**Laporan Praktikum**

Linked List

**Oleh:**

Oscar Javier Abdullah / 5223600001

**Program Studi STr Teknologi Game**

**Departemen Teknologi Multimedia Kreatif**

**Politeknik Elektronika Negeri Surabaya**

**2024**

**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

**Source Code**

#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 = nullptr;

// 1. Menambahkan data 10

buat(head, 10);

tampil(head);

// 2. Menambahkan data 18

buat(head, 18);

tampil(head);

// 3. Menambahkan data 22

buat(head, 22);

tampil(head);

// 4. Menambahkan data 56

buat(head, 56);

tampil(head);

// 5. Menambahkan data 45 setelah 18

sisipkanSetelah(head, 18, 45);

tampil(head);

// 6. Menambahkan data 37 sebelum 10

sisipkanSebelum(head, 10, 37);

tampil(head);

// 7. Mencari data 22

pencari(head, 22);

// 8. Menghapus node ke-2

hpsNodeIndex(head, 2);

tampil(head);

// 9. Menghapus node dengan nilai 22

hpsNodeData(head, 22);

tampil(head);

// 10. Menghapus seluruh linked list

hapus(head);

return 0;

}

// 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 setelah simpul 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;

}

// 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;

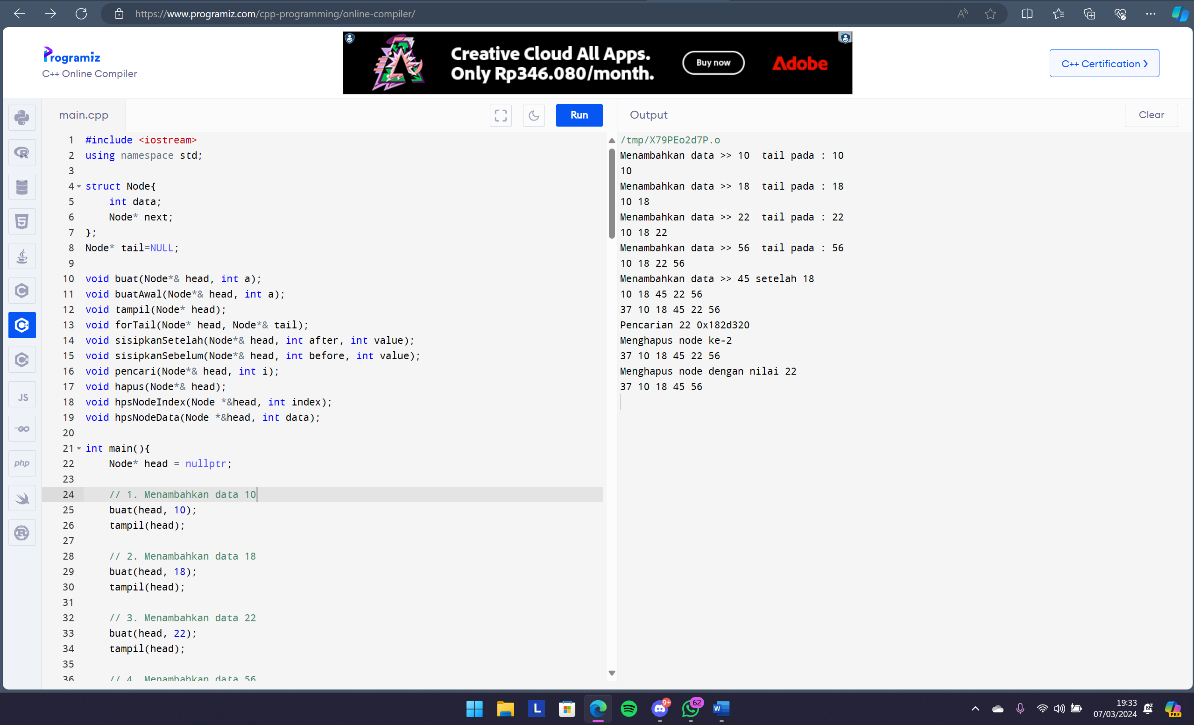
newNode = newNode->next;

bridge->next = newNode->next;

delete newNode;

cout << "Menghapus node dengan nilai " << data << endl;

}

***Gambar 1*** (Output Program)

**PERCOBAAN**

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

ujung(awal) dari linked list.

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

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

tertentu. Tambahkan kondisi jika yang dicari adalah data yang paling depan.

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

terakhir

5. Gabungkan semua operasi di atas dalam sebuah Menu Pilihan.

**Source Code**

#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 hapusNode(Node\*& head, int key, bool byIndex);

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. Menghapus node berdasarkan indeks.\n";

cout << "6. Menghapus node berdasarkan nilai data.\n";

cout << "7. 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 << "Masukkan indeks node yang ingin dihapus: ";

cin >> nilai;

hapusNode(head, nilai, true);

break;

case 6:

cout << "Masukkan nilai data node yang ingin dihapus: ";

cin >> nilai;

hapusNode(head, nilai, false);

break;

case 7:

cout << "Keluar dari program.\n";

break;

default:

cout << "Pilihan tidak valid. Silakan coba lagi.\n";

}

} while (pilihan != 7);

