Struktur Berkait

IF2110/IF2111 – Algoritma dan Struktur Data
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

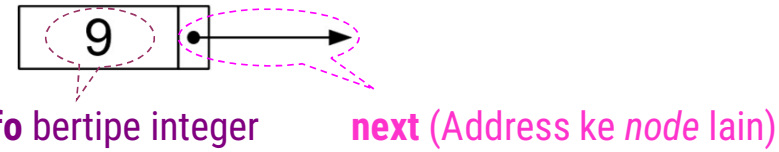
Definisi

Struktur berkait terdiri atas *node* yang terkait dengan *node* lain.

Node merupakan sebuah *tuple* yang terdiri atas dua bagian:

1) Sebuah nilai dengan tipe tertentu (info),
2) Sebuah penunjuk ke *node* lain (next).
   Bisa jadi tidak menunjuk ke mana pun (NIL).

Ilustrasi:



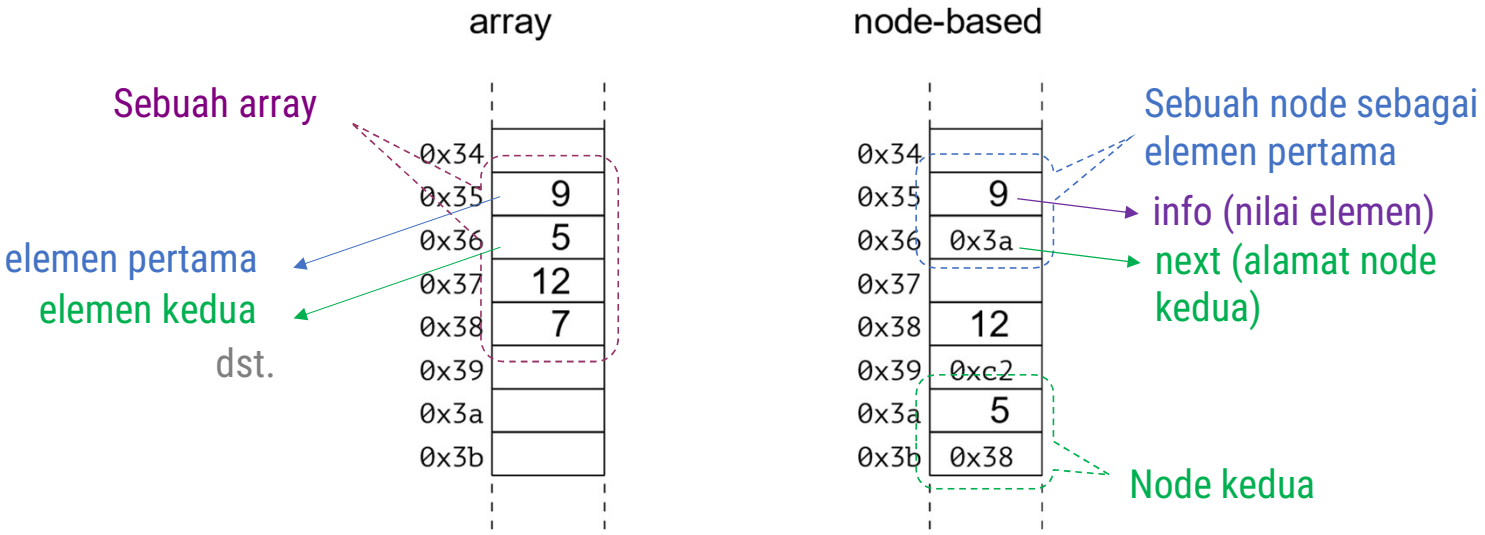
Hal ini memungkinkan penyimpanan elemen-elemen tanpa harus kontigu.

Disebut juga struktur *node-based*, *linked list*, atau *linear list*.

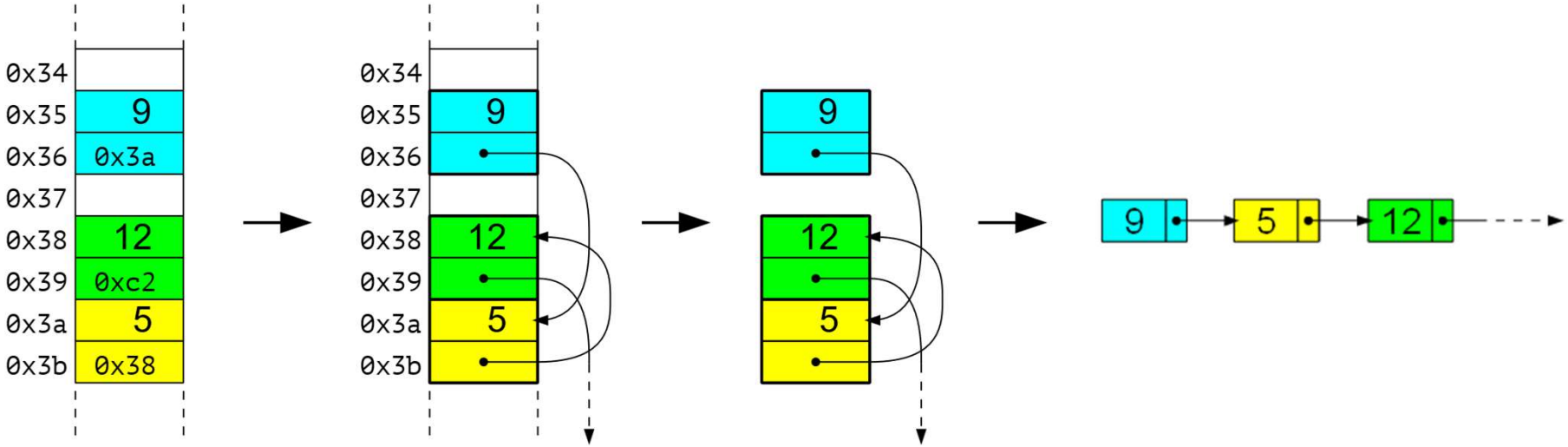
Memori fisik: array vs. node-based

- **Array**: elemen-elemen berada pada lokasi bersebelahan.
- **Node**: "next" mencatat alamat elemen berikutnya.

Contoh, dengan elemen bertipe *byte* (1 elemen mengisi 1 slot memori):



Ilustrasi struktur berkait



Karakteristik struktur data berbasis node

Memori dialokasi sesuai kebutuhan.

