Majalah Ilmiah
Informatika, Komputer, dan Bisnis
ISSN 1410 - 9158

FORMAT

Volume 7, Nomor 1, Januari 2006

SEKOLAH TINGGI MANAJEMEN INFORMATIKA DAN KOMPUTER AKAKOM
YOGYAKARTA

Majalah Ilmiah
Informatika, Komputer, dan Bisnis

FORMAT

PELINDUNG:
Ketua Yayasan Pendidikan Widya Bakti

KETUA UMUM:
Ketua STMIK AKAKOM Yogyakarta

KETUA DEWAN REDAKSI:
Bambang P.D.P., S.E., Akt., S.Kom., M.MSi.

ANGGOTA DEWAN REDAKSI:
Ir. F. Soesianto, B.Sc.E., Ph.D.
Prof. H. Adhi Susanto, M.Sc., Ph.D.
Drs. Tri Prabawa, M.Kom.
Ir. Surjono, M.Phil.
Ir. Sudarmanto, M.T.
Ir. M. Guntara, M.T.
Ir. Totok Suprawoto, M.M.
Budi Sugihardjo, S.E., M.M.
Heru Agus Triyanto, S.E., M.M.

REDAKTUR PELAKSANA:
Indra Yatini Buryadi, S.Kom., M.Kom.

SEKRETARIS:
Al. Agus Subagyo, S.E., M.Si

LAYOUT dan PRODUKSI:
Dison Librado, S.E., M.Kom.

SIRKULASI:
Totok Budioko, S.T.

DOKUMENTASI:
Dra. Torsinawati
Sukar

Majalah Ilmiah FORMAT diterbitkan empat bulan sekali oleh
STMIK AKAKOM dengan ISSN 1410 - 9158

Pendapat yang dinyatakan dalam majalah ini
adalah sepenuhnya pendapat pribadi
Segala sesuatu yang berhubungan dengan penerbitan majalah dapat disampaikan secara
tertulis maupun lisan kepada redaksi

ALAMAT REDAKSI:
STMIK AKAKOM
Jl. Raya Janti, *Ring Road* Timur, Yogyakarta 55198
Telepon : 62-0274-486664
Faksimile : 62-0274-486438 E-mail : format@netexecutive.com

Dari Redaksi

Kami panjatkan puji dan syukur atas rahmat dan berkah dari Tuhan Yang Maha Esa hingga kami dapat menyelesaikan dan menerbitkan majalah Format pada nomor pertama, tahun pertama. Pada tahun yang pertama ini kami mencoba untuk memperluas cakupan materi dari berbagai hasil penelitian dan karya ilmiah, namun tetap sesuai dengan misinya.

Dalam edisi ini para pembaca akan melihat topik-topik mengenai *Teknologi dan Aplikasi Telekonferensi di Indonesia, Mengimplementasikan Total Quality Management di Pendidikan Tinggi, Sistem Pakar Diagnosa Penyakit Saluran Pernafasan, List Berkait dengan Defenisi dan Proses Rekursif, Perbandingan Kinerja Algoritma Kompresi Data Metode Huffman dan Metode Sliding-Window, Perhitungan Rencana Anggaran Biaya Bangunan, Program Bantu Pemilihan Telepon Genggam Berbasis Fuzzy, Analisis Kesesuaian Pemilihan Jurusan Mahasiswa STMIK AKAKOM, Sistem Evaluasi Penerimaan Mahasiswa Baru Menggunakan Basis Data Fuzzy, Konektivitas Handphone T68i Dengan Komputer Berbasis Linux,* yang sekiranya akan menarik untuk diulas.

Harapan kami semoga apa yang kami suguhkan kali ini dapat membawa manfaat bagi peminat, dan menambah referensi pembaca pada bidang-bidang tertentu. Terima kasih diucapkan, atas saran dan masukan yang telah kami terima demi kemajuan majalah ilmiah ini. Saran, ide, dan gagasan dari para pembaca tetap kami tunggu untuk perbaikan pada penerbitan edisi mendatang di abad millenium ini.

Daftar Isi

LIST BERKAIT DENGAN DEFENISI DAN PROSES REKURSIF

Oleh : Febri Nova Lenti

ABSTRAK

List Rekursif adalah suatu ADT list berkait dengan defenisi dan proses rekursif dan di representasikan secara fisik dengan pointer, representasi address dengan pointer, tipe data info adalah integer. Pada list rekursif ini sebuah list dipandang sebagai sebuah list L yang terdiri dari elemen pertama ditambah elemen list berikutnya (next(L)), next(L) juga dipandang sebagai sebuah elemen ditambah dengan next(L), begitu seterusnya secara rekurif. Elemen list berikutnya (next(L)) disebut sebagai Tail (L). Sebagai sebuah tipe data Abstrak (*ADT*), list rekursif memiliki sekumpulan primitif yang dalam konteks prosedural diterjemahkan menjadi fungsi atau prosedur yang juga beberapa diantaranya menggunakan proses rekursif.

Keywords : list berkait, rekursif, list berkait rekursif, *ADT*, primitif, *function*, *procedure*.

PENDAHULUAN

Liem [4], List Berkait adalah sekumpulan elemen yang setiap individu elemennya memuat informasi dan menunjuk alamat suksesornya. List berkait direpresentasikan secara fisik dengan implementasi berkait yaitu sekumpulan data yang penempatannya pada memori komputer dapat terpencar-pencar, namun dapat ditelusuri berkat adanya informasi alamat, yang menghubungkan elemen yang satu dengan elemen yang lain.

Struktur data list Berkait adalah struktur data standar dalam bidang pemrograman. Pemakaian struktur data ini sudah banyak dan umum dalam kasus / persoalan yang sifatnya mengelola sekumpulan obyek.

Liem [3]untuk mendefenisikan koleksi dan elemen dapat dilakukan dengan dua cara, yaitu:

- defenisi iteratif : dengan mendefenisikan setiap elemen
- defenisi rekursif : suatu koleksi objek yang dapat mengandung elemen berupa koleksi itu sendiri

Konsep rekursif juga adalah suatu konsep yang sudah tidak asing lagi dalam dunia pemrograman, bahkan ada beberapa kasus persoalan yang pemecahannya lebih tepat dan alamiah jika memakai defenisi atau proses yag rekursif. Beberapa contoh persoalan tersebut adalah :

- defenisi rekursif : struktur data pohon (*tree*)
- proses rekursif : fungsi pangkat, bilangan fibonacci, kasus menara hanoi dan lain lain.

