Double Linked List

Tujuan Praktikum

1. Memahami Double LinkedList dalam program,
2. Mampu menerapkan Double LinkedList untuk menyelesaikan berbagai kasus.

Dasar Teori

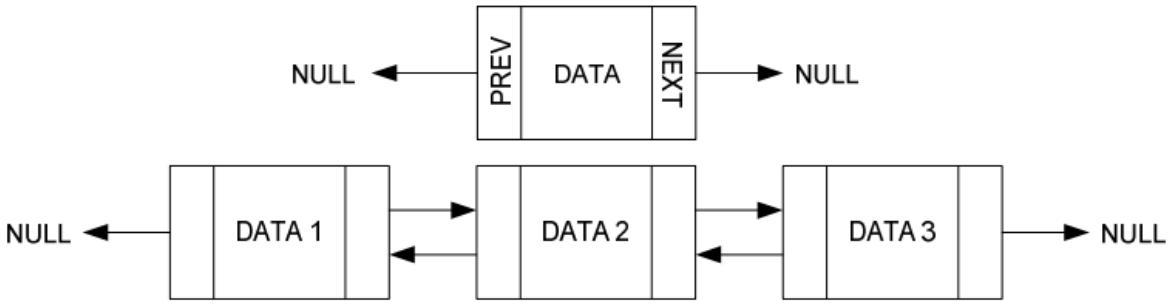
- Pada dasarnya, penggunaan Double Linked List hampir sama dengan penggunaan Single Linked List yang telah kita pelajari pada materi sebelumnya. Hanya saja Double Linked List menerapkan sebuah pointer baru, yaitu **prev**, yang digunakan untuk menggeser mundur selain tetap mempertahankan pointer **next**.
- Keberadaan 2 pointer penunjuk (**next** dan **prev**) menjadikan Double Linked List menjadi lebih fleksibel dibandingkan Single Linked List, namun dengan mengorbankan adanya memori tambahan dengan adanya pointer tambahan tersebut.
- Ada 2 jenis Double Linked List, yaitu: Double Linked List Non Circular dan Double Linked List Circular.

DOUBLE LINKED LIST NON CIRCULAR (DLLNC)

a. **DLLNC**

- DLLNC adalah sebuah Linked List yang terdiri dari dua arah pointer, dengan node yang saling terhubung, namun kedua pointernya menunjuk ke NULL.
- Setiap node pada linked list mempunyai field yang berisi data dan pointer yang saling berhubungan dengan node yang lainnya.

b. **GAMBARAN NODE DLLNC**



Praktik (Guided)

