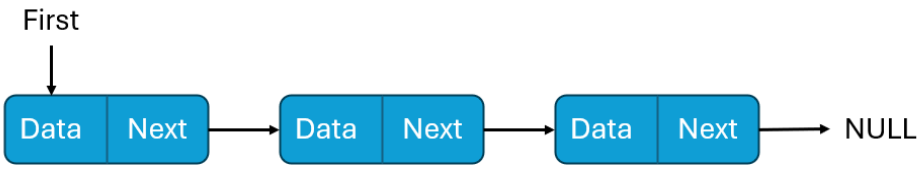
MODUL 7

LINKED LIST 2

Tujuan

1. Praktikan dapat semakin memahami dan menguasai konsep linked list.
2. Praktikan dapat menggunakan pemahaman atas konsep linked list untuk mengembangkan linked list.

Dasar Teori



Modul Sebelumnya Praktikan telah diperkenalkan dengan konsep dari Linked List secara dasar dan Modul ini akan membahas Insert After dan Delete At.

Pada dasarnya, sebuah linked list beserta dengan data-datanya akan tersimpan secara dinamis dalam memory komputer karena menggunakan konsep malloc. Untuk mengakses data tersebut, linked list menggunakan pointer yang menunjuk ke tempat penyimpanan data tersebut. Sebuah elemen di dalam linked list terdiri dari data yang disimpan pada alamat elemen tersebut dan juga alamat untuk data pada elemen selanjutnya di dalam linked list.

Elemen dalam linked list akan berisi 2 tipe data, yaitu datanya sendiri sesuai dengan kebutuhan **(DATA)** dan alamat elemen selanjutnya pada linked list **(NEXT)**. Untuk memanggil elemen pertama, linked list akan menyimpan alamat dari elemen pertama dengan pointer pada bagian head. Untuk mengakses ke data selanjutnya, akan digunakan pemanggilan pointer Next yang menunjuk kepada elemen selanjutnya. Jika sudah berada pada elemen yang terakhir, pointer Next pada elemen tersebut akan menjunjuk ke NULL seperti yang ditunjukan pada gambar diatas.

Operasi-Operasi pada Linked List:

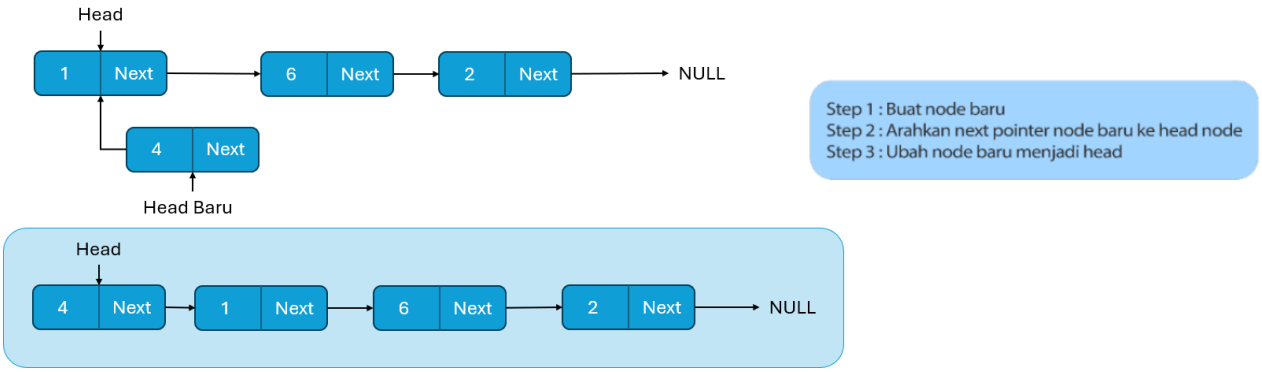
2(dua) Operasi yang paling penting pada linked list adalah insert dan delete dimana **Insert** ditujukan untuk memasukkan data pada linked list, dan **Delete** bertujuan untuk menghapus data dari linked list.

