = true

| **KESAMAAN** | **IsEqual (L1,L2)** |
|---|---|

**DEFINISI DAN SPESIFIKASI**

**IsEqual** : 2 List → boolean

*{ IsEqual (L1,L2) true jika dan hanya jika kedua list sama, yaitu setiap elemen dan urutan kemunculannya sama}*
*{ Basis 0: IsEqual ([],[]) = true*
*IsEqual ([], e o L) = false*
*IsEqual (e o K, []) false }*
*{ Rekurens : terhadap kedua list*
*IsEqual (e1 o K1, e2 o K2) = (e1 = e2) dan K1 = K2 }*

**REALISASI VERSI-1**

```
IsEqual (L1,L2):  {jika hanya salah satu yang kosong, maka false}
   depend on L1, L2
      IsEmpty(L1) and IsEmpty(L1) : true  {Basis }
      IsEmpty(L1) and not IsEmpty(L1) : false   {Basis }
      not IsEmpty(L1) and IsEmpty(L1) : false   {Basis}
      not IsEmpty(L1) and not IsEmpty(L1) : {Rekurens }
                    ((FirstElmt(L1) = FirstElmt(L2)) and then
                                IsEqual (Tail(L1), Tail(L2))
```

**REALISASI VERSI-2 (CEK KEBENARANNYA!!)**

```
IsEqual (L1,L2):
   (IsEmpty(L1) and IsEmpty(L2)) Or else
   ((FirstElmt(L1) = FirstElmt(L2)) and then
                              IsEqual (Tail(L1), Tail(L2))
```

**REALISASI VERSI-3(berdasarkan banyaknya elemen)**

```
{Didefinisikan fungsi antara: CekEqual (L1,L2) yang akan mencek kesamaan
dua buah list yang panjangnya sama}
IsEqual (L1,L2):
   if (NBELmt(L1) ≠  NBElmt(L2) then
         false
    else
        CekEqual(L1,L2)
CekEqual (L1,L2): FirstElmt(L1)= FirstElmt(L2) and then
                     CekEqual(Tail(L1,Tail(L2)
```

**Contoh 6: Elemen ke N sebuah list**

**Persoalan :** Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menghasilkan elemen ke N dari sebuah list. N lebih besar dari Nol, dan list tidak boleh kosong dan N ≤ banyaknya elemen list. **.** Contoh aplikasi dan hasilnya:

ElmtKeN (0, [] ) = tidak terdefinisi karena list kosong tidak dapat menghasilkan elemen; maka analisa rekurens harus berbasis satu dan list tidak kosong

ElmtKeN (1, [a,b,c] ) = a;
Rekurens dilakukan terhadap N, dan juga terhadap list:

**ELEMEN KE N** **ElmtKeN(N,L)**

**DEFINISI DAN SPESIFIKASI**

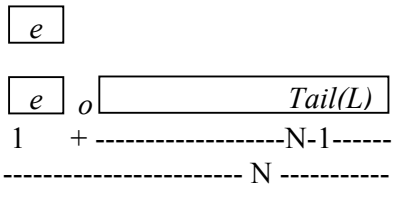
**ElmtKeN (N,L)** : integer ≥ ,List tidak kosong→ elemen

*{ElmtKeN (L) menghasilkan elemen ke-N list L, N ≥ 0, dan N ≤ banyaknya elemen list. }*

**REALISASI: DENGAN KONSO**

*{ Basis 1 : List dengan satu elemen , dan N=1 : elemen pertama list*

*Rekurens:*




*Kasus : N=1 maka e*
*N>1 : bukan e, tetapi ElmtKeN (N-1, Tail(L))*
*}*

```
ElmtKeN(N,L) :
       if N=1 {Basis 1}then
              FirstElmt(L)
       else {Rekurens}
            ElmtKeN(prec(N),Tail (L))
```

**Contoh 7: Konkatenasi dua buah list**

**Persoalan :**

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menerima dua buah list, dan menghasilkan konkatenasi dari kedua list tersebut. Contoh aplikasi dan hasilnya: Concat ([ ], [ ]) = [ ]; Concat ([a], [b,c]) = [a,b,c]

| **KONKATENASI** | **Concat(L1,L2)** |
|---|---|

**DEFINISI DAN SPESIFIKASI**

**Concat (L1,L2)** : List → List

*{Concat (L1,L2)menghasilkan konkatenasi list L1 dan L2}*

**REALISASI : REKURENS TERHADAP L1**

*{ Basis 0: L1 [] : L2*
*Rekurens:*

*e1* *o* *Tail(L1)*

*L2*

*List Hasil: e1 o Hasil konkatenasi dari Tail(L1) dengan L2*

```
Konkat(L1,L2) :
if IsEmpty(L1) then {Basis 0}
      L2
else {Rekurens}
       Konso(FirstElmt(L1), Konkat(Tail(L1),L2) )
```

**REALISASI : REKURENS TERHADAP L2**

*{ Basis 0: L2 [ ] : L1*
*Rekurens:*

*e2* *o* *Tail(L2)*

*L1*

*List Hasil: e1 o Hasil konkatenasi dari Tail(L1) dengan L2*

```
Konkat(L1,L2) :
if IsEmpty(L2) then {Basis 0}
      L1
else {Rekurens}
     Konkat (Kons•(L1, FirstElmt(L2)) , Tail(L2) )
```

**Contoh 8: Apakah banyaknya elemen sebuah list adalah N**

**Persoalan :** Tuliskanlah definisi, spesifikasi dan relaisasi sebuah predikat yang memeriksa apakah banyaknya elemen sebuah list = N, dengan N>=0. Contoh aplikasi dan hasilnya:

IsNBElmtN (0,[]) = true ; IsNBElmtN (3,[a,b,c]) = true;

IsNBElmtN (0,[a]) = false

Rekurens dilakukan terhadap N.

**BANYAKNYA ELEMEN** **IsNbElmtN(L)**

**DEFINISI DAN SPESIFIKASI**

**IsNbElmtN (L,N)** : List, integer >=0 → integer

*{IsNbElmtN (L,N) true jika banyaknya elemen list sama dengan N }*

**REALISASI: DENGAN KONSO**

*{ Basis 0: List kosong: false*

*Rekurens:* [ *e* ] *o* [ *Tail(L)* ]

*# elemen: 1 (N-1) }*

```
IsNbElmtN(L,N) :
if N=0 then {Basis 0}
      if IsEmpty(L) then  true else false
else {Rekurens }
      IsNbElmtN(Tail (L), Prec(N))
```

**REALISASI VERSI-2**

```
{ Realisasi ini memanfaatkan fungsi NbElmt(L) yang sudah didefinisikan }
IsNbElmtN(L,N) :
            NbElmt(L)=N
```

**Contoh 9: Apakah X adalah Elemen ke N sebuah list**

**Persoalan :** Tuliskanlah sebuah predikat yang memeriksa apakah X adalah elemen ke N dari sebuah list. N lebih besar dari Nol, dan list tidak boleh kosong. N positif, dan bernilai 1 s/d banyaknya elemen list

