Algoritma Pemrograman dan Struktur Data

Materi 6: ARRAY dan LINKED LIST

Dosen pengampu:
Suamanda Ika Novichasari, M.Kom.
Imam Adi Nata, M.Kom

PROGRAM STUDI S1 TEKNOLOGI INFORMASI
FAKULTAS TEKNIK
UNIVERSITAS TIDAR

Jl. Kapten Suparman No.39, Tuguran,
Potrobangsan, Kec. Magelang Utara, Kota
Magelang, Jawa Tengah 56116

Learning Objective

Mahasiswa mampu menerapkan Array untuk penyelesaian masalah

Mahasiswa mampu menerapkan Linked List untuk penyelesaian masalah

Course Material

Memory Komputer

Array

Linked List

Pre Test 10 Menit

1. Jelaskan cara kerja memori dalam komputer !

2. Jelaskan perbedaan array dan linked list !

BAB MATERI

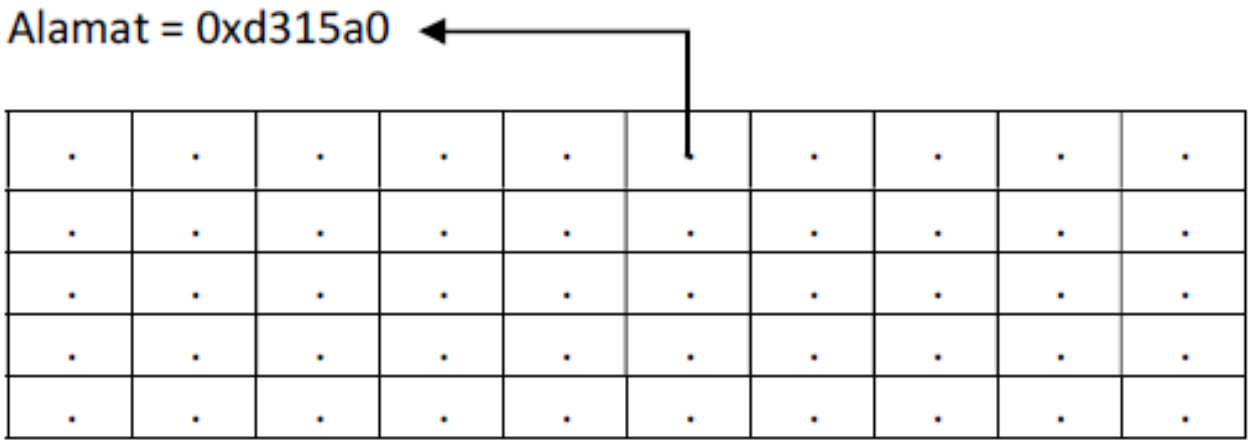
Memori Komputer

Subbab ini mempelajari tentang Cara Kerja Memori komputer

Cara Kerja *Memori* Pada Komputer*?*

- **Memori komputer** diibaratkan seperti lemari besar dengan banyak laci yang masing-masing memiliki alamat tersendiri.
- **Alamat memori** menggunakan format bilangan **hexadecimal.**
- Ketika akan menyimpan sebuah data, komputer akan memberikan alamat slot memori yang bisa digunakan.

Cara Kerja *Memori* Pada Komputer*?*



- Ketika dibutuhkan untuk menyimpan data dalam **jumlah banyak** bisa menggunakan **array** dan **list**.

BAB MATERI

Array

Subbab ini mempelajari tentang Array

Apa itu *Array?*

- **Array** dapat diartikan sebagai sekumpulan pasangan **indeks** dan **nilai**.
- Nilai dalam elemen array memiliki **tipe data** yang **sama**.
- **Indeks** array merepresentasikan **alamat memori** elemen array.
- Indeks elemen array **dimulai** dari **0 (nol)**.

10	7	5	- - -> Nilai Element Array
A[0]	A[1]	A[2]	- - -> Alamat Element Array
0	1	2	- - -> Indeks Element Array

Memory *Array?*

- Saat mendeklarasikan sebuah array, komputer akan memberikan **slot** lokasi memori yang **berurutan** atau **berdampingan** antar elemen satu dengan yang lainnya.
- Sehingga **jumlah** slot memori atau elemen array bersifat **tetap** sesuai jumlah saat pendeklarasian.

