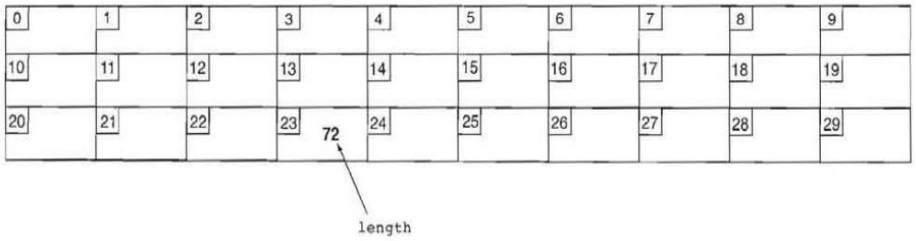
BAB V

STRUKTUR DATA LINEAR

Struktur data linear adalah struktur data yang elemen-elemennya tersusun berurutan sebelah-menyebelah. Secara fisik dapat dilihat seperti deretan elemen atau rantai elemen (*chain*). Hubungan antar elemennya adalah berupa *predecessor* (elemen sebelumnya) atau *successor* (elemen sesudahnya). Elemen yang pertama tidak mempunyai *predecessor*, elemen yang terakhir tidak mempunyai *successor*, dan elemen-elemen diantaranya mempunyai satu *predecessor* dan satu *successor*. Macam-macam struktur data yang termasuk dalam struktur data linear adalah array, list, stack dan queue.

5.1. Konsep Penyimpanan Struktur Data

Sebelum membahas struktur data linear lebih lanjut, akan dibahas lebih dulu bagaimana komputer menyimpan suatu struktur data di dalam memorinya. Seperti telah diketahui, setiap data yang akan diproses akan disimpan di suatu tempat dalam memori komputer. Tempat tersebut mempunyai alamat yang berupa nomor-nomor seperti pada kotak pos. Dengan demikian memori komputer dapat dianalogikan sebagai deretan kotak-kotak untuk menyimpan sebuah data dan setiap kotak tersebut mempunyai nomor sebagai alamatnya. Untuk mengambil data dari suatu kotak, programmer tidak perlu mengetahui nomor kotaknya, karena kotak yang menyimpan data tersebut sudah mempunyai nama simbolik yaitu nama variabel. Programmer hanya perlu mendefinisikan nama variabel beserta tipe datanya, lalu komputer akan menentukan sendiri "kotak" tempat variabel itu atau dengan kata lain alamat dari variabel tersebut di dalam memori. Sebagai contoh, programmer mendefinisikan sebuah variabel length, bertipe interger lalu memberi nilai 72 ke variabel tersebut. Ilustrasi penyimpanan variabel length di memori komputer dapat dilihat pada gambar 5.1 di halaman berikut. Dalam gambar, dapat dilihat kotak nomor 23 diberi nama length. Nama inilah yang akan digunakan programmer untuk mengakses data atau menyimpan data di kotak tersebut.



Gambar 5.1 Ilustrasi penyimpanan varibel length di memori komputer

Ada 2 metode untuk memperoleh alamat dari sebuah elemen dari suatu struktur data, yaitu:

a. metode penghitungan alamat (*computed-address method*): alamat suatu elemen diperoleh dengan menggunakan fungsi pengalamatan (*addressing function*). Fungsi ini digunakan untuk menghitung alamat elemen dalam array atau struct.
b. metode pointer (*link-address* atau *pointer-address method*): alamat suatu elemen diperoleh dengan "melihat" nilai pointer yang menunjuk elemen tersebut. Metode ini digunakan untuk menyimpan elemen-elemen list.

5.1.1 Metode penghitungan alamat

Seperti telah dijelaskan sebelumnya, elemen-elemen array disimpan secara berurutan (sekuensial) dalam memori komputer, karena itu metode ini juga disebut metode alokasi sekuensial Untuk array linier (atau vektor), sejumlah lokasi memori yang berurutan dialokasikan untuk menyimpan elemen-elemen array ketika array didefinisikan. Untuk mengetahui alamat dari suatu elemen dalam array digunakan fungsi pengalamatan (*adressing function*), lihat gambar 5.2.

Fungsi pengalamatan untuk mengetahui posisi dari elemen yang ke i dalam vektor A dapat dirumuskan menjadi:

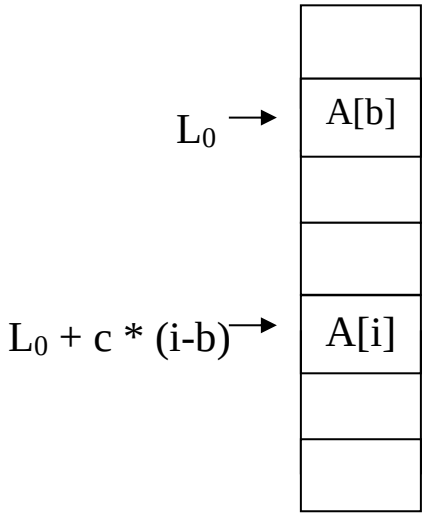
$$Loc(A[i]) = L_0 + c * (i-b) \qquad (5.1)$$

dengan:

L_0 = alamat elemen array A yang pertama

c = banyaknya byte yang dibutuhkan untuk menyimpan sebuah elemen array

b = batas bawah array (indeks dari elemen pertama)



Gambar 5.2 Ilustrasi penyimpanan vektor A dalam memori

Banyaknya byte yang diperlukan untuk menyimpan sebuah elemen, tergantung dari tipe data elemen array serta bahasa pemrograman yang digunakan. Untuk algoritma ini, banyaknya byte untuk menyimpan tiap tipe data dapat dilihat pada tabel 5.1 di bawah ini.

