MODUL 1
LIST BERKAIT

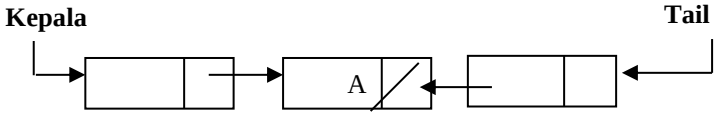
1.1 TUJUAN

Tujuan modul ini, adalah:

- Memperkenalkan penggunaan perangkat lunak bahasa pemrograman untuk mengimplementasikan struktur data (tipe data abstrak list berkait linear).
- Memperkenalkan instruksi-instruksi dan cara pembuatan program yang dapat digunakan untuk menerapkan list berkait linear.
- Melatih mahasiswa menyelesaikan masalah tentang list berkait linear dalam bentuk program aplikasi.

1.2 TEORI LIST BERKAIT SATU POINTER DAN DUA KEPALA (MATERI PERTEMUAN PERTAMA)

List dengan satu pointer dan dua kepala (S*ingle Linked-list & Double Head*) biasanya digunakan untuk memudahkan proses penambahan simpul yang dilakukan pada posisi akhir simpul, sehingga tidak perlu melakukan penelusuran dari simpul pertama sampai dengan simpul terakhir. List dengan satu pointer dan dua kepala (S*ingle Linked-list & Double Head*) adalah list yang memiliki pointer pada simpul *HEAD* (Kepala) diarahkan ke simpul pertama dan pointer pada simpul *TAIL* (ekor) diarahkan ke simpul terakhir. Setiap simpul mempunyai 1 buah pointer (*NEXT*) untuk menunjuk ke simpul sesudahnya (di sebelah kanannya). Pointer (NEXT) untuk simpul terakhir berisi NIL. Hal ini akan mempermudah pencarian elemen terakhir dari suatu list untuk penambahan simpul baru yang akan diletakan di pointer paling belakang atau akhir.



Gambar 1.1. List dengan satu pointer dan dua kepala

Deklarasi simpulnya adalah sebagai berikut :

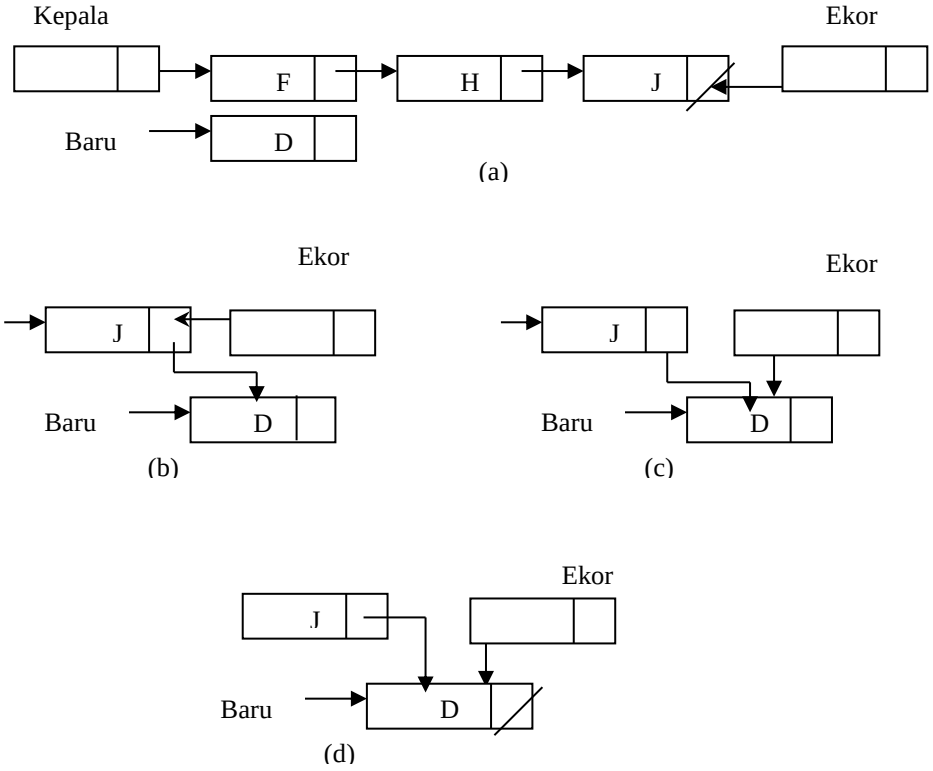
```
typedef struct List{
        char Info;
        List *Next;
        };

List *Head, *Tail;
```

```
//*******************************
// *  Program 1.1                          *
// *  Inisialisasi list dengan satu      *
// *   pointer dan dua kepala              *
// *******************************
void Inisialisasi(List *Head, List *Tail){
     Head = new(List);
     Head -> Next = NULL;
     Tail = new(List);
     Tail -> Next = NULL;
}
```