02

Pendeklarasian Array

- **Bentuk umum** dari pendeklarasian array sebagai berikut :
  - `tipe_data<spasi>nama_array[jml_elemen]`
- Misalkan kita ingin mendeklarasikan sebuah array dengan nama umur yang memiliki 20 elemen dengan tipe data integer, maka bentuk deklarasinya adalah sebagai berikut :
  - `int umur[20];`

02

Akses Elemen Array

- Untuk mengakses elemen, kita harus menuliskan indeks nya. Misalkan kita ingin mengambil nilai yang terdapat pada elemen ke-5 (array yang bernama umur) dan menampung nilai tersebut dalam sebuah variabel yang bertipe integer (misalkan A), maka penulisan kode nya seperti berikut:
  - `A = umur[5];`

02

Array dari Karakter

- Sebelumnya sudah diketahui bahwa sekumpulan karakter disebut string (teks). Untuk mendeklarasikan array dari tipe karakter menggunakan bentuk umum seperti dibawah ini :
  - `char<spasi>nama)array[jumlah_elemen];`
- Contoh :
  - `char huruf[5];`
  - `char nama[6][20];` //array 2 dimensi

02

Array MultiDimensi

- Sama seperti array satu dimensi, pada array multidimensi juga dapat dilakukan inisialisasi nilai dalam elemen-elemennya.
- Array 2 dimensi merupakan array yang mempunyai **baris** dan **kolom**.
- Bentuk umum sebagai berikut :
- `tipe_data<spasi>nama_array[jml_baris][jml_kolom]`
- Contoh :
  - `int A[3][3] = {1,2,3,4,5,6,7,8,9};`
  - `int A[3][3] = { {1,2,3} , {4,5,6} , {7,8,9} };`

Tambah data pada *Array*?

- Pada array proses penambahan data **dapat dilakukan secara langsung** dengan mengakses indeks elemen yang akan di isi atau diubah nilainya. Misalkan terdapat array A dengan 5 elemen, A = {“M”, “A”, “D”} dan akan dilakukan penambahan data pada indeks ke-3 dengan nilai “A”. Sehingga A = {“M”, “A”, “D”, “A”}. Notasi algoritmanya dapat dilakukan seperti berikut :
  - `A[3] ← “A”;`
- Jika ingin menyisipkan “N” pada indeks ke-2, maka nilai di indeks ke-2 sampai ke-3 di pindah ke indeks setelahnya. Sehingga kemudian menjadi A = {“M”, “A”, “N”, “D”, “A”}.
  - `A[4] ← A[3];`
  - `A[3] ← A[2];`

Hapus data pada *Array*?

- Masih dengan menggunakan contoh array sebelumnya, jika ingin menghapus nilai pada indeks ke-2, sehingga kemudian menjadi A = {“M”, “A”, “D”, “A”}.
- Maka proses penghapusan nilai hanya dengan **mengubah nilai** pada indeks ke-2 dengan **Null** seperti berikut :
  - `A[2] ← “”;`