**APAKAH X ELEMEN KE N** **IsXElmtKeN(X,N,L)**

**DEFINISI DAN SPESIFIKASI**

**IsXElmtKeN (N,L)** :elemen, integer ≥ 0, List (tidak kosong) → boolean

*{IsXElmtKeN (L) true jika X adalah elemen ke-N list L, N ≥ 0, dan N ≤ banyaknya elemen list false jika tidak}*

**REALISASI: DENGAN KONSO**

*{ Basis 0: List dengan satu elemen , dan N=1dan e=X : true*

[ *e* ]

*Rekurens:*

[ *e* ] *o* [ *Tail(L)* ]

e =X ----------N-1-------

----------N ----------

*IsXElmtKeN (X,N-1, Tail(L))*

```
IsXElmtKeN(X,N,L) :
       if IsMember (X,L) then {Analisa kasus }
            if N=1 and FirstElmt(L)=X then {Basis 0}
                  true
             else {Rekurens}
                 false or IsXElmtKeN(X,prec(N),Tail (L))
       else {Bukan member, pasti } false
```

**REALISASI:**

```
{ Realisasi ini memanfaatkan fungsi ElmtKeN(L) yang sudah didefinisikan
}
IsXElmtKeN(X,L,N) :
            ElmtKeN(N,L)=X
```

**Contoh 10: Apakah inverse**

**Persoalan :**

Tuliskanlah definisi, spesifikasi dan realisasi dari dari sebuah predikat yang memeriksa apakah sebuah list adalah inverse dari list lain

| **APAKAH INVERSE** | **IsInverse(L1,L2)** |
|---|---|

**DEFINISI DAN SPESIFIKASI**

**IsInverse (L1,L2)** : 2 List → boolean

*{IsInverse (L1,L2) true jika L2 adalah list dengan urutan elemn terbalik dibandingkan L1, dengan perkataan lain adalah hasil inverse dari L1}*

**REALISASI: DENGAN NAMA DAN FUNGSI ANTARA**

```
IsInverse(L) :

              IsEqual (L3,L2)
```

**REALISASI LAIN**

*{ Basis 1 : list dengan satu elemen : true*
*Rekurens: dua buah list sama, jika panjangnya sama dan*

*L1 :* [ *e1* ] *o* [ *Tail(L1) -x1* ] • [ *x1* ]

*L2 :* [ *e2* ] *o* [ *Tail(L2) – x2* ] • [ *x2* ]

*e1=x2 dan Tail(L1) -x1 = Tail(L2)-x2*

```
IsInverse(L) :
       if NbElmt(L1) = NbELmt(L2) then {Analisa kasus }
         if IsEmpty(L1) and IsEmpty(L2) then {Basis 0}
               true
         else {Rekurens }
              ( FirstElmt(L1)=LastElmt(L2)) and
               IsInverse (Head(Tail(L1)), Head(Tail(L2))
      else{panjang tidak sama, pasti bukan hasil inverse }
          false
```

## List of Character (Teks)

Teks adalah list yang elemennya terdiri dari karakter. Karakter adalah himpunan terhingga dari ‘a’..’z’, ‘A..’Z’, ‘0..9’

**Definisi rekursif**

- Basis 0 : Teks kosong adalah teks
- Rekurens : Teks dapat dibentuk dengan menambahkan sebuah karakter pada teks

**TYPE LIST OF CHARACTER**

**DEFINISI DAN SPESIFIKASI TYPE**

**type** Teks : List of character

*{Definisi Teks : sesuai dengan definisi rekursif di atas }*

**DEFINISI DAN SPESIFIKASI KONSTRUKTOR**

**Konso** : character, Teks → Teks

*{Konso(e,T): menghasilkan sebuah list dari e dan T, dengan e sebagai elemen pertama e :*
*e o T → T'}*

**Kons•** : Teks, character → Teks

*{ Kons(T,e): menghasilkan sebuah list dari T dan e, dengan e sebagai elemen terakhir teks :*
*T • e → T'}*

**DEFINISI DAN SPESIFIKASI SELEKTOR**

**FirstChar**: Teks tidak kosong → character

*{FirstElmt(L) Menghasilkan character pertama Teks T }*

**Tail** : Teks tidak kosong → Teks

*{Tail(T) : Menghasilkan list tanpa elemen pertama Teks T }*

**LastChar** : Teks tidak kosong → character

*{LastElmt(T) : Menghasilkan character terakhir Teks T }*

**Head** : Teks tidak kosong → Teks

*{Head(T) : Menghasilkan list tanpa elemen terakhir Teks T}*

**DEFINISI DAN SPESIFIKASI PREDIKAT DASAR**

**(UNTUK BASIS ANALISA REKURENS}**

*{Basis 0 }*

**IsEmpty** : Teks → boolean

*{IsEmpty(T) benar jika list kosong }*

*{Basis 1 }*

**IsOneElmt**: Teks → boolean

*{IsOneElmt (X,T) adalah benar jika teks hanya mempunyai satu karakter }*

**Contoh-1 Teks : Hitung A**

**Persoalan :**

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menghitung kemunculan huruf 'a' pada suatu teks.

| **Hitung A** | **nba (T)** |
|---|---|

**DEFINISI DAN SPESIFIKASI**

**nba** : Teks → integer ≥ 0

*{ nba(T) menghasilkan banyaknya kemunculan 'a' dalam teks T }*

**REALISASI VERSI-1 : DENGAN KONS•**

*{ Basis 0: teks kosong: tidak mengandung 'a', nba ([]) = 0*

*Rekurens:* Head(T) • e

a

*ekspresikan domain berdasarkan elemen lain, selain Basis*

*nba (awt• lastchar) bisa dievaluasi kalau nba (awt) diketahui :*

*nba(T)=nba(Head(T)) + 1 jika e adalah 'a'*

*nba(T) =nba(Head(T)) jika e bukan 'a'}*

```
nba(x) :
     if IsEmpty(x) then {Basis 0}
                    0
     else {Rekurens}
         nba (Head (x)) +  if (Lastchar (x) ='a' then 1 else 0
```

**REALISASI VERSI-2 : DENGAN KONSO**

*{ Basis 0 : teks kosong: tidak mengandung 'a', nba ([]) = 0*

*Rekurens:* e o Tail(T)

e= *a*

*ekspresikan domain berdasarkan elemen lain, selain Basis*

*nba(T)= 1 + nba(Tail(T)) jika e adalah 'a'*

*nba(T) = nba(Tail(T)) jika e bukan 'a'}*

```
nba(x) :  if IsEmpty(x) then {Basis 0}
                    0
          else {Rekurens}
             if (Firstchar (x) ='a' then 1 else  0)
                 + nba(Tail(x))
```

**REALISASI VERSI-3 : BASIS BUKAN TEKS KOSONG**

*{ Basis 1:teks dengan satu karakter :* e

'a'

*jika karakter adalah a, maka 0. Jika bukan 'a', maka 1*

*Rekurens:* Head(T) e1 • e2 •

a

*ekspresikan domain berdasarkan elemen lain, selain Basis*

*nba (awt . e1 . e2) =*

```
        nba (awt . e1) + if (e2 = 'a')
                    then 1 else 0
           selalu terhadap teks tidak kosong. }
nba (T) :
          if lastChar(T) = 'a' then 1 else
             if IsEmpty (Head(T)) then 0
             else nba (Head(T))
```

**Contoh-2 Teks : Banyaknya kemunculan suatu karakter pada teks T**

**Persoalan :**

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menghitung banyaknya kemunculan sebuah karakter dalam teks.