INSERT

Beberapa operasi insert pada linked list adalah sebagai berikut:

- InsertFirst
- InsertLast
- InsertAfter [New]

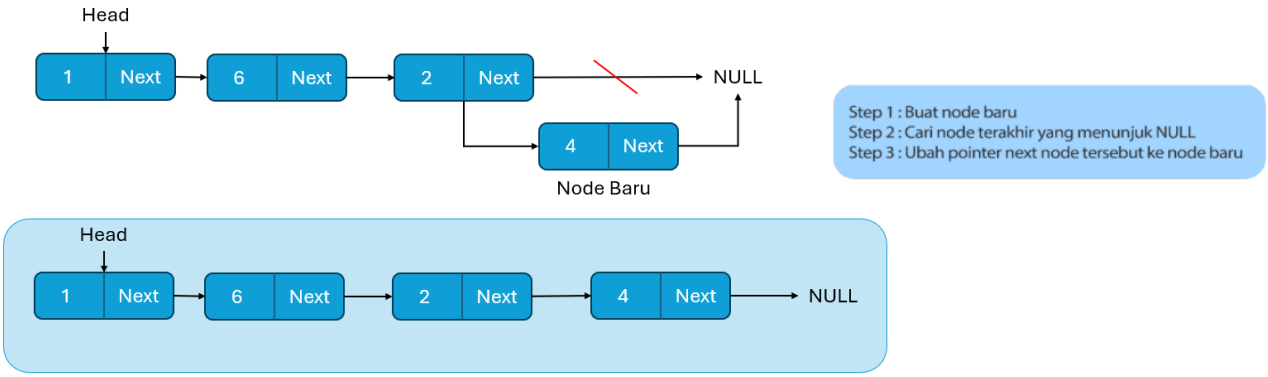
InsertFirst (memasukan data pada elemen utama)



InsertFirst adalah operasi yang berfungsi untuk memasukkan elemen baru menjadi elemen pertama pada linked list. Jika linked list masih kosong, pointer pada head akan terhubung dengan alamat hasil alokasi elemen terbaru. Jika linked list sudah ada isi sebelumnya, maka pointer pada elemen baru akan terlebih dahulu dihubungkan ke data paling awal yang sebelumnya sudah ada, lalu pointer pada head akan dihubungkan dengan alamat hasil alokasi elemen terbaru.

```
void insertFirst(List *L, infoType X)
{   address newNode = alokasi(X);
    newNode->next = (*L).first;
    (*L).first = newNode;
}
```

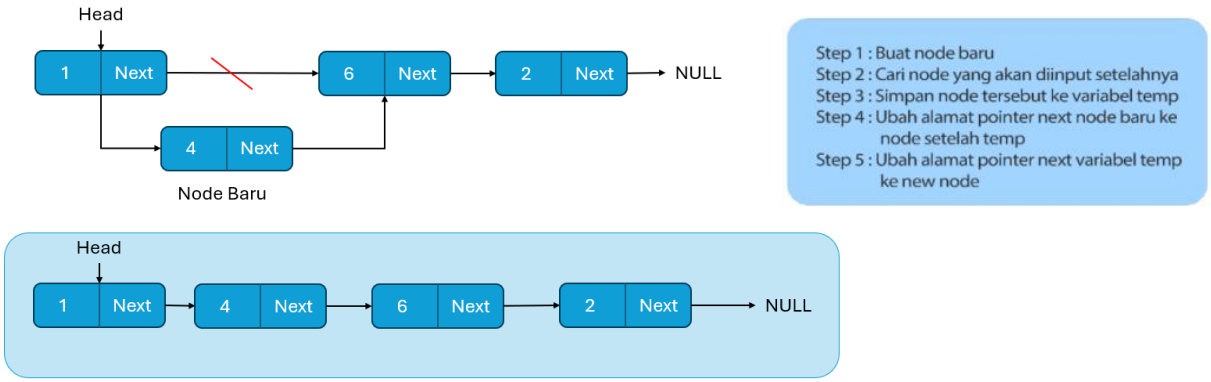
InsertLast (memasukan data sebagai elemen terakhir)



