Laporan Praktikum Algoritma dan Struktur Data

Praktikum 02: Linked List

Oleh:

Muhammad Dimas Ardiansyah / 5223600019

Program Studi STr Teknologi Game

Departemen Teknik Multimedia Kreatif

Politeknik Elektronika Negeri Surabaya

2023/2024

Modul Praktikum Operasi Pada Linked List

Terdapat beberapa Operasi yang penting pada linked list, yaitu:

1. Menyisipkan sebagai simpul ujung(awal) dari linked list.
2. Membaca atau menampilkan.
3. Mencari sebuah simpul tertentu.
4. Menyisipkan sebagai simpul terakhir.
5. Menghapus simpul tertentu.
6. Menyisipkan setelah simpul tertentu.
7. Menyisipkan sebelum simpul tertentu.

Code cpp :

```
#include <iostream>

using namespace std;


struct Node{

    int data;

    Node* next;

};

Node* tail=NULL;


void buat(Node*& head, int a);

void buatAwal(Node*& head, int a);

void tampil(Node* head);

void forTail(Node* head, Node*& tail);

void sisipkanSetelah(Node*& head, int after, int value);

void sisipkanSebelum(Node*& head, int before, int value);

void pencari(Node*& head, int i);

void hapus(Node*& head);

void hpsNodeIndex(Node *&head, int index);

void hpsNodeData(Node *&head, int data);



int main(){

    Node* head=NULL;
```

```
    buat(head, 10);
    tampil(head);
    buat(head, 18);
    tampil(head);
    buat(head, 37);
    tampil(head);
    buat(head, 56);
    tampil(head);
    sisipkanSetelah(head, 18, 45);
    tampil(head);
    sisipkanSebelum(head, 10, 37);
    tampil(head);
    pencari(head, 37);
    tampil(head);
    hpsNodeIndex(head, 2);
    tampil(head);
    hpsNodeData(head, 37);
    tampil(head);
    hapus(head);
}

//1. Menyisipkan sebagai node awal dari linked list.
void buat(Node*& head, int a){
    cout << "Menambahkan data >> " << a;
    Node* newNode = new Node();
    newNode->data = a;
    newNode->next = NULL;

    if (head == NULL) {
        head = newNode;
        tail = head;
    } else {
```