**KEMUNCULAN SUATU KARAKTER nbx(C,T)**

**DEFINISI DAN SPESIFIKASI**

nbx : character, Teks → integer

*{nbx(C,T) menghitung banyaknya kemunculan karakter C pada teks T}*

**REALISASI**

```
{Basis 0}
  (1) nbx (C, [ ]) = 0
  { teks kosong tidak mengandung karakter apapun}
  {Rekurensurence }
  (2) nbx (C, T • e ) :
  { jika e = C maka, 1 + kemunculan karakter pada T
    jika e ≠ C, maka kemunculan karakter pada T}

nbx (C,T):
  if IsEmpty(T) then {Basis 0 }
        0
  else   {Rekurens}
      if LastChar(T)= C then 1 + nbx (C, Head(T))
                        else nbx(C, Head(T))
```

**Contoh-3 Teks : Banyaknya kemunculan suatu karakter pada teks T**

**Persoalan :**

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah predikat yang memeriksa apakah sebuah karakter muncul dalam teks minimal N kali.

**APAKAH MINIMAL ADA N KARAKTER C Atleast(C,N,T)**

**DEFINISI DAN SPESIFIKASI**

Atleast : integer >0, character, teks → boolean

*{ Atleast (N,C,T) benar, jika karakter C minimal muncul N kali pada Teks T }*

**REALISASI**

*{Basis 0}*

*(1) Atleast (0,C,[]) = true*
*{ karakter C pada teks kosong akan muncul 0 kali}*
*(2) Atleast (0, C1, C2oT) = true*
*{ karakter apapun minimal muncul 0 kali pada teks selalu benar }*
*(3) Atleast (1+N, C1, []) = false, N bil. natural.*
*{Rekurens}*
*(4) Atleast (1+n,C1,C2oT)= if (C1=C2 then Atleast (n,C1,T)*
*else Atleast (1+n, C1,T)*

```
Atleast (n,C, T):
   if IsEmpty(T) then {Basis 0}
           0
   else {Rekurens :analisa kasus}
         depend on n
            n = 0 : true
            n > 0 : if (C = FirstChar(T))
                    then Atleast (n,C,Tail(T))
                      else Atleast (1+n, C,Tail(T))
```

**Contoh-4 Teks : Palindrom**

**Persoalan :**

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah predikat yang memeriksa apakah sebuah teks palindrome. Sebuah teks disebut sebagai palindrom jika dibaca dari awal sampai dengan terakhir identik dengan dari karakter terakhir sampai ke awal. Contoh teks palindrom : "NABABAN", "KASUR RUSAK", "KASUR NABABAN RUSAK"

| **Memeriksa palindrome** | **IsPalindrome (T)** |
|---|---|

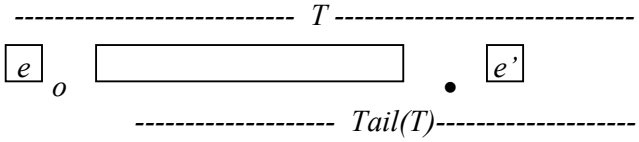
**DEFINISI DAN SPESIFIKASI**

**IsPalindrome** : Text → boolean

*{IsPalindrome(T) benar jika teks T adalah palindrome : jika dibaca dari kiri ke kanan, hasilnya sama dengan jika dibaca dari kanan ke kiri }*

**REALISASI DENGAN O**

*{ Basis 0 : teks kosong adalah palindrome*
*teks dengan satu elemen adalah palindrom*
*Rekurens :*




*}*

```
IsPalindrome (T) :
   depend on (T)
    {Kasus khusus } T = [] : true
     {Basis-1} T = [e] : true
   {Rekurens}  else    : (FirstChar(T) = LastChar(T)) and then
                 IsPalindrome (Head(Tail(T))
```

## List of Integer

**Definisi rekursif list integer**

- Basis: List kosong adalah list bilangan integer
- Rekurens : list bilangan integer dibuat dengan cara menambahkan sebuah integer pada list bilangan integer.

**TYPE LIST INTEGER**

**DEFINISI DAN SPESIFIKASI TYPE**

**type** List of integer

*{Definisi: list yang elemennya integer, sesuai dengan definisi rekursif di atas }*

**DEFINISI DAN SPESIFIKASI KONSTRUKTOR**

**Konso** : integer, List of integer tidak kosong → List of integer

*{Konso(e,L): menghasilkan sebuah list dari e dan L, dengan e sebagai elemen pertama e :*
*e o L → L'}*

**Kons•** : List of integer tidak kosong, integer → List of integer

*{ Kons(L,e): menghasilkan sebuah list dari L dan e, dengan e sebagai elemen terakhir list :*
*L • e → L'}*

**DEFINISI DAN SPESIFIKASI SELEKTOR**

**FirstElmt**: List of integer tidak kosong → integer

*{FirstElmt(L) Menghasilkan elemen pertama list L }*

**Tail** : List of integer tidak kosong → List of integer

*{Tail(L) : Menghasilkan list tanpa elemen pertama list L }*

**LastElmt** : List of integer tidak kosong → integer

*{LastElmt(L) : Menghasilkan elemen terakhir list L }*

**Head** : List of integer tidak kosong → List of integer

*{Head(L) : Menghasilkan list tanpa elemen terakhir list L}*

**DEFINISI DAN SPESIFIKASI PREDIKAT DASAR**

**(UNTUK BASIS ANALISA REKURENS}**

*{Basis 0 }*

**IsEmpty** : List of integer → boolean

*{IsEmpty(L) benar jika list kosong }*

*{Basis 1 }*

**IsOneElmt**: List of integer → boolean

*{IsOneElmt (X,L) adalah benar jika teks hanya mempunyai satu elemen }*

**DEFINISI DAN SPESIFIKASI PREDIKAT KEABSAHAN**

**IsListInt**: List → boolean

*{IsListInt(Lmenghasilkan true jika L adalah list dengan elemen integer }*

**Berikut ini diberikan contoh persoalan yang spesifik terhadap pemrosesan elemen list yang bertype bilangan integer**

**Contoh-1 List integer : Maksimum**

**Persoalan :**

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menghasilkan elemen bernilai maksimum dari sebuah list bilangan integer. Perhatikanlah bahwa fungsi rekursif untuk maksimum didasari atas basis 1, sebab jika list kosong, maka nilai maksimum tidak terdefinisi. Untuk ini predikat dasar untuk mentest adalah predikat IsOneElmt , basis 1

**NILAI MAKSIMUM LIST INTEGER** **maxlist(Li)**

**DEFINISI DAN SPESIFIKASI**

**maxlist** : list of integer tidak kosong → integer

*{ maxlist(Li) : menghasilkan elemen Li yang bernilai maksimm }*

**IsOneElmt** : list of integer tidak kosong → boolean

*{OneElmt(L) benar, jika list L hanya mempunyai satu elemen }*

**REALISASI**

```
maxlist(Li) :
        if IsOneElmt( Li) then {Basis 1}
                LastElmt (Li)
         else {Rekurens}
              max2 (LastElmt (Li), maxList (Head (Li))
```