InserLast adalah operasi yang berfungsi untuk memasukkan elemen baru menjadi elemen terakhir pada linked list. Untuk memasukkan elemen menjadi elemen terakhir, perlu terlebih dahulu dicari data yang terakhir pada linked list. Setelah mencari elemen terakhir, pointer pada elemen terakhir tersebut akan terhubung dengan elemen yang terbaru. Jika belum ada elemen sama sekali di linked list, data di masukkan pada elemen pertama linked list seperti insertfirst.

```
void insertLast(List *L, infoType X){
    address P,newNode;
    newNode = alokasi(X);
    if(isEmpty((*L))){
        insertFirst(L,newNode);
    }else{
        for(P=(*L).first;P->next !=NULL;P=P->next); P->next = newNode;
    }
}
```

InsertAfter (Memasukkan data setelah data tertentu)



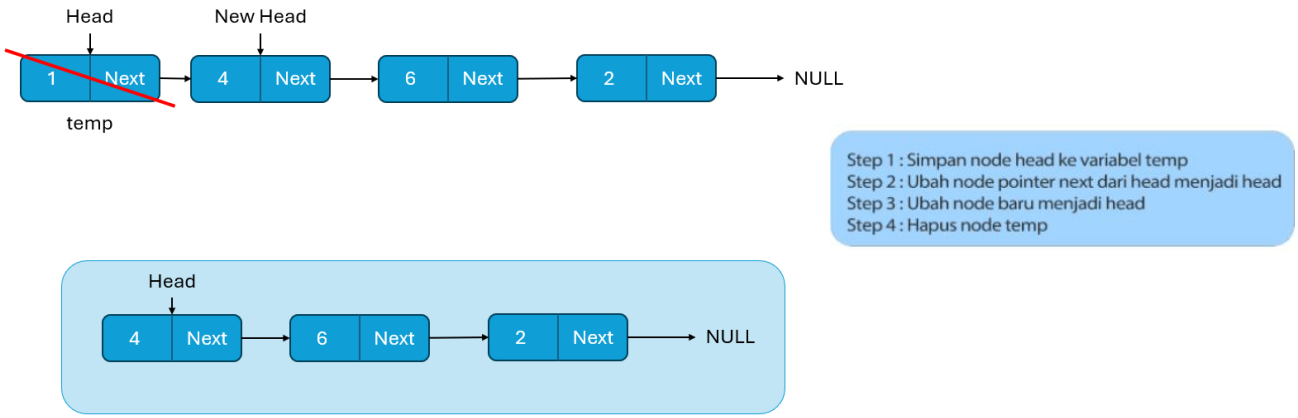
InsertAfter adalah operasi untuk memasukkan elemen baru pada linked list setelah elemen tertentu. Untuk melakukan insertAfter, cari dulu elemen yang akan dimasukkan elemen baru setelahnya. Alokasikan elemen baru, dan hubungkan alamat next dari elemen sebelumnya ke alamat next dari temp, dan masukkan alamat next elemen sebelumnya menjadi alamat temp

```
void insertAfter(address before, infoType X)
{   newNode = alokasi(X);
    if(before!=NULL){
        newNode->next  = before->next;
        before->next = newNode;
    }
}
```

DELETE

- DeleteFirst
- DeleteLast
- DeleteAt [New]

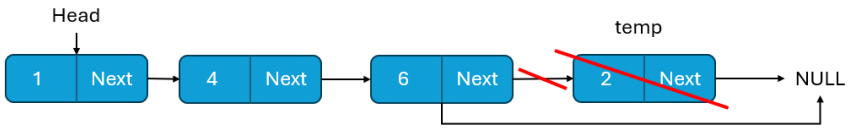
DeleteFirst (menghapus elemen pertama)