- Jika ada 3 elemen maka hanya perlu memori sebesar ukuran *node* ×3.
- Berbeda dengan *array* yang, misalnya sudah dialokasikan 100 elemen maka menggunakan memori sebesar ukuran elemen ×100 meskipun pada suatu waktu hanya 3 elemen yang efektif.

Ukuran memori per elemen menjadi lebih besar.

Ukuran elemen + ukuran *pointer*.

Secara umum mengorbankan efisiensi ruang (memori) demi efisiensi waktu.

(Tidak untuk semua jenis operasi.)

Node dalam Notasi Algoritmik

```
{ Deklarasi tipe bentukan }
  type ElType: integer
  type Address: pointer to Node
  type Node: < info: ElType,
               next: Address >

{ Deklarasi variabel }
  p1: Address
  p2: Address

{ Inisialisasi dan penggunaan variabel }
  p1 ← alokasi(9) { p1 menunjuk ke Node dengan info=9 dan next=NIL }
  p2 ← alokasi(5) { p2 menunjuk ke Node dengan info=5 dan next=NIL }

  p1↑.next ← p2   { Address next pada p1 menunjuk ke node yang ditunjuk p2 }
```

Node dalam Bahasa C (deklarasi tipe bentukan)

```c
/* node.h */
#ifndef NODE_H
#define NODE_H

typedef int ElType;
typedef struct node* Address;
typedef struct node {
  ElType info;
  Address next;
} Node;

#define INFO(p) (p)->info
#define NEXT(p) (p)->next
Address newNode(ElType val);

#endif
```

```c
/* node.c */
#include "node.h"
#include <stdlib.h>

Address newNode(ElType val) {
  Address p = (Address)
              malloc(sizeof(Node));
  if (p!=NULL) {
    INFO(p) = val;
    NEXT(p) = NULL;
  }
  return p;
}
```

Node dalam Bahasa C (contoh penggunaan)

```
/* Deklarasi variabel */
Address p1, p2;

/* Inisialisasi dan penggunaan variabel */
p1 = newNode(9); /* p1 menunjuk ke Node dengan info=9 dan next=NIL */
p2 = newNode(5); /* p2 menunjuk ke Node dengan info=5 dan next=NIL */

NEXT(p1) = p2; /* Address next pada p1 menunjuk ke node yang ditunjuk p2 */
```

Implementasi ADT List dengan Struktur Berkait

IF2110/IF2111 – Algoritma dan Struktur Data
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

Kembali ke definisi List

List, dikenal juga dengan *sequence*, merupakan sekumpulan elemen **bertipe sama** yang memiliki suatu **keterurutan tertentu** *(ordered*, tidak harus *sorted)*.

Operasi-operasi:
- isEmpty
- indexOf
- length
- akses (getElmt, setElmt)
- concat
- insert-
- delete-
- pola traversal

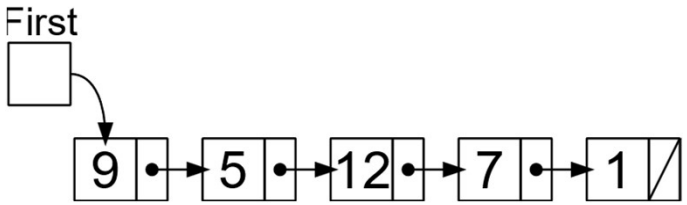
Implementasi List dengan struktur berkait

Elemen-elemen direpresentasikan dengan **Node** ⟨Info, Next⟩ yang saling berkait.

List diacu melalui **Address** elemen pertamanya (**first**).

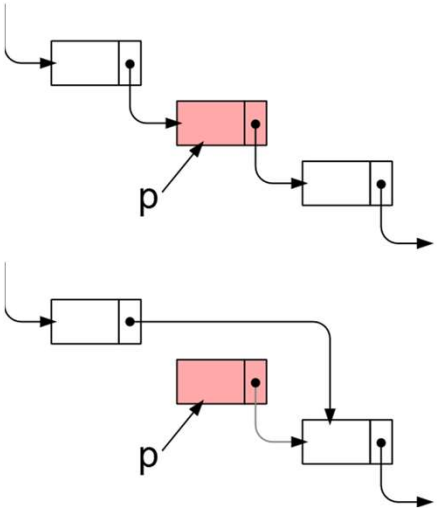
Alamat elemen berikutnya (suksesor) diakses dengan **next**.

Elemen terakhir ditandai dengan Next menunjuk ke **NIL**.

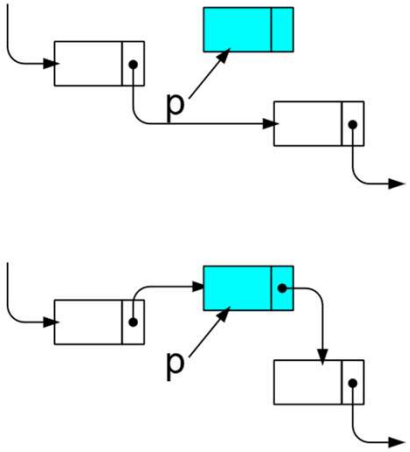


Karakteristik list linier

Penambahan & penghapusan elemen sangat sederhana.



penghapusan elemen



penambahan elemen

Tidak efisien untuk mengakses elemen melalui indeksnya.
(Harus menelusuri mulai dari *node* pertama sambal mencacah.)