BAB MATERI

Linked List

Subbab ini mempelajari konsep big O notation

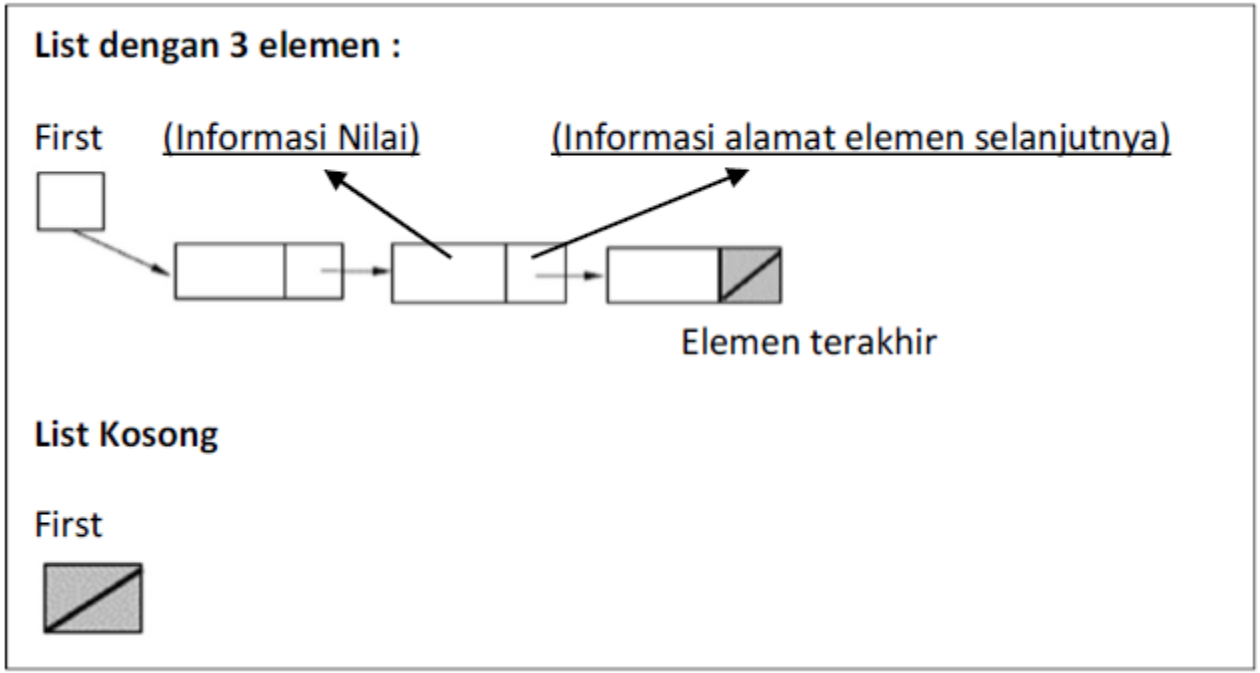
Studi Kasus..

- Misalkan sudah mendeklarasikan array untuk 10 elemen namun ternyata hanya 5 elemen yang terisi. Sisa elemen array yang sudah dideklarasikan akan **terbuang sia-sia** karena tidak akan bisa digunakan untuk data lain karena sudah dipesan oleh array dengan 10 elemen.
- Atau jika sudah mendeklarasikan array untuk 10 elemen namun ternyata yang dibutuhkan adalah 20 elemen. Maka harus dilakukan **pendeklarasian ulang** dengan menambah elemen array menjadi 20 elemen. Hal tersebut tidak efisien.
- Untuk mengatasi permasalahan tersebut dapat menggunakan **linked list**.

Apa itu Linked List?

- **Linked list** adalah sekumpulan data yang **bertipe sama** yang **saling terhubung** satu sama lain.
- 1 elemen linked list terdiri dari **2 bagian** yaitu informasi **nilai** dan informasi **alamat list berikutnya**.