Program 1.1 Prosedur Inisialisasi List Berkait dengan Dua Kepala Satu Pointer

Proses penambahan simpul baru yang ditempatkan setelah simpul terakhir (ekor) dijelaskan sebagai berikut (lihat gambar 1.2.).

- Gambar 1.2 a. adalah keadaan sebelum simpul Baru yang berisi data 'D' ditambahkan. Pointer Kepala menunjuk ke simpul pertama dan pointer ekor menunjuk ke simpul terakhir.
- Gambar 1.2.b sampai Gambar 1.2.d hanya diambil sebagian saja.
- Penambahan dimulai dengan mengatur pointer Next pada simpul Terakhir diarahkan ke simpul Baru (Gambar 1.2.b).
- Gambar 1.2.c menunjukkan pointer Ekor diarahkan ke simpul Baru.
- Dan yang paling akhir adalah pointer Next dari simpul Baru diarahkan ke simpul Ekor (Gambar 1.2.d.).

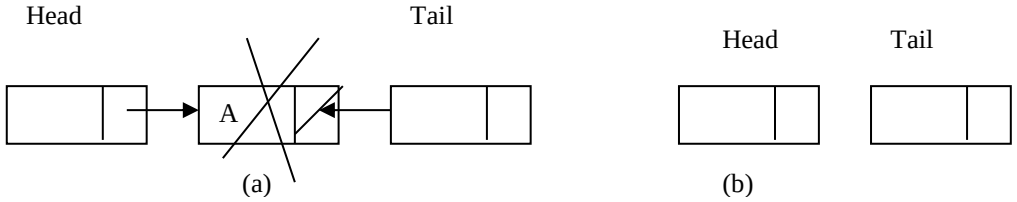


Gambar 1.2. Ilustrasi penambahan simpul di akhir

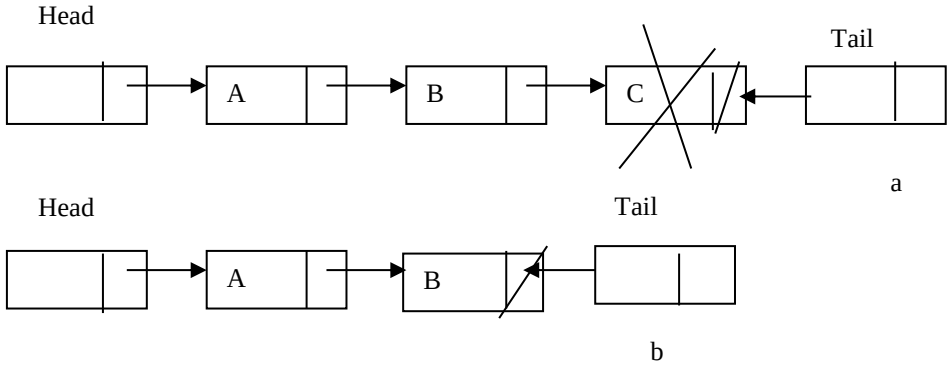
Secara lengkap, prosedur untuk penambahan simpul baru pada list berkait dengan satu pointer dan dua kepala adalah sebagai berikut :

```
//*************************************************************************
//* Program 1.2                                                           *
//* Prosedur untuk penambahan simpul baru di akhir pada list dengan satu pointer  *
//* dan 2 kepala                                                          *
// *************************************************************************
void Tambah(List *Head, List *Tail, char Elemen){
        Baru = new(List);
        Baru ->Info = Elemen;
        if (Head -> Next == NULL) //List masih kosong
        {
            Head -> Next = Baru;
        }
        else // List sudah berisi
        {
             (Tail->Next)->Next = Baru;
        }
        Tail -> Next = Baru;
        Baru -> Next = NULL;
}
```