Implementasi List dengan struktur berkait

Jika l adalah List, dan p adalah Address:

- Karena l diacu melalui alamat node pertamanya, maka first = l.
- Elemen yang diacu oleh p dapat diakses informasinya dengan notasi:
  - **p↑.info**: nilai yang disimpan
  - **p↑.next**: alamat elemen berikutnya

Definisi List Kosong

- List l adalah list kosong: l = **NIL**

Definisi Elemen terakhir

- last↑.next = **NIL**, dengan last adalah alamat elemen terakhir

Dalam notasi algoritmik

```
{  Deklarasi tipe  }

  type ElType: integer
  type Address: pointer to Node
  type Node: < info: ElType,
               next: Address >
  type List: Address


{  Deklarasi variabel  }
  l: List
  p1: Address
  p2: Address

{  Inisialisasi List  }

  CreateList(l)
```

```
{  Akses node pertama:  }

  p1 ← l

{  Cetak isi p1 & akses elemen
   setelah p1  }

  output(p1↑.info)
  p2 ← p1↑.next
```

Dalam bahasa C

```
/* Deklarasi tipe */

  typedef int ElType;
  typedef struct tNode* Address;
  typedef struct tNode {
            ElType info;
            Address next; } Node;
  typedef Address List;

/* Deklarasi variabel */
  List l;
  Address p1;
  Address p2;

/* Inisialisasi List */

  CreateList(*l)
```

```
/* Akses node pertama: */

  p1 = l

/* Cetak isi p1 & akses elemen
   setelah p1 */

  printf("%d\n", p1->info);
  p2 = p1->next;
```

Skema Pemrosesan List Berkait

IF2110/IF2111 – Algoritma dan Struktur Data
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

Skema dasar pemrosesan List berkait

- Traversal
- Pencarian sekuensial (*search*)

Skema traversal

Digunakan untuk memroses setiap elemen List dengan cara yang sama.

Mekanisme: mengunjungi setiap elemen List secara berurutan dimulai dari elemen pertama hingga elemen terakhir.

Jenis traversal:

- Dasar (tanpa perlakuan khusus untuk List kosong)
- Dengan perlakuan khusus untuk List kosong
- Untuk List yang tidak kosong

Skema traversal dasar

```
procedure SKEMAlistTraversal1(input l: List)
{ I.S. List l terdefinisi, mungkin kosong. }
{ F.S. Semua elemen List l "dikunjungi" dan telah diproses. }
{ Traversal sebuah List linier tanpa pemrosesan khusus untuk List kosong. }
KAMUS LOKAL
  p: Address                          { address untuk traversal, type terdefinisi }
  procedure proses(input p: Address) { pemrosesan elemen ber-address p }
  procedure inisialisasi              { aksi sebelum proses dilakukan }
  procedure terminasi                 { aksi sesudah semua pemrosesan elemen selesai }
ALGORITMA
  inisialisasi
  p ← l
  while (p ≠ NIL) do
    proses(p)
    p ← p↑.next
  terminasi
```

Skema traversal dengan penanganan list kosong

```
procedure SKEMAlistTraversal2(input l: List)
{ I.S. List l terdefinisi, mungkin kosong. }
{ F.S. Semua elemen List l "dikunjungi" dan telah diproses. }
{ Traversal sebuah List linier dengan pemrosesan khusus untuk List kosong. }
KAMUS LOKAL
  { SAMA SEPERTI SKEMA 1, tidak ditulis untuk menghemat tempat }
ALGORITMA
  if l = NIL then
    output("List kosong")
  else
    inisialisasi
    p ← l
    repeat
      proses(p)
      p ← p↑.next
    until (p = NIL)
    terminasi
```

Skema traversal untuk List tidak kosong

```
procedure SKEMAlistTraversal3(input l: List)
{ I.S. List l terdefinisi, tidak kosong: minimal mengandung satu elemen. }
{ F.S. Semua elemen List l "dikunjungi" dan telah diproses. }
{ Traversal untuk List linier yang sudah dipastikan tidak kosong. }
KAMUS LOKAL
  { SAMA SEPERTI SKEMA 1 dan 2, tidak ditulis untuk menghemat tempat }
ALGORITMA
  inisialisasi
  p ← l
  iterate
    proses(p)
  stop (p↑.next = NIL)
    p ←  p↑.next
  terminasi
```

Skema pencarian sekuensial

List linier tidak memungkinkan *binary search*.

Mekanisme: mengunjungi elemen-elemen List secara berurutan dimulai dari elemen pertama hingga ditemukan elemen yang memenuhi syarat pencarian, atau semua elemen telah dikunjungi.

Jenis skema pencarian:

- Dengan boolean
- Tanpa boolean

Pencarian berdasarkan nilai elemen (1)

```
procedure SKEMAlistSearch1(input l: List, input x: ElType,
                           output p: Address, output found: boolean)
{ I.S. List linier l sudah terdefinisi dan siap dikonsultasi, x terdefinisi }
{ F.S. p adalah address di mana x ditemukan, p = NIL jika tidak ketemu }
{      found menyatakan apakah nilai x yang dicari ditemukan }
{ Menggunakan skema search dengan boolean }
KAMUS LOKAL
  -
ALGORITMA
  p ← l
  found ← false
  while (p ≠ NIL) and (not found) do
    if (x = p↑.info) then
      found ← true
    else
      p ← p↑.next
  { p = NIL or found}
  { Jika found maka p = Address dari nilai yg dicari }
  { p = NIL jika nilai tidak ditemukan}
```

Pencarian berdasarkan nilai elemen (2)

```
procedure SKEMAlistSearch2(input l: List, input x: ElType,
                           output p: Address, output found: boolean)
{ I.S. List linier l sudah terdefinisi dan siap dikonsultasi, x terdefinisi }
{ F.S. p adalah address di mana x ditemukan, p = NIL jika tidak ketemu }
{      found menyatakan apakah nilai x yang dicari ditemukan }
{ Menggunakan skema search tanpa boolean }
KAMUS LOKAL
  -
ALGORITMA
  p ← l
  if p = NIL then  { List kosong }
    found ← false
  else { List tidak kosong }
    while (p↑.next ≠ NIL) and (p↑.info ≠ x) do
      p ← p↑.next
    { p↑.next = NIL or p↑.info = x }
    found ← (p↑.info = x)
    if (not found) then
      p ← NIL
```

Pencarian elemen yang memenuhi kondisi

```
procedure SKEMAlistSearchX(input l: List, input kondisi(p): boolean,
                           output p: Address, output found: boolean)
{ I.S. List linier l sudah terdefinisi dan siap dikonsultasi, kondisi(p) adalah
       suatu ekspresi boolean yang merupakan fungsi dari elemen beralamat p }
{ F.S. p adalah address di mana kondisi(p) terpenuhi, p = NIL jika tidak ketemu }
{      found menyatakan apakah ada p yang memenuhi kondisi(p) }
KAMUS LOKAL
  -
ALGORITMA
  p ← l
  found ← false
  while (p ≠ NIL) and (not found) do
    if kondisi(p) then
      found ← true
    else
      p ←  p↑.next
  { p = NIL or found}
  { Jika found maka p adalah elemen List dengan kondisi(p) true }
```