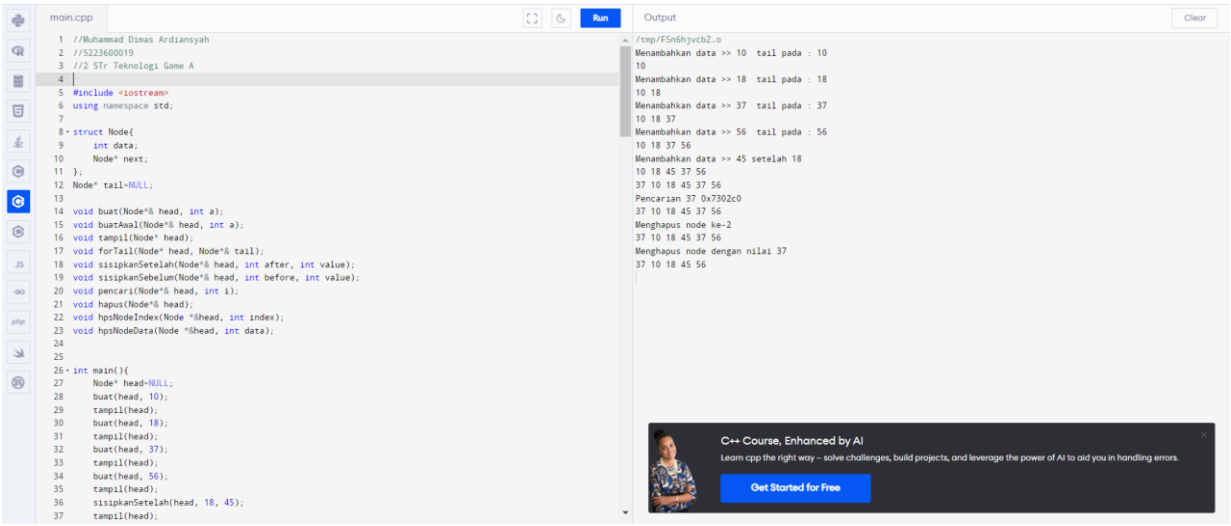
```
        tail->next = newNode;
        tail = newNode;
    }
    forTail(head, tail);
}

void buatAwal(Node*& head, int a){
    Node* newNode = new Node();
    newNode->data = a;
    newNode->next = NULL;

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

//2. Membaca atau menampilkan linked list.
void tampil(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

void forTail(Node* head, Node*& tail){
    tail = head;
```

```
    while(tail->next != NULL){
        tail = tail->next;
    }
    cout << "\ttail pada : " << tail->data << endl;
}

//3. Mencari sebuah simpul tertentu.
void pencari(Node*& head, int i){
    Node* cari = head;
    while(cari->data != i){
        cari = cari->next;
    }
    cout << "Pencarian " << cari->data << " " << cari->next << endl;
}

//4. Menyisipkan sebagai simpul terakhir.
void sisipkanSetelah(Node*& head, int after, int value){
    Node* newNode = new Node();
    newNode->data = value;

    Node* temp = head;
    while (temp != nullptr && temp->data != after) {
        temp = temp->next;
    }

    if (temp != nullptr) {
        newNode->next = temp->next;
        temp->next = newNode;
    } else {
        cout << "Data " << after << " tidak ada" << endl;
    }
```

```
    cout << "Menambahkan data >> " << value << " setelah " << after <<
endl;
}

//5. Menghapus simpul tertentu.
void hapus(Node*& head) {
    while (head != nullptr) {
        Node* temp = head;
        head = head->next;
        delete temp;
    }
}

//6. Menyisipkan sebelum simpul tertentu
void sisipkanSebelum(Node*& head, int before, int value){
    Node* newNode = new Node();
    newNode->data = value;

    Node* temp = head;

    if (head != nullptr && head->data == before) {
        newNode->next = head;
        head = newNode;
        return;
    }

    while (temp != nullptr && temp->next->data != before) {
        temp = temp->next;
    }

    if (temp != nullptr) {
        newNode->next = temp->next;
```

```
        temp->next = newNode;
    } else {
        cout << "Data " << before << " tidak ada" << endl;
    }
    cout << "Menambahkan data >> " << value << " sebelum " << before <<
endl;
}

//7. Menghapus simpul berdasarkan index.
void hpsNodeIndex(Node *&head, int index){
    cout << "Menghapus node ke-" << index << endl;
    Node* newNode = head;
    Node* bridge = head;
    for(int k=0; k < index; k++){
        newNode = newNode->next;
    }
    for (int i = 0; i < index-1; i++){
        bridge = bridge->next;
    }
}

//8. Menghapus simpul berdasarkan data.
void hpsNodeData(Node *&head, int data){
    Node* newNode = head;
    Node* bridge = new Node();
    while(newNode->next->data != data){
        newNode = newNode->next;
    }
    bridge = newNode;
    newNode = newNode->next;
    bridge->next = newNode->next;
```

```
    delete newNode;
    cout << "Menghapus node dengan nilai " << data << endl;
}
```



Pembahasan:

1. Menyisipkan sebagai simpul ujung(awal) dari linked list.

```
//...

void buatAwal(Node*& head, int a){
    Node* newNode = new Node();
    newNode->data = a;
    newNode->next = NULL;

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

```
}

//...
```
2. Membaca atau menampilkan
```
//...

void tampil(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

//...
```
3. Mencari sebuah simpul tertentu.
```
//...

void pencari(Node*& head, int i){
    Node* cari = head;
    while(cari->data != i){
        cari = cari->next;
    }
    cout << "Pencarian " << cari->data << " " << cari->next << endl;
}

//...
Adapun berikut fungsi untuk mencari node tail
//...

void forTail(Node* head, Node*& tail){
```

```
    tail = head;
    while(tail->next != NULL){
        tail = tail->next;
    }
    cout << "\ttail pada : " << tail->data << endl;
}

//...
```

4. Menyisipkan sebagai simpul terakhir

Fungsi berikut secara default saya buat untuk mennyisipkan node ke bagian akhir atau sebagai tail.