Proses penghapusan simpul pada list berkait di atas dapat dilakukan di awal, di tengah, atau di akhir. Penghapusan di awal atau di tengah dapat dilakukan seperti penghapusan pada list berkait satu pointer satu kepala (pembahasan praktikum Algoritma & Pemrograman II). Namun untuk penghapusan di akhir terdapat penambahan satu intruksi, yaitu **menyambungkan Ekor dengan simpul terakhir yang baru.** Sebagai ilustrasi penghapusan dapat dilihat pada gambar 1.3.



Gambar 1.3. Penghapusan List pada List dengan Satu List



Gambar 1.4. Penghapusan List pada List dengan Banyak List

HAPUS_SIMPUL(Kepala, Ekor, Elemen);

```
//****************************************
//* Program 1.3                          *
//* Prosedur untuk menghapus simpul      *
//****************************************
void HAPUS_SIMPUL(List *Head, List *Tail, char Elemen){
   List *Bantu,*Hapus;
   Hapus = Head->Next;
   if (Head->Next == NULL){
        printf("List masih kosong\n");
   }
   else if (Hapus->Info == Elemen)  //simpul pertama dihapus atau simpul tunggal
   {
         if (Head->Next == Tail->Next) //simpul tunggal
         {
                Head->Next = NULL;
                Tail->Next = NULL;
         }
         else  // simpul pertama
         {
                 Head->Next = Hapus->Next;
         }
         printf("Data %d berhasil dihapus\n",NimHapus);
   }
   else{ //menghapus simpul tengah atau terakhir
         Bantu = NULL;
         //mencari simpul yang akan dihapus
         while (Elemen != Hapus->NIM  && Hapus->Next != NULL){
                      Bantu = Hapus;
                      Hapus = Hapus->Next;
          }
          if (Elemen == Hapus->NIM){
               if (Hapus->Next == NULL){ // simpul yang akan dihapus berada di akhir
                    Tail->Next = Bantu;
                }
               else{    // simpul tengah dihapus
                     Bantu->Next = Hapus->Next;
                }
               printf("Data %d berhasil dihapus\n",NimHapus);
           }
           else{ // simpul yang akan dihapus tidak ketemu
                      printf("Simpul yang dihapus tidak ada\n");
           }
      }
}
```

Program 1.3. Prosedur untuk Menghapus List

1.3 TUGAS & LATIHAN (PERTEMUAN PERTAMA)

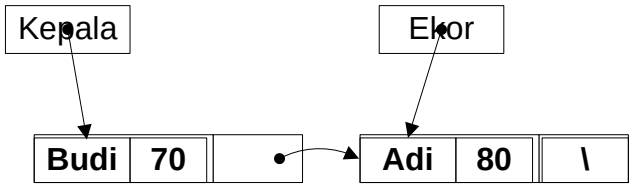
1.3.1 TUGAS PENDAHULUAN

Tugas pendahuluan dikumpulkan pada pertemuan pertama :

Contoh 1 elemen list:

Nama	Nilai	Next

Contoh list lengkap:



Diketahui struktur data untuk menggambarkan list berkait dengan satu pointer dan dua kepala (KEPALA & EKOR) untuk data sesuai contoh di atas adalah sebagai berikut.

```
type List2K : < Nim : integer,
                Nama : string,
                IPK : real,
                Next : ^List2K >
Head : ^List2K
Tail : ^List2K
```

Nim, Nama, dan IPK pada algoritma di atas diganti, disesuaikan dengan kasus kelompok Anda.

1. Dari struktur data tersebut, buat algoritma penambahan simpul pada posisi akhir (ekor) list berkait tersebut!
2. Buatlah algoritma penambahan simpul pada posisi awal list berkait tersebut!
3. Buatlah algoritma untuk menampilkan semua isi list data!
4. Buatlah algoritma untuk melakukan pencarian (searching) data pada list!
5. Buatlah algoritma untuk menghapus simpul pada list tersebut!

