BAB 6
LINKED LIST

1. Tujuan Instruksional Umum

a. Mahasiswa dapat melakukan perancangan aplikasi menggunakan struktur Linked List (Senarai Berkait)
b. Mahasiswa mampu melakukan analisis pada algoritma *Linked List* yang dibuat
c. Mahasiswa mampu mengimplementasikan algoritma *Linked List* pada sebuah aplikasi secara tepat dan efisien

2. Tujuan Instruksional Khusus

a. Mahasiswa dapat menjelaskan mengenai *Linked List*
b. Mahasiswa dapat membuat dan mendeklarasikan Abstraksi Tipe Data *Linked List*
c. Mahasiswa mampu menerapkan operasi *Single Linked List Non Circular* : *Insert, update, dan delete*
d. Mahasiswa mampu menerapkan operasi *Single Linked List Circular* : *Insert, update, dan delete*

Pengertian Linked List

Salah satu bentuk struktur data yang berisi kumpulan data yang tersusun secara sekuensial, saling bersambungan, dinamis dan terbatas adalah senarai berkait (*linked list*). Suatu senarai berkait (*linked list*) adalah suatu simpul (*node*) yang dikaitkan dengan simpul yang lain dalam suatu urutan tertentu. Suatu simpul dapat berbentuk suatu struktur atau *class*. Simpul harus mempunyai satu atau lebih elemen struktur atau *class* yang berisi data.

Secara teori, *linked list* adalah sejumlah node yang dihubungkan secara linier dengan bantuan *pointer*. Dikatakan *single* (*singly*) linked apabila hanya ada satu pointer yang menghubungkan setiap node. *single* artinya *field pointer*-nya hanya satu buah saja dan satu arah.

Senarai berkait adalah struktur data yang paling dasar. Senarai berkait terdiri atas sejumlah unsur-unsur dikelompokkan, atau terhubung, bersama-sama di suatu deret yang spesifik. Senarai berkait bermanfaat di dalam memelihara koleksi-koleksi data, yang serupa dengan *array*/larik yang sering digunakan. Bagaimanapun juga, senarai berkait memberikan keuntungan-keuntungan penting yang melebihi array/larik dalam banyak hal. Secara rinci, senarai berkait lebih efisien di dalam melaksanakan penyisipan-penyisipan dan penghapusan-penghapusan. Senarai berkait juga menggunakan alokasi penyimpanan secara dinamis, yang merupakan penyimpanan yang dialokasikan pada *runtime*. Karena di dalam banyak aplikasi, ukuran dari data itu tidak diketahui pada saat kompile, hal ini bisa merupakan suatu atribut yang baik juga. Setiap *node* akan berbentuk *struct* dan memiliki satu buah *field* bertipe *struct* yang sama, yang berfungsi sebagai *pointer*. Dalam menghubungkan setiap node, kita dapat menggunakan cara *first-create-first-access* ataupun *first-create-last-access*. Yang berbeda dengan deklarasi *struct* sebelumnya adalah satu *field* bernama *next*, yang bertipe *struct tnode*. Hal ini sekilas dapat membingungkan. Namun, satu hal yang jelas, variabel *next* ini akan menghubungkan kita dengan *node* di sebelah kita, yang juga bertipe *struct tnode*. Hal inilah yang menyebabkan *next* harus bertipe *struct tnode*.

Bentuk Umum :

```
typedef struct telmtlist
    {
      infotype info;
      address next;
    }elmtlist;
```

infotype → sebuah tipe terdefinisi yang menyimpan informasi sebuah elemen list
next → address dari elemen berikutnya (suksesor)
Jika L adalah list, dan P adalah address, maka alamat elemen pertama list L dapat diacu dengan notasi : `first (L)`
Sebelum digunakan harus dideklarasikan terlebih dahulu :

```
#define First(L) (L)
```

Elemen yang diacu oleh P dapat dikonsultasi informasinya dengan notasi :

```
info(P) deklarasi #define info(P) P->info
next(P) deklarasi #define next(P) P->next
```

Beberapa Definisi :
1. List l adalah list kosong, jika First(L) = Nil
2. Elemen terakhir dikenali, dengan salah satu cara adalah karena Next(Last) = Nil
Nil adalah pengganti Null, perubahan ini dituliskan dengan #define Nil Null

Senarai berkait tunggal (Single Linked List)
Senarai berkait yang paling sederhana, di mana unsur-unsur terhubung oleh suatu pointer. Struktur ini mengizinkan senarai dilintasi dari elemen pertama sampai elemen terakhir.

