**Laporan Praktikum Singled Linked**

Praktikum Algoritma dan Struktur Data

****

**Oleh:**

Samudero Dirgantoro / 5223600016

**Program Studi D4 Teknologi Game**

**Departemen Teknologi Multimedia Kreatif**

**Politeknik Elektronika Negeri Surabaya**

**2023/2024**

1. Percobaan
2. Implementasikan operasi dasar Double linked list: Menyisipkan sebagai simpul ujung(awal) dari linked list.

#include <iostream>

struct Node {

int data;

Node\* next;

Node\* prev;};

Node\* createNode(int data) {

Node\* newNode = new Node;

newNode->data = data;

newNode->next = NULL;

newNode->prev = NULL;

return newNode;}

void insertNode(Node\*\* head, int data) {

Node\* newNode = createNode(data);

newNode->next = \*head;

if (\*head != NULL)

(\*head)->prev = newNode;

\*head = newNode;}

void printList(Node\* head) {

Node\* temp = head;

while (temp != NULL) {

std::cout << temp->data << " ";

temp = temp->next;}}

int main() {

Node\* head = NULL;

insertNode(&head, 3);

insertNode(&head, 2);

insertNode(&head, 1);

printList(head);}

1. Implementasikan operasi dasar Double linked list : Membaca atau menampilkan.

#include <iostream>

struct Node {

int data;

Node\* next;

Node\* prev;};

Node\* createNode(int data) {

Node\* newNode = new Node;

newNode->data = data;

newNode->next = NULL;

newNode->prev = NULL;

return newNode;}

void insertNode(Node\*\* head, int data) {

Node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;}

else {

Node\* temp = \*head;

while (temp->next != NULL) {

temp = temp->next;}

temp->next = newNode;

newNode->prev = temp;}}

void Tampilkan(Node\* head) {

Node\* current = head;

while (current != NULL) {

std::cout << "Data: " << current->data << "\n";

current = current->next;}}

int main() {

Node\* head = NULL;

insertNode(&head, 1);

insertNode(&head, 2);

insertNode(&head, 3);

Tampilkan(head);}

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

#include <iostream>

struct Node {

std::string nama;

int nrp;

Node\* next;

Node\* prev;};

Node\* createNode(const std::string& nama, int nrp) {

Node\* newNode = new Node;

newNode->nama = nama;

newNode->nrp = nrp;

newNode->next = NULL;

newNode->prev = NULL;

return newNode;}

void insertNode(Node\*\* head, const std::string& nama, int nrp) {

Node\* newNode = createNode(nama, nrp);

if (\*head == NULL) {

\*head = newNode;}

else {

Node\* temp = \*head;

while (temp->next != NULL) {

temp = temp->next;}

temp->next = newNode;

newNode->prev = temp;}}

Node\* searchNode(Node\* head, const std::string& targetNama) {

Node\* current = head;

while (current != NULL) {

if (current->nama == targetNama) {

return current;}

current = current->next;}

return NULL;}

int main() {

Node\* head = NULL;

insertNode(&head, "Sam", 61);

insertNode(&head, "Dirga", 16);

insertNode(&head, "Mas", 20);

std::string targetNama = "Dirga"; //<-- Tempat cari nama

Node\* foundNode = searchNode(head, targetNama);

if (foundNode != NULL) {

std::cout << "Node ditemukan dengan nama: " << foundNode->nama << ", NRP: " << foundNode->nrp << "\n";}

else {

std::cout << "Node dengan Nama '" << targetNama << "' tidak ditemukan." << '\n';}}

1. Implementasikan operasi dasar *Double linked list* : Menghapus simpul tertentu.  Tambahkan kondisi jika yang dihapus adalah data yang paling depan atau data  yang paling terakhir.

#include <iostream>

struct Node {

int data;

Node\* next;

Node\* prev;};

Node\* createNode(int data) {

Node\* newNode = new Node;

newNode->data = data;

newNode->next = NULL;

newNode->prev = NULL;

return newNode;}

void insertNode(Node\*\* head, int data) {

Node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;}

else {

Node\* temp = \*head;

while (temp->next != NULL) {

temp = temp->next;}

temp->next = newNode;

newNode->prev = temp;}}

void deleteNode(Node\*\* head, int targetData) {

if (\*head == NULL) {

std::cout << "Linked list kosong." << “\n”;

return;}

Node\* current = \*head;

// mencari data yang akan dihapus

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

current = current->next;}

if (current->prev == NULL) {

\*head = current->next;}

else {

current->prev->next = current->next;}

if (current->next != NULL) {

current->next->prev = current->prev;}

delete current;}

void Tampilkan(Node\* head) {

Node\* current = head;

while (current != NULL) {

std::cout << "Data: " << current->data << "\n";

current = current->next;}}

int main() {

Node\* head = NULL;

insertNode(&head, 1);

insertNode(&head, 2);

insertNode(&head, 3);

std::cout << "Sebelum penghapusan:" << "\n";

Tampilkan(head);

deleteNode(&head, 1);

std::cout << "\nSetelah penghapusan:" << "\n";

Tampilkan(head);}

1. Gabungkan semua operasi di atas dalam sebuah Menu Pilihan

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

Node\* prev;};

Node\* buatnode(int data) {

Node\* newNode = new Node;

newNode->data = data;

newNode->next = NULL;

newNode->prev = NULL;

return newNode;}

void nodeawal(Node\*\* head, int data) {

Node\* newNode = buatnode(data);

if (\*head == NULL) {

\*head = newNode;}

else {

newNode->next = \*head;

(\*head)->prev = newNode;

