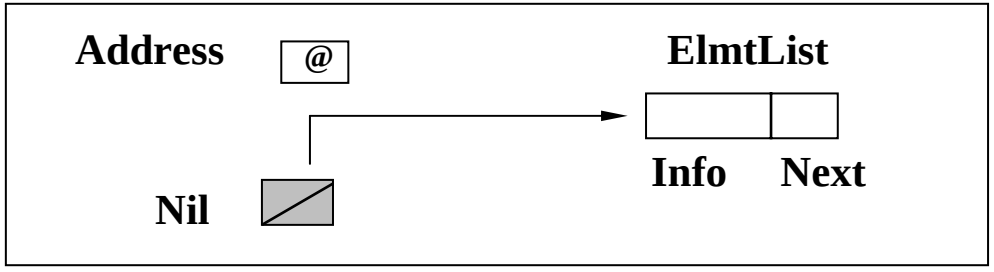
Representasi Fisik List Linier

IF2030/Algoritma dan Struktur Data

Representasi Lojik

- List linier:
  - Sekumpulan elemen ber-type sama yang mempunyai keterurutan tertentu dan setiap elemen terdiri atas 2 bagian:
    - Informasi mengenai elemen (Info)
    - Informasi mengenai alamat elemen suksesor (Next)

```
type ElmtList : < Info : InfoType,
                  Next : address   >
```



Representasi Lojik List Linier

- Jika **L** adalah list, dan **P** adalah address:
  - Alamat elemen pertama list L dapat diacu dengan notasi: **First(L)**
  - Elemen yang diacu oleh P dapat dikonsultasi informasinya dengan notasi **Selektor**:
    - **Info(P)**
    - **Next(P)**

Representasi Fisik

- Implementasi dalam struktur data yang nantinya dapat ditangani oleh pemroses bahasa-bahasa pemrograman
- Tidak semua bahasa dapat mengimplementasi semua struktur fisik yang diuraikan
- Implementasi struktur fisik harus sesuai permasalahan dan ketersediaan bahasa → untuk memperoleh algoritma yang optimal
- Pada tahapan analisis, yang harus dipakai sebagai pegangan adalah struktur lojik
  - mempermudah analisis masalah dan pengembangan algoritma.
- Setelah suatu struktur fisik dipilih, algoritma dalam struktur lojik yang telah dibahas tinggal diterjemahkan ke dalam struktur fisik yang dipilih.

Representasi Fisik
List Linier

- Representasi Berkait
  - Representasi Berkait dengan “Pointer”
  - Representasi Berkait dengan Tabel
- Representasi Kontigu → dengan Tabel

Representasi Berkait Dengan Pointer (1/2)

- Memanfaatkan struktur “pointer” yang disediakan bahasa pemrograman
- Representasi First secara **eksplisit**

```
KAMUS
{ List direpresentasi dengan pointer }
        type InfoType : ... { terdefinisi }
        type ElmtList : < Info : InfoType,
                          Next : address >
        type Address : pointer to ElmtList
        type List : <First : address>
        L : List  { Alamat elemen pertama list }
{ Deklarasi  nama untuk variabel kerja }
        P : address { address untuk traversal }
{ Maka penulisan First(L) menjadi L.First
                  Next(P) menjadi P↑.Next
                  Info(P) menjadi P↑.Info }
```

Representasi Berkait Dengan Pointer (2/2)

Representasi First secara **implisit**

```
KAMUS
{ List direpresentasi dengan pointer }
        type InfoType : ... { terdefinisi }
        type ElmtList : < Info : InfoType,
                          Next : address >
        type Address : pointer to ElmtList
        type List : address
        First : List  { Alamat elemen pertama list }
{ Deklarasi  nama untuk variabel kerja }
        P : address { address untuk traversal }
{ Maka penulisan First(L) menjadi First
                  Next(P) menjadi P↑.Next
                  Info(P) menjadi P↑.Info }
```

Representasi Berkait dg. Tabel (1/5)

- Jika bahasa pemrograman tidak menyediakan struktur pointer, maka kita dapat melakukan implementasi fisik alamat dengan indeks tabel.
- Pengelolaan “memori” yang dilakukan oleh sistem dengan alamat memori yang sebenarnya sekarang diacu melalui nama tabel
  - Kita harus mendefinisikan suatu tabel GLOBAL, yang setiap elemennya adalah elemen list yang diacu oleh alamat

Representasi Berkait dg. Tabel (2/5)

