

Implementasi ADT List dengan Array (elemen kontigu)

IF1210 – Algoritma dan Pemrograman 1
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

Implementasi ADT List dengan Array

Alat dan bahan:

- 1) Sebuah *array* dengan ukuran tertentu.
- 2) Cara untuk mengetahui berapa elemen yang sedang terisi & elemen mana saja yang sudah terisi:
 - alt-1: implisit – beri nilai khusus untuk elemen yang sedang kosong.
 - alt-2: eksplisit – simpan jumlah elemen efektif.

Implisit vs. eksplisit

Implisit (alt-1)

Array kosong harus diinisialisasi memorinya dengan nilai khusus yang disebut *mark*.

Elemen terisi harus kontigu: tidak boleh ada *mark* di antara nilai terisi.

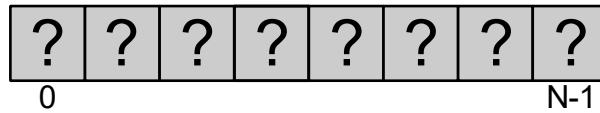
Eksplisit (alt-2)

Semua elemen *array* dapat berisi nilai yang valid namun yang dipedulikan hanya elemen pada indeks-indeks yang efektif ($0..n_{\text{Eff}}-1$).

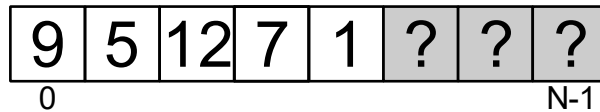
Implisit vs. eksplisit: ilustrasi

Implisit (alt-1)

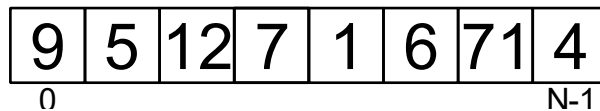
Contoh *array* kosong:



Contoh *array* terisi sebagian:

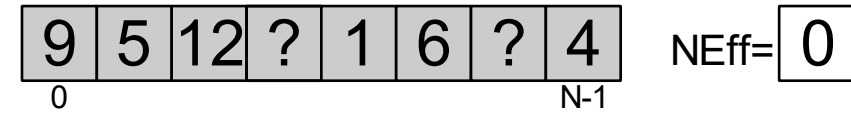


Contoh *array* penuh:

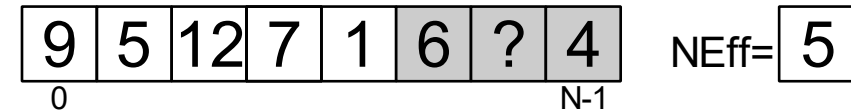


Eksplisit (alt-2)

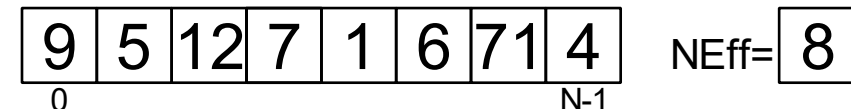
Contoh *array* kosong:



Contoh *array* terisi sebagian:



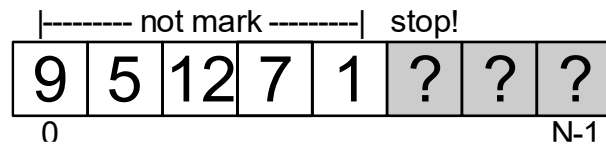
Contoh *array* penuh:



Implisit vs. eksplisit: karakteristik operasi

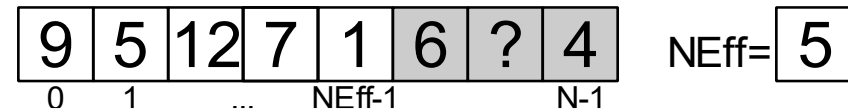
Implisit (alt-1)

Menggunakan pola “*while* belum ketemu *mark*”