Tabel 5.1 Contoh banyaknya byte dan range dari tipe data atomik

Tipe data	Banyaknya byte	Range
short	2	-32768 … 32767
int	2	-32768 … 32767
long	4	-2147482684 … 2147482683
unsigned short	2	0 … 65535
unsigned int	2	0 … 65535
unsigned long	4	0 … 429496793
float/real	4	3.4×10^{-38} … 3.4×10^{38}
double	8	1.7×10^{-308} … 1.7×10^{308}
char	1	a … z, A..Z, dan lain-lain

Untuk array 2 dimensi atau matriks, elemen-elemennya juga disimpan secara linier baris perbaris (*row major order*) atau kolom perkolom (*column major order*). Untuk array A yang terdiri dari n baris dan m kolom maka posisi dari elemen array pada baris ke i dan kolom ke j dapat dirumuskan sebagai berikut:

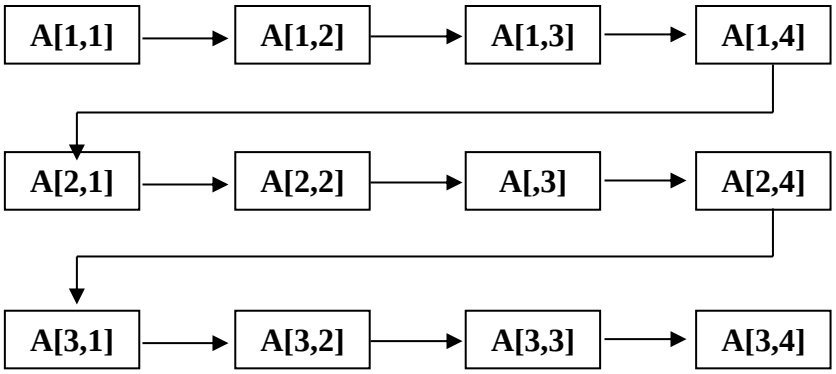
a. Penyimpanan matriks A[n,m] secara **Row major:**

$$Loc(A[i,j]) = L_0 + ((i-b_1)*r_2 + (j-b_2)) * c \qquad (5.2)$$

dengan:
L_0 = alamat elemen A yang pertama ($A[b_1,b_2]$)
c = banyaknya byte yang dibutuhkan untuk menyimpan sebuah elemen array
$r_2 = u_2 - b_2 + 1$ (= range kolom)
b_1 = batas bawah baris
b_2 = batas bawah kolom
u_1 = batas atas baris
u_2 = batas atas kolom

Ilustrasi penyimpanan matriks A[1..3,1..4] secara *row major* dapat dilihat pada gambar 5.3 di bawah ini.

```
A[1,1]
A[1,2]
A[1,3]
A[1,4]
A[2,1]
A[2,2]
A[2,3]
A[2,4]
A[3,1]
A[3,2]
A[3,3]
A[3,4]
```



Gambar 5.3 Penyimpanan matriks A 3x4 secara row major

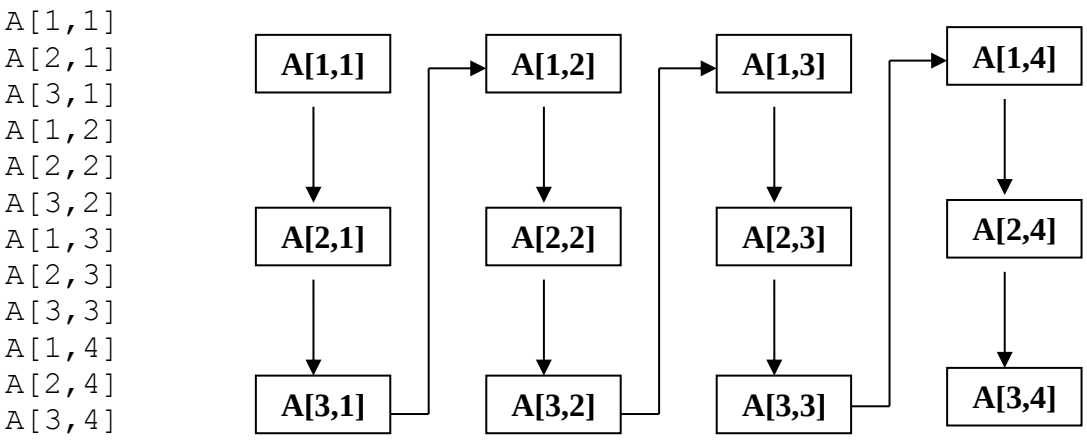
b. Penyimpanan matriks A[m,n] secara **Column major:**

$$Loc(A[i,j]) = L_0 + ((j-b_2)*r_1 + (i-b_1)) * c \qquad (5.3)$$

dengan:

L_0 = alamat elemen matriks A yang pertama ($A[b_1,b_2]$)
c = banyaknya byte yang dibutuhkan untuk menyimpan sebuah elemen array
$r_1 = u_1 - b_1 + 1$ (= range baris)
b_1 = batas bawah baris
b_2 = batas bawah kolom
u_1 = batas atas baris
u_2 = batas atas kolom

Ilustrasi penyimpanan matriks A[1..3,1..4] secara column major dapat dilihat pada gambar 5.4 di halaman berikut.



Gambar 5.4 Penyimpanan matriks A 3x4 secara column major

Untuk array multi dimensi lainnya: alamat dari elemen A[s1, s2, …, sn] yang disusun secara *row major* dengan batas bawah (b) dan batas atas (u) dengan $b_i \le s_i \le u_i$ untuk $1 \le i \le n$ maka lokasi elemen A[s1, s2, …, sn] adalah:

$$\text{Loc}(A[s_1, s_2, \ldots, s_n]) = L_0 + \sum_{1 \le i \le n} p_i (s_i - b_i) * c \qquad (554)$$

dengan $p_i = \prod_{i < j \le n} (u_j - b_j + 1)$

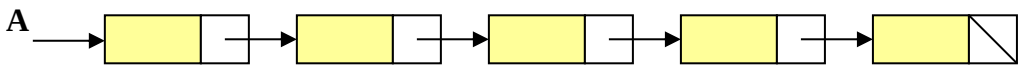
Dengan cara yang sama didapat *addressing function* untuk array multi dimensi yang disusun secara *column major*.

5.2.2 Metode pointer

Seringkali ketika menuliskan program, banyaknya data yang akan disimpan tidak dapat diketahui dengan pasti. Dengan demikian dapat terjadi memori yang dialokasikan untuk menyimpan data mungkin terlalu banyak atau bahkan kurang. Untuk masalah ini kurang efisien jika penyimpanan data dilakukan dengan menggunakan metoda pengalokasian sekuensial yang telah dibahas. Tetapi akan lebih efisien (dari segi banyaknya memori) jika penyimpanan data dilakukan dengan Dynamic Memory Allocation dengan menggunakan pointer atau alokasi berantai (*linked allocation*) karena memori yang dialokasikan untuk menyimpan data, sesuai dengan banyaknya data. Jika ada penambahan data, maka dialokasikan tambahan

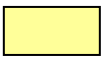
memori untuk menyimpan data yang baru. Sebaliknya jika ada data yang dihapus maka alokasi memori dikurangi.