\*head = newNode;}}

void menampilkandata(Node\* head) {

if (head == NULL) {

cout << "Linked list kosong." << "\n";

return;}

Node\* current = head;

while (current != NULL) {

cout << "Data: " << current->data << "\n";

current = current->next;}}

Node\* carinode(Node\* head, int targetData) {

Node\* current = head;

while (current != NULL) {

if (current->data == targetData) {

return current;}

current = current->next;}

return NULL;}

void hapusnode(Node\*\* head, int targetData) {

if (\*head == NULL) {

cout << "Linked list kosong." << "\n";

return;}

Node\* current = \*head;

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

current = current->next;}

if (current == NULL) {

cout << "Simpul dengan data '" << targetData << "' tidak ditemukan." << "\n";

return;}

if (current->prev == NULL) {

\*head = current->next;}

else {

current->prev->next = current->next;}

if (current->next != NULL) {

current->next->prev = current->prev;}

delete current;}

int main() {

Node\* head = NULL;

int choice;

int data, searchData, deleteData;

do {

cout << "\nPilihan\n 1. Menyisip Ujung Awal\n 2. Membaca dan menampilkan\n 3. Mencari sebuah simpul\n 4. Menghapus simpul\n 5. Exit\nInput: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Masukkan data baru: ";

cin >> data;

nodeawal(&head, data);

break;

case 2:

cout << "Isi dari linked list:\n";

menampilkandata(head);

break;

case 3:

cout << "Masukkan data yang ingin dicari: ";

cin >> searchData;

if (Node\* foundNode = carinode(head, searchData)) {

cout << "Simpul dengan data '" << searchData << "' ditemukan." << "\n";}

else {

cout << "Simpul dengan data '" << searchData << "' tidak ditemukan." << "\n";}

break;

case 4:

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

cin >> deleteData;

hapusnode(&head, deleteData);

break;

case 5:

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

break;

default:

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

break;

}

} while (choice != 5);}

1. Latihan
2. Bangunlah Double linked dengan prinsip LIFO.

#include <iostream>

using namespace std;

struct Node {

std::string nama;

int nrp;

Node\* next;

Node\* prev;};

Node\* nodebaru(const std::string& nama, int nrp) {

Node\* nodeBaru = new Node;

nodeBaru->nama = nama;

nodeBaru->nrp = nrp;

nodeBaru->next = nullptr;

nodeBaru->prev = nullptr;

return nodeBaru;}

void nodeawal(Node\*\* ujung, const std::string& nama, int nrp) {

Node\* nodeBaru = nodebaru(nama, nrp);

if (\*ujung == nullptr) {

\*ujung = nodeBaru;}

else {

nodeBaru->next = \*ujung;

(\*ujung)->prev = nodeBaru;

\*ujung = nodeBaru;}}

void hasildata(Node\* ujung) {

Node\* current = ujung;

while (current != nullptr) {

cout << "Nama: " << current->nama << ", NRP: " << current->nrp << '\n';

current = current->next;}}

int main() {

Node\* ujung = nullptr;

int j = 0;

string nama;

int nrp;

cout << "Nama: ";

cin >> nama;

cout << "NRP : ";

cin >> nrp;

// memasukkan node awal (LIFO)

nodeawal(&ujung, nama, nrp);

cout << "Hasil input: " << '\n';

hasildata(ujung);}

1. Bangunlah sebuah *single linked list* dentan prinsip FIFO(First In First Out)

#include <iostream>

using namespace std;

struct Node {

std::string nama;

int nrp;

Node\* next;

Node\* prev;};

Node\* nodebaru(const std::string& nama, int nrp) {

Node\* nodeBaru = new Node;

nodeBaru->nama = nama;

nodeBaru->nrp = nrp;

nodeBaru->next = nullptr;

nodeBaru->prev = nullptr;

return nodeBaru;}

void nodeakhir(Node\*\* ujung, const std::string& nama, int nrp) {

Node\* nodeBaru = nodebaru(nama, nrp);

if (\*ujung == nullptr) {

\*ujung = nodeBaru;}

else {

Node\* current = \*ujung;

while (current->next != nullptr) {

current = current->next;}

current->next = nodeBaru;

nodeBaru->prev = current;}}

void hasildata(Node\* ujung) {

Node\* current = ujung;

while (current != nullptr) {

cout << "Nama: " << current->nama << ", NRP: " << current->nrp << '\n';

current = current->next;}}

int main() {

Node\* ujung = nullptr;

int j = 0;

string nama;

int nrp;

cout << "Nama: ";

cin >> nama;

cout << "NRP : ";

cin >> nrp;

// memasukkan node akhir (FIFO)

nodeakhir(&ujung, nama, nrp);

cout << "Hasil input: " << '\n';

hasildata(ujung);}

1. Kesimpulan

Double linked list dan single linked list adalah dua jenis struktur data yang digunakan untuk menyimpan dan mengorganisir data secara berurutan. Perbedaan utamanya terletak pada cara setiap simpul terhubung satu sama lain. Pada single linked list, setiap simpul hanya memiliki satu pointer yang menunjuk ke simpul berikutnya, sedangkan pada double linked list, setiap simpul memiliki dua pointer, satu untuk menunjuk ke simpul sebelumnya dan satu lagi untuk menunjuk ke simpul berikutnya. Karena memiliki pointer tambahan, double linked list membutuhkan lebih banyak memori, tetapi memberikan keuntungan dalam operasi seperti penghapusan dan penambahan simpul di akhir atau di tengah list, yang dapat dilakukan dengan lebih efisien karena tidak perlu mencari simpul sebelumnya. Single linked list umumnya lebih sederhana dan memakan memori yang lebih sedikit, namun operasi tertentu seperti menghapus simpul tertentu membutuhkan pencarian dari awal, yang dapat memakan waktu dan menjadi tidak efisiensi.