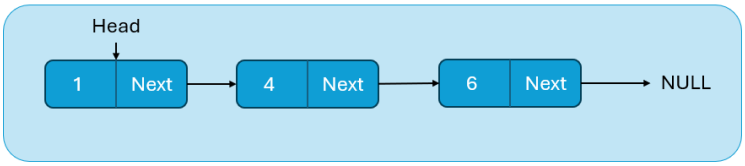
DeleteFirst adalah operasi yang digunakan untuk menghapus data pertama pada linkedList. Operasi ini akan melakukan penghapusan dengan terlebih dahulu menghubungkan linkedList pada elemen pertamanya kepada elemen setelah elemen pertama, atau elemen kedua. Lalu dilakukan pembebasan pada elemen pertama.

```
void deleteFirst(List *L){
    if(!isEmpty((*L))){
        address del = (*L).first;
        (*L).first = (*L).first-
>next;  free(del);
    }
}
```

DeleteLast (Menghapus elemen terakhir)



Step 1 : Cari node yang next keduanya menunjuk NULL
Step 2 : Masukkan node terakhir ke variabel temp
Step 3 : Hapus node temp
Step 4 : Ubah pointer next node yang ditemukan tadi menjadi NULL

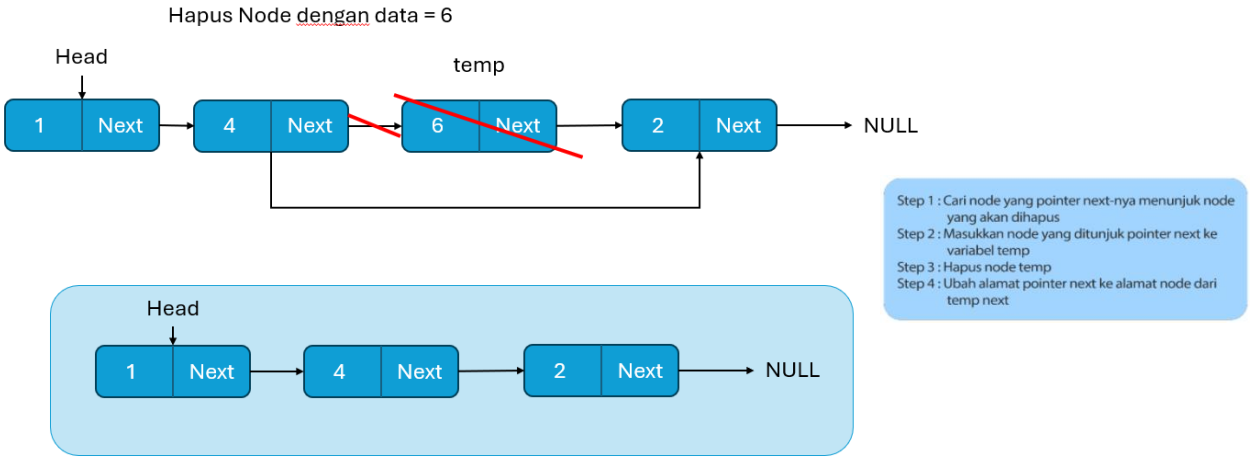


DeleteLast adalah operasi untuk menghapus data terakhir dari linked list. Cara menghapusnya adalah dengan mencari elemen sebelum elemen terakhir, lalu membebebaskan elemen terakhir. Elemen sebelum elemen terakhir tersebut akan dihubungkan ke null karena sudah tidak ada lagi data di elemen next/selanjutnya dari elemen tersebut

```
void deleteLast(List *L){
    address P;

    if(!isEmpty((*L))){
        if(isOneElement((*L))){
            deleteFirst(L);
        }else{
            for(P = (*L).first;P->next->next != NULL;P=P->next);
            free(P->next);
            P->next = NULL;
        }
    }
}
```

DeleteAt (menghapus elemen tertentu)



Operasi DeleteAt akan menghapus data pada elemen tertentu. Untuk melakukan penghapusan ini, perlu dicari terlebih dahulu elemen sebelum elemen yang ingin dihapus. Elemen tersebut yang berada sebelum elemen yang ingin dihapus akan dihubungkan ke elemen selanjutnya dari elemen yang dihapus. Lalu hapus elemen yang ingin dihapus. Untu melakukan operasi ini, perlu digunakan operasi FindNode(Bisa dilihat di guided) untuk mencari data yang ingin dihapus terlebih dahulu.

```
void deleteAt(List *L, infoType X){
    address P, del;
    del = findNode((*L),X);
    if(del != null){
      if((*L).first == del){
          deleteFirst(L);
      }else{
          P=(*L).first;
          while(P->next !=  del){
              P = P->next;
          }
          P->next = del->next;
          free(del);
      }
    }
}
```