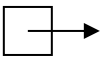
Metode pointer menggunakan pointer atau link untuk mereferensikan suatu elemen dalam struktur data. Dengan metode ini suatu elemen, selain menyimpan informasi juga menyimpan alamat dari elemen berikutnya (*successor element*). Elemen ini disebut dengan node. Tiap node terdiri dari 2 bagian, yaitu bagian yang menyimpan informasi dan bagian yang menyimpan alamat. Bagian yang menyimpan alamat disebut field pointer atau link. Dengan adanya pointer, node-node yang berurutan tidak perlu terletak bersebelahan secara fisik dalam memori seperti pada alokasi sekuensial. Gambar 5.5 merupakan ilustrasi dari 5 data yang disimpan dengan metode pointer. A adalah pointer yang menunjuk node yang pertama.



Gambar 5.5 Ilustrasi penyimpanan 5 data dengan metode pointer

Keterangan:


: bagian node yang berisi informasi


: bagian node yang berisi alamat dari node berikutnya


: node yang terakhir mempunyai pointer null

Banyaknya memori yang dibutuhkan untuk menyimpan data dengan metode pointer tergantung dari banyaknya memori untuk menyimpan informasi ditambah banyaknya memori untuk menyimpan pointer. Pointer biasanya bertipe integer sehingga membutuhkan 2 byte untuk tiap nodenya. Untuk gambar 5.5 di atas, misalnya informasi bertipe real dan pointer bertipe integer, maka banyaknya memori yang perlu dialokasikan adalah:

Σ memori = (bagian node untuk informasi) + (bagian node untuk pointer)

= 5 x 4 byte + 5 x 2 byte = 30 byte

Untuk contoh ini terlihat bahwa banyaknya memori yang dialokasikan untuk pointer, cukup besar yaitu 10 byte atau 33.33% dari memori keseluruhan. Tetapi jika informasi yang disimpan dalam tiap node membutuhkan 50 byte atau lebih (misalnya berupa struct) maka penambahan alokasi memori untuk pointer tidaklah signifikan.

Latihan Soal Subbab 5.2

1. Diketahui alamat awal array berikut ini adalah 1000.
   - A adalah array linier integer dengan indeks –50 sampai 40
   - B adalah array linier real dengan indeks –25 sampai 60
   - C adalah array linier karakter dengan indeks –35 sampai 35
   - D adalah array matriks integer dengan indeks 0 sampai 8 dan 1 sampai 12
   - E adalah array matriks real dengan indeks 1 sampai 6 dan -1 sampai 7
   - F adalah array matriks karakter dengan indeks -3 sampai 5 dan 2 sampai 11
   - G adalah array 3 dimensi integer dengan indeks 0 sampai 2, 1 sampai 4, dan 1 sampai 8
   - H adalah array 3 dimensi real dengan indeks 1 sampai 5, 0 sampai 6, dan 1 sampai 4
   - I adalah array 3 dimensi karakter dengan indeks 1 sampai 4, 1 sampai 7, dan 0 sampai 3

a. Hitunglah banyaknya elemen dan banyaknya memori yang dibutuhkan oleh masing-masing array di atas (lihat tabel 5.1).

b. Tentukan alamat dari

- A[0]	- A[-5]	- A[20]
- B[0]	- B[-5]	- B[20]
- C[0]	- C[-5]	- C[20]

c. Tentukan alamat dari

- D[5,5]	- D[7,3]
- E[5,5]	- E[3,4]
- F[5,5]	- F[0,10]

d. Tentukan alamat dari

- G[2,2,2]	- G[1,3,5]
- H[2,2,2]	- H[2,4,2]
- I[2,2,2]	- I[3,6,1]

2.a Diketahui 50 data bertipe integer disimpan dengan metode pointer. Hitunglah banyaknya memori yang dibutuhkan. Jika ke 50 data disimpan dalam array A, metode manakah yang lebih menghemat memori? Jelaskan.

2.b Diketahui 50 data bertipe real disimpan dengan metode pointer. Hitunglah banyaknya memori yang dibutuhkan. Jika ke 50 data disimpan dalam array B, metode manakah yang lebih menghemat memori? Jelaskan.

2.c Diketahui 50 data bertipe character disimpan dengan metode pointer. Hitunglah banyaknya memori yang dibutuhkan. Jika ke 50 data disimpan dalam array C, metode manakah yang lebih menghemat memori? Jelaskan.

5.3. LIST

Struktur data list didefinisikan sebagai himpunan elemen yang banyaknya dapat bervariasi (bersifat dinamik) karena adanya penambahan atau penghapusan elemen-elemen list. Ada beberapa bentuk list yaitu linear list, generalized list, stack dan queue. Yang akan dibahas disini adalah struktur linear list atau disebut list saja.

Dilihat secara fisik, struktur data list mirip dengan array, yaitu berupa deretan elemen. Tetapi jika dilihat secara keseluruhan maka terdapat perbedaan, terutama pada operasinya, seperti dapat dilihat pada tabel 4.2 berikut.

Tabel 5.2 Perbedaan struktur data array dengan list

Struktur data array	Struktur data list
Bersifat statik (banyaknya elemen yang dapat disimpan dalam array tidak dapat berubah)	Bersifat dinamik (banyaknya elemen yang disimpan dalam list dapat berubah-ubah)
Operasi yang didefinisikan membaca data dan menulis data dari/ke array.	Operasi yang didefinisikan lebih banyak, diantaranya insert dan delete elemen.

Operasi-operasi yang didefinisikan untuk struktur data list adalah:

a. Membuat list kosong (create)
b. Apakah list kosong (isEmpty)
c. Menentukan ukuran/panjang/jumlah elemen dalam list (size).

d. Mencari elemen tertentu dalam list.
e. Menghapus elemen yang ke i.
f. Menyisipkan elemen ke posisi i.