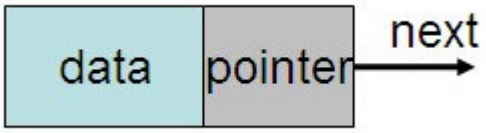
Senarai berkait ganda (Double Linked List)
Senarai berkait di mana unsur-unsur yang terhubung oleh dua pointer sebagai gantinya. Struktur ini mengizinkan senarai untuk dilintasi kedua-duanya secara maju mundur.

Senarai sirkular (Circular List)
Senarai berkait di mana elemen yang terakhir terhubung dengan elemen yang pertama sebagai ganti set ke NULL. Struktur ini mengizinkan senarai untuk dilintasi secara lingkar.

Abstraksi Tipe Data Single Linked List Non Circular

Pembuatan struct bernama tnode berisi 2 field, yaitu field data bertipe integer dan field next yang bertipe pointer dari tnode.
Deklarasi node dengan struct:

```
struct tnode
{
 int data;
 struct tnode *next;
}
```



Gambar 1. Sebuah node pada Single Linked List

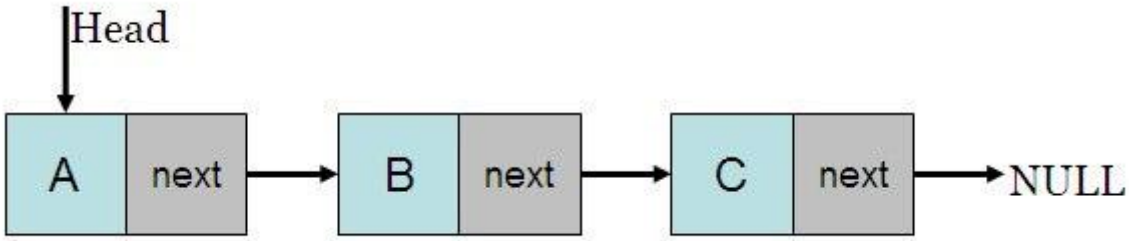
Asumsikan kita memiliki sejumlah *node* yang selalu menoleh ke sebelah dalam arah yang sama. Atau, sebagai alat bantu, kita bisa mengasumsikan beberapa orang yang bermain kereta api. Yang belakang akan memegang yang depan, dan semuanya menghadap arah yang sama. Setiap pemain adalah *node*. Dan tangan pemain yang digunakan untuk memegang bahu pemain depan adalah variabel *next*. Sampai di sini, kita baru saja mendeklarasikan tipe data dasar sebuah *node*. Selanjutnya, kita akan mendeklarasikan beberapa variabel *pointer* bertipe *struct tnode*. Beberapa variabel tersebut akan kita gunakan sebagai awal dari *linked list*, *node* aktif dalam *linked list*, dan *node* sementara yang akan digunakan dalam pembuatan *node* di *linked list*. Berikan nilai awal *NULL* kepada mereka. Deklarasi node untuk beberapa keperluan, seperti berikut ini:

```
struct tnode *head=NULL, *curr=NULL, *node=NULL;
```

Dengan demikian, sampai saat ini, telah dimiliki tiga *node*. Satu sebagai kepala (*head*), satu sebagai *node* aktif dalam *linked list* (*curr*) dan satu lagi *node* sementara (*node*).

Sekarang telah siap untuk membuat *single linked list*. Untuk itu, akan dimasukkan nilai tertentu sebagai nilai untuk *field* data dengan cara melakukan perulangan sehingga campur tangan *user* tidak diperlukan.