```
//...

void buat(Node*& head, int a){
    cout << "Menambahkan data >> " << a;
    Node* newNode = new Node();
    newNode->data = a;
    newNode->next = NULL;

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

//...
```

5. Menghapus simpul tertentu

Dalam menghapus node, saya membuat dua cara yaitu, pertama menggunakan index node

```
//...

void hpsNodeIndex(Node *&head, int index){
    cout << "Menghapus node ke-" << index << endl;
    Node* newNode = head;
    Node* bridge = head;
    for(int k=0; k < index; k++){
        newNode = newNode->next;
    }
    for (int i = 0; i < index-1; i++){
        bridge = bridge->next;
    }
}

//...
```

code di atas akan menghapus node berdasarkan index yang diinputkan, kemudian cara yang kedua adalah menghapus node berdasarkan data atau value yang dimiliki oleh suatu node

```
//...

void hpsNodeData(Node *&head, int data){
    Node* newNode = head;
    Node* bridge = new Node();
    while(newNode->next->data != data){
        newNode = newNode->next;
    }
    bridge = newNode;
    newNode = newNode->next;
    bridge->next = newNode->next;

    delete newNode;
    cout << "Menghapus node dengan nilai " << data << endl;
}
```

```
//...
```

6. Menyisipkan setelah simpul tertentu

Code ini akan menysipkan node setelah node yang memiliki data atau value tertentu

```
//...


void sisipkanSetelah(Node*& head, int after, int value){
    Node* newNode = new Node();
    newNode->data = value;


    Node* temp = head;
    while (temp != nullptr && temp->data != after) {
        temp = temp->next;
    }


    if (temp != nullptr) {
        newNode->next = temp->next;
        temp->next = newNode;
    } else {
        cout << "Data " << after << " tidak ada" << endl;
    }
    cout << "Menambahkan data >> " << value << " setelah " << after << endl;
}


//...
```

7. Menyisipkan setelah simpul tertentu

Code ini akan menysipkan node sebelum node yang memiliki data atau value tertentu

```
//...


void sisipkanSebelum(Node*& head, int before, int value){
```

```
    Node* newNode = new Node();
    newNode->data = value;

    Node* temp = head;

    if (head != nullptr && head->data == before) {
        newNode->next = head;
        head = newNode;
        return;
    }

    while (temp != nullptr && temp->next->data != before) {
        temp = temp->next;
    }

    if (temp != nullptr) {
        newNode->next = temp->next;
        temp->next = newNode;
    } else {
        cout << "Data " << before << " tidak ada" << endl;
    }
    cout << "Menambahkan data >> " << value << " sebelum " << before <<
endl;
}

//...
```

Percobaan:

Implementasikan operasi dasar Single linked list : Menyisipkan sebagai simpul ujung(awal) dari linked list.

Implementasikan operasi dasar Single linked list : Membaca atau menampilkan

Implementasikan operasi dasar Single linked list : Mencari sebuah simpul tertentu. Tambahkan kondisi jika yang dicari adalah data yang paling depan.

Implementasikan operasi dasar Single linked list : Menyisipkan sebagai simpul terakhir

Gabungkan semua operasi di atas dalam sebuah Menu Pilihan.

Code full versi cpp :

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

struct Node{
    int data;
    Node* next;
};
Node* tail=NULL;

void buat(Node*& head, int a);
void buatAwal(Node*& head, int a);
void tampil(Node* head);
void forTail(Node* head, Node*& tail);
void sisipkanSetelah(Node*& head, int after, int value);
void sisipkanSebelum(Node*& head, int before, int value);
void pencari(Node*& head, int i);
void hapus(Node*& head);
void hpsNodeIndex(Node *&head, int index);
void hpsNodeData(Node *&head, int data);

int main(){
    Node* head=NULL;
```

```
int nilai,after, before;

int pilihan;
do{
    cout << "\nMenu Pilihan:\n";
    cout << "1. Menyisipkan sebagai node dari linked list.\n";
    cout << "2. Membaca atau menampilkan linked list.\n";
    cout << "3. Mencari sebuah simpul tertentu.\n";
    cout << "4. Menyisipkan sebagai simpul terakhir.\n";
    cout << "5. Keluar.\n";
    cout << "Pilihan Anda: ";
    cin >> pilihan;

    switch (pilihan) {
        case 1:
            cout << "Masukkan nilai yang ingin disisipkan: ";
            cin >> nilai;
            buatAwal(head, nilai);
            break;
        case 2:
            tampil(head);
            break;
        case 3:
            cout << "Masukkan nilai yang ingin dicari: ";
            cin >> nilai;
            pencari(head, nilai);
            break;
        case 4:
            cout << "Masukkan nilai yang ingin disisipkan: ";
            cin >> nilai;
            buat(head, nilai);
            break;
```

```
            case 5:
                cout << "Keluar dari program.\n";
                break;
            default:
                cout << "Pilihan tidak valid. Silakan coba lagi.\n";
        }
    } while (pilihan != 5);

    return 0;
}
```