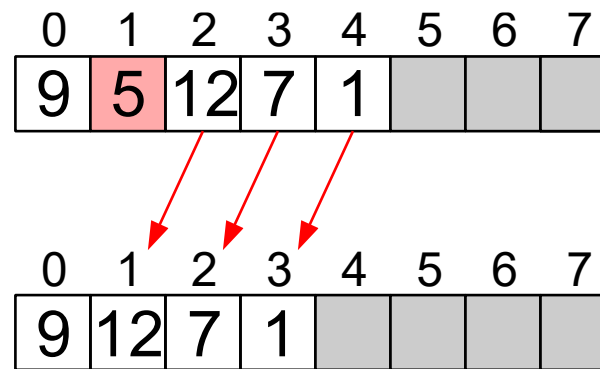
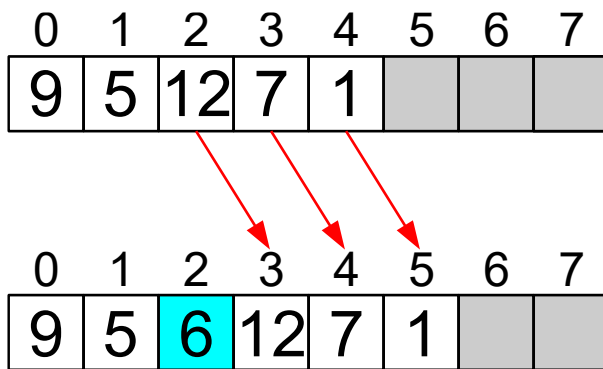
Eksplisit (alt-2)

Menggunakan pola traversal dari indeks 0 sampai $n_{\text{Eff}}-1$



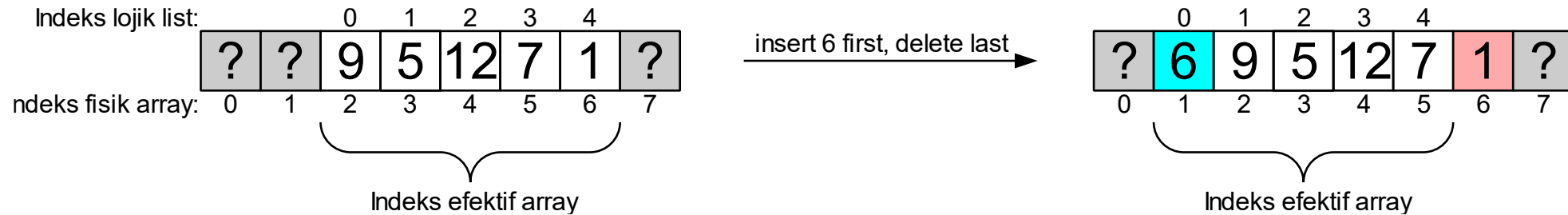
Insert & delete

Membutuhkan penggeseran elemen-elemen setelah posisi penambahan/ penghapusan elemen → *insert first* perlu menggeser nEff buah elemen.



Bagaimana agar lebih efisien?

Ide: elemen tidak harus “rata kiri”



- *Insert first/last* tidak perlu geser elemen jika masih ada ruang kosong di depan/belakang.
- *Delete first/last* tidak perlu geser elemen.

Indeks (fisik) efektif pertama?

- alt-1: skip semua elemen yang bernilai *mark*.
- alt-2: perlu menyimpan informasi indeks pertama.

Ringkasan: 4 alternatif

alt-1a: implisit, rata kiri

alt-2a: eksplisit, rata kiri

alt-1b: implisit, tidak rata kiri

alt-2b: eksplisit, tidak rata kiri

Contoh: rata kiri

(alt-1a, alt-2a)

Contoh: alt-1a (Notasi Algoritmik)

constant CAPACITY: integer = 100

constant IDX_UNDEF: integer = -1

constant MARK: integer = -9999

type ElType: integer { *elemen List* }

type List: < contents: array [0..CAPACITY-1] of ElType > { *penyimpanan elemen List.* }

{ Contoh deklarasi:

l: List

Contoh akses elemen array pada indeks i:

l.contents[i]

}

{ Konstruktor }

procedure CreateList(output l: List)

{ Membentuk List kosong sesuai kapasitas. }

Contoh: alt-1a (Notasi Algoritmik)

constant CAPACITY: integer = 100

constant IDX_UNDEF: integer = -1

constant MARK: integer = -9999

type ElType: integer { *elemen List* }

type List: < contents: array [0..CAPACITY-1] of ElType > { *penyimpanan elemen List.* }

{ *Contoh deklarasi:*

l: List

Contoh akses elemen array pada indeks i:

l.contents[i]

}

{ *Konstruktor* }

procedure CreateList(output l: List)