5.4. Alokasi Struktur Data List

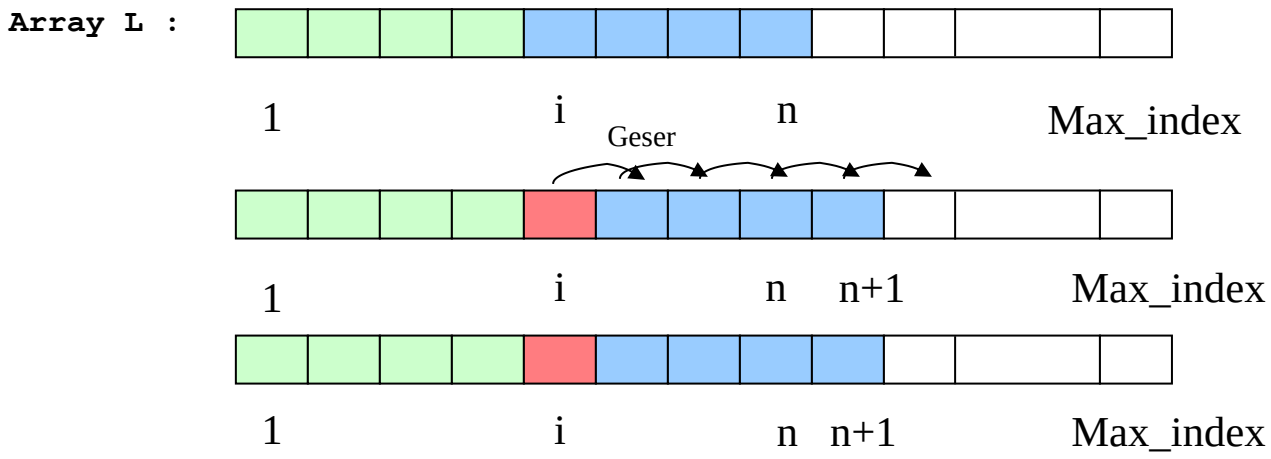
Ada dua cara untuk mengalokasikan struktur data list yaitu dengan menggunakan alokasi sekuensial dan alokasi berantai (linked). Dengan menggunakan alokasi sekuensial, struktur data list dialokasikan dalam struktur data array, sehingga banyaknya elemen dalam struktur data list akan menjadi terbatas yaitu sesuai dengan banyaknya memori yang dialokasikan untuk array. Selain itu, untuk operasi insert dan delete perlu dilakukan banyak pergeseran..

Dengan alokasi berantai, struktur data list tetap bersifat dinamik, dapat bertambah panjang sesuai kebutuhan. Dan operasi insert dan delete dilakukan tanpa perlu melakukan pergeseran elemen, cukup dengan manipulasi field pointer saja. Lihat ilustrasi pada gambar 4.6.b dan 4.7.b. Perbandingan operasi insert dan delete untuk kedua alokasi tersebut dapat dilihat pada tabel 4.3 berikut.

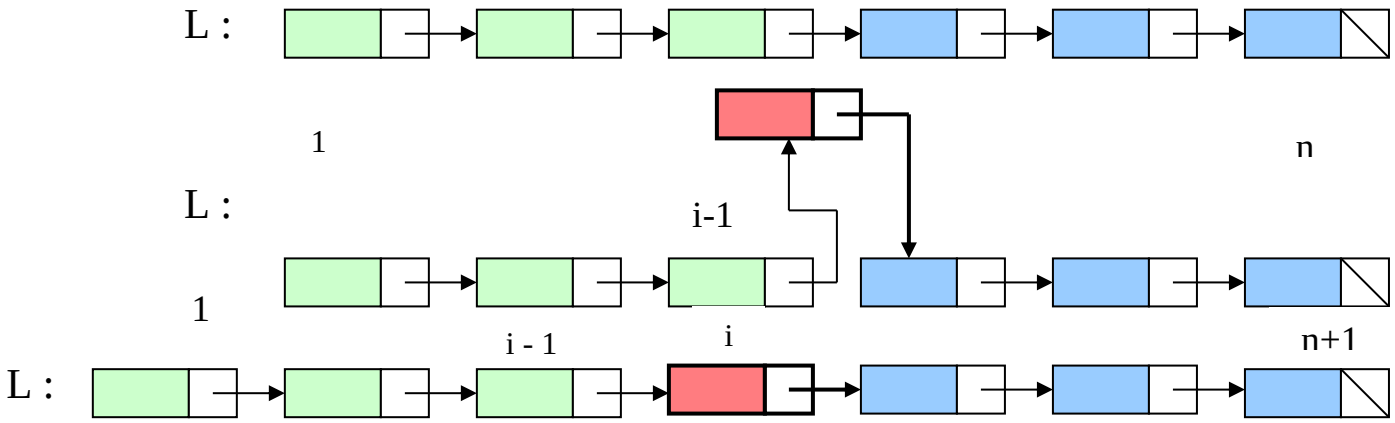
Tabel 5.3 Perbandingan operasi linear list dengan alokasi sekuensial dan alokasi berantai

Operasi	Alokasi Sekuensial	Alokasi berantai
Insert elemen baru ke posisi i: insert(i,x)	• Geser mulai elemen n, n-1, .., i ke n+1, n, .., i+1 • insert elemen baru di posisi i	• buat pointer pada node baru menunjuk ke node i+1 • buat pointer pada node i menunjuk ke node baru
Delete elemen pada posisi i: delete(i)	• Geser mulai elemen i+1, i+2, .., n ke i, i + 1, .., n-1	• buat pointer pada node i-1 menunjuk ke node i+1