//1. Implementasikan operasi dasar Single linked list : Menyisipkan sebagai simpul

```
// ujung(awal) dari linked list.
void buatAwal(Node*& head, int a){
    Node* newNode = new Node();
    newNode->data = a;
    newNode->next = NULL;

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

// 4. Implementasikan operasi dasar Single linked list : Menyisipkan sebagai simpul

```
// terakhir
void buat(Node*& head, int a){
    cout<<"menambahkan data >> "<<a;
    Node* newNode = new Node();
    newNode->data = a;
```

```
    newNode->next = NULL;


    if (head == NULL) {
        head = newNode;
        tail = head;
    } else {
        tail->next = newNode;
        tail = newNode;
    }
    forTail(head, tail);
}
```

// 2. Implementasikan operasi dasar Single linked list : Membaca atau menampilkan

```
void tampil(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
void forTail(Node* head, Node*& tail){
    tail=head;
    while(tail->next != NULL){
        tail=tail->next;
    }
    cout<<"\ttail pada : "<<tail->data<<endl;
}
```

// 3. Implementasikan operasi dasar Single linked list : Mencari sebuah simpul

```
// tertentu. Tambahkan kondisi jika yang dicari adalah data yang paling depan.
void pencari(Node*& head, int i){
```

```
    Node* cari=head;
    while(cari->data!=i){
        cari=cari->next;
    }
    cout<<"pencarian "<<cari->data<<" "<<cari->next<<endl;
}
void sisipkanSetelah(Node*& head, int after, int value){
    Node* newNode = new Node();
    newNode->data = value;

    Node* temp = head;
    while (temp != nullptr && temp->data != after) {
        temp = temp->next;
    }

    if (temp != nullptr) {
        newNode->next = temp->next;
        temp->next = newNode;
    } else {
        cout << "data " << after << " tidak ada" << endl;
    }
    cout<<"menambahkan data >> "<<value<<" setelah "<<after<<endl;
}
void sisipkanSebelum(Node*& head, int before, int value){
    Node* newNode = new Node();
    newNode->data = value;

    Node* temp = head;

    if (head != nullptr && head->data == before) {
        newNode->next = head;
        head = newNode;
```

```
        return;
    }

    while (temp != nullptr && temp->next->data != before) {
        temp = temp->next;
    }

    if (temp != nullptr) {
        newNode->next = temp->next;
        temp->next = newNode;
    } else {
        cout << "data " << before << " tidak ada" << endl;
    }
    cout<<"menambahkan data >> "<<value<<" sebelum "<<before<<endl;
}
void hapus(Node*& head) {
    while (head != nullptr) {
        Node* temp = head;
        head = head->next;
        delete temp;
    }
}

void hpsNodeIndex(Node *&head, int index){
    Node* newNode=head;
    Node* bridge=head;
    for(int k=0;k<index;k++){
        newNode=newNode->next;
    }
    for (int i = 0; i < index-1; i++){
        bridge=bridge->next;
    }
```

```
    bridge->next=newNode->next;

    delete newNode;
    cout<<"menghapus node ke-"<<index<<endl;
}

void hpsNodeData(Node *&head, int data){
    Node* newNode=head;
    Node* bridge=new Node();
    while(newNode->next->data!=data){
        newNode=newNode->next;
    }
    bridge=newNode;
    newNode=newNode->next;
    bridge->next=newNode->next;

    delete newNode;
    cout<<"menghapus node dengan nilai "<<data<<endl;
}
```