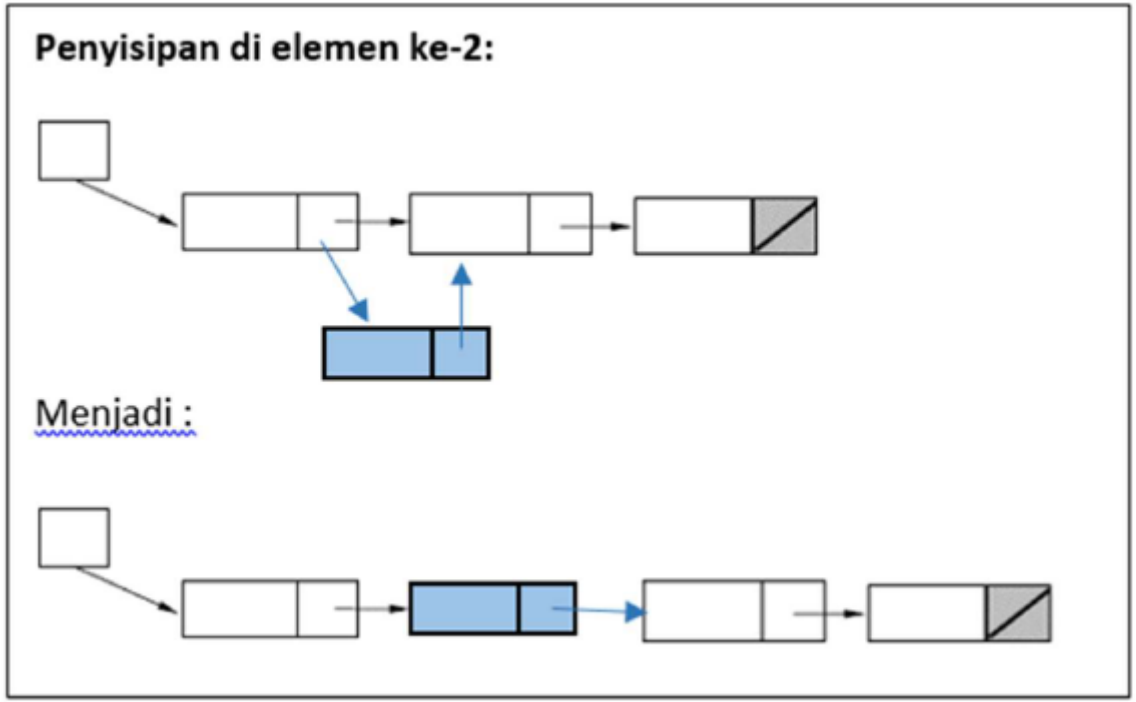
Ilustrasi ..


List dengan 3 elemen :
First
(Informasi Nilai)
(Informasi alamat elemen selanjutnya)
Elemen terakhir
List Kosong
First

Tambah data pada *Linked List*?

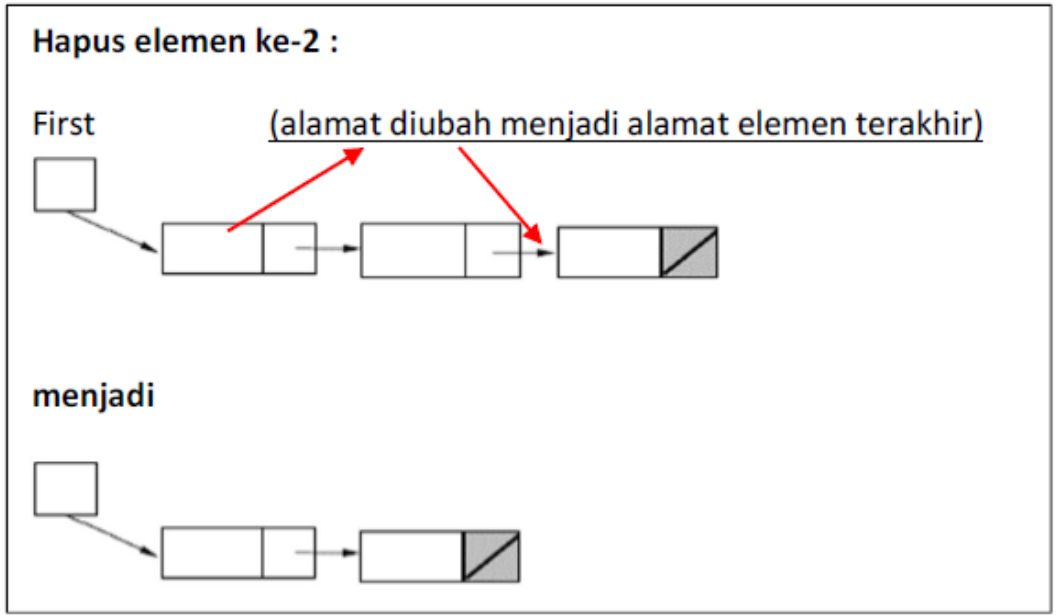
- Proses penyisipan data di tengah dapat dilakukan dengan lebih baik dengan hanya mengubah alamat dari elemen yang disisipi seperti ilustrasi berikut.
- Proses penyisipan hanya dapat dilakukan jika **diketahui** alamat elemen yang akan disisipi.
- Sebelum proses penyisipan nilai, dilakukan **alokasi alamat** (pencarian slot memori yang kosong untuk menyimpan data baru) terlebih dahulu.
- **Penyisipan** akan **berhasil** jika **alokasi** alamat **berhasil**.