*{dengan max2(a,b) adalah fungsi yang mengirimkan nilai maksimum dari dua buah integer a dan b yang pernah dibahas pada bagian I. }*

**Contoh-2 List integer : Ukuran (Dimensi) List**

**Persoalan :**

Tuliskanlah definisi, spesifikasi dan realisasi dari dari sebuah fungsi yang menghasilkan banyaknya elemen (dimensi) sebuah list integer

| **UKURAN LIST** | **Dimensi(Li)** |
|---|---|

**DEFINISI**

Dimensi : list of integer → integer > 0

*{ DIM(Li) menghasilkan banyaknya elemen list integer }*

*{ Basis 0 : dimensi list kosong = 0*

*Rekurens : dimensi (Li)*

*e* o *Tail(Li)*

*1 + Dimensi (Tail(Li))*

**REALISASI**

```
Dimensi (Li) :
        if IsEmpty( Li)  then 0 else 1 + Dimensi(Tail(Li))
```

Catatan : ukuran list ini juga berlaku untuk list dengan elemen apapun. Namun karena akan dipakai untuk operasi lainnya, maka direalisasi.

**Contoh-3 List integer : Penjumlahan dua buah list integer:**

**Persoalan :**

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menjumlahkan dua buah list integer dan menghasilkan sebuah list integer.

Perhatikanlah bahwa contoh ini adalah contoh rekurens terhadap kedua list.

**PENJUMLAHAN DUA LIST INTEGER** **List+(L1,L2)**

**DEFINISI**

**List+** : 2 list of integer ≥ 0 → list of integer ≥ 0

*{ List+ (Li1,Li2):*
*menjumlahkan setiap elemen list integer yang berdimensi sama, hasilnya berupa list integer berdimensi tsb. }*
*{ Basis 0 : Dimensi(Li1)=0 and Dimensi(Li2)= 0 : []*

*Rekurens : Dimensi(Li1) = 0 and Dimensi(Li2) = 0 :*

*e1* *o* *Tail(Li1)*

*e2* *o* *Tail(Li2)* +

---

*e1+e2 o List+(Tail(Li1), Tail(Li2))*
*}*

**REALISASI**

```
List+ (Li1, Li2) :
  if Dimensi(Li1) = 0 then  {Basis 0}
     []
  else {Rekurens}
       Konso(FirstElmt(Li1)+FirstElmt(Li2),
             List+(Tail(Li1),Tail(Li2) )
```

**Contoh-4 List of integer : INSERTION SORT**

Persoalan :

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menghasilkan list terurut **membesar** dari sebuah list bilangan integer sembarang dengan metoda "insertion sort"

| **INSERTION SORT** | **Insort(Li)** |
|---|---|

**DEFINISI**

**Insort** : list of integer → list of integer

*{ Insort(Li) menghasilkan list integer yang terurut dengan metoda insertion sort }*

*{ Basis 0 : list kosong sudah terurut, hasilnya list kosong*

*Rekurens : Insort (Li)*

| *e* | o | *Tail(Li)* |
|---|---|---|

*Hasil: Insert e ke Tail(Li) yang sudah terurut dg Insertion sort }*

**Insert** : integer, list of integer tidak kosong, terurut membesar → list of integer terurut membesar

*{ Insert(x,Li) melakukan insert x ke Li menghasilkan list terurut membesar*

*{ Basis 0 : list kosong, insert sebuah elemen x ke list kosong : [x]*

*Rekurens :*

| *e* | o | *Tail(Li)* |
|---|---|---|

$x \leq e$ : *x oLi*

$x > e$ : *e o Insert(x,Tail(Li))*

*Catatan : insert tidak pernah "merusak" keterurutan elemen list!}*

**REALISASI**

```
Insert (x,Li) :
   if  IsEmpty(Li) then {Basis 0}
          [x]
   else {Rekurens}
          if (x  ≤ FirstElmt(Li) )
          then Konso(x,Li)
         else Konso(FirstElmt(Li),Insert (x,Tail(Li))
Insort (Li) :
   if IsEmpty( Li)  then {Basis 0}
        [ ]
   else {Rekurens}
       Insert(First(Li),Insort(Tail(Li)))
```

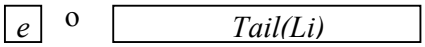
**Contoh-5 List integer : Banyaknya kemunculan nilai maksimum**

**Fungsi dengan *range* type bentukan tanpa nama**

**Persoalan :**

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menghasilkan nilai maksimum **dan** banyaknya kemunculan dari nilai maksimum dari sebuah list bilangan integer positif. Nilai maksimum hanya terdefinisi jika list tidak kosong. Contoh ini adalah contoh fungsi yang mengembalikan type komposisi tanpa nama dengan parameter masukan sebuah list. Perhatikanlah realisasi dalam dua versi sebagai berikut dan pelajarilah implementasinya dengan bahasa pemrograman fungsional yang dipakai. Contoh : List [11 3 4 5 11 6 8 11] menghasilkan <11,3>

Solusi versi-1 : dengan analisa rekurens berdasarkan definisi
Analisis rekurens :
Basis 1: List dengan satu elemen e: menghasilkan <e,1>
Rekurens : List dengan struktur e oTail(Li) harus dianalisis sebagai berikut :




Nilai maks= m, #kemunculam m=n

Jika m adalah nilai maksimum Tail(Li) dan n adalah banyaknya kemunculan m pada Tail(Li), maka ada tiga kemungkinan :
m < e : e adalah maksimum "baru", maka hasilnya <e,1>
m = e : terjadi kemunculan m, maka hasilnya <m,n+1>
m > e : e dapat diabaikan, maka hasilnya <m,n>

**KEMUNCULAN MAKSIMUM (versi-1)**

**DEFINISI DAN SPESIFIKASI**

**maxNb** : list of integer → <integer, integer>

*{maxNb(Li) menghasilkan <nilai maksimum, kemunculan nilai maksimum> dari suatu list integer Li ; <m,n> =m adalah maks x dari n # occurence m dalam list}*

**REALISASI VERSI-1**

```
MaxNb(Li) : {menghasilkan nilai maksimum dan kemunculannya }
  if OneElmt(Li) then {Basis 1}
       <FirstElmt(Li),1>
  else
     let <m,n> = MaxNb (Tail(Li))
     in    depend on m,FirstElmt(Li)
                m < FirstElmt(Li) : < FirstElmt(Li),1>
                m = FirstElmt(Li) : < m,1+n>
                m > FirstElmt(Li) : <m,n>
```

Versi kedua : dengan realisasi fungsi antara yang menghasilkan:

- Nilai maksimum
- Banyaknya kemunculan nilai maksimum

**KEMUNCULAN MAKSIMUM (versi-2)**

**DEFINISI DAN SPESIFIKASI**

**maxNb** : list of integer → <integer, integer>

*{maxNb(Li) menghasilkan <nilai maksimum, kemunculan nilai maksimum> dari suatu list integer Li ; <m,n> =m adalah maks x dari n # occurence m dalam list}*