// Hapus semua node sebelum keluar dari program

while (head != NULL) {

Node\* temp = head;

head = head->next;

delete temp;

}

return 0;

}

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;

}

}

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 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;

}

void pencari(Node\*& head, int i) {

Node\* cari = head;

while (cari->data != i) {

cari = cari->next;

}

cout << "pencarian " << cari->data << " " << cari->next << endl;

}

void hapusNode(Node\*& head, int key, bool byIndex) {

if (head == NULL) {

cout << "Linked list kosong." << endl;

return;

}

Node\* prev = NULL;

Node\* current = head;

if (byIndex) {

if (key == 0) {

head = head->next;

delete current;

cout << "Menghapus node ke-" << key << endl;

return;

}

int currentIndex = 0;

while (current != NULL && currentIndex < key) {

prev = current;

current = current->next;

currentIndex++;

}

}

else {

while (current != NULL && current->data != key) {

prev = current;

current = current->next;

}

}

if (current != NULL) {

if (prev != NULL) {

prev->next = current->next;

}

else {

head = current->next; // Jika node yang dihapus adalah node pertama

}

delete current;

if (byIndex)

cout << "Menghapus node ke-" << key << endl;

else

cout << "Menghapus node dengan nilai " << key << endl;

}

else {

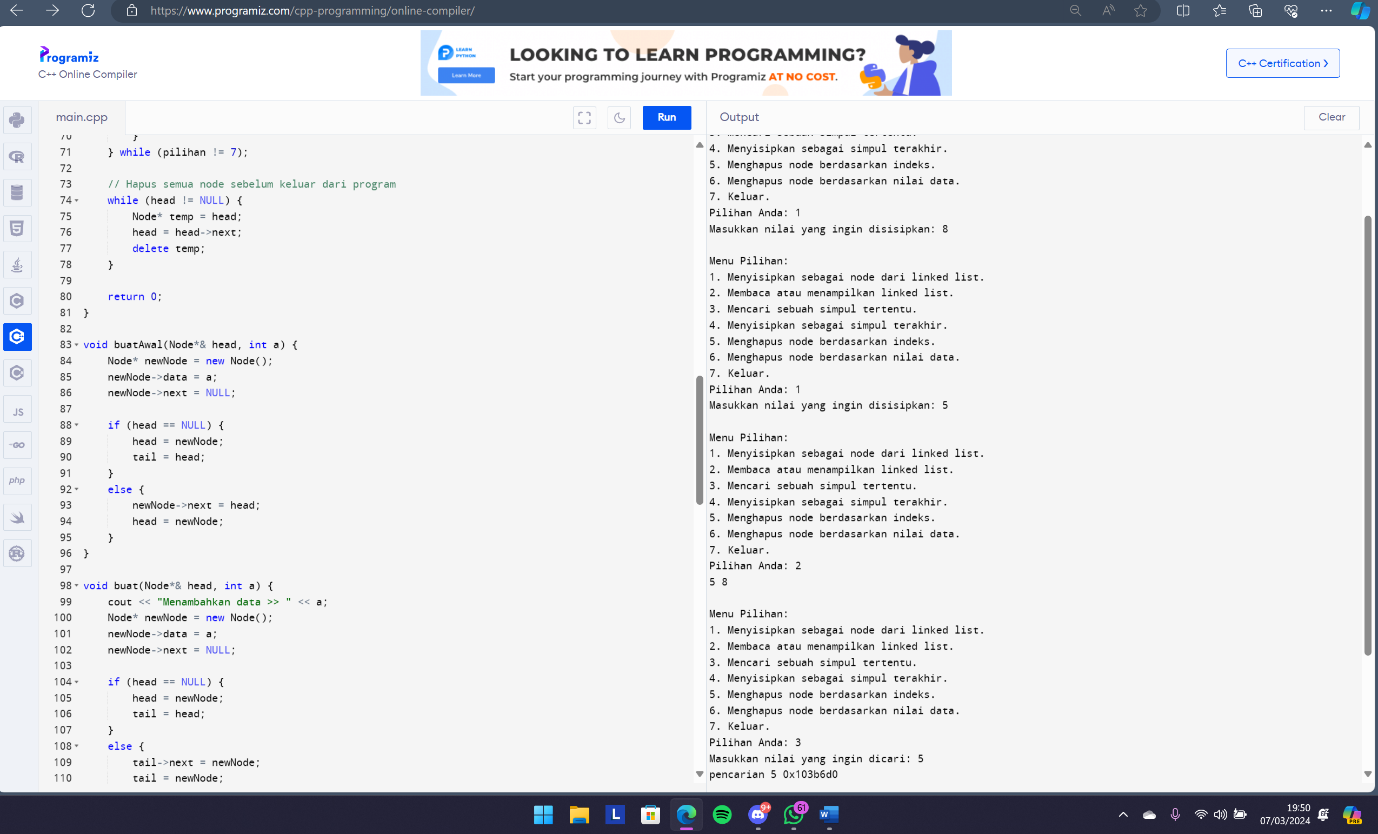
if (byIndex)

cout << "Indeks melebihi ukuran linked list." << endl;

else

cout << "Node dengan nilai " << key << " tidak ditemukan." << endl;

}

****}

***Gambar 2*** (Output Program)