{ *Membentuk List kosong sesuai kapasitas.* }

Semua elemen diinisialisasi
dengan mark

Contoh: alt-2a (Notasi Algoritmik)

```
constant CAPACITY: integer = 100  
constant IDX_UNDEF: integer = -1
```

```
type ElType: integer { elemen List }  
type List: < contents: array [0..CAPACITY-1] of ElType { penyimpanan elemen List. }  
          nEff: integer  $\geq$  0 { jumlah elemen efektif List. } >
```

{ Contoh deklarasi:

l: List

Contoh akses elemen array pada indeks i:

l.contents[i]

}

{ Konstruktor }

```
procedure CreateList(output l: List)  
  { Membentuk List kosong sesuai kapasitas. }
```

Contoh: alt-2a (Notasi Algoritmik)

```
constant CAPACITY: integer = 100  
constant IDX_UNDEF: integer = -1
```

```
type ElType: integer { elemen List }  
type List: < contents: array [0..CAPACITY-1] of ElType { penyimpanan elemen List. }  
          nEff: integer ≥ 0 { jumlah elemen efektif List. } >
```

{ Contoh deklarasi:

l: List

Contoh akses elemen array pada indeks i:

l.contents[i]

}

{ Konstruktor }

```
procedure CreateList(output l: List)  
  { Membentuk List kosong sesuai kapasitas. }
```

nEff diinisialisasi
dengan 0

Selektor

```
{ Selektor }  
function isEmpty(l: List) → boolean  
  { Prekondisi: l terdefinisi.  
    Memeriksa apakah l kosong. }  
function length(l: List) → integer  
  { Prekondisi: l terdefinisi.  
    Mengirimkan banyaknya elemen efektif l, 0 jika list kosong. }  
function getElmt(l: List, i: integer): ElType  
  { Prekondisi: l tidak kosong, i di antara 0..length(l).  
    Mengirimkan elemen list l yang ke-i (indeks logik). }  
procedure setElmt(output l: List, input i: integer, input v: ElType)  
  { I.S. l tidak kosong, i di antara 0..length(l).  
    F.S. Elemen l yang ke-i bernilai v.  
    Mengeset nilai elemen list yang ke-i sehingga bernilai v. }
```

Selektor

```
{ Selektor }  
function isEmpty(l: List) → boolean  
  { Prekondisi: l terdefinisi.  
    Memeriksa apakah list l kosong. }  
function isEmpty(l: List) → boolean  
  { Prekondisi: l terdefinisi.  
    Menghitung jumlah elemen yang bukan bernilai mark. }  
function isEmpty(l: List) → boolean  
  { Prekondisi: l terdefinisi.  
    Menghitung jumlah elemen yang bukan bernilai mark. }  
function isEmpty(l: List) → boolean  
  { Prekondisi: l tidak kosong, i di antara 0..length(l).  
    Mengirimkan elemen list l yang ke-i (indeks logik). }  
procedure setElmt(output l: List, input i: integer, input v: ElType)  
  { I.S. l tidak kosong, i di antara 0..length(l).  
    F.S. Elemen l yang ke-i bernilai v.  
    Mengeset nilai elemen list yang ke-i sehingga bernilai v. }
```

Operasi-operasi (1/2)

{ Operasi-operasi }

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

{ Prekondisi: l, x terdefinisi.

*Mengembalikan indeks elemen pertama l yang bernilai x (jika ada),
atau mengembalikan IDX_UNDEF jika tidak ada. }*

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

{ I.S. l terdefinisi, mungkin kosong.

F.S. x menjadi elemen pertama l. }

procedure insertAt(input/output l: List, input x: 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 menempa elemen di i). }

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

{ I.S. l terdefinisi, mungkin kosong.

F.S. x menjadi elemen terakhir l. }

Operasi-operasi (2/2)

{ Operasi-operasi }

procedure deleteFirst(input/output l: List, output e: ElType)

{ I.S. l terdefinisi, tidak kosong.

F.S. e diset dengan elemen pertama l, elemen pertama l dihapus dari l. }

procedure deleteAt(input/output l: List, input idx: integer, output e: ElType)

{ I.S. l terdefinisi, tidak kosong, i merupakan indeks yang valid di l.

F.F. e diset dengan elemen l pada indeks ke-idx.

Elemen l pada indeks ke-idx dihapus dari l. }

procedure deleteLast(input/output l: List, output e: ElType)

{ I.S. l terdefinisi, tidak kosong.