**max** : list of integer → integer

*{ max(Li) menghasilkan nilai maksimum dari elemen suatu list integer Li }*

**Vmax** : list of integer → integer

*{ VMax(Li) adalah NbOcc(max(Li),Li) yaitu banyaknya kemunculan nilai maksimum dari Li , dengan aplikasi terhadap NbOcc (max(Li),Li)}*

**NbOcc** : integer, list of integer → integer > 0

*{ NbOcc(X ,Li) yaitu banyaknya kemunculan nilai X pada Li }*

**REALISASI VERSI-2**

```
max (Li) : {nilai maskimum list }
   if  IsOneELmt(Li) then {Basis 1}
      [x]
   else {Rekurens : analisa kasus }
          if (FirstElmt(Li) > max(Tail(Li))
          then FirstElmt(Li)
         else max(Tail(Li))
Vmax (Li) : { Banyaknya kemunculan nilai maks }
     NBOcc(max(Li), Li)
NbOcc(X, Li) : { banyaknya kemunculan nilai
   if  IsOneELmt(Li) then {Basis 1, analisa kasus}
      if X=FirstElmt(Li) then
         1
      else
         0
   else {Rekurens : analisa kasus }
      if X=FirstElmt(Li) then
         1 + NbOcc(Tail(Li))
      else
         NbOcc(Tail(Li))
 MaxNb(Li) : <max(Li), NbOcc(max(Li), Li)>
```

## Himpunan (Set)

**Definisi :**
Himpunan *(set)* adalah sebuah list yang setiap elemennya hanya muncul sekali (unik). List "kosong" adalah himpunan kosong.

Contoh :
[apel, jeruk, pisang] adalah himpunan
[apel, jeruk, mangga, jeruk] bukan himpunan

**TYPE SET (HIMPUNAN)**

**DEFINISI DAN SPESIFIKASI TYPE**

*{**Set** adalah List dengan tambahan syarat bahwa tidak ada elemen yang sama }*

*{ Semua konstruktor, selektor dan fungsi pada List berlaku untuk Himpunan }*

**DEFINISI DAN SPESIFIKASI KONSTRUKTOR HIMPUNAN DARI LIST**

*{ Himpunan dibentuk dari list }*

**MakeSet** (L) : list → set

*{ membuat sebuah set dari sebuah list }*
*{ yaitu membuang semua kemunculan yang lebih dari satu kali}*
*{ List kosong tetap menjadi list kosong }*

**DEFINISI DAN SPESIFIKASI PREDIKAT**

**IsSet** : list → boolean

*{ IsSet(L) true jika L adalah set }*

**IsSubSet** : 2 set → boolean

*{ IsSubSet (H1,H2) true jika H1 adalah subset dari H2: semua elemen H1 adalah juga merupakan elemen H2 }*

**DEFINISI DAN SPESIFIKASI OPERASI TERHADAP HIMPUNAN**

**MakeIntersect** : 2 set → set

*{ Intersect (H1,H2) membuat interseksi H1 dengan H2 : yaitu set baru dengan anggota elemen yang merupakan anggota H1 dan juga anggota H2 }*

**MakeUnion** : 2 set → set

*{ Union (H1,H2) membuat union H1 dengan H2 : yaitu set baru dengan semua anggota elemen H1 dan anggota H2 }*

Untuk kasus himpunan berikut, dibutuhkan beberapa fungsi terhadap list sebagai berikut :

**IsMember** : elemen, list → boolean

*{ IsMember(e,L) true jika e adalah elemen list L }*

**Rember** : elemen, list → list

*{ Rember (x,L) menghapus sebuah elemen bernilai x dari list }*
*{ list yang baru berkurang SATU elemennya yaitu yang bernilai e }*
*{ List kosong tetap menjadi list kosong }*

**MultiRember** : elemen, list → list

*{ MultiRember (x,L) menghapus semua elemen bernilai x dari list }*
*{ list yang baru tidak lagi mempunyai elemen yang bernilai x }*
*{ List kosong tetap menjadi list kosong }*

**Contoh 1: Menghapus SEBUAH elemen sebagai anggota list**

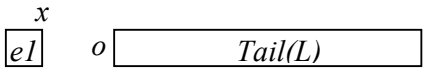
**Predikat ini dibutuhkan untuk membentuk himpunan**

**HAPUS1ELEMEN** **Rember(e.L)**

**DEFINISI**

**Rember** : elemen, list → list

*{ Rember (x,L) menghapus sebuah elemen bernilai x dari list }*
*{ list yang baru berkurang SATU elemennya yaitu yang bernilai e }*
*{ List kosong tetap menjadi list kosong }*
*{ Base : list kosong : → list kosong*
*Rekurens :*




*e = x : hasil adalah Tail(L) ,*
*e ≠x : e1 o Hasil rember(e,Tail(L) ) }*

**REALISASI**

```
Rember(x,L) :
          if IsEmpty(L)  then {Basis }
                  L
          Else {Rekurens : analisa kasus }
               if FirstElmt(L)=x then Tail(L)
              else Konso (FirstElmt(L),Rember(x,Tail(L))
```

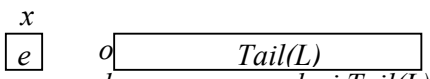
**Contoh 2: Menghapus SEMUA elemen e sebagai anggota list :**
**Predikat ini dibutuhkan untuk membentuk himpunan**

| **HAPUS SEMUA ELEMEN** | **Multirember(e,L)** |
|---|---|

**DEFINISI**

**MultiRember** : elemen, list → list

*{ MultiRember (x,L) menghapus semua elemen bernilai x dari list }*
*{ list yang baru tidak lagi mempunyai elemen yang bernilai x }*
*{ List kosong tetap menjadi list kosong }*
*{ Base : list kosong : → List kosong*
*Rekurens :*




*e = x : hapus semua x dari Tail(L) ,*
*e ≠ x : e1 o hasil penghapusan semua x dari Tail(L)}*

**REALISASI**

```
MultiRember(x,L) :
    if IsEmpty(L)  then {Basis}
          L
    else {Rekurens : analisa kasus }
          if FirstElmt(L)=x
          then MultiRember(x,Tail(L))
          else Konso (FirstElmt(L),MultiRember(x,Tail(L))
```

**Contoh 3: Mentest apakah sebuah list adalah himpunan**

**Persoalan:**

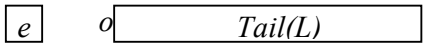
Tuliskanlah definisi, spesifikasi dan realisasi dari sebuh predikat yang akan mentest apakah sebuah list adalah Himpunan

| **APAKAH SET** | **IsSet(L)** |
|---|---|

**DEFINISI PREDIKAT**

**IsSet** : list → boolean

*{ Set(L) true jika L adalah set }*
*{ Base : list kosong adalah set*
*Rekurens :*




*merupakan set jika Tail(L) tidak mengandung e}*

**REALISASI VERSI-1**

```
IsSet(L) :
    if IsEmpty(L)  then {Basis: list kosong adalah himpunan kosong }
```

```
          true
   else {Rekurens : analisa kasus }
        if IsMember(FirstElmt(L),Tail(L))  then false
        else IsSet(Tail(L))
```

