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

A. Percobaan

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

```
#include <iostream>
struct Node {
   int data;
   Node* next;};

Node* createNode(int data) {
   Node* newNode = new Node;
   newNode->data = data;
   newNode->next = NULL;
   return newNode;}

void insertNode(Node** head, int data) {
   Node* newNode = createNode(data);
   newNode->next = *head;
   *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);}
```

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

```
#include <iostream>
struct Node {
  int data;
  Node* next;};

Node* createNode(int data) {
  Node* newNode = new Node;
  newNode->data = data;
  newNode->next = 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;}}

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);}
```

3. Implementasikan operasi dasar Single 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* createNode(const std::string& nama, int nrp) {
    Node* newNode = new Node;
    newNode->nama = nama;
    newNode->nrp = nrp;
    newNode->next = nullptr;
    return newNode;}

void insertNode(Node** head, const std::string& nama, int
nrp) {
    Node* newNode = createNode(nama, nrp);
    if (*head == nullptr) {
        *head = newNode;}
    else {
        Node* temp = *head;
        while (temp->next != nullptr) {
            temp = temp->next;}
        temp->next = newNode;}}
```

```
Node* searchNode(Node* head, const std::string&
targetNama) {
    Node* current = head;
    while (current != nullptr) {
        if (current->nama == targetNama) {
            return current;}
        current = current->next;}
    return nullptr; }

int main() {
    Node* head = nullptr;

    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 != nullptr) {
        std::cout << "Node ditemukan dengan nama: " <<
foundNode->nama << ", NRP: " << foundNode->nrp <<
std::endl;}
        else {
        std::cout << "Node dengan Nama '" << targetNama <<
"'tidak ditemukan." << '\n';}}
```

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

```
#include <iostream>

struct Node {
    int data;
    Node* next;};

Node* createNode(int data) {
    Node* newNode = new Node;
    newNode->data = data;
    newNode->next = NULL;
    return newNode;}
void insertNode(Node** head, int data) {
    Node* newNode = createNode(data);
    if (*head == NULL) {
        *head = newNode;}
    else {
        Node* current = *head;
        while (current->next != NULL) {
            current = current->next;}
        current->next = newNode;}}

void printList(Node* head) {
    Node* current = head;
    while (current != NULL) {
        std::cout << current->data << " ";
        current = current->next;}
    std::cout << std::endl;}

int main() {
    Node* head = NULL;
    insertNode(&head, 1);
    insertNode(&head, 2);
    insertNode(&head, 3);
    printList(head);}
```

5. Membuat menu pilihan gabungan:

```
#include <iostream>
```

```
#include <string>

struct Node {
    int data;
    Node* next;};

Node* buatnode(int data) {
    Node* nodebaru = new Node;
    nodebaru->data = data;
    nodebaru->next = NULL;
    return nodebaru;}

void nodeawal(Node** head, int data) {
    Node* nodebaru = buatnode(data);
    nodebaru->next = *head;
    *head = nodebaru;}

void nodeakhir(Node** head, int data) {
    Node* nodebaru = buatnode(data);
    if (*head == NULL) {
        *head = nodebaru;}
    else {
        Node* current = *head;
        while (current->next != NULL) {
            current = current->next;}
        current->next = nodebaru;}}

void menampilkandata(Node* head) {
    Node* current = head;
    while (current != NULL) {
        std::cout << current->data << " ";
        current = current->next;}
    std::cout << '\n';}

Node* nodecari(Node* head, int targetData) {
    Node* current = head;
    while (current != NULL) {
        if (current->data == targetData) {
            return current;}
        current = current->next;}
    return NULL;}

void tampilkan(Node** head) {
    int choice;
    int data;
    Node* menemukanNode;

    do {
        std::cout << "\nPilihan\n 1. Menyisip Ujung Awal\n 2. Menyisip Ujung Akhir\n 3. Data yang Disimpan\n 4. Cari Data\n 5. Exit\nInput: ";
        std::cin >> choice;
        switch (choice) {
            case 1:
                std::cout << "Masukkan data: ";
                std::cin >> data;
                nodeawal(head, data);
                break;
            case 2:
                std::cout << "Masukkan data: ";
                std::cin >> data;
                nodeakhir(head, data);
                break;
            case 3:
                std::cout << "Data: ";
                menampilkandata(*head);
                break;
            case 4:
                std::cout << "Cari: ";
                std::cin >> data;
                menemukanNode = nodecari(*head, data);
                if (menemukanNode != NULL) {
                    std::cout << "Data dengan nama " << data << "
ditemukan.\n";}
                else {
                    std::cout << "Data dengan nama " << data << "
tidak ditemukan.\n";}
                break;
            case 5:
                std::cout << "Telah Keluar Program.\n";
                break;
            default:
                std::cout << "Input invalid\n";}
    } while (choice != 5);}

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

B. Latihan

1. Bangunlah Single linked dengan prinsip LIFO.

```
#include <iostream>

struct Node {
    std::string nama;
    int nrp;
```

```
   Node* next;
};

Node* nodebaru(const std::string& nama, int nrp) {
   Node* nodeBaru = new Node;
   nodeBaru->nama = nama;
   nodeBaru->nrp = nrp;
   nodeBaru->next = NULL;
   return nodeBaru;
}

void nodeawal(Node** ujung, const std::string& nama, int
nrp) {
   Node* nodeBaru = nodebaru(nama, nrp);
   nodeBaru->next = *ujung;
   *ujung = nodeBaru;
}

void hasildata(Node* ujung) {
   Node* current = ujung;
   while (current != NULL) {
      std::cout << "Nama: " << current->nama << ", NRP: "
<< current->nrp << '\n';
      current = current->next;
   }
}

int main() {
   Node* ujung = NULL;
   Node* tampung = NULL;

   int j = 0;
```

```
Node* nodeBaru = nodebaru("John Doe", 123);

if (nodeBaru == NULL) {
    std::cout << "Alokasi gagal" << std::endl;
} else {
    std::cout << "Nama: ";
    std::cin >> nodeBaru->nama;
    std::cout << "NRP : ";
    std::cin >> nodeBaru->nrp;

    if (j == 0) {
        tampung = nodeBaru;}

    // memasukkan node awal (LIFO)
    nodeawal(&ujung, nodeBaru->nama, nodeBaru->nrp);

    std::cout << "Hasil input: " << '\n';
    hasildata(ujung);}}
```

C. Kesimpulan

Percobaan dan latihan pada single linked list memberikan pemahaman yang kuat terkait konsep dasar dan operasional struktur data. Single linked list merupakan struktur data dinamis untuk penyisipan dan penghapusan elemen. Konsep prinsip LIFO dapat diaplikasikan dengan menyisipkan dan menghapus simpul di awal linked list. Dengan kelebihan dan kekurangan terkait pencarian data atau elemen, pemahaman terhadap single linked list penting untuk mengambil keputusan yang tepat dalam pemilihan struktur data berdasarkan kebutuhan.