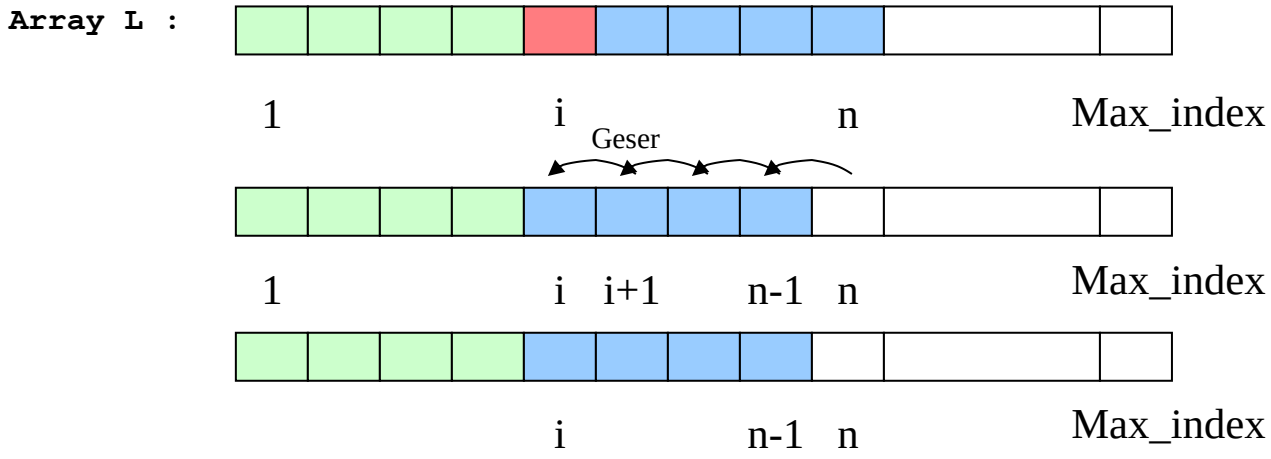
Sebagai ilustrasi, array L[1..Max_index] digunakan sebagai alokasi sekuensial dari list yang mempunyai n elemen. Perhatikan, pada array L, mulai dari index n+1 sampai Max_index, tidak digunakan oleh list, tetapi hanya sebagai tempat cadangan jika list bertambah panjang. Sebaliknya pada alokasi berantai, banyaknya node yang dialokasikan sesuai dengan panjang list L. Sebagai ilustrasi untuk operasi insert(i,x) dan delete(i) untuk kedua alokasi tersebut dapat dilihat pada gambar 5.6 dan 5.7.



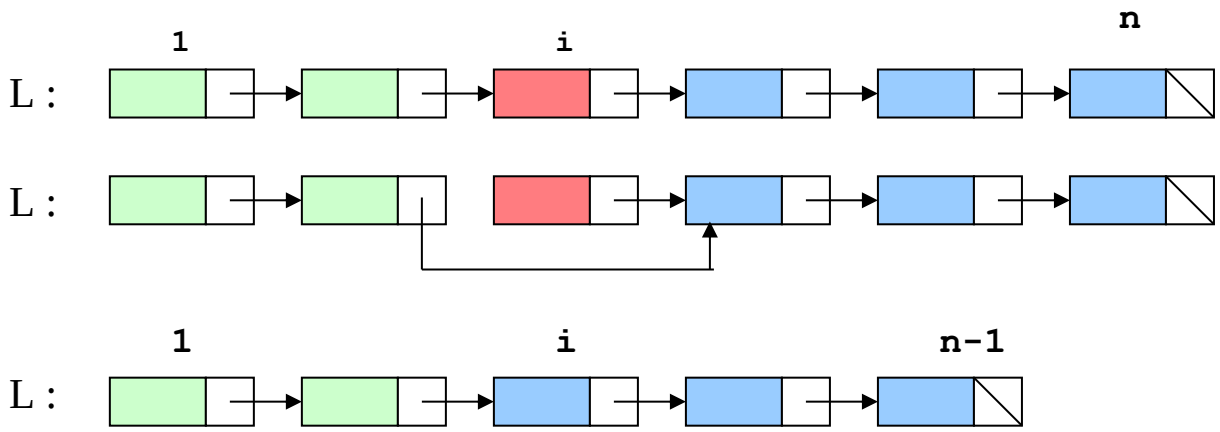
Gambar 5.6.a Operasi insert(i,x) untuk list yang dialokasikan secara sekuensial



Gambar 5.6.b Operasi insert(i,x) untuk list yang dialokasikan secara berantai



Gambar 5.7.a Operasi delete(i) untuk list yang dialokasikan secara sekuensial



Gambar 5.7.b. Operasi delete(i) untuk list yang dialokasikan secara berantai

5.5. Singly Linked Linear List

Linear list yang dialokasikan dengan metode berantai disebut dengan linked list. Berdasarkan banyaknya link yang ada pada node, dibedakan menjadi:

a. Single linked linear list atau singly linked list: tiap node mempunyai sebuah pointer yang menunjuk node berikutnya.
b. Double linked linear list atau double linked list: tiap node mempunyai dua pointer, yaitu yang menunjuk node berikutnya dan yang menunjuk node sebelumnya.

5.5.1. Terminologi

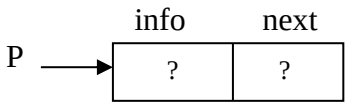
Beberapa istilah dan perintah yang digunakan bersama struktur data list:

a. Node: elemen dari list. Node berbentuk struct yang terdiri dari 2 field yaitu field INFO untuk menyimpan data/informasi dan field NEXT untuk menyimpan pointer yang menunjuk ke node berikutnya. Contoh definisi struct untuk node yang menyimpan data integer:

```
struct node {
        int             info
        struct node  *next
}
typedef struct  node *NODEPTR
```

b. new: perintah untuk mengalokasikan/membuat node baru.
   Contoh membuat node baru yang ditunjuk oleh pointer P:

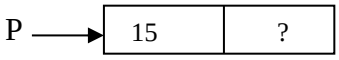
   P = new node



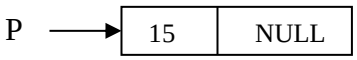
   Field info dan next dari node masih kosong (belum berisi data – simbol ?)

c. pointer NULL: pointer yang tidak menunjuk alamat apapun. Field next dari node yang terakhir pada list diisi pointer NULL.

d. P → info: menunjukkan field info dari node yang ditunjuk oleh pointer P
   Contoh: P→ info = 15
   ➔ field info dari node yang ditunjuk oleh pointer P, diisi angka 15

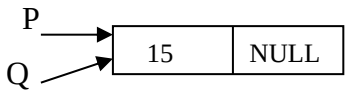


e. P->next: menunjukkan field next dari node yang ditunjuk oleh pointer P
   Contoh: P-> next = NULL
   ➔ field next dari node yang ditunjuk oleh pointer P diisi NULL



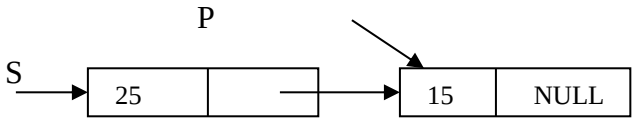
f. Untuk membuat sebuah pointer Q menunjuk node yang ditunjuk oleh pointer P:

   Q = P



g. Untuk membuat field next dari node yang ditunjuk oleh pointer S menunjuk node yang ditunjuk oleh pointer P.

   S → next = P



h. freenode(X): menghapus node yang ditunjuk oleh pointer X. Setelah perintah ini dilaksanakan pointer X dan node yang ditunjuk oleh pointer X tidak dapat diakses lagi.