**REALISASI VERSI-2**

```
IsSet(L) :
   if IsEmpty(S) then {Basis }
      true
    else {Rekurens:}
        not IsMember(FirstElmt(L),Tail(L))  or then IsSet(Tail(L))
```

**REALISASI**

```
IsSet(L) :
   Isempty(S) or then not IsMember(FirstElmt(L),Tail(L))
              or then IsSet(Tail(L))
```

**Contoh 4: Membuat sebuah set dari sebuah list**

**Persoalan :**

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang akan membentuk sebuah Himpunan dari elemen-elemennya, yaitu dengan meniadakan duplikasi elemen

**MEMBENTUK SET (versi-1)** **MakeSet (L)**

**DEFINISI**

MakeSet1 (L) : list → set

*{ membuat sebuah set dari sebuah list }*
*{ yaitu membuang semua kemunculan yang lebih dari satu kali}*
*{ List kosong tetap menjadi list kosong }*
*{ Base : list kosong : → List kosong*
*Rekurens :*

| *e* | *o* | *Tail(L)* |
|---|---|---|

*Untuk setiap e :*
*e adalah Member dari Tail(L) : MakeSet(Tail(L))*
*e bukan Member dari Tail(L) : e o MakeSet(Tail(L)) }*

**REALISASI**

```
MakeSet1(L) :
   if IsEmpty(L)  then {Basis}
      L
   else {Rekurens}
      if IsMember(FirstElmt(L),Tail(L)) then
            MakeSet(TAIL(L))
      else Konso (FirstElmt(L),MakeSet(Tail(L))
```

**APLIKASI**

⇒ MakeSet( [apel, sirsak, per, mangga, apel, jeruk, sirsak])

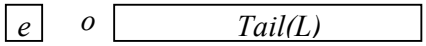
{Himpunan hasil adalah : [per, mangga, apel, jeruk, sirsak] }

**MEMBENTUK SET (versi-2)** **MakeSet (L)**

**DEFINISI**

MakeSet2 : list → set

*{ MakeSet2 (S) membuat sebuah set dari sebuah list }*
*{ yaitu mempertahankan elemen pertama yang muncul, dan membuang kemunculan elemen tersebut pada sisa jika muncul lebih dari satu kali}*
*{ List kosong tetap menjadi list kosong }*
*{ Base : list kosong : → ()*
*Rekurens :*




*e o MakeSet(Tail(L)) dengan Tail(L) yg tidak lagi mengandung e}*

**REALISASI**

```
MakeSet2(L) :
   if IsEmpty(L) then {Basis }
        L
   Else {Rekurens }
      Konso (FirstElmt(L),MakeSet(MULTIRember(FirstElmt(L),Tail(L))
```

**APLIKASI**

⇒ MakeSet( [apel, sirsak, per, mangga, apel, jeruk, sirsak])

{Himpunan hasil : [apel, sirsak, per, mangga, jeruk] }

**Contoh 5: Mentest apakah sebuah set merupakan subset dari set yang lain**

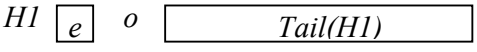
**Persoalan :**
Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah predikat yang akan mentest apakah sebuah himpunan merupakan himpunan bagian dari himpunan lain yang diberikan

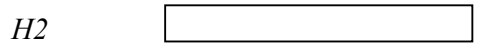
**APAKAH SUBSET** **IsSubSet(H1,H2)**

**DEFINISI PREDIKAT**

IsSubSet : 2 set → boolean

*IsSubSet (H1,H2) true jika H1 adalah subset dari H2: semua elemen H1 adalah juga merupakan elemen H2 }*
*{ List kosong adalah subset dari set apapun}*
*{ Base : list kosong : → true*
*Rekurens :*



*Setiap karakter H1 harus dicek thd H2 :*
*e anggota dari H2 : adalah subset jika Tail(H1) adalah subset H2*
*e bukan anggota H2: H1 pasti bukan subset H2 }*

**REALISASI**

```
IsSUBSet(H1,H2):
 {Basis}  if Isempty(H1)  then true
 {Rekurens} else {analisa kasus }
              if not IsMember(FirstElmt(H1),H2) then false
            else { e anggota H2 }
                 IsSubSet (Tail(H1),H2)
```

Sebagai latihan, buatlah realisasi yang hasilnya identik, namun dengan memanfaatkan ekspresi boolean dengan operator **and then** dan **or else**

**Contoh 6 :Mentest kesamaan dua buah set**

**Persoalan :**
Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah predikat yang akan mentest apakah dua buah himpunan identik.

| **KESAMAAN DUA SET** | **IsEQSet (H1,H2)** |
|---|---|

**DEFINISI PREDIKAT**

**IsEQSet** : 2 set → boolean

*{ IsEQSet (H1,H2) true jika H1 "sama dengan" H2, yaitu jika semua elemen H1 juga merupakan elemen H2, tanpa peduli urutannya }*
*{ H1==H2 jika dan hanya jika H1 adalah subset H2 dan H2 adalah subset H1}*

**REALISASI**

```
IsEQSet(H1,H2): IsSUBSet(H1,H2) and then IsSUBSet(H2,H1)
```

**Contoh 7: Mentest apakah dua buah set berinterseksi**

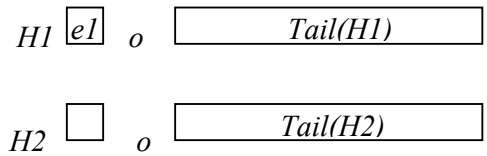
**Persoalan :**
Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah predikat yang akan mentest apakah dua buah himpunan saling beririsan

| **APAKAH INTERSEKSI** | **Intersect (H1,H2)** |
|---|---|

**DEFINISI PREDIKAT**

IsIntersect : 2 set → boolean

*{ IsIntersect (H1,H2) true jika H1 berinterseksi dengan H2 : minimal ada satu anggota yang sama. Himpunan kosong bukan merupakan himpunan yang berinterseksi dengan himpunan apapun }*
*{ Base : Salah satu kosong : → false*
*Rekurens :*




*e1 adalah Member dari H2 : true*
*e1 bukan Member dari H2 : Intersect(Tail(H1),H2) }*

**REALISASI**

```
IsIntersect (L) :
   depend on H1, H2
     {Basis : }
     Isempty(H1) and Isempty(H2)     : false
     not Isempty(H1) and Isempty(H2) : false
     Isempty(H1) and not Isempty(H2) : false
     not IsEmpty(H1) and not IsEmpty(H2) : { Rekurens}
         IsMember(FirstElmt(H1),H2) or then IsIntersect(Tail(H1),H2)
```