1. LIST BERKAIT

Liem [3] sebuah list adalah sekumpulan elemen bertipe sama yang rnempunyai "keterurutan" tertentu, dan setiap elemennya terdiri dari dua bagian, yaitu informasi mengenai elemennya dan informasi mengenai alamat elemen suksesornya. List dapat direprtesentasikan secara fisik dengan dua cara implementasi**: kontigu** atau **berkait**. List yang diimplementasikan secara berkait disebut sebagai **List Berkait**.

List berkait direpresentasikan secara fisik dengan implementasi berkait yaitu sekumpulan data yang penempatannya pada memori komputer dapat terpencar-pencar, namun dapat ditelusuri berkat adanya informasi alamat, yang menghubungkan elemen yang satu dengan elemen yang lain.

Liem [3] alamat yang sudah didefenisikan disebut sudah "di-alokasi". Didefenisikan suatu konstanta **Nil**, yang artinya alamat yang tidak terdefenisi. Alamat ini nantinya akan didefenisikan secara lebih konkret ketika list diimplimentasi pada struktur data fisik. Jadi, sebuah list dikenali:

- **Elemen pertamanya,** biasanya melalui alamat elemen pertama yang disebut : First
- Alamat **elemen berikutnya** (suksesor), jika kita mengetahui alamat sebuah elemen, yang dapat diakses melalui informasi NEXT. NEXT mungkin ada secara **eksplisit,** atau secara **implisit** yaitu lewat kalkulasi atau fungsi suksesor
- Setiap elemen mempunyai **alamat**, yaitu tempat elemen disimpan dapat diacu. Untuk mengacu sebuah elemen, alamat harus terdefenisi. Dengan alamat tersebut informasi yang tersimpan pada elemen list dapat diakses.
- **Elemen terakhirnya**. Ada berbagai cara untuk mengenali elemen akhir

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** :

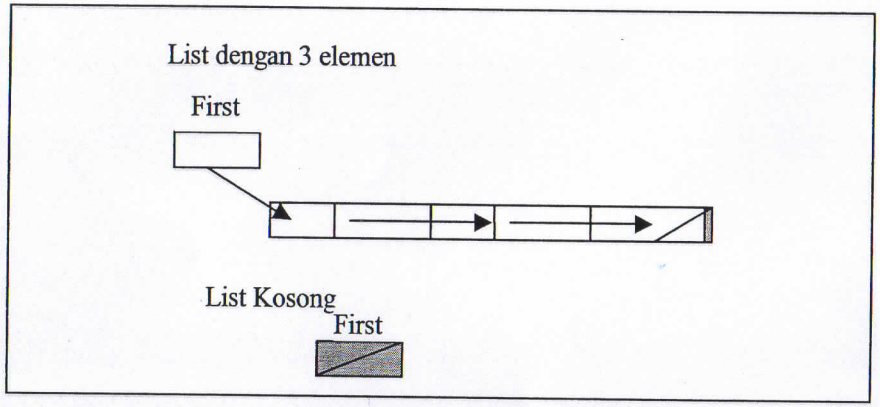
Info(P)

Next(P)

Beberapa defenisi :

- List L adalah list kosong, jika First(L) = Nil
- Elemen terakhir dikenali, misalnya jika Last adalah alamat elemen terakhir, maka Next(Last)=Nil

Biasanya list paling sederhana digambarkan dengan ilustrasi yang ditunjukkan pada gambar 2.1.



Gambar 2.1. Ilustrasi sebuah list

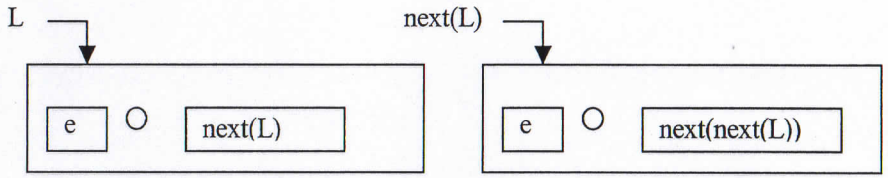
2. REKURSIF

Rekursif adalah suatu mekanisme yang menggunakan dirinya sendiri (Kernighan [1]). Dalam Pemrograman dikenal dua macam rekursif, yaitu defenisi rekursif dan proses rekursif (Liem [3]). Defenisi rekursif adalah suatu defenisi obyek dengan menggunakan dirinya sendiri. Sedang yang dimaksud dengan fungsi rekursif adalah suatu fungsi yang didalamnya ada statemen untuk memanggil dirinya sendiri.

Lewis [2] rekursif tidak selalu lebih jelek daripada iteratif. Ada kalanya sebuah fungsi rekursif justru mempermudah penyelesaian masalah yang ditemui pada kasus iteratif (pengulangan). Kelemahan pada proses rekursif antar lain, memerlukan tempat penampungan stack yang cukup besar. Karena setiap kali pemanggilan fungsi, register – register seperti cs (untuk memory far) dan ip harus disimpan, belum lagi untuk penanganan local variable serta parameter fungsi yang tentunya membutuhkan ruang untuk stack lebih banyak lagi. Selain itu karena setiap pemanggilan fungsi, register dan memory harus di push ke stack maka setelah selesai pemanggilan perlu diadakannya pop stack untuk mengembalikan memory dan register kembali ke keadaan awal, ini sering disebut sebagai *overhead*.

PEMBAHASAN

List Rekursif adalah suatu ADT list berkait dengan defenisi dan proses rekursif dan dengan representasi fisik pointer, representasi address dengan pointer, tipe data info adalah integer. Pada list rekursif ini sebuah list dipandang sebagai sebuah list L yang terdiri dari elemen pertama ditambah elemen list berikutnya (next(L)), next(L) juga dipandang sebagai sebuah elemen ditambah dengan next(L), begitu seterusnya secara rekurif seperti ditunjukkan pada gambar 4.1.