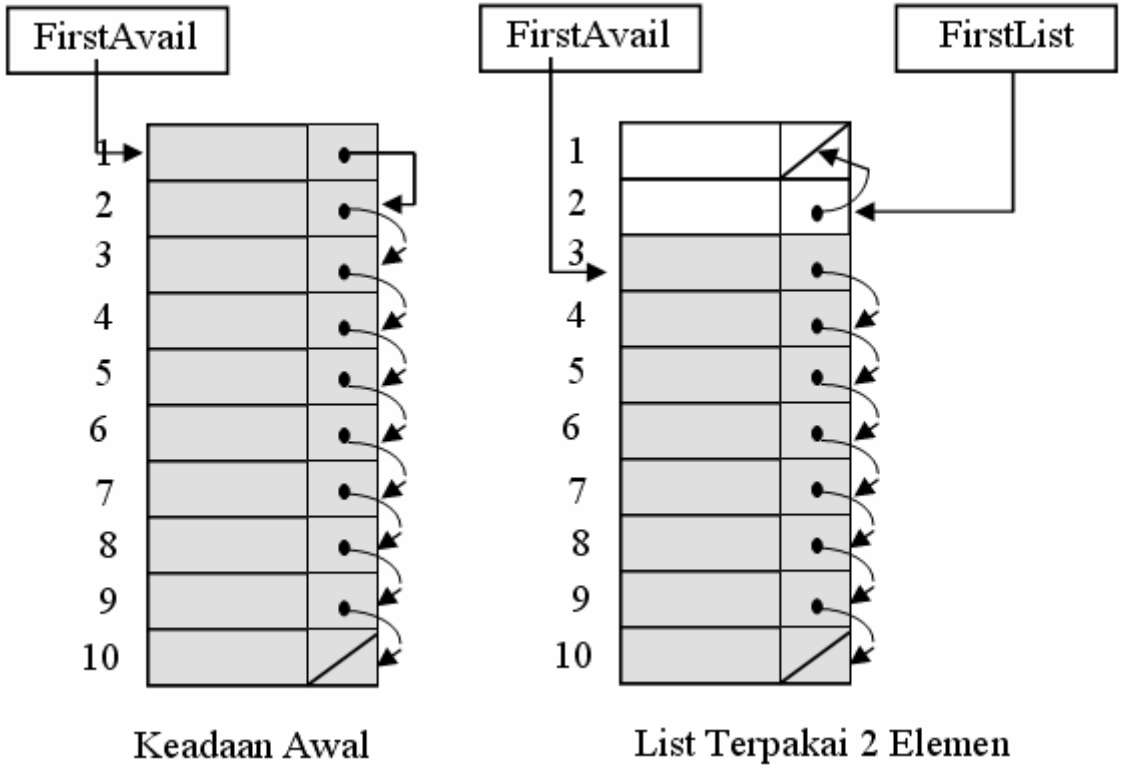
```
KAMUS
{ List direpresentasi secara berkait dengan tabel }
  type InfoType : ... { terdefinisi }
  type ElmtList : < Info : InfoType, Next : Address >
  type Address  : integer [IndexMin..IndexMax, Nil]
{ TABEL MEMORI LIST, GLOBAL}
  constant IndexMin : integer = 1
  constant IndexMax : integer = 100
  constant Nil : integer = 0
  { Nil : address tak terdefinisi,
    di luar [IndexMin..IndexMax] }
  TabElmt : array [IndexMin..IndexMax] of ElmtList
  FirstAvail : Address { alamat pertama tabel memori siap pakai }

  type List : <First : Address>
  L : List
  { Deklarasi  nama untuk variabel kerja }
  P : Address { address untuk traversal }
  { Maka penulisan First(L) menjadi L.First
                   Next(P) menjadi TabElmt(P).Next
                   Info(P) menjadi TabElmt(P).Info }
```

Representasi Berkait dg. Tabel (3/5)

```
function MemFull
{ Mengirim true jika memori list sudah “habis” : FirstAvail=Nil }
procedure InitTab
{ Inisialisasi tabel yang akan dipakai sebagai memori list }
{ I.S. Sembarang. }
{ F.S. TabElmt[IndexMin..IndexMax] siap dipakai sebagai elemen list
  berkait, Elemen pertama yang available adalah FirsAvail=1.
  Next(i)=i+1 untuk i[IndexMin..IndexMax-1], Next(IndexMax)=Nil }
procedure AllocTab (output P : address)
{ Mengambil sebuah elemen siap pakai P  pada awal list FirstAvail }
{ I.S. FirstAvail mungkin kosong. }
{ F.S. Jika  FirstAvail tidak Nil, P adalah  FirstAvail dan
       FirstAvail yang baru adalah  Next(FirstAvail) }
{      Jika FirstAvail =Nil, P=Nil,
       tulis pesan 'Tidak tersedia lagi elemen siap pakai }
procedure DeAllocTab (input P : address)
{ Mengembalikan sebuah elemen  P  pada awal list FirstAvail}
{ I.S. FirstAvail mungkin kosong. P tidak Nil }
{ F.S. FirstAvail = P }
```