Operasi Primitif List Linier

IF2110/IF2111 – Algoritma dan Struktur Data
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

Definisi ADT

```
KAMUS UMUM
  type ElType: integer
  type Address: pointer to Node
  type Node: < info: ElType,
               next: Address >
  type List: Address

{ Konstruktor }
procedure CreateList(output l: List)
{ I.S. Sembarang
  F.S. Terbentuk list l kosong: l diinisialisasi dengan NIL }
KAMUS LOKAL
  -
ALGORITMA
  l ← NIL
```

Operasi-operasi

- isEmpty
- indexOf
- length
- akses (getElmt, setElmt)
- insert-
  - -First
  - -At
  - -Last
- delete-
  - -First
  - -At
  - -Last
- concat

isEmpty

```
function isEmpty(l: List) → boolean
{ Tes apakah sebuah list l kosong.
  Mengirimkan true jika list kosong, false jika tidak kosong. }
KAMUS LOKAL
  -
ALGORITMA
  → (l = NIL)
```

indexOf

```
function indexOf(l: List, val: ElType) → integer
{ Prekondisi: l, x terdefinisi. Mengembalikan indeks elemen pertama l yang bernilai x
  (jika ada), atau mengembalikan IDX_UNDEF jika tidak ada. }
KAMUS LOKAL
  idx: integer;   p: Address;   found: boolean
ALGORITMA
  p ← l; found ← false; idx ← 0
  while p≠NIL and not found do
    if p↑.info=val then
      found ← true
    else
      idx ← idx+1
      p ← p↑.next
  if found then
    → idx
  else
    → IDX_UNDEF
```

length

```
function length(l: List) → integer
{ Prekondisi: l terdefinisi.
  Menghasilkan banyaknya elemen pada list l, 0 jika list kosong. }
KAMUS LOKAL
  ctr: integer
  p: Address
ALGORITMA
  ctr ← 0
  p ← l
  while p≠NIL do
    ctr ← ctr+1
    p ← p↑.next
  { p=NIL }
  → ctr
```

akses (getElmt, setElmt)

```
function getElmt(l: List,
                 idx: integer) → ElType
{ Prekondisi: l terdefinisi,
    idx indeks yang valid dalam l,
    yaitu 0..length(l).
  Mengirimkan nilai elemen l pada
  indeks idx. }
KAMUS LOKAL
  ctr: integer
  p: Address
ALGORITMA
  ctr ← 0
  p ← l
  while ctr<idx do
    ctr ← ctr+1
    p ← p↑.next
  {ctr=idx}
  → p↑.info
```

```
procedure setElmt(input/output l: List,
   input idx: integer, input val: ElType)
{ I.S. l terdefinisi, idx indeks yang
       valid dalam l, yaitu 0..length(l).
  F.S. elemen l pada indeks ke-idx
       diganti nilainya menjadi val. }

KAMUS LOKAL
  ctr: integer
  p: Address
ALGORITMA
  ctr ← 0
  p ← l
  while ctr<idx do
    ctr ← ctr+1
    p ← p↑.next
  {ctr=idx}
  p↑.info ← val
```

insertFirst

```
procedure insertFirst(input/output l: List, input val: ElType)
{ I.S. l terdefinisi, mungkin kosong.
  F.S. x menjadi elemen pertama l. }
KAMUS LOKAL
  p: Address
ALGORITMA
  p ← newNode(val)
  if p≠NIL then { alokasi berhasil }
    p↑.next ← l
    l ← p
```

insertAt

```
procedure insertAt(input/output l: List, input val: ElType, input idx: integer)
{ I.S. l terdefinisi, tidak kosong, i merupakan indeks yang valid di l.
  F.S. x disisipkan dalam l pada indeks ke-i (bukan menimpa elemen di i). }
KAMUS LOKAL
  ctr: integer
  p, loc: Address
ALGORITMA
  if idx=0 then
     insertFirst(l,val)
  else
    p ← newNode(val)
    if p≠NIL then { alokasi berhasil }
       ctr ← 0
       loc ← l
       while ctr<idx-1 do
          ctr ← ctr+1
          loc ← loc↑.next
       {ctr=idx-1}
       p↑.next ← loc↑.next
       loc↑.next ← p
```

insertLast

```
procedure insertLast(input/output l: List, input val: ElType)
{ I.S. l terdefinisi, mungkin kosong.
  F.S. x menjadi elemen terakhir l. }
KAMUS LOKAL
  p, last: Address
ALGORITMA
  if isEmpty(l) then
    insertFirst(l,val)
  else { List tidak kosong */ }
    p ← newNode(val)
    if p≠NIL then { alokasi berhasil }
      last ← l
      while (last↑.next≠NIL) do { cari alamat node terakhir }
        last ← last↑.next
      {last↑.next=NIL}
      last↑.next ← p
```

deleteFirst

```
procedure deleteFirst(input/output l: List, output val: ElType)
{ I.S. l terdefinisi, tidak kosong.
  F.S. e diset dengan elemen pertama l, elemen pertama l dihapus dari l. }
KAMUS LOKAL
  p: Address
ALGORITMA
  p ← l
  val ← p↑.info
  l ← p↑.next
  dealokasi(p)
```

deleteAt

```
procedure deleteAt(input/output l: List, input idx: integer, output val: ElType)
{ I.S. l terdefinisi, tidak kosong, i merupakan indeks yang valid di l.
  F.S. e diset dengan elemen l pada indeks ke-idx.
       Elemen l pada indeks ke-idx dihapus dari l. }
KAMUS LOKAL
  ctr: integer
  p, loc: Address
ALGORITMA
  if idx=0 then
     deleteFirst(l,val)
  else
    ctr ← 0
    loc ← l
    while ctr<idx-1 do
      ctr ← ctr+1
      loc ← loc↑.next
    {ctr=idx-1}
    p ← loc↑.next
    val ← p↑.info
    loc↑.next ← p↑.next
    dealokasi(p)
```

deleteLast

```
procedure deleteLast(input/output l: List, output val: ElType)
{ I.S. l terdefinisi, tidak kosong.
  F.S. e diset dengan elemen terakhir l, elemen terakhir l dihapus dari l. }
KAMUS LOKAL
  p, loc: Address
ALGORITMA
  p ← l
  loc ← NIL
  while p↑.next≠NIL do
    loc ← p
    p ← p↑.next
 {p↑.next=NIL}
  if loc=NIL then
    l ← NIL
  else
    loc↑.next ← NIL
  val ← p↑.info
  dealokasi(p)
```

concat

```
function concat(l1: List, l2: List) → List
{ Prekondisi: l1 dan l2 terdefinisi, mungkin kosong.
  Mengembalikan hasil Konkatenasi ("Menyambung") dua buah list, l2 ditaruh di
  belakang l1 }
KAMUS LOKAL
  p: Address;  l3: List
ALGORITMA
  CreateList(l3)
  p ← l1
  while p≠NIL do
    insertLast(l3,p↑.info)
    p ← p↑.next
  {p=NIL}
  p ← l2
  while p≠NIL do
    insertLast(l3,p↑.info)
    p ← p↑.next
 {p=NIL}
  → l3
```