Gambar 4.1. List L sebagai list berkait rekursif

Elemen list berikutnya (next(L)) disebut sebagai Tail (L). Sebagai sebuah tipe data Abstrak (*ADT*), list rekursif memiliki sekumpulan primitif yang dalam konteks prosedural diterjemahkan menjadi fungsi atau prosedur, yang dikelompokkan menjadi :

1. Konstruktor ; CreateList dan Alokasi
2. Destruktor ; Dealokasi
3. Validator ; ListEmpty, IsOneElmt, Search, Fsearch, SearchPrec
4. Selektor ; Konso, Kons, FirstElmt, Tail, DelVFirst, DelVLast, DelIAll
5. Baca/Tulis ; Printinfo
6. Aritmatika ; NbElmt, InversList, CopyList, CpAlokList, Konkat
7. Operator relasional ; Max, Min

1. STRUKTUR DATA

Mendefenisikan representasi fisik struktur data List Rekursif pada dasarnya hanya ada dua macam implementasi fisik yaitu dengan pointer atau tabel. Pada tulisan ini List Rekursif direpresentasikan dengan pointer. Pada bahasa pemrograman C representasi secara berkait ini dilakukan dengan defenisi sebagai berikut:

```
typedef int infotype;
typedef struct tElmntlist *address;
typedef struct tElmntlist
                  { infotype info;
                    address next;
                  }ElmntList;

typedef address List;
```

2. FUNGSI PRIMITIF

Beberapa fungsi primitif list rekursif yang dibahas pada tulisan ini dengan spesifikasinya masing-masing adalah sebagai berikut :

```
1. void CreateList(List *L);
    /* I.S :sembarang /*
    /* F.S : terbentuk list kosong */

2. address Alokasi(infotype X);
    /* mengirimkan address hasil alokasi sebuah elemen */
    /* jika alokasi berhasil, maka address tidak Nil,
       dan                misalnya menghasilkan */
   /* P, maka info(P)=X, Next(P)=Nil */
   /* jika alokasi gagal, mengirimkan nil */

3. void Dealokasi(address P);
   /*I.S : P terdefenisi */
   /*F.S : P dikembalikan ke sistem */

4. boolean ListEmpty(List L);
    /* mengirim true jika list kosong */
```



```
5. boolean IsOneElmt(List L);
   /* mengrimkan true jika list hanya memiliki satu
elemen*/

6. address Search(List L, infotype X);
   /* mencari apakah ada elemen list dengan info(P)=X */
   /* jika ada, mengirimkan address elemen tersebut */
   /* jika tidak ada, mengirimkan Nil */

7. boolean Fsearch(List L, address P);
   /* mencari apakah ada elemen list yang beralamat P */
   /* mengirimkan true jika ada, false jika tidak ada */

8. address SearchPrec(List L, infotype X);
   /*mengirimkan address elemen sebelum elemen yang
      nilainya X */
   /* mencari apakah ada elemen list dengan info(P)=X */
   /* jika ada, mengirimkan address Prec, dengan
      Next(Prec)=P dan Info(P)=X */
   /* jika tidak ada, mengirimkan nil*/
   /* jika P adalah elemen pertama, maka Prec=Nil */
   /* Search dengan speifikasi seperti ini menghindari
      traversal ulang */
   /* jika setelah search akan dilakukan operasi lain */

9. List Konso(List L, infotype X);
   /* I. S : L mungkin kosong */
   /* F.S  : melakukan alokasi sebuah elemen dan
      menambahkan elemen pertama dengan nilai X jika
      alokasi berhasil */

10.List Kons(List L, infotype X);
    /* I. S : L mungkin kosong */
                /* F.S  : melakukan alokasi sebuah
           elemen dan menambahkan elemen list di akhir
           elemen terakhir yang baru bernilai X jika
           alokasi berhasil */
    /*          jika alokasi gagal I.S = F.S */

11.infotype FirstElmt(List L);
   /* I.S : sembarang, P sudah dialokasi */
```

```
   /* F.S : menambahkan elemen beraddress P sebagai
      elemen pertama */

12.List Tail(List L);
   /* I.S : sembarang, P sudah dialokasi */
   /* F.S : ditambahkan sebagai elemen terakhir yang
baru */

13.void DelVFirst(List *L,infotype *X);
   /* I.S : list tidak kosong */
   /* F.S :elemen pertama list dihapus; nilai info
      disimpan pada X dan alamat elemen terakhir di
      dealokasi*/

14.void DelVLast(List *L,infotype *X);
    /* I.S : list tidak kosong */
    /* F.S :elemen terakhir list dihapus; nilai info
       disimpan pada X */
    /* dan alamat elemen terakhir di dealokasi */

15.void DelAll(List *L);
    /* Delete semua elemen list dan alamat elemen di-
       dealokasi */

16.void Printinfo(List L);
    /* I.S : list mungkin kosong */
    /* F.S : jika list tidak kosong, semua elemen list
       diprint */
    /*       jika list kosong, hanya menuliskan "list
       kosong" */

17.int NbElmt(List L);
     /* Mengirimkan banyaknya elemen list; mengirimkan 0
        jika list kosong */

18.infotype Max(List L);
     /* mengirimkan nilai info(P)yang maximum */

19.infotype Min(List L);
    /* mengirimkan nilai info(P) yang minimum */
```

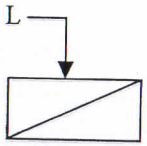


```
20.void InversList(List *L);
    /* I.S : sembarang */
    /* F.S : elemen list dibalik; elemen terakhir
       menjadi elemen pertama */
    /*    membalik elemen list, tanpa melakukan alokasi/
       dealokasi */

21.List CopyList(List L);
     /* mengirimkan list yang merupakan salinan L */

22.void CpAlokList(List Lin, List *Lout){
    /*I.S:Lin sembarang */
    /*F.S:jika semua alokasi berhasil, maka Lout berisi
       hasil copy Lin */
    /*jika ada alokasi yg gagal, maka Lout=Nil dan semus
       elemen yg  terlanjur dialokasi, didealokasi */
    /*dengan melakukan alokasi elemen */
    /*Lout adalah list kosong jika ada alokasi elemen yg
       gagal */

23.void konkat(List L1, List L2, List *L3){
    /* I.S: L1 dan L2 sembarang */
    /* F.S: L1dan L2 tetap,L3 adalah hasil konkatenasi
       L1&L2*/
    /* jika semua alokasi berhasil, maka L3 adalah hasil
       konkatenasi */
    /* jika ada alokasi yg gagal, semua elemen yg sdh
       dialokasi harus di dealokasi dan L3=Nil */
```