Seperti yang diungkapkan sebelumnya, bahwa akan dibuat *Single Linked List* (SLL) dengan cara first-create-first-access. Dengan cara ini, node yang dibuat pertama akan menjadi head. Node-node yang dibuat setelahnya akan menjadi node-node pengikut. Untuk mempermudah pengingatan, ingatlah gambar anak panah yang mengarah ke kanan. Head akan berada pada pangkal anak panah, dan node-node berikutnya akan berbaris ke arah bagian anak panah yang tajam.



Gambar 2 Single Linked List

Apabila diperhatikan, setiap node memiliki petunjuk untuk node sebelahnya. Node terakhir akan diberikan nilai NULL. Dengan demikian, setiap node kebagian jatah.

```
int i;
for (i=0; i<5; i++)
{
   node = (struct tnode *)
   malloc (sizeof(struct tnode));
   node -> data = i;
   if (head == NULL)
   {
        head = node;
        curr = node;
   }else
   {
        curr -> next = node;
        curr = node;
   }
}
curr -> next = NULL;
```

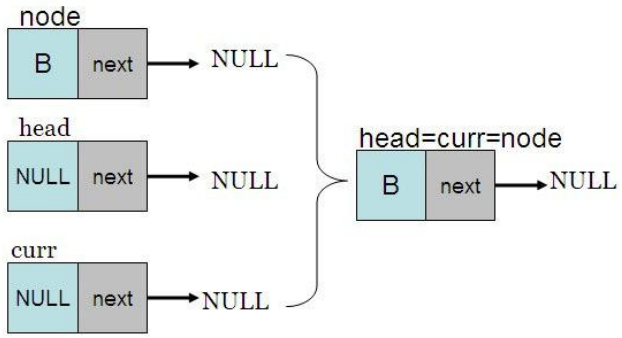
Berikut adalah penjelasan kode-kode pembuatan singly linked list tersebut. Pertama-tama, akan dibuat perulangan dari 0 sampai 4, yang dimaksudkan untuk membuat lima buah node yang masing-masing field data nya berisikan nilai dari 0 sampai 4. Pembuatan node dilakukan dengan fungsi malloc().

```
for (i=0; i<5; i++)
{
      node = (struct tnode *)
      malloc (sizeof(struct tnode));
      node -> data = i;
      ...
      ... }
```

Setelah itu, perlu dibuat node dan penghubung. Pertama-tama, akan diuji apakah head bernilai NULL. Kondisi head bernilai NULL hanya terjadi apabila belum dimiliki satu node pun. Dengan demikian, node tersebut akan dijadikan sebagai head. Node aktif (curr), juga kita dapat dari node tersebut.

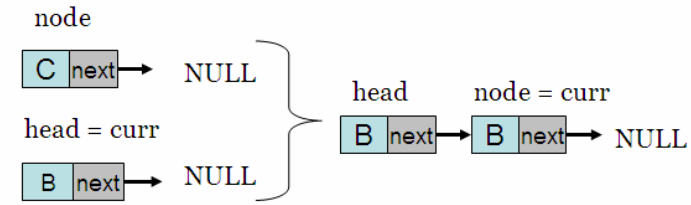
Kalau head tidak bernilai NULL alias telah dimiliki satu atau lebih node, yang pertama dilakukan adalah menghubungkan pointer next dari node aktif (curr) ke node yang baru saja dibuat. Dengan demikian, baru saja dibuat penghubung antara rantai lama dengan mata rantai baru. Atau, dalam permainan kereta api, pemain paling depan dalam barisan lama akan menempelkan tangannya pada bahu pemain yang baru bergabung. Node aktif (curr) kemudian dipindahkan ke node yang baru dibuat.

```
if (head == NULL)
{
      head = node;
      curr = node;
}
```



Gambar 3 Membuat elemen pertama SLL

```
else
{
      curr->next=node;
      curr = node;
}
```