Tambah data pada *Linked List*?



Hapus data pada *Array*?

- Sama seperti penyisipan, proses hapus data pada linked list lebih mudah dari array, hanya dengan mengubah alamat dari elemen sebelumnya.
- Proses penyisipan dapat gagal jika sudah tidak ada memori untuk disisipi, namun proses hapus data pasti akan selalu berhasil



Definisi Fungsional *Linked List*

Diberikan L, L1 dan L2 adalah list linier dengan elemen ElmtList:

```
IsEmptyList : L → boolean      { Tes apakah list kosong }
CreateList  : → L              { Membentuk sebuah list linier
                                 kosong }
Insert      : ElmtList x L → L { Menyisipkan sebuah elemen
                                 ke dalam list }
Delete      : L → L x ElmtList { Menghapus sebuah elemen
                                 list }
UpdateList  : ElmtList x L → L { Mengubah informasi sebuah
                                 elemen list linier }
Concat      : L1 x L2 → L      { Menyambung L1 dengan L2 }
```

Operasi Primitif Linked List

List Kosong	Penyisipan	Penghapusan	Penggabungan
• Cek List Kosong • Pembuatan List Kosong	• Insert First • Insert After • Insert Last	• Delete First • Delete After • DeleteP • Delete Last	• Konkatenasi • Update List

List Kosong

```
function IsEmptyList (L : List) → boolean
{ Tes apakah sebuah list L kosong. Mengirimkan true jika
  list kosong, false jika tidak kosong }
KAMUS LOKAL
ALGORITMA
  → (First(L) = Nil)
```

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

Insert First *Linked List*

Insert First jika dikatahui alamatnya

```
procedure InsertFirst (input/output L : List, input P : address)
{ I.S. List L mungkin kosong, P sudah dialokasi, P ≠ Nil,
   Next(P)=Nil }
{ F.S. P adalah elemen pertama list L }
{ Insert sebuah elemen beralamat P sebagai elemen pertama list
   linier L yang mungkin kosong }
KAMUS LOKAL
ALGORITMA
  Next(P) ← First(L)
  First(L) ← P
```

Insert First *Linked List*

Insert First jika dikatahui nilainya

```
procedure InsFirst (input/output L : List, input InfoE : InfoType)
{ I.S. List L mungkin kosong }
{ F.S. Sebuah elemen dialokasi dan menjadi elemen pertama list L, jika
   alokasi berhasil. Jika alokasi gagal list tetap seperti semula. }
{ Insert sebuah elemen sbg. elemen pertama list linier L yang mungkin
   kosong. }
KAMUS LOKAL
  function Alokasi (X : infotype) → address
  { Menghasilkan address yang dialokasi. Jika alokasi berhasil,
  Info(P)=InfoE, dan Next(P)=Nil. Jika alokasi gagal, P=Nil }
  P : address
ALGORITMA
  P ← Alokasi(InfoE)
  if P ≠ Nil then
    Next(P) ← First(L); First(L) ← P
```

Insert After *Linked List*

```
procedure InsertAfter (input P, Prec : address)
{ I.S. Prec adalah elemen list, Prec ≠ Nil, P sudah dialokasi, P
   ≠ Nil, Next(P) = Nil }
{ F.S. P menjadi suksesor Prec }
{ Insert sebuah elemen beralamat P pada List Linier L }
KAMUS LOKAL
ALGORITMA
  Next(P) ← Next(Prec)
  Next(Prec) ← P
```