Representasi Fisik List Linier:
Struktur Berkait dengan Pointer

IF2110/IF2111 – Algoritma dan Struktur Data
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

Representasi implisit dan eksplisit

Sebelumnya kita telah menjumpai list berkait dengan representasi **implisit**

yaitu struktur data list berkait di mana *menunjuk ke sebuah list* adalah sama dengan *menunjuk ke elemen pertamanya*:

```
type ElType: ...
type Address: pointer to Node
type Node: < info: ElType,
             next: Address >
type List: Address
```

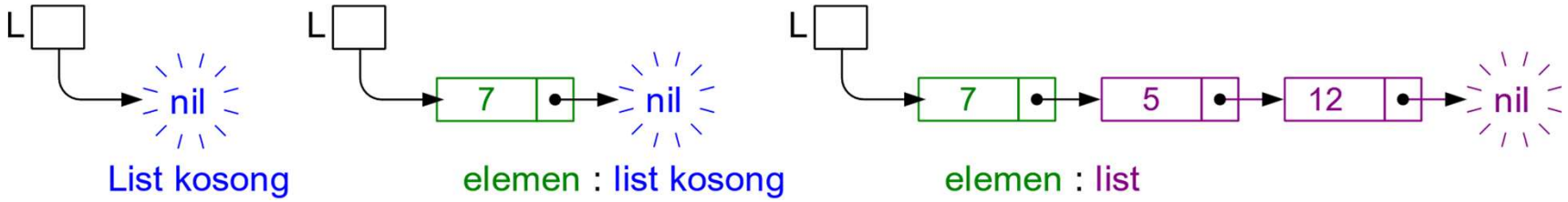
Terdapat representasi lain yaitu **eksplisit**, di mana elemen pertama list merupakan *bagian dari* struktur data list:

```
type List: < first: Address >
```

Representasi implisit

Representasi implisit mewakili definisi rekursif sebuah list linier seperti dalam paradigma fungsional:

- List kosong adalah list.
- List tidak kosong terdiri atas sebuah elemen yang diikuti list.



Elemen pertama list L, First = L.

Next dari First harus merupakan list juga, ∴ `type List: Address`

Representasi implisit

Representasi implisit mewakili definisi rekursif sebuah list linier seperti dalam paradigma fungsional:

- List kosong adalah list.
- List tidak kosong terdiri atas sebuah elemen yang diikuti list.

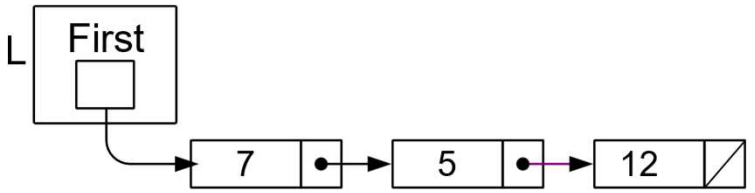
Elemen pertama list L, First = L.

Next dari First harus merupakan list juga, ∴ `type List: Address`

Representasi eksplisit

Dalam representasi eksplisit, First merupakan *bagian* dari struktur data list.

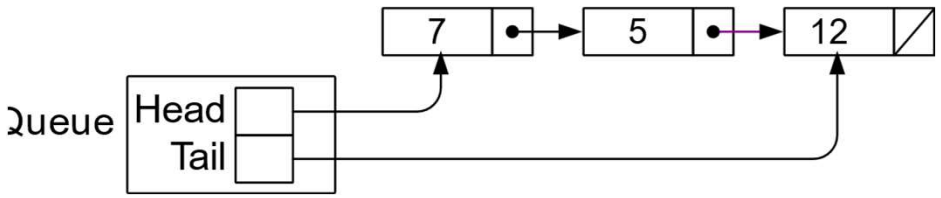
```
type List: < first: Address >
```



∴ akses ke elemen pertama `l` adalah `l.first`.

Representasi ini berguna misalnya pada implementasi Queue memanfaatkan struktur list berkait:

```
type Queue: < head: Address,
              tail: Address >
```



Proses rekursif

```
type ElType: ...
type Address: pointer to Node
type Node: < info: ElType,
             next: Address >
type List: Address

{ Representasi Implisit }
```

```
procedure displayList(input l: List)
{ I.S. l terdefinisi
  F.S. Setiap elemen l di-print }
KAMUS LOKAL
ALGORITMA
  if (isEmpty(l)) then { Basis 0 }
    { tidak melakukan apa-apa }
  else { Rekurens }
    output(l↑.info)
    PrintList(l↑.next)
```

```
...

displayList(l)
```

Proses rekursif

```
type ElType: ...
type Address: pointer to Node
type Node: < info: ElType,
             next: Address >
type List: < first: Address >

{ Representasi Eksplisit }
```

```
{ PROBLEMS:
  l tidak memiliki info.
  l tidak memiliki next. }
```

```
procedure displayList(input l: List)
{ I.S. L terdefinisi
  F.S. Setiap elemen list diprint }
KAMUS LOKAL
ALGORITMA
  if (isEmpty(l)) then { Basis 0 }
    { tidak melakukan apa-apa }
  else { Rekurens }
    output(l↑.info)
    PrintList(l↑.next)
```

```
...

displayList(l)
```