Gambar 4. Penambahan elemen dibelakang SLL

Setelah semuanya dibuat, sudah saatnya bagi kita untuk menghubungkan pointer next untuk mata rantai terakhir ke NULL.

```
curr -> next = NULL;
```

Single linked list baru saja diciptakan. Agar kita tahu bahwa linked list yang kita ciptakan adalah linked list yang benar, salah satu cara untuk mengetahuinya adalah dengan mencetak field x untuk semua node. Dari head sampai node terakhir.

```
curr = head;
while (curr != NULL)
{
printf("%d ", curr -> data);
curr = curr -> next;
}
printf("\n");
```

Pertama-tama, kita meletakkan node aktif (curr) ke posisi head. Setelah itu, kita akan pindahkan kunjungi satu per satu node dengan memindahkan node aktif (curr) ke posisi sebelahnya. Semua kunjungan tersebut akan kita lakukan apabila node aktif (curr) tidak menemui NULL. Anda mungkin ingin menambahkan pemanggilan free() untuk melakukan dealokasi.

Kode selengkapnya:

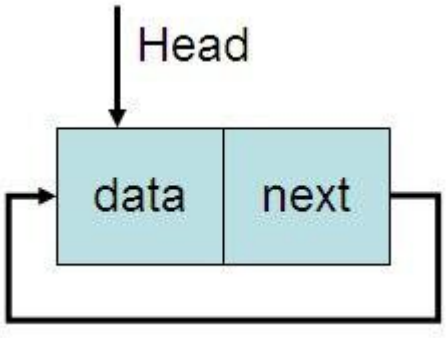
```
#include <stdio.h>
#include <stdlib.h>
int main()
{
      struct tnode
{
      int data;
      struct tnode *next;
};
struct tnode *head=NULL,
*curr=NULL, *node=NULL;
int i;
for (i=0; i<5; i++)
{
      node = (struct tnode *)
      malloc (sizeof(struct tnode));
      node -> data = i;
      if (head == NULL)
      {
           head = node;
           curr = node;
      }else
      {
           curr -> next = node;
           curr = node;
      }
}
curr -> next = NULL;
curr = head;
while (curr != NULL)
{
      printf("%d ", curr -> data);
      curr = curr -> next;
}
printf("\n");
free();
return 0;
}
```

Abstraksi Tipe Data Single Linked List Circular

Hampir sama dengan single linked list non circular, bahwa dibutuhkan sebuah kait untuk menghubungkan node-node data yang ada, dimana pada node terakhir atau tail yang semula menunjukkan NULL diganti dengan menunjuk ke kepala atau head. Dimana inisialisasi senarai berkait tunggal sirkular menggunakan struc adalah sebagai berikut:

Deklarasi Single Linked List Circular:

```
Struct tnode
{
      int data;
      tnode *next;
};
void main()
{
      head = new tnode;
      head->next = head;
}
```