1. Compile program dibawah ini !
2. Berikan penjelasan pada masing-masing fungsi yang terdapat pada program. Jelaskan apa kegunaan masing-masing fungsi !

```
#include <iostream>
using namespace std;

// Deklarasi double linked list
struct Node {
    int data;
    Node *prev;
    Node *next;
};

Node *head, *tail;

void init() {
    head = NULL;
    tail = NULL;
}

int totalNode() {
    Node *cur;

    cur = head;
    int total = 0;
    while (cur != NULL) {
        total++;
        cur = cur->next;
    }
    return total;
}

void removeFirst() {
```

```
    if (head != NULL) {
        Node *delNode;

        if (head->next != NULL) {
            delNode = head;
            head = head->next;
            head->prev = NULL;
            delete delNode;
        } else {
            head = NULL;
        }
    } else {
        cout << "Cannot remove any node. Empty List!" << endl;
    }
}

void removeLast() {
    if (head != NULL) {
        Node *delNode;

        if (tail->prev != NULL) {
            delNode = tail;
            tail = tail->prev;
            tail->next = NULL;
            delete delNode;
        } else {
            tail = NULL;
        }
    } else {
        cout << "Cannot remove any node. Empty List!" << endl;
    }
}

void removeMiddle(int position) {
    if (head != NULL) {
        if (position == 0) {
            removeFirst();
        } else if (position == totalNode()-1) {
            removeLast();
        } else if (position < 0 || position > totalNode()-1) {
            cout << "Cannot add node. Unreachable index" << endl;
        } else {
            Node *delNode, *cur, *afterNode;

            cur = head;
            int index = 0;
            while (index < position-1) {
                cur = cur->next;
                index++;
            }

            delNode = cur->next;
            afterNode = delNode->next;
            cur->next = afterNode;
```

```
            afterNode->prev = cur;
            delete delNode;
        }
    } else {
        cout << "Cannot remove any node. Empty List!" << endl;
    }
}

void addFirst(int value) {
    Node *newNode;
    newNode = new Node();

    newNode->data = value;
    newNode->prev = NULL;
    newNode->next = NULL;

    if (head == NULL) {
        head = newNode;
        head->next = NULL;
        head->prev = NULL;
        tail = head;
    } else {
        newNode->prev = NULL;
        newNode->next = head;
        head->prev = newNode;
        head = newNode;
    }
}

void addLast(int value) {
    Node *newNode;
    newNode = new Node();

    newNode->data = value;
    newNode->prev = NULL;
    newNode->next = NULL;

    if (head == NULL) {
        head = newNode;
        head->next = NULL;
        head->prev = NULL;
        tail = head;
    } else {
        newNode->prev = tail;
        newNode->next = NULL;
        tail->next = newNode;
        tail = newNode;
    }
}

void addMiddle(int position, int value) {
    if (head != NULL) {
        if (position == 0) {
            addFirst(value);
```

```
        } else if (position == totalNode()-1) {
            addLast(value);
        } else if (position < 0 || position > totalNode()-1) {
            cout << "Cannot add node. Unreachable index" << endl;
        } else {
            Node *newNode, *cur, *afterNode;
            newNode = new Node();

            newNode->data = value;
            newNode->prev = NULL;
            newNode->next = NULL;

            cur = head;
            int index = 0;
            while (index < position-1) {
                cur = cur->next;
                index++;
            }

            afterNode = cur->next;
            newNode->prev = cur;
            newNode->next = afterNode;
            cur->next = newNode;
            afterNode->prev = newNode;
        }
    } else {
        addFirst(value);
    }
}

void printList() {
    if (head != NULL) {
        Node *cur;
        cur = head;
        while (cur != NULL) {
            cout << "(" << cur->data << ")" << " ";
            cur = cur->next;
        }
        cout << endl;
    } else {
        cout << "Empty list" << endl;
    }
}

void clear() {
    Node *cur, *delNode;
    cur = head;

    while (cur != NULL) {
        delNode = cur;
        cur = cur->next;
        delete delNode;
    }
```

```
    head = NULL;
}

int main() {
    init();

    addLast(22);
    addLast(66);
    addLast(87);

    addFirst(90);
    addFirst(10);
    addFirst(11);

    printList();

    removeFirst();
    printList();

    removeLast();
    printList();

    addMiddle(3, 89);
    printList();

    addMiddle(0, 66);
    printList();

    addMiddle(5, 277);
    printList();

    addMiddle(-1, 237);
    printList();

    addMiddle(100, 44);
    printList();

    removeMiddle(3);
    printList();

    removeMiddle(-1);
    printList();

    removeMiddle(200);
    printList();

    clear();
    printList();
}
```

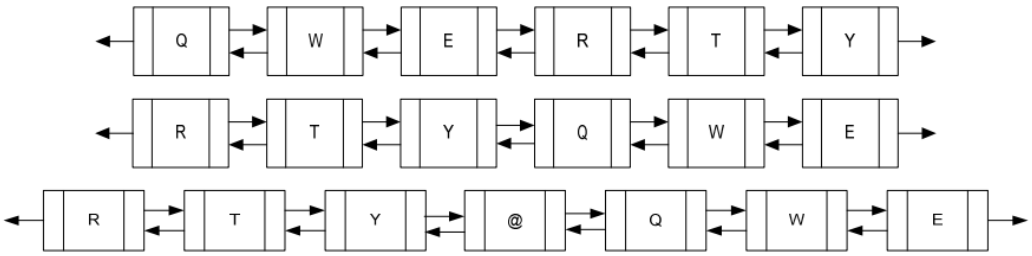
Latihan (Unguided)

1. Setelah deklarasi node dilakukan, dan semua fungsi sudah tersedia. Sekarang gabungkan setiap fungsi yang ada pada sebuah program penuh dengan spesifikasi :

    - Pada program utama (main) berisi sebuah menu yang berisi fitur-fitur yang terdapat dari setiap fungsi yang sudah ada ada sebelumnya, yaitu : tambah data, hapus data, cek data kosong, dan cetak semua data.
    - Pada struct hanya terdapat 1 tipe data saja yaitu integer.
    - Sesuaikan fungsi-fungsi yang ada dengan program yang Anda buat (jangan langsung copy-paste dan digunakan)

3. Buat program untuk enkripsi dan dekripsi password yang memanfaatkan Linked List, dengan spesifikasi :

    - Panjang password minimal 6 digit.
    - Isi password terserah dari user dan password diinputkan terlebih dahulu sebelumnya (penambahan data di belakang).
    - Enkripsi dilakukan dengan memindahkan 3 node terakhir, menjadi node terdepan. Kemudian sisipkan 1 karakter baru (kunci) setelah node ketiga dari yang dipindahkan tersebut.
    - Ilustrasi :



    - Lakukan juga proses dekripsi-nya.
    - Berikan juga fitur untuk menampilkan password.