F.S. e diset dengan elemen terakhir l, elemen terakhir l dihapus dari l. }

function concat(l1, l2: List) → List

{ Prekondisi: l1 dan l2 terdefinisi, mungkin kosong.

Mengembalikan hasil Konkatenasi ("Menyambung") dua buah List, l2 ditaruh di belakang l1 }

Contoh algoritma: length

```
function length(l: List) → integer
{ ... }
KAMUS LOKAL { untuk alt-1a }
  i: integer
ALGORITMA { untuk alt-1a }
  i ← 0
  while l.contents[i]≠MARK AND i<CAPACITY do
    i ← i+1
  { l.contents[i]=MARK OR i≥CAPACITY }
  → i
```

```
KAMUS LOKAL { untuk alt-2a }
-
ALGORITMA { untuk alt-2a }
  → l.nEff
```

Contoh algoritma: insertAt

```
procedure insertAt(input/output l: List, input x: ElType, input idx: integer)  
{ ... }
```

KAMUS LOKAL

i: integer

ALGORITMA { untuk alt-1a }

if length(l)<CAPACITY then

 i traversal [length(l)..idx+1]

 l.contents[i] = l.contents[i-1]

 l.contents[idx] = x

ALGORITMA { untuk alt-2a }

if length(l)<CAPACITY then

 i traversal [length(l)..idx+1]

 l.contents[i] = l.contents[i-1]

 l.contents[idx] = x

 l.nEff \leftarrow l.nEff+1

**Contoh: tidak rata kiri
(alt-1b, alt-2b)**

Membutuhkan fungsi antara

{ Fungsi antara }

function **firstIdx**(l: List) → integer

{ Prekondisi : l tidak kosong

*Mengirimkan indeks **fisik** elemen pertama }*

function **lastIdx**(l: List) → integer

{ Prekondisi : l tidak kosong

*Mengirimkan indeks **fisik** elemen terakhir}*

function **isIdxValid**(l: List, i: integer) → boolean

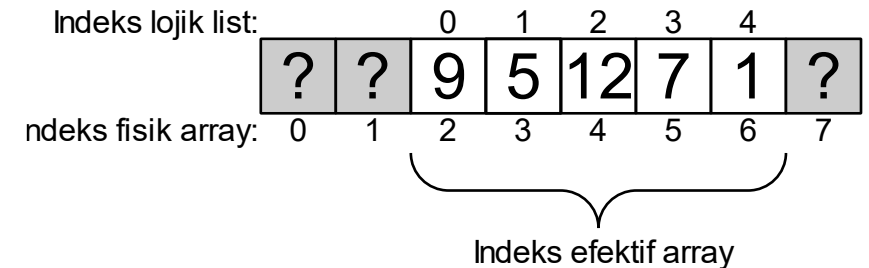
{ Prekondisi: l terdefinisi.

*Mengirimkan true jika i adalah indeks **fisik** yang valid utk kapasitas l, yaitu antara 0..MaxEl-1. }*

function **isIdxEff**(l: List, i: integer) → boolean

{ Prekondisi: l terdefinisi.

*Mengirimkan true jika i adalah indeks **fisik** yang terdefinisi untuk l, yaitu antara firstIdx(l)..lastIdx(l). }*



Membutuhkan fungsi antara

```

{ Fungsi antara }
function firstIdx(l: List) → integer
{ Prekondisi: l tidak kosong
  Mengirimkan true jika i adalah indeks fisik yang valid utk kapasitas l,
  yaitu antara 0..MaxEl-1. }
function lastIdx(l: List) → integer
{ Prekondisi: l tidak kosong
  Mengirimkan true jika i adalah indeks fisik yang terdefinisi untuk l,
  yaitu antara firstIdx(l)..lastIdx(l). }
function isIdxEff(l: List, i: integer) → boolean
{ Prekondisi: l terdefinisi.
  Mengirimkan true jika i adalah indeks fisik yang valid utk kapasitas l,
  yaitu antara 0..MaxEl-1. }
function isIdxEff(l: List, i: integer) → boolean
{ Prekondisi: l terdefinisi.
  Mengirimkan true jika i adalah indeks fisik yang terdefinisi untuk l,
  yaitu antara firstIdx(l)..lastIdx(l). }

```