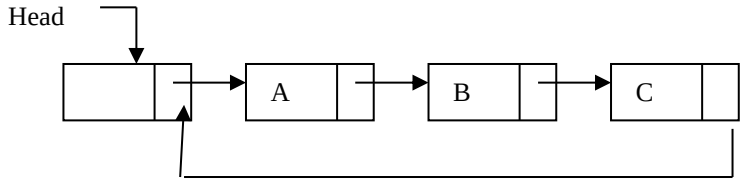
1.3.2 LATIHAN PRAKTIKUM

Latihan untuk pertemuan pertama :

1. Implementasikan struktur data dan algoritma pada tugas pendahuluan nomor 1-5 menggunakan C/C++ !
2. Satukan program tersebut dalam satu file dengan menggunakan program utama pemanggil fungsi-fungsi nomor 1-5. Lakukan pengamatan dan bandingkan dengan bentuk algoritma yang telah saudara buat pada tugas pendahuluan !
3. [Bonus] Kembangkan program saudara untuk menampilkan nilai maksimum-minimum dan nilai rata-rata !

1.3.3 TUGAS RUMAH

Tugas pendahuluan dikumpulkan pada pertemuan kedua :



Diketahui struktur data untuk menggambarkan list berkait sirkuler satu kepala untuk data sesuai contoh di atas adalah sebagai berikut.

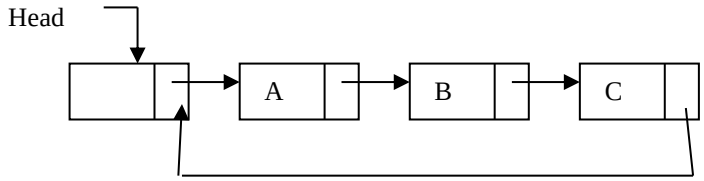
```
type ListS : < Nim : integer,
               Nama : string,
               IPK : real,
               Next : ^ListS >
Head : ^ListS
```

Nim, Nama, dan IPK pada algoritma di atas diganti, disesuaikan dengan kasus kelompok Anda.

1. Dari struktur data tersebut, buat algoritma penambahan simpul pada posisi akhir (ekor) list berkait tersebut!
2. Buatlah algoritma penambahan simpul pada posisi awal list berkait tersebut!
3. Buatlah algoritma untuk menampilkan semua isi list data!
4. Buatlah algoritma untuk melakukan pencarian (searching) data pada list!
5. Buatlah algoritma untuk menghapus simpul pada list tersebut!

1.4 TEORI LIST BERKAIT SIRKULER SATU KEPALA (MATERI PERTEMUAN KEDUA)

List berkait sirkuler satu kepala adalah list biasa dimana setiap simpulnya mempunyai 1 buah pointer (*NEXT*), dengan pointer *NEXT* dari simpul terakhir menunjuk ke HEAD. List ini **dapat dimanfaatkan untuk mempermudah pembacaan/penelusuran seluruh simpul yang tidak hanya dimulai dari posisi simpul awal list berkait.** Apabila pointer sedang menunjuk simpul di posisi tengah atau akhir, penelusuhan seluruh simpul tetap dapat dilakukan tanpa harus mengembalikan posisi pointer ke simpul awal terlebih dahulu.



Gambar 1.5. List berkait sirkuler dengan 1 kepala

Dengan keadaan seperti di atas, maka kita perlu menyesuaikan deklarasi simpulnya, yaitu kita ubah menjadi :

```
typedef struct{
                    char Info;
                    List *Next; // pointer ke kanan
            }List;
List *Head;
```

Pada deklarasi di atas, pointer Back adalah pointer untuk menunjuk ke simpul sebelumnya atau simpul sebelah kiri; pointer Next adalah pointer untuk menunjuk ke simpul sesudahnya atau simpul sebelah kanan. Untuk inisialisasi kita menginginkan pointer Back dan Next pada simpul kepala bernilai nil, sehingga kita bisa membuat prosedurnya :

```
//*********************************
//*  Program 1.4                  *
//*  Inisialisasi list sirkuler 1 kepala   *
//*********************************}
void INISIALISASI (List *Head){
        Head = new(List);
        Head->Next=NULL;
}
```