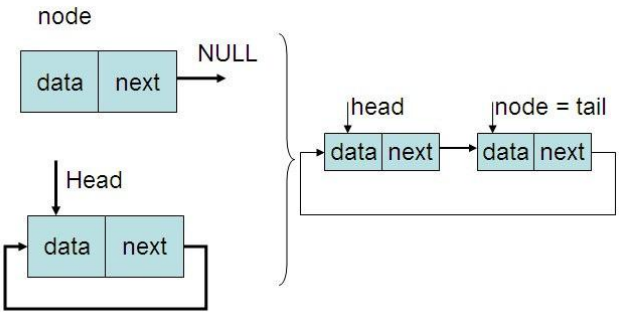


Gambar 5. Single Linked List circular

Menambah node dan membuat tail dari single linked list circular

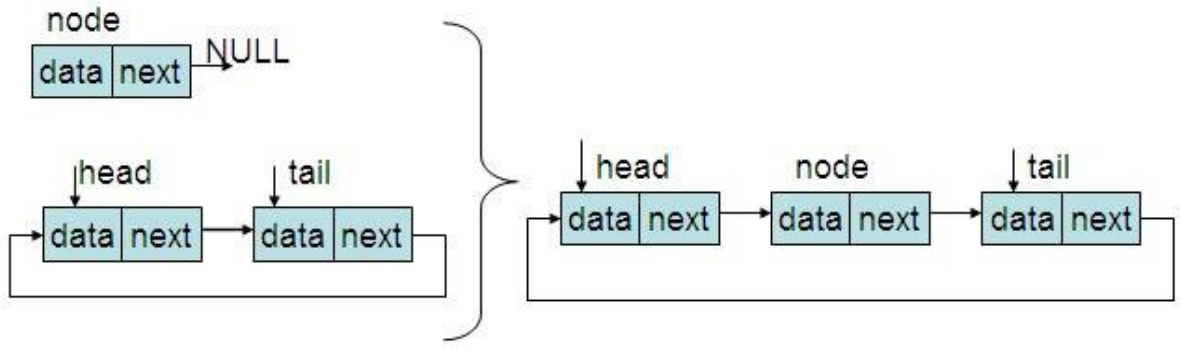
Deklarasi penambahan node baru:

```
Void main()
{
      node = new tnode;
      tail = new tnode;
      node->next = head->next;
      head->next = node;
      tail = node;
}
```



Gambar 6. Penambahan Node baru

Menyisipkan Node baru

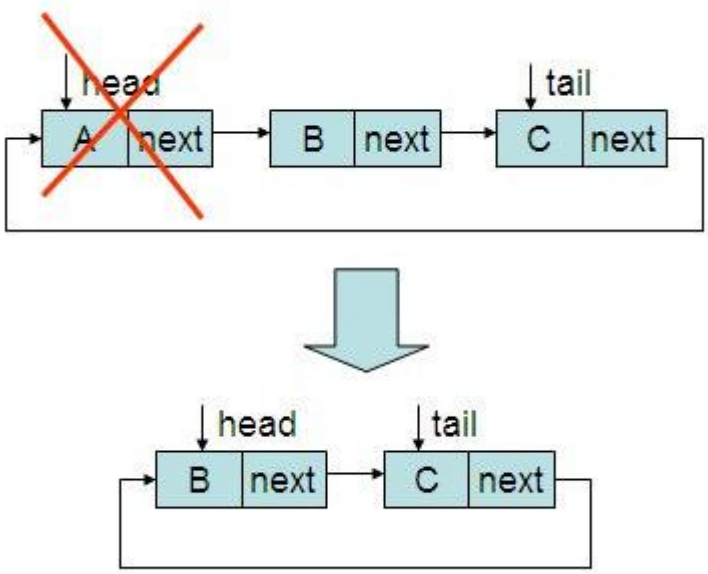


Gambar 7 Menyisipkan Node Baru

Deklarasi menyisipkan node baru menggunakan sintak berikut:

```
Void main()
{
      node = new tnode;
      node->next = head->next;
      head->next = node;
}
```

Menghapus Node dari Single Linked List Circular



Gambar 8. Menghapus node dari SLLC

Deklarasi menghapus node dari single linked list circular, menggunakan sintaks berikut:

```
Void main()
{
      hapus = new tnode;
      if( head != tail)
      {
            hapus = head;
            head = head->next;
            tail->next = head;
            delete hapus;
      }else
      {
            head = NULL;
            tail = NULL;
      }
 }
```