Dari primitif primitif di atas diimplementasikan sebagai berikut:

```
void CreateList(List *L){
    (*L)=Nil;
}
```

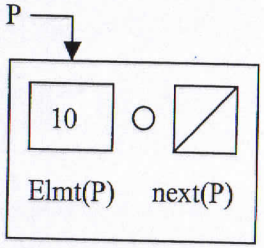
pemanggilan prosedur **CreateList (&L)** di atas akan menggambarkan list L dengan ilustrasi sebagai berikut :



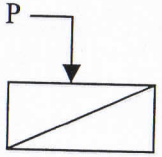
L=Nil

```
address Alokasi(infotype X){
    address P;
   P=(address)malloc(sizeof(ElmtList));
   if (P!=Nil)
   {
      info(P)=X;
      next(P)=Nil;
   }
      return P;
}
```

pemanggilan fungsi **Alokasi (10)** akan menggambarkan suatu node P sebagai berikut:



Jika alokasi berhasil



Jika alokasi gagal

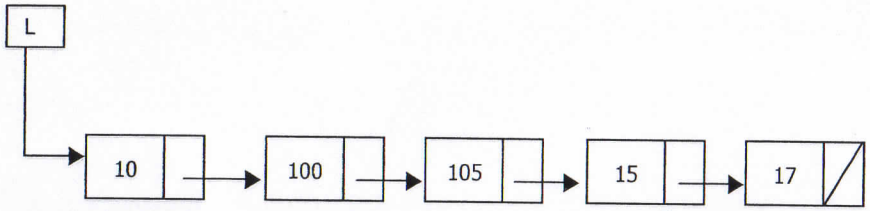
```
void Dealokasi(address P){
   free(P);
}
```


Pemanggilan prosedur Dealokasi(P) akan menghapus node P dan membebaskan memorinya.

```
boolean ListEmpty(List L){
    return (L==Nil)
}
boolean IsOneElmt(List L){
    return (tail(L)==Nil));
}
address Search(List L, infotype X){
    if (ListEmpty(L))
    {
       return Nil;
    }
    else
     {
       if(FirstElmt(L)==X)
       {
          return L;
       }
       else
       {
          Search(Tail(L),X);
        }
    }
}
```

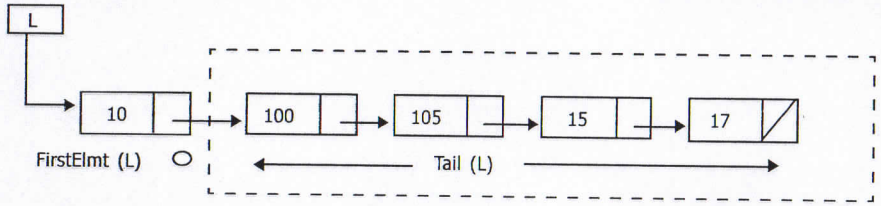
Pada fungsi search di atas yang dicek adalah elemen pertama, jika elemen pertama sama dengan yang dicari maka alamatnya akan dikirimkan, tetapi jika tidak ketemu dan list belum berakhir maka dilakukan proses secara rekursif kembali terhadap Tail dari list.

Contoh : jika list L adalah sebagai berikut :

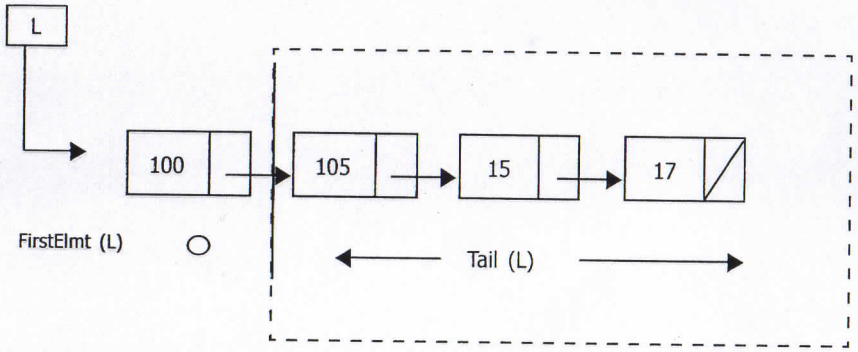


Maka ketika elemen 15 dicari dengan fungsi pemanggilan Search(L,15) proses pencariannya adalah sebagai berikut :

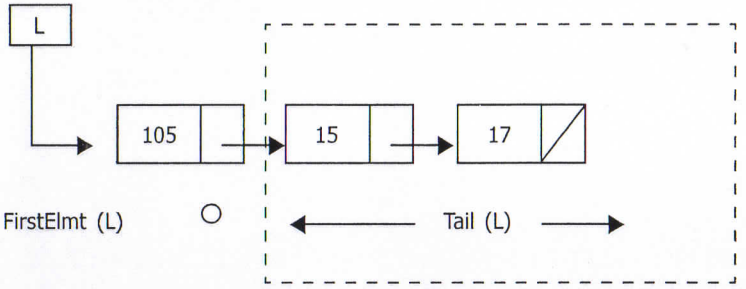
```
Search(L,15)

FirstElmt(L) == 15   ?                    /* 10 "" 15, false */
```




```
Search(Tail(L,15) , list L menjadi :
FirstElmt(L) ==15 ?                       /* 100 "" 15, false  */
```

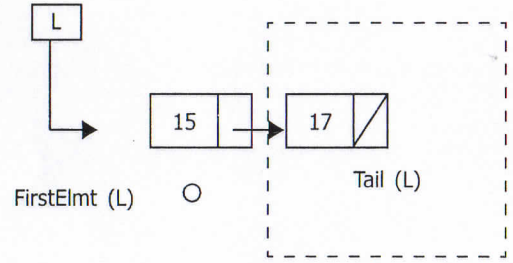


Search (Tail(L),15), list L menjadi :




```
FirstElmt(L)==15  ?                       /* 105 "" 15, false */
```

Search(Tail(L),15), list L menjadi :




```
FirstElmt(L)==15?                         /* 15=15, true */
```

Maka alamat elemen 15 akan dikirimkan oleh fungsi tersebut.

```
boolean Fsearch(List L, address P){
        if(ListEmpty(L))
        {
            return false;
        }
         else
        {
```

```
        if(L==P)
        {
          return true;
        }
        else
        {
          Fsearch(Tail(L),P);
        }
      }
  }
```

Fungsi ini sama seperti fungsi di atas menggunakan proses rekursif, hanya saja keluaran dari fungsi ini adalah boolean yaitu bernilai true atau false.