**Contoh 8:Membuat interseksi dari dua buah set :**

**Persoalan :**

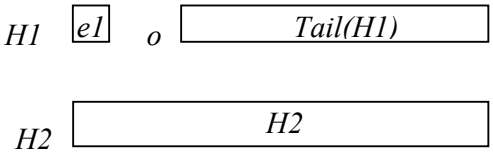
Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menghasilkan sebuah himpunan yang elemennya adalah hasil irisan dari dua buah himpunan

| **BUAT INTERSEKSI** | **Intersect(H1,H2)** |
|---|---|

**DEFINISI**

MakeIntersect : 2 set → set

*{ Intersect (H1,H2) membuat interseksi H1 dengan H2 : yaitu set baru dengan anggota elemen yang merupakan anggota H1 dan juga anggota H2 }*
*{ Base : Jika salah satu kosong , hasilnya set kosong*
*Rekurens :*




*e1 adalah Member dari H2 : e1 o MakeIntersect(Tail(H1),H2)*
*e1 bukan Member dari H2 : MakeIntersect(Tail(H1),H2)}*

**REALISASI**

```
Make Intersect(H1,H2) :
      depend on H1, H2
      {Basis }
         IsEmpty(H1) and IsEmpty(H2)     : []
         not IsEmpty(H1) and IsEmpty(H2) : []
         IsEmpty(H1) and not IsEmpty(H2) : []
         not IsEmpty(H1) and not IsEmpty(H2) : {Rekurens}
               if   IsMember(FirstElmt(H1),H2)
               then Konso (FirstElmt(H1), MakeIntersect(Tail(H1),H2))
               else MakeIntersect(Tail(H1),H2))
```

**Contoh 9: Membuat Union dari dua buah set**

**Persoalan :**
Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menghasilkan himpunan yang elemennya merupakan hasil union (gabungan) dari dua buah himpunan

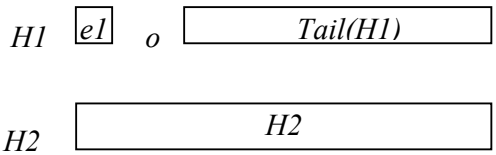
| **SET UNION** | **MakeUnion (H1,H2)** |
|---|---|

**DEFINISI**

**MakeUnion** : 2 set → set

*{ MakeUnion (H1,H2) membuat union H1 dengan H2 : yaitu set baru dengan semua anggota elemen H1 dan anggota H2 }*

*{ Base : Jika salah satu kosong, hasilnya adalah himpunan yang tidak kosong*
*Kedua set kosong , hasilnya sebuah set yang kosong*
*Rekurens :*




*e1 adalah Member dari H2 : buang e1, MakeUnion(Tail(H1),H2)*
*e1 bukan Member dari H2 : e1 o MakeUnion(Tail(H1),H2)}*

**REALISASI**

```
MakeUnion(H1,H2) :
   depend on H1, H2
     {Basis }
     IsEmpty(H1) and IsEmpty(H2)     : []
     not IsEmpty(H1) and IsEmpty(H2) : H1
     IsEmpty(H1) and not IsEmpty(H2) : H2
     not IsEmpty(H1) and not IsEmpty(H2) : {Rekurens}
                 if    IsMember(FirstElmt(H1),H2)
                 then MakeUnion(Tail(H1),H2))
                 else Konso(FirstElmt(H1),Union(Tail(H1),Tail(H2))
```

## List of List

### Definisi rekursif

List of list adalah list yang :

- mungkin kosong,
- mungkin terdiri dari sebuah elemen yang disebut **atom dan sisanya adalah list of list**,
- mungkin terdiri dari sebuah elemen berupa **list dan sisanya adalah list of list**

Jadi List of List adalah list S yang elemennya adalah list. Dalam bahasa LISP, type ini disebut sebagai S-expression

Untuk membedakan antara list dengan atom: List dituliskan di antara tanda kurung (), sedangkan Atom dituliskan tanpa tanda kurung

Contoh List :

[ ] adalah list kosong

[ad , a , b ] adalah list dengan elemen berupa 3 atom

[ [ ], [a ,b ,c] ,[ d ,e] ,f ] adalah :

list dengan elemen list kosong, L1, L2 dan sebuah atom

L1 adalah list dengan elemen 3 buah atom a, b, c

L2 adalah list dengan elemen 2 buah atom d,e

Untuk list khusus ini, nama Konstruktor dan Selektor tidak dibedakan dengan list yang hanya mengandung elemen dasar, namun diperlukan predikat tambahan sebagai berikut: yaitu untuk mengetahui apakah sebuah ekspresi adalah Atom atau List.

Atom dapat berupa:

- atom numerik (yang dapat dipakai sebagai operan dalam ekspresi aritmatik)
- atom simbolik

**TYPE LIST-OF-LIST**

**DEFINISI DAN SPESIFIKASI PREDIKAT KHUSUS UNTUK LIST OF LIST**

**IsEmpty** : list of list → boolean

*{IsEmpty(S) benar jika S adalah list of list kosong}*

**IsAtom** : list of list → boolean

*{IsAtom(S) menghasilkan true jika list adalah atom, yaitu terdiri dari sebuah atom }*

**IsList** : list of list → boolean

*{ IsList(S) menghasilkan true jika S adalah sebuah list (bukan atom)}*

**DEFINISI DAN SPESIFIKASI KONSTRUKTOR**

**KonsLo** : List, List of list → List of list

*{ KonsLo(L,S) diberikan sebuah List L dan sebuah List of List S, membentuk list baru dengan List yang diberikan sebagai elemen pertama List of list: L o S → S'}*

**KonsL•** : List of list , List → List of list

*{KonsL• (S,L) diberikan sebuah List of list S dan sebuah list L, membentuk list baru dengan List yang diberikan sebagai elemen terakhir list of List: S • L → S'}*

**DEFINISI DAN SPESIFIKASI SELEKTOR**

**FirstList**: List of list tidak kosong → List

*{FirstList(S) Menghasilkan elemen pertama list, mungkin sebuah list atau atom }*

**TailList** : List of list tidak kosong → List of list

*{TailList(S) Menghasilkan "sisa" list of list S tanpa elemen pertama list S }*

**LastList** : List of list tidak kosong → List of list

*{LastList(S) : Menghasilkan elemen terakhir list of list S, mungkin list atau atom }*

**HeadList** : List of list tidak kosong → List of list

*{HeadList(S) Menghasilkan "sisa" list of list tanpa elemen terakhir list }*

**Contoh-1 : Mencek kesamaan dua buah list of list:**

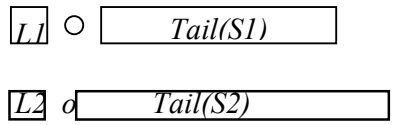
**Persoalan :**
Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah predikat yang mencek kesamaan dua buah list of list

| **KESAMAAN** | **IsEqS (L1,S2)** |
|---|---|

**DEFINISI PREDIKAT**

**IsEqS** : 2 List of list → boolean

*{IsEqS (S1,S2 ) true jika S1 identik dengan S2 : semua elemennya sama }*
*{ Basis : kedua list kosong : → true*
*salah satu list kosong : → false*
*Rekurens :*




*L1 dan L2 adalah atom : L1=L2 and IsEqS(TailList(S1),TailList(S2))*
*L1 dan L2 adalah list : IsEqS(S1,S2) and IsEqS(TailList(S1),TailList(S2))*
*else : false*
*}*