Program 1.4. Inisialisasi list berkait sirkuler 1 kepala

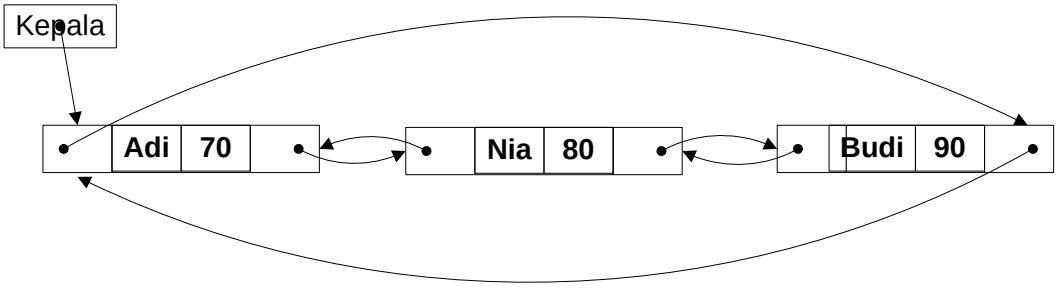
1.5 LATIHAN PRAKTIKUM (PERTEMUAN KEDUA)

Latihan untuk pertemuan kedua :

1. Implementasikan struktur data dan algoritma pada tugas pendahuluan nomor 1-5 menggunakan C/C++ !
2. Satukan program tersebut dalam satu file dengan menggunakan program utama pemanggil fungsi-fungsi nomor 1-5. Lakukan pengamatan dan bandingkan dengan bentuk algoritma yang telah saudara buat pada tugas pendahuluan !
3. [Bonus] Kembangkan program saudara untuk menampilkan nilai maksimum-minimum dan nilai rata-rata !

1.5.1 TUGAS RUMAH

Tugas pendahuluan dikumpulkan pada pertemuan ketiga :



Diketahui struktur data untuk menggambarkan list berkait dengan satu pointer dan dua kepala (KEPALA & EKOR) untuk data sesuai contoh di atas adalah sebagai berikut.

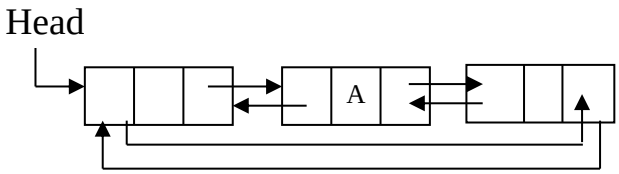
```
type List2P : < Nim : integer,
                Nama : string,
                Nilai : real,
                Back : ^List2P >
                Next : ^List2P >
Head : ^List2P
```

Nim, Nama, dan Nilai pada algoritma di atas diganti, disesuaikan dengan kasus kelompok Anda.

1. Buat struktur data untuk menggambarkan list berkait dua pointer (back & next) seperti contoh di atas!
2. Dari struktur data pada nomor 1, buat algoritma penambahan simpul pada list berkait tersebut agar terbentuk list berkait yang terurut dari kecil ke besar !
3. Buatlah algoritma untuk menampilkan semua isi list data!
4. Buatlah algoritma untuk melakukan pencarian (searching) data pada list!
5. Buatlah algoritma untuk menghapus simpul pada list tersebut!

1.6 TEORI LIST BERKAIT DUA POINTER DAN SIRKULER (MATERI PERTEMUAN KETIGA)

List ganda dengan dua pointer adalah list biasa dimana setiap simpulnya mempunyai 2 buah pointer (*NEXT & BACK*), dengan pointer *BACK* menunjuk ke simpul sebelumya (di sebelah kirinya) dan pointer *NEXT* menunjuk ke simpul sesudahnya (di sebelah kanannya). Hal ini akan mempermudah pembacaan (bisa dilakukan dari kiri ke kanan atau dari kanan ke kiri) dan penambahan simpul baru di akhir (tidakperlu dilakukan penelusuran dari simpul awal sampai akhir).