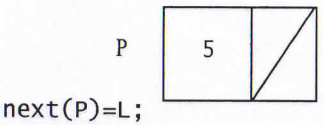
```
address SearchPrec(List L, infotype X){
      if(IsOneElmt(L))
      {
         return Nil;
      }
      else
      {
         if(info(Tail(L)==X)
         {
              return L;
         }
         else
         {
            SearchPrec(Tail(L),X);
          }
      }
  }
```


Fungsi ini juga fungsi pencarian tetapi fungsi ini mengirimkan alamat elemen sebelum elemen yang dicari, biasanya fungsi ini digunakan jika setelah search akan dilakukan operasi lain. Skema pemrosesannya juga rekursif hanya saja pada fungsi ini yang dicek adalah **nilai dari tail elemen** dari posisi *current*.

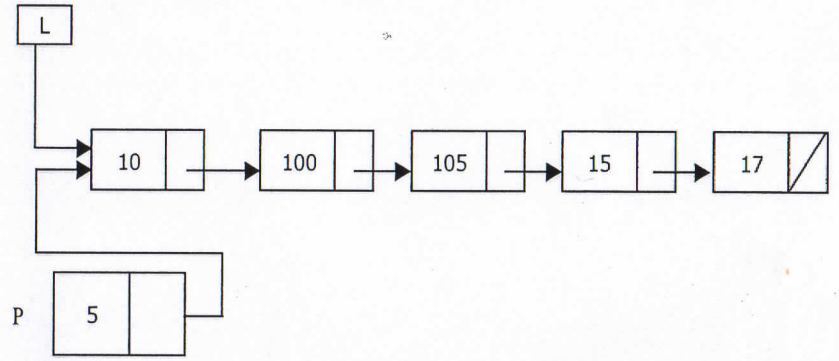
```
List Konso(List L, infotype X){
      address P;
      P=Alokasi(X);
      if(P==Nil)                    /* alokasi gagal */
      {
         return L;
      }
      else
      {
              next(P)=L;
              return P;
      }
  }
```

Untuk list L diatas, jika akan dilakukan sisip depan maka pemanggilan fungsi Konso (L,5) prosesnya adalah sebagai berikut :

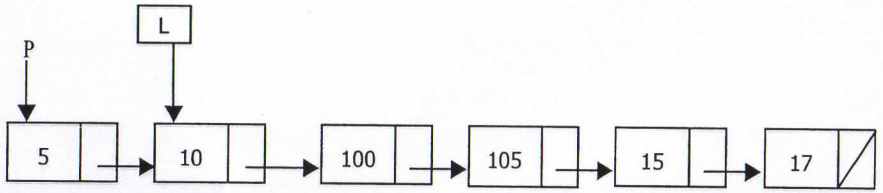
```
P=Alokasi(X);
```




```
next(P)=L;
```



return P; akan mengirimkan List P melalui parameter L;

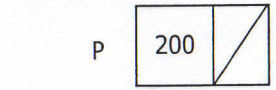



```
List Kons(List L, infotype X){
       address p,last;
       P=Alokasi(X);
       if(L==Nil)      /* list kosong */
       {
          return P;
       }
       else
       {
          last=L;
          while(next(last)!=Nil)
          {
                last=next(last);
           }
              next(Last)=P;
              return L;
       }
   }
```

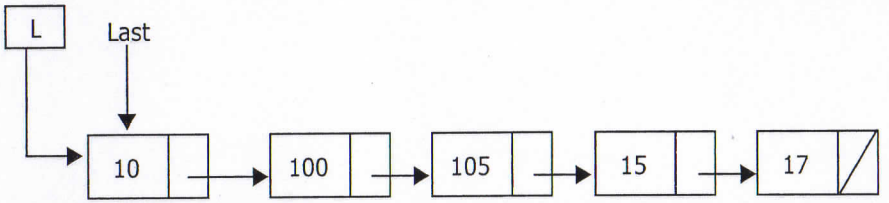
Fungsi ini adalah sisip belakang, pemanggilan fungsi ini terhadap list L berikut ini :

Kons(L,200) prosesnya adalah :

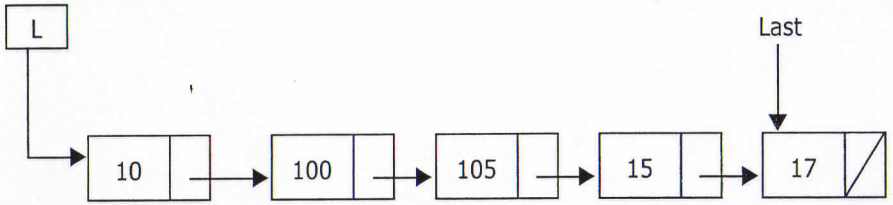
P=Alokasi(X);



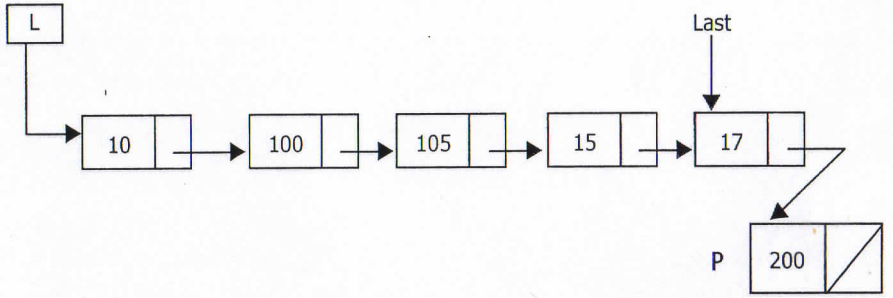
last=L;