GUIDED 7

Header.h

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <conio.h>

/*tipe data yang akan diterima oleh linked list adalah infotype*/
typedef int Infotype;// pendeklarasian bahwa infotype adalah int

/*address digunakan untuk menyimpan alamat pointer dari node*/
typedef struct node *address;
/*nodes akan menyimpan data dan juga menunjuk ke alamat Nodes selanjutnya*/
typedef struct node{ // disini node bekerja sebagai sebuat tag sehingga bisa langsung dipanggil sebelum dideklarasi
    Infotype X;
    address next;
}Nodes;

/*list menunjuk ke data pertama yakni first*/
typedef struct{
    address first;
}List;

void createEmpty(List *L);                          //menginitkan / membuat list kosong
bool isEmpty(List L);                               //mengecek apakah list itu kosong
bool isOneElement(List L);                          //mengecek apakah list hanya memiliki 1 data saja
address alokasi(Infotype X);                        //mengalokasikan data ke sebuah address beserta membuat node baru

void insertFirst(List *L, address newNode);         //memasukan data ke awal data linked list
void insertAfter(address before, address newNode);  //memasukan data ke linked list setelah node tertentu
void insertLast(List *L, address newNode);          //memasukan data ke akhir data linked list

void deleteFirst(List *L);                          //menghapus data node pertama
void deleteAt(List *L, address del);                //menghapus data node tertentu
void deleteLast(List *L);                           //menghapus data node terakhir

void printData(List L);                             //menampilkan data dalam linked list

address findNode(List L, Infotype X);               //mencari data pada linked list menggunakan tipe data tertentu
```

Source.c

```c
#include "header.h"

void createEmpty(List *L)
{
    L->first = NULL;
}

bool isEmpty(List L)
{
    return L.first == NULL;
}

bool isOneElement(List L)
{
    return !isEmpty(L) && L.first->next == NULL;
}

address alokasi(Infotype X)
{
    address temp;
    temp = (Nodes*)malloc(sizeof(Nodes));
    temp->X = X;
    temp->next = NULL;
    return temp;
}

void createEmpty(List *L)
{
    L->first = NULL;
}
void insertFirst(List *L, address newNode)
{
    newNode->next = L->first;
    L->first = newNode;
}

void insertLast(List *L, address newNode)
{
    address temp = L->first;
    if (isEmpty(*L))
    {
        insertFirst(&(*L), newNode);
    }
    else
    {
        while (temp->next != NULL)
        {
            temp = temp->next;
        }
        temp->next = newNode;
    }
}
```

Source.c (Part 2)

```c
void insertAfter(address before, address newNode)
{
    if (before != NULL)
    {
        newNode->next = before->next;
        before->next = newNode;
    }
}

void deleteFirst(List *L)
{
    if (!isEmpty(*L))
    {
        address del = (*L).first;
        (*L).first = (*L).first->next;
        free(del);
    }
}
void deleteAt(List *L, address del)
{
    address P;
    if (!isEmpty(*L))
    {
        if ((*L).first == del)
        {
            deleteFirst(L);
        }
        else
        {
            P = (*L).first;
            while (P->next != del)
            {
                P = P->next;
            }
            P->next = del->next;
            free(del);
        }
    }
}
```

Source.c (Part 3)