Gambar 1.5. List berkait ganda dengan 1 simpul

Dengan keadaan seperti di atas, maka kita perlu menyesuaikan deklarasi simpulnya, yaitu kita ubah menjadi :

```
typedef struct List{
        char Info;
        List *Next;
        List *Back;
};
List *Head;
```

Pada deklarasi di atas, pointer Back adalah pointer untuk menunjuk ke simpul sebelumnya atau simpul sebelah kiri; pointer Next adalah pointer untuk menunjuk ke simpul sesudahnya atau simpul sebelah kanan. Untuk inisialisasi kita menginginkan pointer Back dan Next pada simpul kepala bernilai nil, sehingga kita bisa membuat prosedurnya :

```
//*********************************
//*  Program 1.5                  *
//*  Inisialisasi list berkait ganda   *
//*********************************}
void INISIALISASI (List *Head){
      Head = new(List);
      Head -> Next = NULL;
      Head -> Back = NULL;
}
```

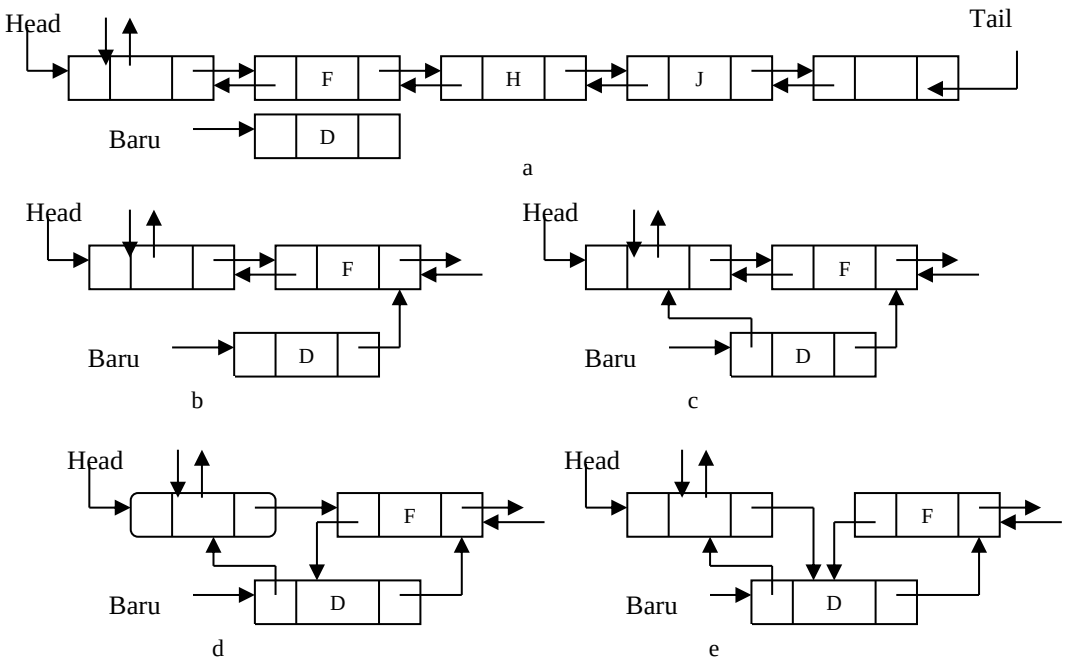
Program 1.5. Inisialisasi list berkait ganda

Proses penambahan simpul baru yang ditempatkan sesudah simpul kepala bisa dijelaskan sebagai berikut (lihat gambar 1.6).

- Gambar 1.6.a. adalah keadaan sebelum simpul Baru ditambahkan. Pointer Head menunjuk ke simpul kepala.
- Gambar 1.6.b sampai Gambar 1.6.e hanya diambil sebagian saja.
- Penambahan dimulai dengan mengatur pointer Next pada simpul Baru diarahkan ke simpul yang ditunjuk oleh pointer Next dari simpul Head (Gambar 1.6.b).
- Gambar 1.6.c menunjukkan pointer Back dari simpul Baru diarahkan ke simpul kepala.
- Selanjutnya pointer Back dari simpul sesudah simpul Kepala diarahkan ke simpul Baru (Gambar 1.6.d), dan yang paling akhir adalah pointer Next dari simpul Kepala diarahkan ke simpul Baru.