Insert Last *Linked List*

Insert Last jika dikatahui alamatnya

```
procedure InsertLast (input/output L : List, input P : address)
{ I.S. List L mungkin kosong, P sudah dialokasi, P ≠ Nil, Next(P)=Nil }
{ F.S. P adalah elemen trerakhir list L }
{ Insert sebuah elemen beralamat P sbg elemen terakhir dari list linier L
   yg mungkin kosong }
KAMUS LOKAL
  Last : address { address untuk traversal,
  pada akhirnya address elemen terakhir }
ALGORITMA
  if First(L) = Nil then { insert sebagai elemen pertama }
    InsertFirst (L,P)
  else
    { Traversal list sampai address terakhir }
    { Bagaimana menghindari traversal list untuk mencapai Last? }
    Last ← First(L)
    while (Next(Last) ≠ Nil) do
      Last ← Next(Last)
    { Next(Last) = Nil, Last adalah elemen terakhir }
    { Insert P after Last }
    InsertAfter(P,Last)
```

Insert Last *Linked List*

Insert Last jika dikatahui nilainya

```
procedure InsLast (input/output L : List, input InfoE : InfoType)
{ I.S. List L mungkin kosong }
{ F.S. Jika alokasi berhasil, InfoE adalah nilai elemen terakhir
   L }
{ Jika alokasi gagal, maka F.S. = I.S. }
{ Insert sebuah elemen beralamat P (jika alokasi berhasil)
   sebagai elemen terakhir dari list linier L yg mungkin kosong }
KAMUS LOKAL
  function Alokasi(X : InfoType) → address
  { Menghasilkan address yang dialokasi. Jika alokasi berhasil,
  Info(P)=InfoE, dan Next(P)= Nil. Jika alokasi gagal, P=Nil }
  P : address
ALGORITMA
  P ← Alokasi(InfoE)
  if P ≠ Nil then { insert sebagai elemen pertama }
    InsertLast(L,P)
```

Delete First *Linked List*

```
procedure DeleteFirst (input/output L : List, output P : address)
{ I.S. List L tidak kosong, minimal 1 elemen, elemen pertama
   pasti ada }
{ F.S. First "maju", mungkin bernilai Nil (list menjadi kosong) }
{ Menghapus elemen pertama L, P adalah @ elemen pertama L sebelum
   penghapusan, L yang baru adalah Next(L) }
KAMUS LOKAL
ALGORITMA
  P ← First(L)
  First(L) ← Next(First(L))
  Next(P) ← Nil
  { Perhatikan bahwa tetap benar jika list menjadi kosong }
```

Delete First *Linked List*

```
procedure DeleteFirst (input/output L : List, output E :
   InfoType)
{ I.S. List L tidak kosong, minimal 1 elemen, elemen pertama
   pasti ada }
{ F.S : Menghapus elemen pertama L, E adalah nilai elemen
   pertama L sebelum penghapusan, L yang baru adalah Next(L)
   }
KAMUS LOKAL
  procedure DeAlokasi (input P : address)
  { I.S. P pernah dialokasi. F.S. P=Nil }
  { F.S. Mengembalikan address yang pernah dialokasi. P=Nil}
  P : address
ALGORITMA
  P ← First(L); E ← Info(P)
  First(L) ← Next(First(L)) { List kosong : First(L)
   menjadi Nil }
  Next(P) ← Nil; Dealokasi(P)
```

Delete After *Linked List*

```
procedure DeleteAfter (input Prec : address, output P : address)
{ I.S. List tidak kosong, Prec adalah elemen list, Next(Prec)
  ≠ Nil }
{ F.S. Next(Prec), yaitu elemen beralamat P dihapus dari List.
Next(P)=Nil }
{ Menghapus suksesor Prec, P adalah @ suksesor Prec sebelum
  penghapusan, Next(Prec) yang baru adalah suksesor dari
  suksesor Prec sebelum penghapusan }
KAMUS LOKAL
ALGORITMA
  P ← Next(Prec)
  Next(Prec) ← Next(Next(Prec))
  Next(P) ← Nil
```