5.5.2 Operasi Linked List

Operasi linked list yang akan dibahas disini adalah operasi menyisipkan node baru (insert) ke dalam linked list, menghapus node dari linked list (delete) dan menyalin linked list (copy). Masing-masing operasi akan dibuat fungsi tersendiri.

Khusus untuk operasi insert dan delete akan diperinci lagi berdasarkan letak/posisi node yang terkena operasi.

Untuk operasi insert dibagi menjadi 5 kasus yaitu:

1. Insert front: node yang baru diletakkan di depan node yang pertama dari linked list sehingga node baru menjadi node yang pertama.

   Function INS_FRONT(FIRST, X)
   FIRST adalah pointer yang menunjuk list. X adalah isi field info dari node yang akan disisipkan. NEW adalah pointer yang menunjuk node yang akan disisipkan (node baru). Node baru akan menjadi node yang pertama.
   {
   1. [membuat node baru yang ditunjuk oleh pointer NEW]
      NEW = new node
   2. [field info pada node baru diisi dengan X]
      NEW->info = X
   3. [field next pada node baru dibuat menunjuk node yang ditunjuk oleh pointer FIRST]
      NEW->next = FIRST
   4. [kembali ke program utama]
      Return (NEW)
   }

2. Insert end: node yang baru diletakkan di belakang node yang terakhir dari linked list sehingga node baru menjadi node yang terakhir.

   Function INS_END(FIRST, X)
   FIRST adalah pointer yang menunjuk list. X adalah isi field info dari node yang akan disisipkan. NEW adalah pointer yang menunjuk node yang akan disisipkan (node baru). S adalah pointer. Node baru akan menjadi node yang terakhir.
   {
   1. [membuat node baru yang ditunjuk oleh pointer NEW]
      NEW = new node
   2. [field info pada node baru diisi dengan X]
      NEW->info = X
   3. [field next pada node baru diisi oleh pointer NULL]
      NEW->next = NULL
   4. [jika list masih kosong)
      If (FIRST = = NULL)
      { Return(NEW) [node baru menjadi node yang pertama]
      }
   5. [buat pointer S menunjuk list]
      S = FIRST
   6. [dengan menggunakan pointer S, telusuri list untuk mencari node yang terakhir]
      While (S->next != NULL)
      { S = S->next
      }
   7. [buat node terakhir menunjuk node baru]

```
    S->next  =  NEW
8.  [kembali ke program utama]
    Return (FIRST)
}
```

3. Insert last: sama seperti insert end, tapi alamat node yang terakhir disimpan dalam variabel pointer LAST

```
Function INS_LAST(FIRST, LAST, X)
FIRST adalah pointer yang menunjuk list (node pertama). LAST adalah pointer
yang menunjuk node terakhir. X adalah isi field info dari node yang akan
disisipkan. NEW adalah pointer yang menunjuk node yang akan disisipkan. Node
baru akan menjadi node yang terakhir dan ditunjuk oleh pointer LAST.
{
1. [membuat node baru yang ditunjuk oleh pointer NEW]
   NEW = new node
2. [field info pada node baru diisi dengan X]
   NEW->info = X
3. [field next pada node baru diisi oleh pointer NULL]
   NEW->next = NULL
4. [jika list masih kosong)
   If (FIRST == NULL)
   {  LAST = NEW          [pointer LAST menunjuk node yang baru]
      Return(NEW)         [node baru menjadi node yang pertama dan terakhir]
   }
5. [fileld next pada node yang ditunjuk pointer LAST dibuat menunjuk node
   baru]
   LAST->next = NEW
6. [buat pointer LAST menunjuk node baru]
   LAST  =  NEW
7. [kembali ke program utama]
   Return (FIRST)
}
```

4. Insert mid: node yang baru diletakkan di antara node pertama dan node terakhir.

```
Function INS_MID(FIRST, MID, X)
FIRST adalah pointer yang menunjuk list (node pertama). MID adalah pointer
yang menunjuk node pada list tempat node baru disisipkan. X adalah isi field
info dari node yang akan disisipkan. NEW adalah pointer yang menunjuk node
yang akan disisipkan. Node baru akan menjadi successor dari node yang
ditunjuk oleh pointer MID.
{
1. [membuat node baru yang ditunjuk oleh pointer NEW]
   NEW = new node
2. [field info pada node baru diisi dengan X]
   NEW->info = X
3. [field next pada node baru dibuat menunjuk node successor]
   NEW->next = MID->next
```

4. [field next pada node yang ditunjuk oleh pointer MID dibuat menunjuk node baru]
   MID->next = NEW
5. [kembali ke program utama]
   Return (FIRST)

}

5. Insert order: node yang baru diletakkan di posisi yang sesuai dalam ordered linked list (linked list yang bagian informasinya sudah terurut).

```
Function INS_ORDER(FIRST, X)
FIRST adalah pointer yang menunjuk list (node pertama). X adalah isi field info
dari node yang akan disisipkan. NEW adalah pointer yang menunjuk node yang
akan disisipkan. S adalah pointer.
{
1. [membuat node baru yang ditunjuk oleh pointer NEW]
   NEW = new node
2. [field info pada node baru diisi dengan X]
   NEW->info = X
3. [jika list masih kosong]
   If (FIRST == NULL)
   {       Return(NEW)          [node baru menjadi node yang pertama]
   }
4. If (FIRST->info >= X)
   {       NEW->next = FIRST    [node baru menjadi node yang pertama]
           Return(NEW)        [karena isi field info pada node pertama >= X]
   }
5. [buat pointer S menunjuk list]
   S = FIRST
6. [dengan menggunakan pointer S, telusuri list untuk mencari node yang isi
   info field >= X]
   While (S->next != NULL && ((S->next)->info) <= (NEW -> info)))
   {       S = S->next
   }
7. [menyisipkan node baru di posisinya yang sesuai]
   If (S-> next == NULL)
   {       S ->next = NEW
           NEW->next = NULL}
   Else
   {       NEW->next = S->next
           S->next = NEW}
8. [kembali ke program utama]
   Return (FIRST)
}
```

Sedangkan untuk operasi delete, akan dibuat satu fungsi, tetapi dalam fungsi ada 3 kasus, yaitu:

1. List kosong, operasi delete tidak dapat dilakukan
2. Delete first: menghapus node yang pertama dari linked list
3. Delete node lainnya: menghapus node yang berada setelah node yang pertama.

```
Function DELETE(FIRST, D)
FIRST adalah pointer yang menunjuk list (node pertama). D adalah pointer yang
menunjuk node yang akan dihapus. P adalah pointer.
{
1. If (FIRST == NULL)
   {  write ('List kosong, operasi delete tidak dilakukan')
      Return(FIRST)
   }
2. [Pointer D menunjuk node pertama]
   If ( D == FIRST)
   {  FIRST = FIRST->next
      freenode (D)
      Return(FIRST)
   }
3. [pointer P menunjuk node pertama pada list]
   P = FIRST
4. [dengan pointer P, menelusuri list untuk menemukan pointer D]
   While (P->next != NULL && P->next != D)
   {  P = P->next
   }
5. [pointer P menunjuk node prodesesor dari node yang ditunjuk oleh pointer D]
   If (P->next != NULL)
   {  P->next = D->next      [field next pada node yang ditunjuk oleh pointer P,]
                  [menunjuk node successor dari node yang ditunjuk oleh pointer D]
      freenode(D)            [hapus node yang ditunjuk oleh pointer D]
   }
   else
   {  Write('Node yang akan dihapus tidak ketemu')
   }
6. [kembali ke algoritma utama]
   Return(FIRST)
}
```

Untuk menyalin suatu linked list menjadi linked list yang baru dilakukan dengan menyalin seluruh node satu persatu.

```
Function COPY(FIRST)
Fungsi untuk menyalin seluruh node dalam list. FIRST adalah pointer yang
menunjuk list asal. SECOND adalah pointer yang menunjuk node pertama dari
list hasil copy. P dan S adalah pointer.
{
1. IF (FIRST == NULL
   {   write ('List asal kosong')
```

```
        Return(NULL)
    }
2.  [membuat node baru yang ditunjuk oleh pointer NEW]
    NEW = new node
3.  [salin field info pada node pertama list ke field info pada node baru]
    NEW->info = FIRST->info
4.  [pointer SECOND menunjuk node baru, node pertama dari list hasil copy]
    SECOND = NEW
5.  [pointer S menunjuk list asal]
    S = FIRST
6.  [menelusuri list asal dengan menggunakan pointer S untuk melakukan
    proses copy]
    While (S->next != NULL)
    {
6.a     [pointer P menunjuk node baru]
        P = NEW
6.b     [pointer S menunjuk node successor]
        S = S->next
6.c     [buat node baru lalu salin field info dari list asal ke node baru]
        NEW = new node
        NEW->info = S->info
6.d     [menyambungkan node baru ke list hasil copy]
        P->next = NEW
    }
7.  [field next dari node terakhir pada list hasil copy diisi pointer NULL]
    NEW->next = NULL
8.  [kembali ke program utama]
    Return(SECOND)
}
```

Latihan Soal Subbab 5.5

1. Diketahui linear linked list L yang node-nya berisi data integer. A dan K adalah pointer. Gambarkan list L setelah operasi berikut ini:

a. L = null
   L = INS_FRONT(L, 25)
   L = INS_FRONT(L, 10) K = L
   L = INS_END(L, 30)
   L = INS_MID(L, K, 20)
   L = INS_MID(L, K, 15)
   L = INS_FRONT(L, 5)
   L = INS_END(L,35)

b. L = NULL
   A = L
   L = INS_LAST(L, A, 20)

K = A
L = INS_FRONT(L, 15)
L = INS_FRONT(L, 10)
L = INS_LAST(L, A, 35)
L = INS_MID(L, K, 30)
L = INS_MID(L, K, 25)
L = INS_FRONT(L, 5)

c. L = NULL
S = L
L = INS_LAST(L, S, 5)
L = INS_LAST(L, S, 10)
L = INS_FRONT(L, 15)
P = L
L = INS_FRONT(L, 20)
L = INS_FRONT(L, 25)
L = INS_MID(L, P, 30)
L = INS_MID(L, P, 35)

2. Tulis algoritma untuk memasukkan data berikut ini:
   15, 20, 5, 25, 10
   menjadi sebuah linear linked list jika data tersebut dimasukkan dengan menggunakan fungsi berikut dan gambarkan prosesnya:
   a. Ins_Front(First,X)
   b. Ins_End(First,X)
   c. Ins_Last(First, Last, X)
   d. Ins_Order(First, X)

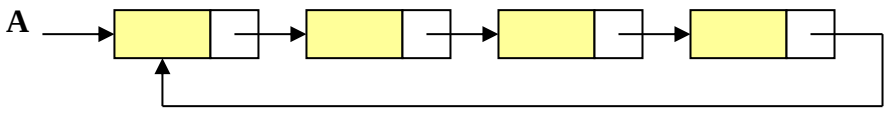
Untuk soal nomor 3 – 10, buat program untuk fungsi-fungsi berikut ini dalam bahasa C++ lalu lengkapi dengan algoritma utama yang berisi perintah untuk memanggil fungsi, memasukkan data dan mencetak isi linked list.

3. Buat fungsi untuk menghitung banyaknya node dalam sebuah linked list.
4. Buat fungsi untuk mengganti isi field INFO dari node yang ke k dalam sebuah linked list.
5. Buat fungsi untuk menyisipkan node baru tepat setelah node yang ke I dalam sebuah linked list.
6. Buat fungsi untuk menyisipkan node baru tepat sebelum node yang ke I dalam sebuah linked list.
7. Buat fungsi untuk menggabungkan dua buah linked list yang masing-masing ditunjuk oleh pointer FIRST dan SECOND, menjadi sebuah linked list yang ditunjuk oleh pointer THIRD.