Sehingga secara garis besar bisa dituliskan sebagai (anda perlu memperhatikan urutannya agar jangan sampai pointernya terputus sehingga simpulnya tidak bisa dimasuk lagi) :

```
Baru ->Next   =  Head ->Next;
Baru -> Back      = Head;
(Kepala->Next)->Back = Baru;
Kepala ->Next = Baru;
```



Gambar 1.6. Ilustrasi penambahan simpul sesudah simpul kepala

Untuk penambahan simpul baru di tengah list, kita memerlukan bantuan pointer lain, misalnya Bantu dimana simpul Bantu ini diperoleh setelah dilakukan penelusuran untuk mencari posisi yang tepat untuk simpul Baru. List Baru akan kita sisipkan sebelum simpul yang ditunjuk oleh pointer Bantu. Dengan demikian kita bisa menggunakan ilustrasi pada Gambar 1.6. di atas, dengan mengganti pointer Kepala menjadi Bantu. Sehingga secara garis besar, proses penyisipannya adalah sebagai berikut (sekali lagi perlu diperhatikan supaya pointernya tidak terputus).

```
Baru ->Next := Bantu;
Baru ->Back := Bantu->Back;
(Bantu -> Back)->Next := Baru;
Bantu ->Back := Baru;
```

Dengan menggunakan gambar 1.6. sebagai ilustrasi maka untuk **penambahan simpul baru yang ditempatkan di akhir list**, dapat dilakukan sebagai berikut :

```
Baru->Next := Kepala;
Baru->Back := Kepala^.Back;
(Baru->Back)->Next := Baru;
Kepala->Back := Baru;
```

Secara lengkap, prosedur untuk penambahan simpul baru pada list berkait ganda adalah sebagai berikut :

```
//*****************************************************************
//* Program 1.6                                                   *
//* Prosedur untuk penambahan simpul baru pada list berkait ganda 2 kepala  *
//* dan sirkuler                                                  *
// *****************************************************************
void TAMBAH_SIMPUL(List *Head, char  Elemen){
   List *Bantu;
   Bantu = new(List);

   Baru = new(List);
   Baru -> Info = Elemen;
   if (Head -> Next == NULL) //List masih kosong
   {
      Head->Next = Baru;
      Baru->Back = Head;
      Baru->Next = Head;
      Head->Back = Baru;
   }
   else // List sudah berisi
   {
      if (Elemen > (Head->Back)->Info) { //penambahan simpul terakhir
            Baru->Next = Head;
            Baru->Back = Head->Back;
            (Head->Back)->Next = Baru;
            Head->Back = Baru;
      }
      else {
            Bantu = Head->Next;
            while (Bantu->Info < Elemen) { //mencari lokasi untuk simpul baru
               Bantu = Bantu->Next;
            }

            if (Bantu != Head->Next) {  //simpul di awal atau di tengah
                  Baru->Next = Bantu;
                  Baru->Back = Bantu->Back;
                  (Bantu->Back)->Next = Baru;
                  Bantu->Back = Baru;
            }
            else {  //simpul di awal
                  Baru->Next = Head->Next;
                  Baru->Back = Head;
                  (Head->Next)->Back = Baru;
                  Head->Next = Baru;
            }

      }
   }
}
```

Program 1.6 Prosedur penambahan simpul baru pada list berkait ganda dua kepala

1.7 LATIHAN PRAKTIKUM (PERTEMUAN KETIGA)

Latihan untuk pertemuan ketiga :

1. Implementasikan struktur data dan algoritma pada tugas pendahuluan nomor 1-5 menggunakan C/C++ !
2. Satukan program tersebut dalam satu file dengan menggunakan program utama pemanggil fungsi-fungsi nomor 1-5. Lakukan pengamatan dan bandingkan dengan bentuk algoritma yang telah saudara buat pada tugas pendahuluan !
3. [Bonus] Kembangkan program saudara untuk menampilkan nilai maksimum-minimum dan nilai rata-rata !