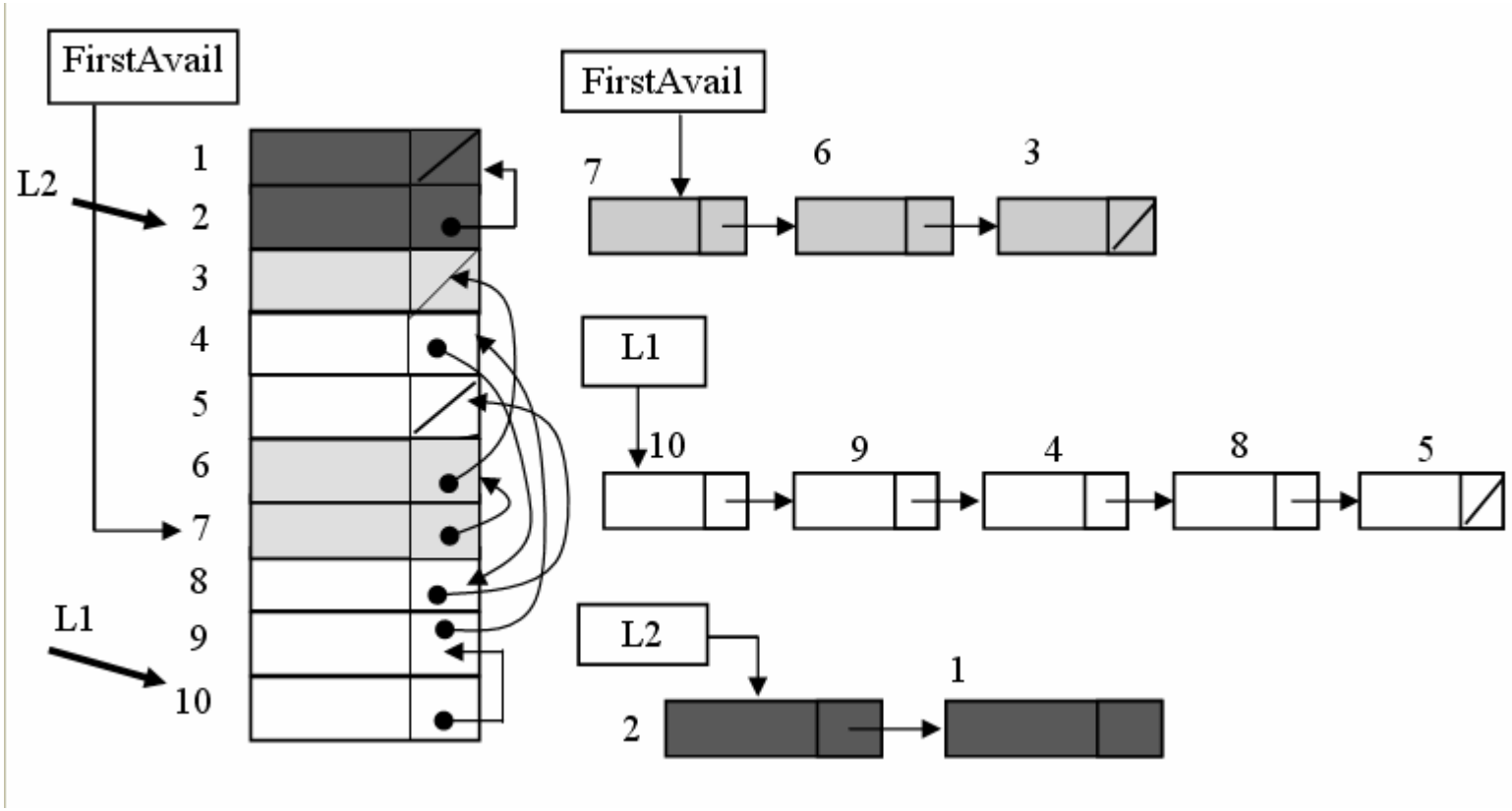
Representasi Berkait dg. Tabel (4/5)

Ilustrasi Pemakaian Memory List



Representasi Berkait dg. Tabel (5/5)