8. Buat fungsi untuk memisahkan sebuah linked list yang ditunjuk oleh pointer AWAL menjadi 2 linked list yang masing-masing ditunjuk oleh pointer FIRST dan SECOND. Pemisahan dilakukan dengan cara:
   a. Menghitung dulu jumlah node dalam linked list AWAL. Linked list FIRST berisi (N div 2) node yang pertama dan linked list SECOND berisi node-node berikutnya.
   b. Linked list FIRST berisi node yang ke 1, 3, 5… dari linked list AWAL dan linked list SECOND berisi node-node yang ke 2, 4, 6, … dari linked list AWAL.
9. Buat fungsi untuk membalikkan urutan sebuah linked list. Pada akhir proses linked list mula-mula masih tetap ada (tidak berubah).
10. Buat fungsi untuk menghapus node ke I sampai node ke J dari sebuah linked list

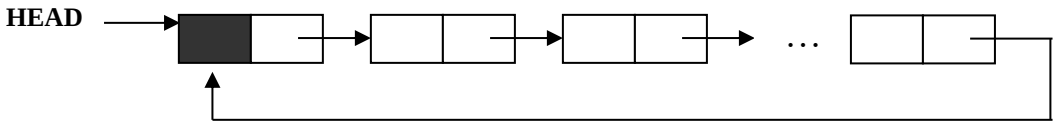
5.6. Circular List

Singly linked linear list yang telah dibahas, hanya dapat ditelusuri dari node awal sampai node terakhir dan tidak dapat kembali lagi. Agar penelusuran list lebih fleksibel, field next pada node terakhir dibuat menunjuk node pertama. List yang demikian disebut circularly linked list atau circular list saja (lihat gambar 5.8).



Gambar 5.8 Circular List

Pada circular list tiap node dapat diakses dari node mana saja, tetapi jika tidak hati-hati dapat terjadi *infinite loop* karena circular list tidak mempunyai ujung. Hal ini dapat diatasi dengan menambahkan node khusus di awal list yang disebut node HEAD (lihat gambar 5.9). Node HEAD adalah node yang menunjuk node pertama dari list.



Gambar 5.9 Circular list dengan node HEAD

Untuk menyisipkan node baru di awal circular list yang ditunjuk oleh pointer HEAD, fungsi INS_FRONT untuk linear list dimodifikasi menjadi seperti berikut:

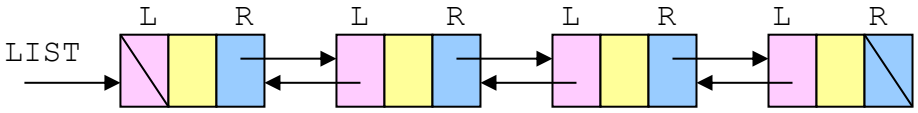
```
Function INS_FRONT_CL(HEAD, X)
HEAD adalah pointer yang menunjuk node pertama dari circular list. X adalah
isi field info dari node yang akan disisipkan. NEW adalah pointer yang menunjuk
node yang akan disisipkan (node baru). Node baru akan menjadi node yang
pertama.
{
1. [membuat node baru yang ditunjuk oleh pointer NEW]
   NEW = new node
2. [field info pada node baru diisi dengan X]
   NEW->info = X
3. [jika list masih kosong maka node yang baru menjadi node yang pertama]
   If ( HEAD->next == NULL )
   {    NEW->next = NULL
        HEAD->next = NEW
   }
4. [node yang baru berada setelah node HEAD]
   NEW -> next = HEAD -> next
   HEAD -> next = NEW
5. [kembali ke program utama]
   Return (HEAD)
}
```

Perhatikan pada fungsi di atas, node baru (node pertama) disisipkan setelah node HEAD. Sebagai latihan, buatlah fungsi untuk menyisipkan node baru di akhir dan di tengah circular linked list.

5.7. Doubly Linked Linear List

Dalam beberapa aplikasi list perlu dijalani dari kiri ke kanan dan sebaliknya. Untuk itu tiap node mempunyai 2 link field yaitu right link yang digunakan untuk menunjuk node berikutnya (*successor node*) dan left link yang digunakan untuk menunjuk node sebelumnya (*predecessor node*). List yang mempunyai 2 link ini disebut doubly linked linear list atau two way chain. Left link dari node yang paling

kiri bernilai NULL dan right link dari node yang paling kanan bernilai NULL (lihat gambar 5.10).



Gambar 5.10 Doubly linked list

Keterangan:

: link ke predesesor node

: tempat data

: link ke successor node

Contoh deklarasi untuk node dari double linked list ini adalah:

```
struct node {
        int             info
        struct node     *RP
        struct node     *LP
}
typedef struct  node *NODEPTR
```

Perhatikan pada deklarasi di atas, ada 2 pointer untuk setiap node, yaitu RP yang menunjuk node successor dan LP yang menunjuk ke node predecessor. Dengan adanya 2 buah link ini, maka operasi untuk double linked list ini menjadi lebih rumit karena setiap operasi node ke dua link tersebut harus diatur. Sebagai gambaran akan dibuat fungsi untuk menyisipkan node baru pada double linked list setelah node yang ditunjuk oleh pointer MID.

Function INS_MID(FIRST, MID, X)
FIRST adalah pointer yang menunjuk list (node pertama). MID adalah pointer yang menunjuk node pada list tempat node baru disisipkan. X adalah isi field info dari node yang akan disisipkan. NEW adalah pointer yang menunjuk node yang akan disisipkan. Node baru akan menjadi successor dari node yang ditunjuk oleh pointer MID. RP adalah pointer yang menunjuk succesor dari suatu node dan LP adalah pointer yang menunjuk predecessor dari suatu node. T pointer yang menunjuk node successor dari node yang ditunjuk oleh pointer MID

```
{
1. [membuat node baru yang ditunjuk oleh pointer NEW]
   NEW = new node
2. [field info pada node baru diisi dengan X]
   NEW->info = X
3. [membuat pointer T menunjuk node successor dari pointer MID]
   T = MID->RP
4. [menyisipkan node baru setelah node yang ditunjuk pointer MID]
   NEW->LP = MID   [field LP dari node baru dibuat menunjuk node predecessor]
   NEW->RP = T     [field RP dari node baru dibuat menunjuk node successor]
   MID->RP = NEW   [field RP dari node yang ditunjuk oleh pointer MID dibuat]
                           [menunjuk node baru]
   T->LP = NEW     [field LP dari node yang ditunjuk oleh pointer T dibuat]
                           [menunjuk node baru]
5.  [kembali ke program utama]
   Return (FIRST)
}
```

Perhatikan, pada fungsi di atas pointer MID diasumsikan menunjuk salah satu node yang menjadi elemen dari LIST. Dengan adanya 2 buah ponter (RP dan LP) maka operasi yang dilakukan menjadi lebih banyak, tetapi programmer mempunyai kemudahan dalam menelusuri list. Sebagai latihan buatlah fungsi untuk menyisipkan node di awal dan di akhir double linked list, serta fungsi untuk menghapus node dari double linked list.