Latihan

1. Membuat Single Linked List

```
#include<iostream.h>
#include<stdlib.h>
#include<malloc.h>
#include<conio.h>

#define Nil NULL
#define info(P) P->info
#define next(P) P->info
#define First(L) NULL
```

```
typedef int InfoType;
typedef struct telmtlist address;
typedef struct telmtlist
{
   InfoType info;
   address next;
}elmtlist;

typedef address list;
void CiptaSenarai(list *L)
{
   First(L)=Nil;
}

list NodBaru(int m)
{
   list n;
   n=(list) (sizeof(elmtlist));
   if(n!=NULL)
   {
        info(n)=m;
        next(n)=Nil;
   }
   return n;
}

void SisipSenarai(list *L,list t,list p)
{
   if(p==Null)
   {
        t->next=*L;
        *L=t;
   }
   else
   {
        t->next=p->next;
        p->next=t;
   }
}

void CetakSenarai(list L)
{
   list ps;
   for(ps=L;ps!=Nil;ps=ps->next)
   {
        cout<<"    "<<info()<<"-->";
   }
   cout<<"NULL"<<endl;
}
```

```
int main()
{
   list pel;
   list n;
   int i,k,nilai;

   CiptaSenarai(&pel);
   cout<<"Masukkan banyak data=";
   cin>>k;
   for(i=1;i<=k;i++)
   {
        cout<<"Masukkan Data Senarai ke-"<<i<<" = ";
        cin>>nilai;
        n=NodBaru(nilai);
        SisipSenarai(pel,n,NULL);
   }

   CetakSenarai(pel);
   return 0;
}
```

2. Pencarian Nilai Terkecil dan Nilai Terbesar dalam sebuah Single Linked List

```
#include<iostream.h>
#include<stdlib.h>
#include<malloc>
#include<conio.h>

#define Nil NULL
#define info(P) P->info
#define next(P) P->info
#define First(L) (L)

typedef int InfoType;
typedef struct telmtlist *address;
typedef struct elmtlist
{
   InfoType info;
   address next;
}elmtlist;

typedef address list;
void CiptaSenarai(list *L)
{
   First(*L)=Null;
}

list NodBaru(int m)
{
   list n;
   n=(list) malloc(size(elmtlist));
```

```
    if(n!=NULL)
    {
         info(n)=m;
         next(n)=Nil;
    }
    return n;
}

void SisipSenarai(list *L,list t,list p)
{
    if(p==Nil)
    {
         t->next=*L;
         *L=t;
    }
    else
    {
         t->next=p->info;
         p->next=t;
    }
}

void CetakSenarai(list L)
{
    list ps;
    for(ps=L;ps!=Nil;ps=ps->next)
    {
         cout<<"    "<<info(ps)<<"-->";
    }
    cout<<"NULL";
}

InfoType Max(list L)
{
    address Pmax,Pt;

    Pmax=First(L);

    if(next(Pmax)==Nil)
    {
         return(info(Pmax));
    }
    else
    {
         Pt=next(Pmax);
         while(Pt!=Nil)
         {
              if(info(Pmax)<info(Pt))
              {
                   Pmax=Pt;
```

```
            }
            else
            {
                  Pt=next(Pt);
            }


      return(info(Pmax));
   }
}

InfoType Min(list L)
{
   address Pmin,Pt;

   Pmin=First (L);

   if(next(Pmin)==Nil)
   {
      return(info(Pmin));
   }
   else

      Pt=next(Pmin);
      while(Pt!=Nil)
      {
            if(info(Pmin)>info(Pt))
            {
                  Pmin=Pt;
            }
            else
            {
                  Pt=next(Pt);
            }
      }

      return(info(Pmin));
   }
}

void main()
{
   list pel;
   list n;
   int i,k,nilai,maks,min;

   CiptaSenarai(&pel);
   cout<<"Masukkan banyak data=";
   cin>>k;
   for(i=1;i<=k;i++)
```

```
    {
        cout<<"Masukkan Data Senarai ke-"<<k<<" = ";
        cin>>nilai;
        n=NodBaru(nilai);
        SisipSenarai(&pel,n,NULL);
    }

    cout<<endl;
    CetakSenarai(pel);
    maks=Max(pel);
    min=Min(pel);

    cout<<endl;
    cout<<"Nilai Terbesar:"<<max;
    cout<<endl;
    cout<<"Nilai Terkecil:"<<min;
}
```

TUGAS

Pembuatan Single Linked List dapat menggunakan 2 metode:

- LIFO (Last In First Out), aplikasinya : Stack (Tumpukan)
- FIFO (First In First Out), aplikasinya : Queue (Antrean)

Buatlah 1 program dengan menggunakan 2 metode tersebut!