Ilustrasi Pemakaian Memory List



Representasi Fisik secara Kontigu dengan Tabel

- Setiap elemen tabel mengandung informasi info,
- Informasi mengenai Next tidak perlu lagi disimpan secara eksplisit, karena secara implisit sudah tersirat dalam struktur data yang menjadi tempat penyimpanannya → indeks
- Cara untuk mengetahui elemen terakhir adalah dari alamatnya : P=N,
  - N adalah lokasi pada tabel tempat menyimpan elemen terakhir.
- Representasi list bukan murni seperti di atas, tetapi harus mengandung First(L) dan Last(L), seperti yang pernah dibahas pada Queue

```
KAMUS
        constant IndexMin : integer  = 1
        constant IndexMax : integer = 100
        constant Nil : integer = 0
        type InfoType : ...  { Elemen Type : terdefinisi }
        type Info : InfoType { tidak perlu mengandung Next,
                               karena dapat dikalkulasi }
        TabElmtList : array [IndexMin..IndexMax] of Info
        type Address : integer [IndexMin..IndexMax, Nil]

{ Deklarasi  nama untuk variabel  kerja }
        N : Address  { alamat elemen terakhir.
                       Karena field NEXT tidak ada secara
                       eksplisit, maka satu-satunya jalan untuk
                       mengenali elemen terakhir adalah dengan @-nya}
        type List : Address
        First : List
{ Deklarasi alamat }
        P : Address    {address untuk traversal }
{ Maka First(L)..Last(L) adalah indeks efektif elemen tabel
  anggota list
  Next(P) P ← P + 1   Next(P) tidak terdefinisi untuk P = N }
  Info(P) menjadi TabElmtList(P).Info }
```

Mana yang dipilih?

- Pemilihan representasi fisik dari list linier akan sangat mempengaruhi kinerja dari algoritma
- Pada bagian-bagian selanjutnya, akan dipelajari (melalui kasus-kasus), kapan masing-masing representasi cocok untuk dipakai

Ringkasan Representasi Berkait

Representasi lojik Berkait	Representasi fisik berkait dengan Pointer	Representasi fisik berkait dengan Tabel
KAMUS UMUM: L : List P : address {kamus belum terdef.}	KAMUS UMUM: type infotype:{terdef} type address: pointer to Elmt type ElmtList: < info:infotype, Next:address> type List:{rep.terdef} L : List P : address	KAMUS UMUM: type infotype:{terdef} type address:integer type ElmtList: < info:infotype, Next:address > {variabel GLOBAL} constant Nil=0 constant NMAx:address=... TabMem:array[Nil..NMax] of ElmtList FirstAvail:address type List:{rep.terdef} L : List P : address

Ringkasan Representasi Berkait

Representasi lojik Berkait	Representasi fisik berkait dengan Pointer	Representasi fisik berkait dengan Tabel
AKSES: First(L) Next(P) Info(P)	AKSES: tergantung deklarasi P↑.Next P↑.Info	AKSES: tergantung deklarasi TabMem(P).Next TabMem(P).Info
PRIMITIF ALOKASI/DEALOKASI:	PRIMITIF ALOKASI/DEALOKASI: (tidak perlu realisasi, sistem)	PRIMITIF ALOKASI/DEALOKASI: (harus direalisasi) MemFull InitTab AllocTab(P) DeallocTab(P)
Pengenal elemen terakhir	Next(P) = Nil	

Ringkasan
Representasi Kontigu

Representasi lojik kontigu	**Representasi fisik kontigu dengan tabel**
KAMUS UMUM: L : List P : address	KAMUS UMUM: type address : integer type ElmtList : < info : infotype > constant First : address = ... constant Last : address = ... type List : < TabMem : array [First..Last] of ElmtList > L : List P : address
AKSES: First(L) Next(P) Info(P)	AKSES: tergantung deklarasi P ← P + 1 TabELmt(P).Info
PRIMITIF ALOKASI/DEALOKASI:	PRIMITIF ALOKASI/DEALOKASI: { tidak perlu dilakukan, pada saat pendefinisian tabel, secara statis sudah ditentukan. Kecuali jika deklarasi tabel secara dinamis seperti dalam bahasa C }
Pengenal elemen terakhir	Last

Implementasi Bahasa C

- Berkait:
  - Dengan pointer
  - Dengan tabel berkait
- Kontigu (tabel)

- Lihat Diktat halaman 89-92

Representasi Berkait dg. Pointer


```
#define Nil NULL

typedef int infotype;
typedef struct tElmtlist *address;
typedef struct tElmtlist {
     infotype info;
     address next;
} ElmtList;

typedef struct {
     address First;
} List;

/* Selektor */
#define Info(P) (P)->info
#define Next(P) (P)->next
#define First(L) ((L).First)
```

PR

- Modul pra-praktikum bagian prosedural
  - P-10. ADT List Berkait Linier
    - Bagian 1. Representasi Fisik Pointer – Type List dengan First Eksplisit
    - Bagian 2. Representasi Fisik Pointer – Type List dengan First Implisit
    - Bagian 3. Representasi Fisik dengan Tabel Berkait
    - Bagian 4. Representasi Fisik dengan Tabel Kontigu