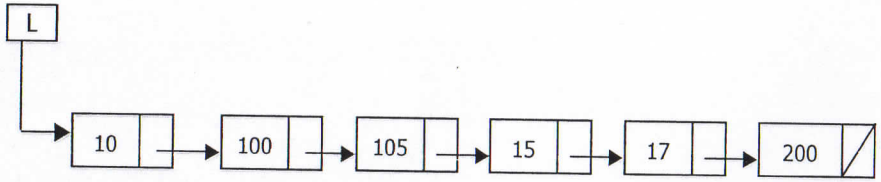

```
while(next(last)!=Nil)
   {
        last=next(last);
   }
```



next(Last)=P;



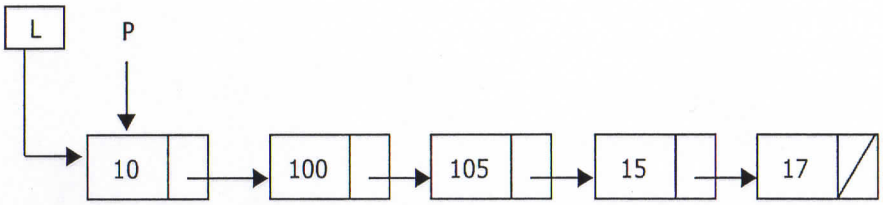
```
Return L;
```



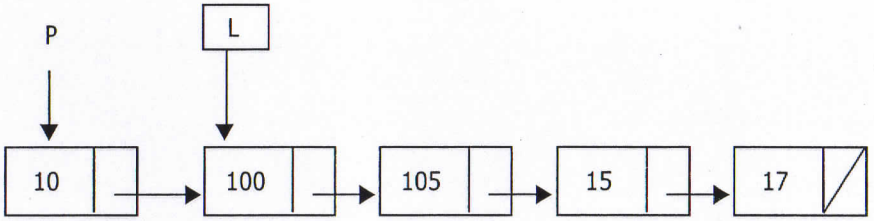

```
infotype FirstElmt(List L){
    return(info(L));
}

List Tail(List L){
   return(next(L));
}

void DelVFirst(List *L,infotype *X){
    address P;
     P=(*L);
     (*X)= FirstElmt(P);
     (*L)= Tail(P);
     Dealokasi(P);
}
```

jika dilakukan pada list L, maka pemanggilan prosedur di atas akan berakibat:

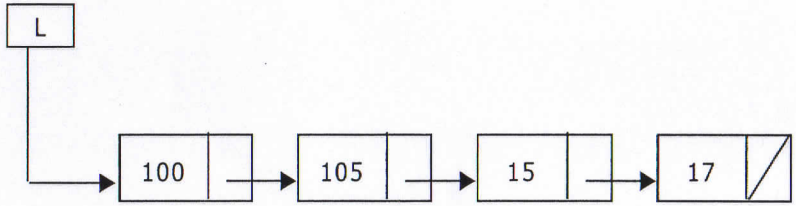
```
DelVFirst(&L, &X);

P=(*L);
```


```
(*X)=FirstElmt(P);                /* *X= 10   */
(*L)=Tail(P);
```




```
Dealokasi(P);
```

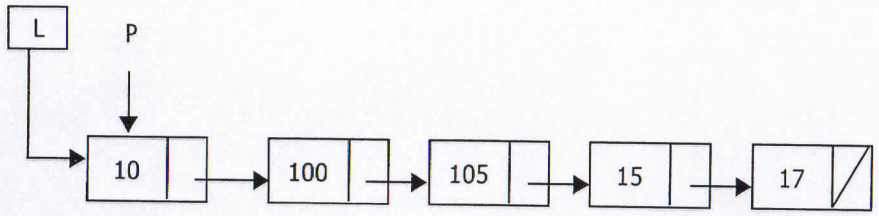



```
void DelVLast(List *L,infotype *X){
      address P;
      if(IsOneElmt(L)
      {
         *X=info(L);
        Dealokasi(L);
      }
      else
      (
```

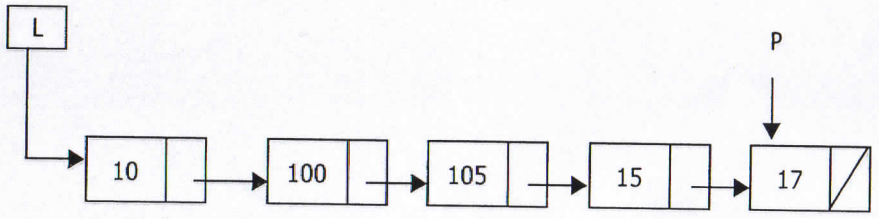
```
        P=*L;
        while (next(P)!= Nil)
        {
           P=next(P);
        }
        *X=info(P);
        Dealoaksi(P);
    }
}
```

untuk prosedur ini, jika list bukan 1 elemen maka address P akan travelsal sampai pada elemen terakhir list. Berikut adalah ilustrasi hapus elemen terakhir list untuk list L diatas. Karena L bukan list dengan satu elemen maka yang diproses adalah mulai dengan instruksi berikut:

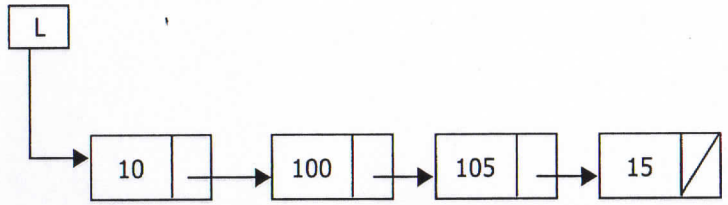
```
P=*L;
```




```
while (next(P)!= Nil)
 {
      P=next(P);
 }                                  /* next(P)=Nil*/
```




```
*X=info(P);          /* *X=17 */
Dealoaksi(P);
```




```
void DelAll(List *L){
   infotype X;
   while(!ListEmpty(*L))
   {
     DelVFirst(&L,&X);
   }
   /* List Empty */
}

void printinfo(List L){
       if (ListEmpty(L))
       {
          printf("[]");
       }
       else
       {
            printf("[%d] ",info(L));
          printinfo(next(L));
       }
       printf("\n");
   }

int NbElmt(List L){
       if (ListEmpty(L))
        {
```

```
            return 0;
          }
        else
        {
           return(1+ NBElmt(next(L)));
        }
    }

infotype Max(List L){
        if (info(L)> Max(next(L))
        {
           return info(L);
        }
        else
        {
           return Max(next(L));
        }
    }

infotype Min(List L){
        if (info(L)< Min(next(L))
        {
           return info(L);
        }
        else
        {
           return Min(next(L));
        }
    }

void InversList(List *L){
        if(ListEmpty(L))
        {
           CreateList(&L);
        }

        else

         {
```