DeleteP *Linked List*

```
procedure DeleteP (input/output L : List, input P : address)
{ I.S. List L tidak kosong, P adalah elemen list L }
{ F.S. Menghapus P dari list L, P mungkin elemen pertama,
   "tengah", atau terakhir }
KAMUS LOKAL
  Prec : address { alamat predesesor P }
ALGORITMA
  { Cari predesesor P }
  if (P = First(L)) then { Delete list dengan satu elemen }
    DeleteFirst(L,P)
  else
    Prec ← First(L)
    while (Next(Prec) ≠ P) do
      Prec ← Next(Prec)
    { Next(Prec) = P, hapus P }
    DeleteAfter(Prec,P)
```

DeleteP *Linked List*

```
procedure DeleteLast (input First : List, output P : address)
{ I.S. List L tidak kosong, minimal mengandung 1 elemen }
{ F.S. P berisi alamat elemen yang dihapus. Next(P)=Nil. List berkurang
elemennya }
{ Menghapus elemen terakhir dari list L, list mungkin menjadi kosong }
KAMUS LOKAL
  Last, PrecLast : address { address untuk traversal, type terdefinisi
  pada akhirnya Last adalah alamat elementerakhir dan PrecLast adalah
    alamat sebelum yg terakhir }
ALGORITMA
  { Find Last dan address sebelum Last }
  Last ← First(L)
  PrecLast ← Nil { predesesor dari L tak terdefinisi }
  while (Next(Last) ≠ Nil) do
  { Traversal list sampai @ terakhir }
    PrecLast ← Last
    Last ← Next(Last)
  { Next(Last) = Nil, Last adalah elemen terakhir }
  { PrecLast = sebelum Last }
  P ← Last
  if (PrecLast = Nil) then { list dengan 1 elemen, jadi kosong }
    First(L) ← Nil
  else
    Next(PrecLast) ← Nil
```

Konkatenasi *Linked List*

```
procedure CONCAT (input L1, L2 : List, output L3 : List)
{ I.S. L1 ≠ L2, L1 ≠ L3, dan L3 ≠ L2; L1, L2 mungkin kosong }
{ F.S. L3 adalah hasil Konkatenasi ("Menyambung") dua buah list linier, L2
ditaruh di belakang L1 }
KAMUS LOKAL
  Last1 : address { alamat elemen terakhir list pertama }
ALGORITMA
  CreateList(L3) { inisialisasi list hasil }
  if First(L1) = Nil then
    First(L3) ← First(L2)
  else { Traversal list 1 sampai address terakhir, hubungkan Last1 dengan
   First(L2) }
    First(L3) ← First(L1)
    Last1 ← First(L1)
    while (Next(Last1) ≠ Nil) do
      Last1 ← Next(Last1)
      { Next(Last1) = Nil, Last adalah elemen terakhir }
      Next(Last1) ← First(L2)
```

Perbedaan Array dan Linked List

Perbedaan	Array	Linked List
Jumlah elemen	Tetap, diatur pada awal pembentukan	Berubah sesuai kebutuhan
Cara akses	Akses acak	Akses urut
Elemen	Indeks dan nilai (value)	Nilai (value) dan alamat memori elemen selanjutnya
Komplesitas waktu baca data	O(1)	O(n)
Komplesitas waktu penyisipan data	O(n)	O(1)
Komplesitas waktu hapus data	O(n)	O(1)

Rangkuman

Memori komputer seperti sekumpulan laci raksasa yang masing-masing laci memiliki alamat tersendiri dengan format hexadecimal.

Untuk menyimpan banyak elemen digunakan array atau *linked list*.

Array akan menyimpan semua elemen tepat di samping satu sama lain secara berurutan.

Linked list menyimpan elemen yang tersebar di mana-mana, dan satu elemen menyimpan alamat elemen berikutnya.

Semua elemen dalam array harus bertipe sama (semua int, semua ganda, dan seterusnya).

Array mengakses data secara acak sehingga memungkinkan membaca data dengan cepat.

Linked list mengakses data secara berurutan dari awal sampai akhir.

Linked list memungkinkan penyisipan dan penghapusan dengan cepat.

Tugas !

Buatlah pseudocode dari algoritma *binary search,* dan *simple search* dengan menggunakan array atau *linked list* !

TERIMA KASIH

#NEXT... Rekursif