Proses rekursif

```
type ElType: ...
type Address: pointer to Node
type Node: < info: ElType,
             next: Address >
type List: < first: Address >

{ Representasi Eksplisit }
```

```
{ PROBLEM:
  l.first↑.next merupakan Address,
  tidak bisa di-pass ke displayList yang
  menerima sebuah List. }
```

```
procedure displayList(input l: List)
{ I.S. L terdefinisi
  F.S. Setiap elemen list diprint }
KAMUS LOKAL
ALGORITMA
  if (isEmpty(l)) then { Basis 0 }
    { tidak melakukan apa-apa }
  else { Rekurens }
    output(l.first↑.info)
    PrintList(l.first↑.next)
```

```
...

displayList(l)
```

Proses rekursif

```
type ElType: ...
type Address: pointer to Node
type Node: < info: ElType,
             next: Address >
type List: < first: Address >

{ Representasi Eksplisit }
```

```
{ PROBLEM:
  p merupakan Address, tidak bisa di-pass
  ke isEmpty yang menerima sebuah List. }
```

```
procedure displayList (input p: Address)
{ I.S. p terdefinisi
  F.S. Setiap elemen list diprint }
KAMUS LOKAL
ALGORITMA
  if (isEmpty(p)) then { Basis 0 }
    { tidak melakukan apa-apa }
  else { Rekurens }
    output(p↑.info)
    displayList(p↑.next)

...

displayList(l.first)
```

Representasi Fisik List Linier:
Struktur Berkait dengan Array

IF2110/IF2111 – Algoritma dan Struktur Data
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

Kekurangan struktur berkait

Pada pembahasan implementasi ADT List menggunakan struktur berkait, setiap Node dialokasikan satu demi satu.

- Persoalan: alokasi & dealokasi memori adalah operasi yang “mahal” pada sistem operasi.
- Akan lebih efisien jika dapat dilakukan alokasi beberapa Node sekaligus.

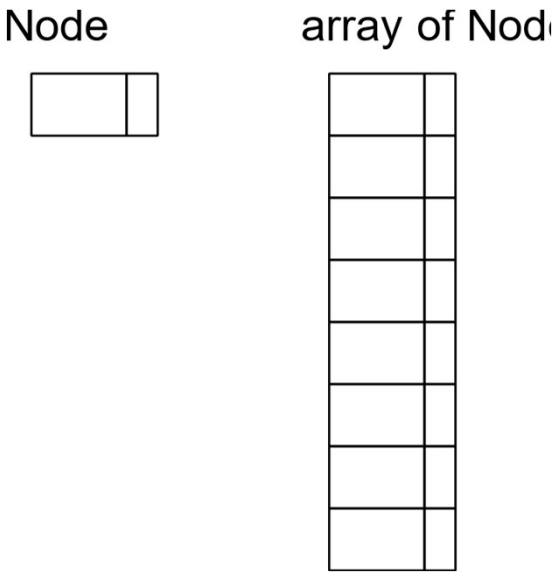
Persoalan lain: bahasa pemrograman yang digunakan mungkin tidak mendukung pointer.

Alternatif: array of Node

Banyak Node dialokasi dengan satu kali pemanggilan ke sistem operasi, dalam bentuk array.

Bagian Next dari Node kini bukan mengacu pada alamat fisik memori melainkan indeks array.

Array of Node dapat dideklarasikan secara global untuk digunakan oleh beberapa List sekaligus.



Array of Node

Saat inisialisasi, bagian Next setiap Node diisi dengan indeks elemen array berikutnya (`nodeArray[i].next = i+1`).

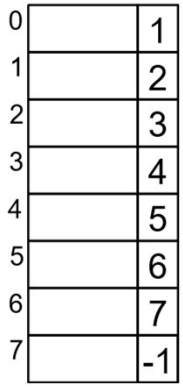
Untuk Node terakhir, diisi dengan indeks yang tidak valid (konstanta, misal -1).

Diperlukan sebuah pencatat Node pertama yang kosong.

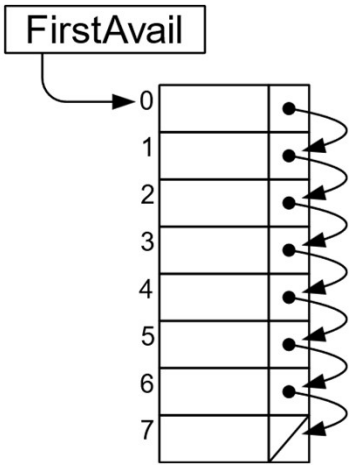
Saat inisialisasi, diisi dengan indeks pertama array (0).