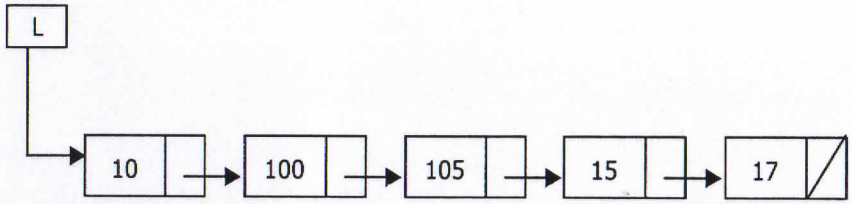
```
            return Kons(InversList(Tail(L)), FirstElmt(L));
        }
    )
```

Pemanggilan prosedur InversList(&L) pada list L akan berakibat sebagai berikut :

Intruksi pertama yang dijalankan adalah :

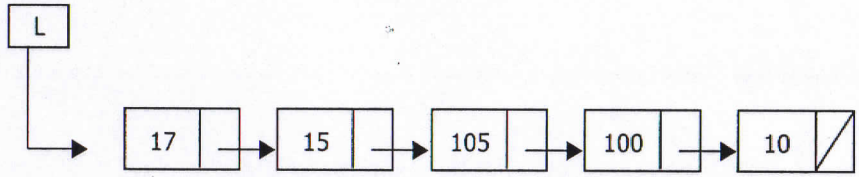
```
return Kons(InversList(Tail(L)), FirstElmt(L));
```

yaitu sisip belakang elemen pertama list dengan tail (L). List L sebelum proses adalah:

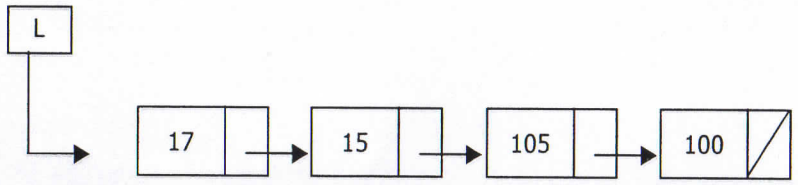


Kemudian dilakukan proses rekursif yang ditunjukkan pada bagian ini dari bawah ke atas sebagai berikut :

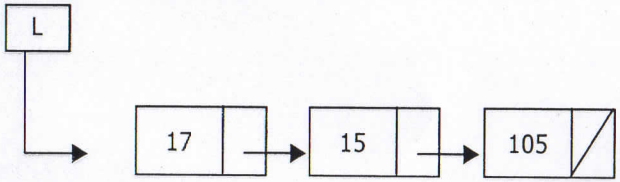
```
return Kons(Kons(Kons(Kons(Kons(InversList(Tail(L)),
FirstElmt(L)));
```



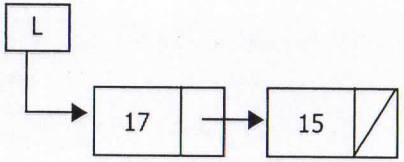

```
return
Kons(Kons(Kons(Kons(InversList(Tail(L)),FirstElmt(L))));
```



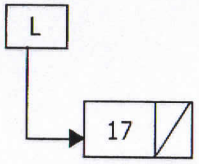

```
return Kons(Kons(Kons(InversList(Tail(L)),FirstElmt(L))));
```



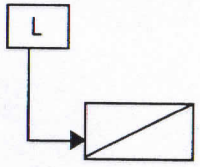

```
return Kons(Kons(InversList(Tail(L)),FirstElmt(L)));
```




```
return Kons(InversList(Tail(L)),FirstElmt(L));
```




```
CreateList(&L);
```



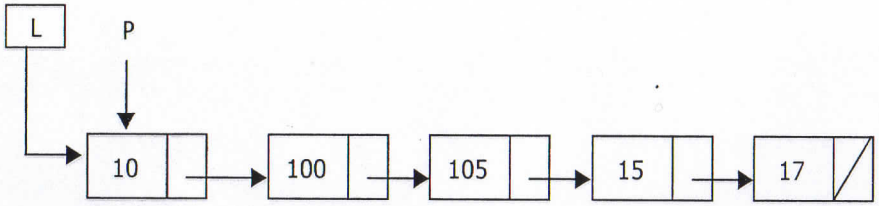

```
List CopyList(List L){
       if(ListEmpty(L))
       {
          CreateList(&L);
       }
       else
       {
          return Konso(CopyList(Tail(L)), FirstElmt(L));
       }
   )
```

Pada fungsi copy ini mirip dengan proses InversList, hanya saja pada fungsi ini dilakukan operasi sisip depan (Konso).

```
void CpAlokList(List Lin, List *Lout){
   address P;
   boolean berhasil;
   P= Lin;
   berhasil=true;
   while((P!=Nil)&& berhasil)
   {
      Kons(Lout, info(P));
      If (P==Nil)
      {
          berhasil=false
      }
   }
   /* P=Nil or alokasi tdk berhasil */
   if(!berhasil)
   {
      DelAll(Lout);
   }
}
```

Misal dilakukan pemanggilan pada fungsi CpAlokList(L,&L2) maka prosesnya diilustrasikan sebagai berikut:

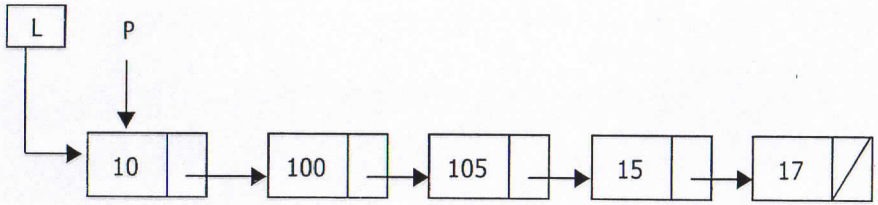
```
P = L;
```

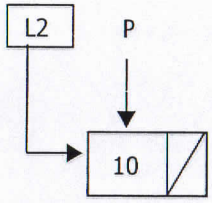



```
while((P!=Nil)&& berhasil)
 {
    Kons(Lout, info(P));
    If (P==Nil)
    {
        berhasil=false
     }
 }
 /* P=Nil or alokasi tdk berhasil */
```