```c
void deleteLast(List *L)
{
    address P;
    if (!isEmpty(*L))
    {
        if (isOneElement(*L))
        {
            deleteFirst(L);
        }
        else
        {
            for (P = (*L).first; P->next->next != NULL; P = P->next)
                ;
            free(P->next);
            P->next = NULL;
        }
    }
}

void printData(List L){
    address temp = L.first;
    while(temp !=NULL){
        printf("[%d]->",temp->X);
        temp = temp->next;
    }
    printf("NULL");
    /*
    alternatif
    for(temp=L.first;temp!=NULL;temp=temp->next){
        printf("[%d]",temp->X);
    }
    */
}

address findNode(List L, Infotype X)
{
    address bantu = L.first;
    while (bantu != NULL && bantu->X != X)
    {
        bantu = bantu->next;
    }
    return bantu;
}
```

Main.c

```c
#include "header.h"

int main(int argc, char *argv[]) {
    List L;
    address temp;
    char menu;
    int bil;
    createEmpty(&L);
    do{
        system("cls");
        if(!isEmpty(L)){
            printData(L);
        }else{
            printf("\n\t Data Masih Kosong");
        }

        printf("\n\n\t Guided Linked List 2 - [XXXXX]\n");// di isi oleh 5 digit npm
        printf("\n\t 1. Insert First");
        printf("\n\t 2. Insert After");
        printf("\n\t 3. Insert Last");
        printf("\n\t 4. Delete First");
        printf("\n\t 5. Delete At");
        printf("\n\t 6. Delete Last");
        printf("\n\t 0. Exit");
        printf("\n\t Input Menu: >>\t"); menu = getch();
        printf("\n\t\n");
        switch(menu){
            case '1':
                printf("\n\t Masukan Angka: "); scanf("%d",&bil);
                printf("\t%d", bil);
                insertFirst(&L,alokasi(bil));
                printf("\n\t Insert First Berhasil");
                break;
            case '2':
                printf("\n\t Masukan Angka Setelah: "); scanf("%d",&bil);
                /*perhatikan findNode di source. fungsi findNode adalah untuk mencari node manakah yang memiliki data yang sesuai dengan inputan diatas.

                inputan diatas merupakan tempat node sebelum node data yang ingin dimasukan
                */
                temp = findNode(L,bil);
                if(temp == NULL){
                    printf("\n\t[!] Angka Tidak Ditemukan");
                    break;
                }
                printf("\n\t Masukan Angka: "); scanf("%d",&bil);
                printf("%d", bil);
                insertAfter(temp, alokasi(bil));
                printf("\n\t Insert After Berhasil");
                break;
            case '3':
                printf("\n\t Masukan Angka: "); scanf("%d",&bil);
                insertLast(&L,alokasi(bil));
                printf("\n\t Insert Last Berhasil");
                break;
```

Main.c (Part 2)

```
            case '4':
                if(isEmpty(L)){
                    printf("\n\t [!] List Masih Kosong");
                }
                deleteFirst(&L);
                printf("\n\t Delete First Berhasil");
                break;
            case '5':
                if(isEmpty(L)){
                    printf("\n\t [!] List Masih Kosong");
                }
                printf("\n\t Masukan Angka yang ingin di Delete: "); scanf("%d",&bil);
                temp = findNode(L,bil);
                if(temp == NULL){
                    printf("\n\t[!] Angka Tidak Ditemukan");
                    break;
                }
                deleteAt(&L, temp);
                printf("\n\t Delete At Berhasil");
                break;
            case '6':
                if(isEmpty(L)){
                    printf("\n\t [!] List Masih Kosong");
                }
                deleteLast(&L);
                printf("\n\t Delete Last Berhasil");
                break;
            case '0':
                // Ubah XXXXX jadi 5 digit terakhir NPM
                // Ubah A menjadi Kelas
                // Ubah NAMAPRAKTIKAN menjadi nama kalian
                printf("\n\t Exit [XXXXX - A - NAMAPRAKTIKAN]");
                break;
            default:
                printf("\n\t [!] Menu Tidak Ada");
                break;
        }
        getch();
    }while(menu!='0');

    return 0;
}
```

KETENTUAN PENGUMPULAN

Kumpulkan maksimal **H-1** SEBELUM PRAKTIKUM

Format pengumpulan:

GD7_X_YYYYY

X = kelas

Y = 5 digit NPM

Tips:

Pelajari cara memindah-mindahkan elemen dalam linked list