Double Linked List Circular

Tujuan Praktikum

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

Dasar Teori

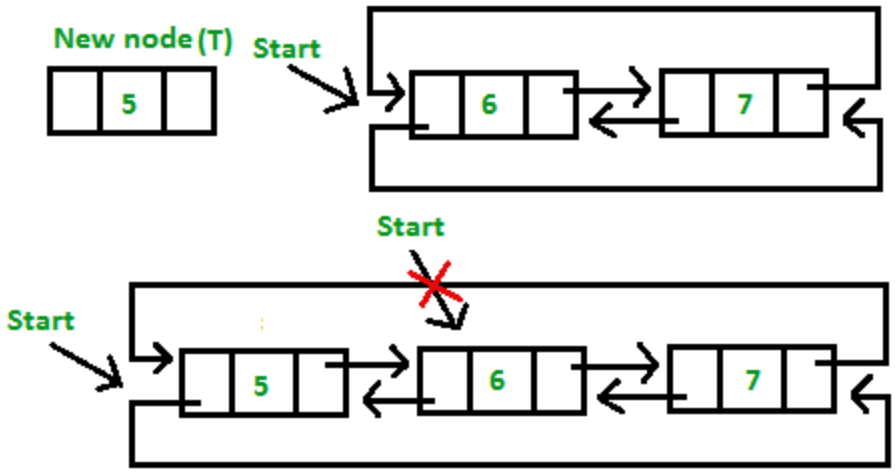
- Menggunakan 1 pointer head
- Head selalu menunjuk node pertama

DOUBLE LINKED LIST CIRCULAR (DLLC)

a. **DLLC**

- **Double Linked List Circular** adalah **linked list** dengan menggunakan pointer
- Setiap node memiliki 3 field, yaitu: - 1 field pointer yang menunjuk pointer berikutnya (next), - 1 field menunjuk pointer sebelumnya (prev), - serta sebuah field yang berisi data untuk node tersebut.

b. **GAMBARAN NODE DLLC**



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;

// Prototype fungsi
void init();
int totalNode();
void addFirst(int);
void addMiddle(int, int);
void addLast(int);
void removeFirst();
void removeMiddle(int);
void removeLast();
void printList();
void clear();

// Deklarasi node
struct Node {
    int data;
    Node *prev;
    Node *next;
};

Node *head, *tail;

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

// DONE
int totalNode() {
    Node *cur;

    cur = head;
    int total = 0;

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

// DONE
```

```
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 = head;
        head->prev = head;
        tail = head;
    } else {
        newNode->prev = tail;
        newNode->next = head;
        head->prev = newNode;
        tail->next = newNode;
        head = newNode;
    }
}

// UNGUIDED
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 {
        cout << "Cannot add any node. Empty list!" << endl;
```

```
    }
}

// DONE
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 = head;
        head->prev = head;
        tail = head;
    } else {
        newNode->prev = tail;
        newNode->next = head;
        head->prev = newNode;
        head->next = newNode;
        tail = newNode;
    }
}

// TODO
void removeFirst() {
    if (head != NULL) {
        Node *delNode;

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

// DONE
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 add any node. Empty list!" << endl;
    }
}

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

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

// DONE
void printList() {
    Node *cur;
    cur = head;
    do {
        cout << "(" << cur->data << ")" << " ";
        cur = cur->next;
    } while (cur != head);
    cout << endl;
}

// TODO
void clear() {
    Node *cur, *delNode;
```

```
    cur = head;

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

    head = NULL;
}

int main() {
    init();

    addLast(88);
    addLast(99);
    printList();

    addFirst(33);
    addFirst(66);
    printList();

    removeFirst();
    printList();

    removeMiddle(1);
    printList();
}
```

Latihan (Unguided)

1. Buatlah ilustrasi dari masing-masing potongan program.
2. Buat program lengkap dari potongan-potongan program yang ada diatas! Buat agar menjadi seperti menu.
3. Buat program untuk memasukkan node baru tetapi diantara node yang sudah ada. Tentukan node yang baru akan berada pada antrian keberapa.