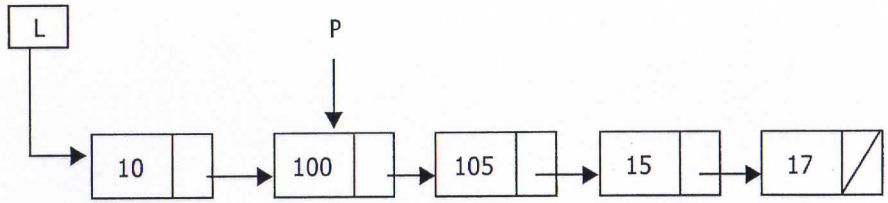
jika alokasi P selalu berhasil maka prosesnya adalah

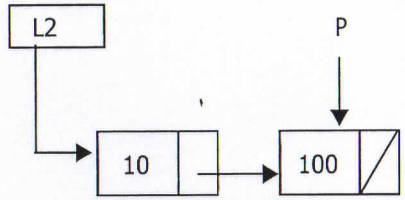
loop 1 : `Kons(Lout, info(P));`



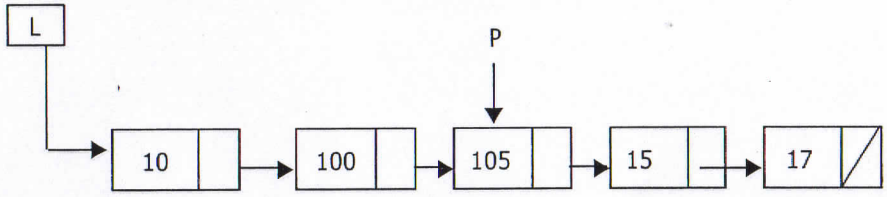


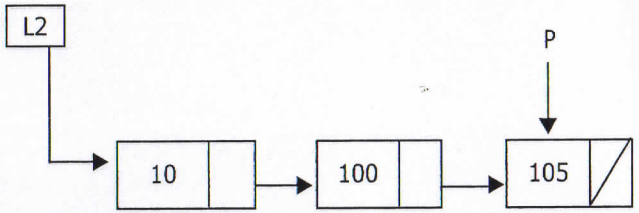
Loop2 : `Kons(Lout, info(P));`



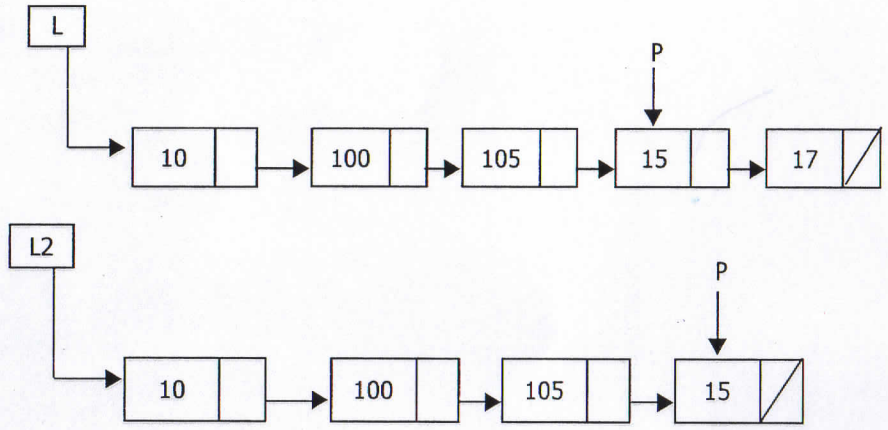


Loop3 : `Kons(Lout, info(P));`

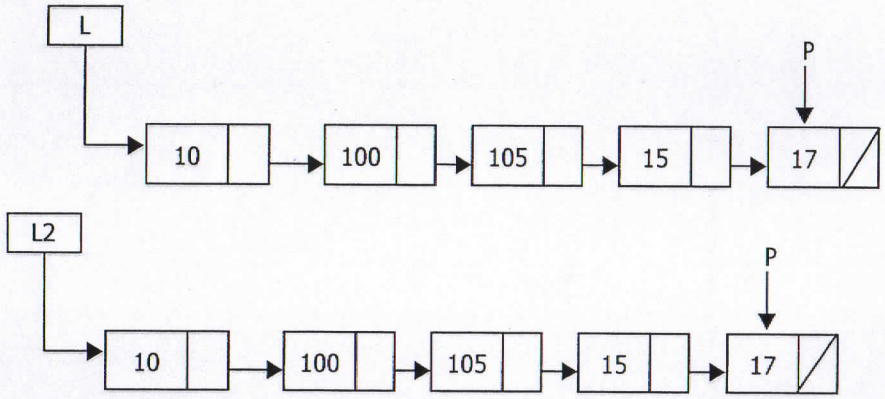




```
Loop4 : Kons(Lout, info(P));
```




```
Loop5 : Kons(Lout, info(P));
```



Loop 6 : P=Nil, loop stop


```
void konkat(List L1, List L2, List *L3){
   address P, PAlok;
   CreateList(L3);
   CpAlokList(L1,L3);
   CpAlokList(L2,L3);
}
```

proses pada fungsi ini sama dengan fungsi CpAlokList, hanya saja dilakukan terhadap list L2 dan L3

PENUTUP

ADT List berkait rekursif di atas memiliki defenisi rekursif dan memiliki 24 fungsi primitif. Dari 24 fungsi primitif 9 diantaranya memiliki proses rekursif. Struktur data yang dipakai untuk representasi fisik list adalah berkait dengan menggunakan pointer.

DAFTAR PUSTAKA

1. Kernighan B.W., Ritchie D.M., *The C Programming Language*, Prentice Hall, Second Edition, 1988.
2. Lewis, H.R., Denenberg, L., *Data Structures & Their Algorithms*, Addison-Wesley, 1991.
3. Liem I., *Diktat Kuliah IF 223 Algoritma dan Pemrograman*, Jurusan Teknik Informatika ITB, 1999
4. Liem I., *Diktat Struktur Data*, Jurusan Teknik Informatika ITB, 2001
5. Wirth, N., *Algorithms & Data Structures*, Prentice Hall, 1986