**REALISASI**

```
IsEqS(S1,S2) :
    depend on S1, S2
         IsEmpty(S1) and IsEmpty(S2) : true
         not IsEmpty(S1) and IsEmpty(S2) : false
         IsEmpty(S1) and not IsEmpty(S2) : false
         not IsEmpty(S1) and not IsEmpty(S2) :
           depend on FirstList(S1), FirstList(S2)
                IsAtom(FirstList(S1) and IsAtom(FirstList(S1)) :
                       FirstList(S1) = FirstList(S1) and
                       IsEqS (TailList(S1), TailList(S2))
                IsList(FirstList(S1) and IsList(FirstList(S1):
                     IsEqS(FirstList(S1),FirstList(S2)) and
                     IsEqS(TailList(S1),TailList(S2))
                  Else {atom dengan list pasti tidak sama}
                       false
```

**Contoh-2 : Mencek apakah sebuah atom merupakan elemen sebuah list yang elemennya list:**

**Persoalan :**
Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah predikat yang mencek keanggotaan sebuah elemen terhadap list of list

**KEANGGOTAAN** **IsMemberS (A,S)**

**DEFINISI PREDIKAT**

**IsMemberS** : elemen, List of list → boolean

*{ IsMemberS (A,S) true jika A adalah anggota S }*
*{ Basis : list kosong : → false*
*Rekurens :*

| *L1* | ○ | *Tail(S)* |
|---|---|---|

*L1 adalah atom dan A = L1 : true*
*L1 bukan atom : A anggota L1 or IsMemberS(A, TailList(S))*
*}*

**REALISASI**

```
IsMemberS(A,S) :
   depend on S
        IsEmpty(S): false
        Not IsEmpty(S) :
            depend on FirstList(S)
                 IsAtom(FirstList(S)): A = FirstList(S)
                 IsList(FirstList(S)) : IsMember(A,FirstList(S))
                    or IsMemberS(A,TailList(S))
{ dengan IsMember(A,L) adalah fungsi yang mengirimkan true jika A adalah
elemen list L}
```

**Contoh-3 : Mencek apakah sebuah List merupakan elemen sebuah list yang elemennya list**

**Persoalan :**
Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang mencek keanggotaan sebuah list terhadap list of list.

**KEANGGOTAAN** **IsMemberLS (L,S)**

**DEFINISI PREDIKAT**

**IsMemberLS** : List, List of list → boolean

*{ IsMemberLS (L,S2) true jika L adalah anggota S }*
*{ Basis : L dan S list kosong : → true*
*L atau S tidak kosong : false*
*Rekurens :*

| L1 | ○ | Tail(S) |
|---|---|---|

*L1 adalah atom : IsMemberLS(L,TailList(S))*
*L1 bukan atom : L1=L : true*
*L1≠L : IsMemberLS(L,TailList(S))*
*}*

**REALISASI**

```
IsMemberLS(L,S) :
   depend on S
        IsEmpty(L) and IsEmpty(S): true
        not IsEmpty(L) and IsEmpty(S) : false
        IsEmpty(L) and not IsEmpty(S) : false
         not IsEmpty(L) and not IsEmpty(S) :
            if (IsATOM(FirstList(S))) then IsMemberLS(Taillist(L,S))
             else {   IsLIST(FirstList(S)) }
                    If IsEqual(L,FirstList(S)) then true
                    else      IsMemberLS(L,Taillist(S))
{ dengan IsEqual(L1,L2) adalah fungsi yang mengirimkan true jika list L1
sama dengan list L}
```

**Contoh 4: Menghapus SEBUAH elemen (atom) dari list of list :**

**Persoalan :**
Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menentukan menghapus senuah kemunculan atom tertenut yang ada pada sebuah list of list

**HAPUS*ELEMEN** **Rember*(a,S)**

**DEFINISI**

**Rember***: elemen, List of list → List of list

*{ Rember* (a,S) menghapus sebuah elemen bernilai a dari semua list S }*

*{ List kosong tetap menjadi list kosong }*
*{ Basis : list kosong : → ( )*
*Rekurens :*

*a*

*S* | *L1* *o* | *Tail(S)*

*L1 adalah atom : L1 = a : TailList(S) tanpa a*
*L1 ≠ a : L1 o (TailList(S) tanpa a)*
*L1 adalah list : (L1 tanpa a) o (TailList(S) tanpa a )*
*}*

**REALISASI**

```
Rember*(a,L) :
   if IsEmpty(S)  then S
   else if IsList(FirstList(S))
        then KonsLo(Rember*(a,FirstList(S)), Rember*(a,TailList(S)))
        else { elemen pertama S adalah atom }
             if FirstElmt(S) = a then
               Rember*(a,TailList(S))
            else
                 KonsLo (FirstElmt(S),Rember*(a,Tail(S))
```

**Contoh 5 : Maksimum dari list of list dengan atom integer:**

**Persoalan :**

Tuliskanlah definisi, spesifikasi dan realisasi dari sebuah fungsi yang menentukan nilai maksimum dari atom-atom yang ada pada sebuah list of list yang tidak kosong.

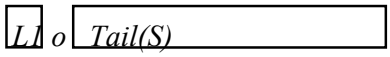
**ELEMEN BERNILAI MAKSIMUM** **Max(S)**

**DEFINISI**

**Max** List of list tidak kosong → integer

*{ Max (S) menghasilkan nilai elemen (atom) yang maksimum dari S }*

*{ Basis : list dengan satu elemen E1*
*E1 adalah atom : nilai E1*
*E1 adalah list : Max(E1)*
*Rekurens :*
*a*




*L1 adalah atom : Max2(L1,Max(Tail(S))*
*L1 adalah list : Max2 (Max(L1), Max(Tail(S))*
*}*
*{Fungsi antara }*

**Max2** 2 integer → integer

*{Max2(a,b) menghasilkan nilai maksimum a dan b }*

**REALISASI**

```
Max2(a,b) :
    If a>=b then
         a
    Else
        b

Max(S) :
   if IsOneElmt(S)  then  {Basis 1 }
         if IsAtom(FirstList(S))  then
                FirstList(S)
         Else   { List }
                Max(FirstList(S))
   Else  {Rekurens }
       if IsAtom(FirstList(S))  then {First elemen adalah atom }
           Max2(FirstList(S),Max(TailList(S))
        Else   { Firs element adalah List }
           Max2(Max(FirstList(S)), Max(TailList(S))
```

## Resume dari analisa rekurens terhadap list

**Rekurens terhadap bilangan integer n: f(n)**

Basis : n = 0 : ekspresi basis
Rekurens : f(prec(n))

**Rekurens terhadap list(L) : f(L)**

Basis kosong :

Basis : IsEmpty(L) : ekspresi basis

Rekurens : f(Tail(L))

Basis satu :

Basis : IsOneElmt(L) : ekspresi basis

Rekurens : f(Tail(L))  { minimal dua elemen }

**Rekurens terhadap list of list (S) : f(S)**

Basis : IsEmpty(S) : ekspresi basis

Rekurens :

depend on FirstList(S)

IsAtom(FirstList(S)) : g(ekspresi terhadap atom ,f(TailList(S)))

IsList(FirstList(S)): g(ekspresiterhadaplist, f(Tail(S)))