Ilustrasi: setelah inisialisasi

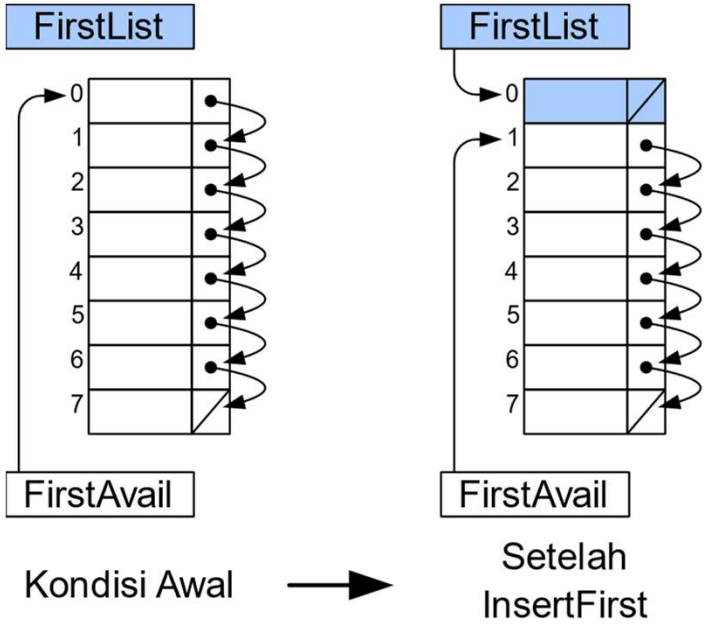
FirstAvail = 0



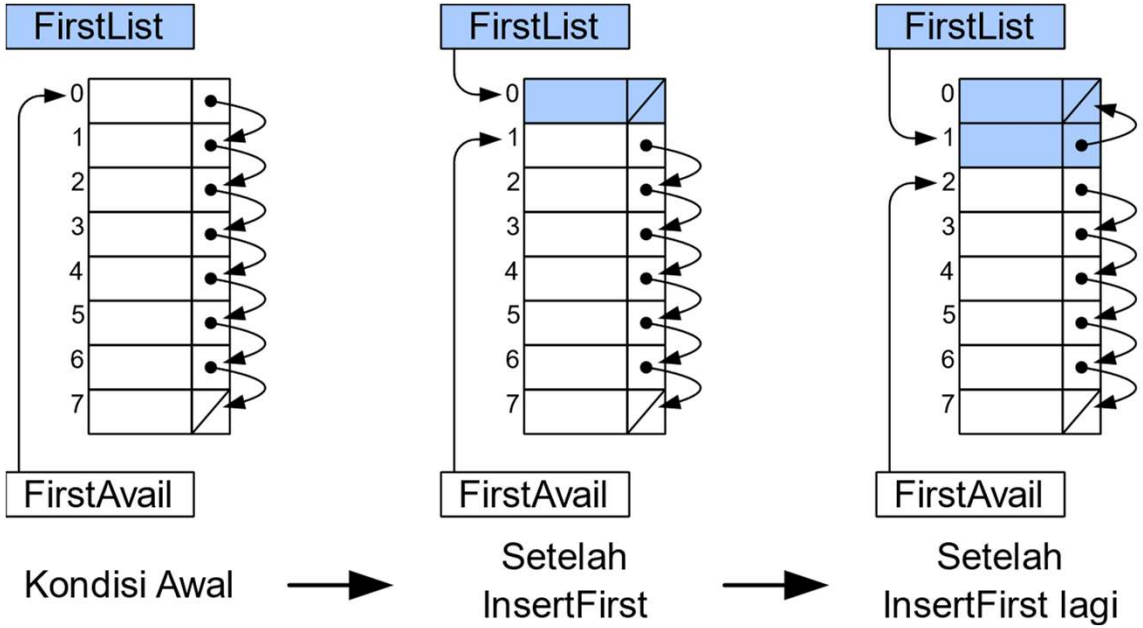
atau



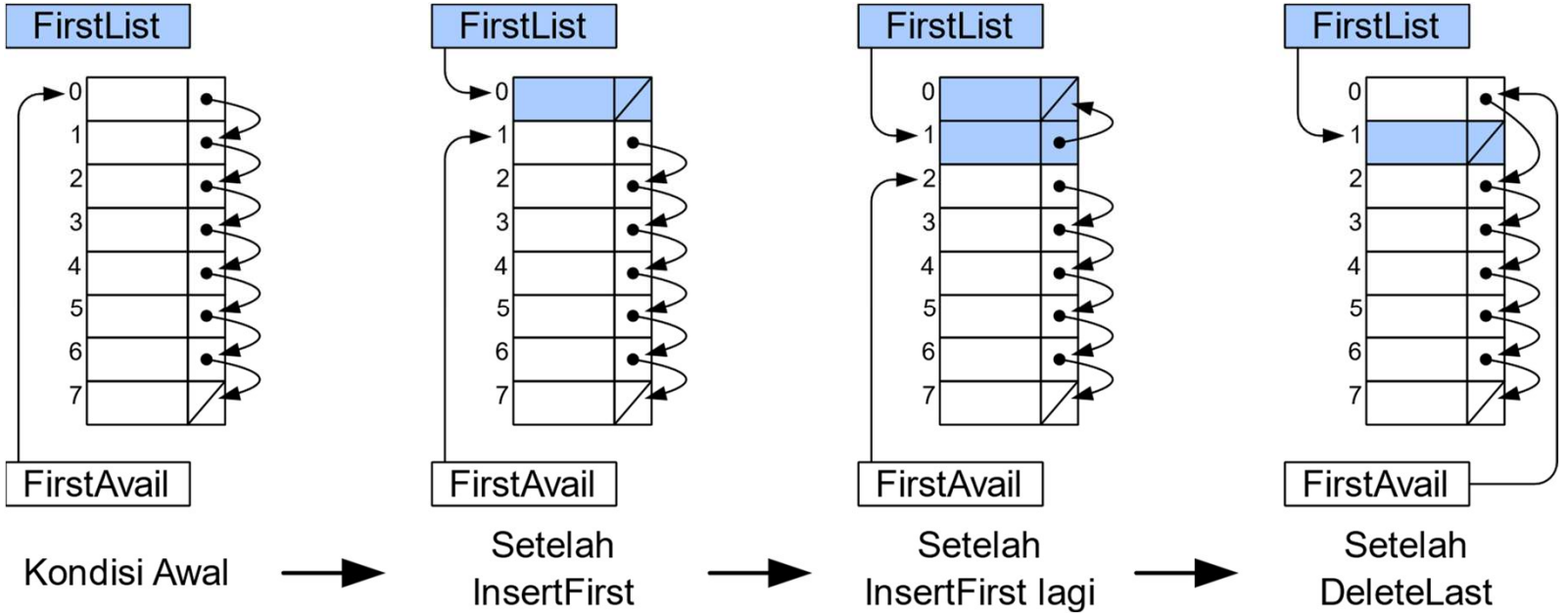
Ilustrasi: pemakaian memori list



Ilustrasi: pemakaian memori list

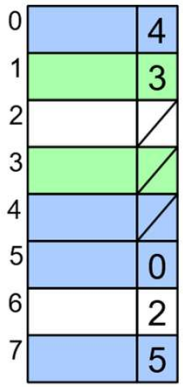


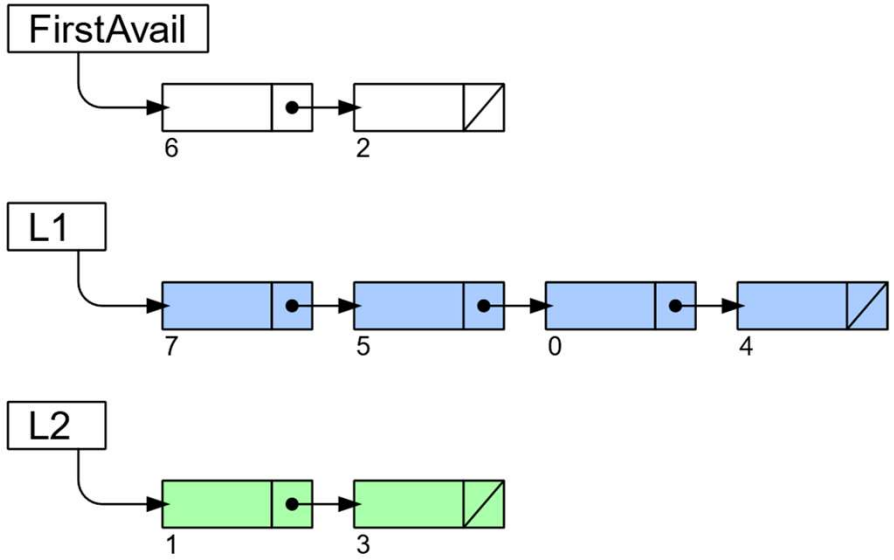
Ilustrasi: pemakaian memori list



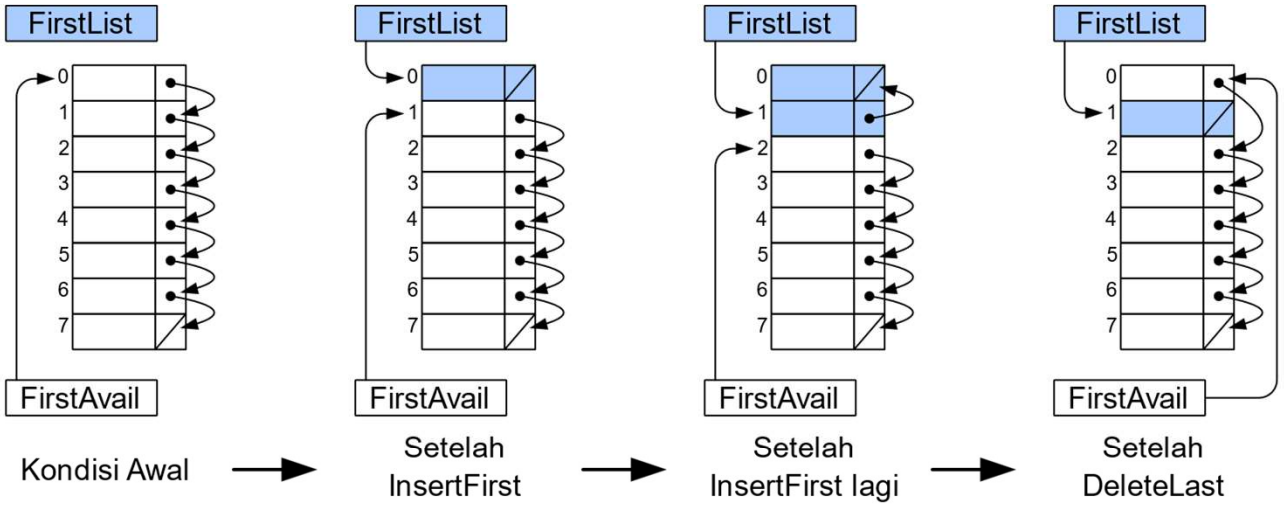
Ilustrasi: array digunakan oleh dua list

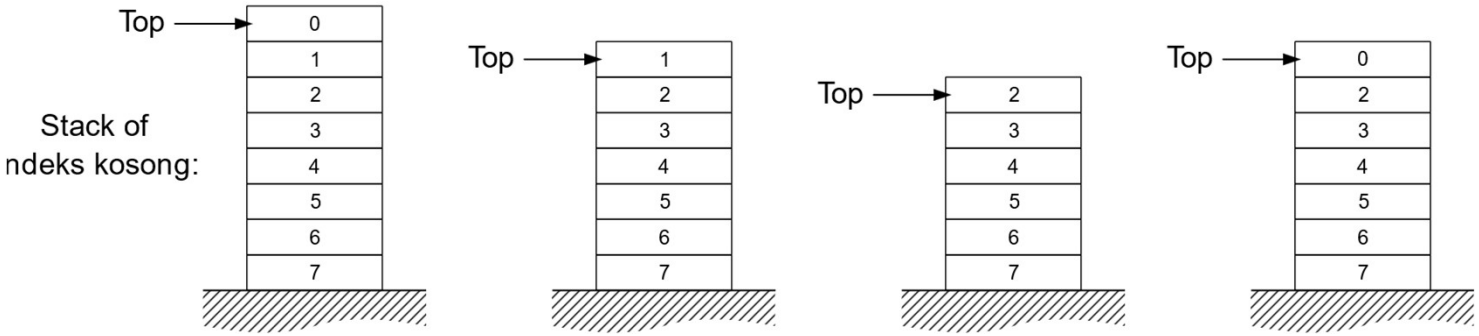
FirstAvail = 6
First L1 = 7
First L2 = 1





Indeks yang kosong membentuk sebuah Stack!





Latihan Soal ADT List dengan Struktur Berkait

IF2110/IF2111 – Algoritma dan Struktur Data
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

Latihan Soal

1. Buatlah fungsi **countPos** yang menghitung banyaknya kemunculan bilangan positif (>0) dari sebuah list of integer l

```
function countPos(l: List) → integer
```

2. Buatlah fungsi **max** yang menghasilkan nilai maksimum dari suatu list of integer l yang tidak kosong

```
function max(l: List) → integer
```

3. Buatlah fungsi **searchPos** yang menghasilkan address di mana nilai positif pertama kali ditemukan di list of integer l

```
function searchPos(l: List) → address
```

Latihan Soal

4. Buatlah prosedur **deleteNeg** yang menghapus semua elemen bernilai negatif (<0) pada sebuah list of integer l. List l boleh kosong dan setiap elemen yang dihapus harus dilakukan dealokasi.

```
procedure deleteNeg(input/output l:List)
```

5. Buatlah prosedur **copyPos** yang menyalin semua elemen bernilai positif (>0) dari sebuah list of integer l1 menjadi l2

```
procedure copyPos(input l1:List, output l2:List)
```

6. Buatlah prosedur **sortedInsert** yang menambahkan sebuah elemen x pada sebuah list of integer l yang terurut menaik

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

Latihan Soal

7. Buatlah prosedur **updateList** yang menerima sebuah infotype x dan y dan sebuah list l dan kemudian mengganti elemen pertama l yang bernilai x dengan y jika x ada di l. Jika x tidak ada di l, l tetap.

```
procedure updateList(input x,y: infotype, input/output l: List)
```