Nama: Ardy Teguh Yulanda

NRP : 5223600021

Logbook DLL-MENU

Saya akan menjelaskan tentang bagaimana code saya bekerja

Pertama Code tersebut akan membuat Struct dengan nama **Node** yang berisikan data **int** dengan nama data, Node lainnya yang terhubung dengan nama **next** & **prev**

struct Node {

int data;

Node \*next;

Node \*prev;

};

Lalu akan membuat 2 variabel node dengan nama **head** & **tail** yang akan di buat null

Node \*head = NULL, \*tail = NULL;

Lalu ada Function *addHead* yang membutuhkan **int** agar bisa di panggil

void addHead(int data){

Function tersebut berfungsi untuk menambahkan **node baru** pada list sebagai **head**, ketika di panggil function akan mengecek apakah ada **head** dan **tail** pada list dan jika tidak ada maka **node baru** akan menjadi **head**

Node\* newNode = new Node();

if (head == NULL && tail == NULL){

head = newNode;

newNode->data = data;

newNode->next = NULL;

newNode->prev = NULL;

Jika ada **tail** dan tidak ada **head** maka node baru akan menjadi **head** dan akan menyambung dengan **node tail**

} else if (head == NULL && tail != NULL){

head = newNode;

newNode->data = data;

newNode->next = tail;

newNode->prev = NULL;

tail->prev = newNode;

Jika ada **head** dan tidak ada **tail** maka **node baru** akan di sambung di bagian depan **head** dan **head** lama akan menjadi **tail** lalu **node baru** menjadi **head**

} else if (head != NULL && tail == NULL){

newNode->data = data;

newNode->next = head;

head->prev = newNode;

tail = head;

head = newNode;

Dan jika ada keduanya maka **node baru** akan di sambung dengan head di bagian depan dan akan menjadi head baru

} else {

newNode->data = data;

newNode->next = head;

head->prev = newNode;

head = newNode;

}

Selanjutnya adalah Function *addTail* Function ini memiliki kegunaan dan fungsi yang kurang lebih mirip dengan *addHead* hanya berbeda hasil saja dan tempat menyambung Node

void addTail(int data) {

Node \*newNode = new Node();

if (head == NULL && tail == NULL){//mengecek apakah ada head & tail

tail = newNode;//jika tidak ada keduanya akan jadi tail

newNode->data = data;

newNode->next = NULL;

newNode->prev = NULL;

} else if ( head != NULL && tail == NULL){

tail = newNode;//jika ada head akan di sambung dan jadi tail

newNode->data = data;

newNode->prev = head;

newNode->next = NULL;

head->next = newNode;

} else if (head == NULL && tail != NULL){

newNode->data = data;//jika ada tail

newNode->prev = tail;//akan di sambung di belakang tail

newNode->next = NULL;

tail->next = newNode;

head = tail;//tail akan jadi head

tail = newNode;//node baru akan jadi tail

} else{ //node baru akan di sambung di belakang tail

newNode->data = data;//dan menjadi tail baru

newNode->prev = tail;

newNode->next = NULL;

tail->next = newNode;

tail = newNode;

}

}

Selanjutnya adalah Function *addNodeAfter* yang membutuhkan Node dan int agar bisa di panggil

void addNodeAfter(Node\* prevNode, int data){

Function ini bekerja dengan mengecek apakah **Node** yang di panggil adalah tail atau tidak lalu jika iya maka **node baru** akan di sambung di belakangnya

if(prevNode == tail){

prevNode->next = newNode;

newNode->prev = prevNode;

newNode->next = NULL;

Lalu setelah itu kode akan mengecek apakah **head** nya null atau tidak jika tidak **node baru** akan menjadi **tail**, tetapi jika iya maka **tail** akan menjadi **head** dan **node baru** akan menjadi **tail**

**if (head != NULL){**

**tail = newNode;**

**} else {**

**head = tail;**

**tail = newNode;**

**}**

Lalu jika **node** tersebut adalah **head** dan tail nya kosong maka kode akan langsung memanggil function AddTail

} else if (prevNode == head && tail == NULL){

addTail(data);

Sedangkan jika Node yang dipanggil bukan keduanya, maka kode akan memanggil next dari **Node** tersebut sebagai **temp** lalu menyambung kedua node tersebut dengan **node baru** sebagai di tengahnya

} else {

Node \*Temp = prevNode->next;

prevNode->next = newNode;

newNode->prev = prevNode;

newNode->next = Temp;

Temp->prev = newNode;

Selanjutnya adalah Function *addNodeBefore* yang fungsinya mirip dengan *addNodeAfter* tetapi pada function ini **node** yang di panggil akan di cek apakah node tersebut **head** atau tidak jika iya maka **node baru** akan di tambahkan di depan **head** dan setelah di sambung **node baru** akan menjadi **head**

void addNodeBefore(Node\* NextNode, int data){

Node\* newNode = new Node();

newNode->data = data;

if(NextNode == head) {//mengecek apakah nodenya head

newNode->next = head;//Jika iya maka Node baru akan di sambung

newNode->prev = NULL;//di depan head

if (tail != NULL){//lalu mengecek apakah tail ada

head = newNode;//jika iya node baru akan menjadi head

} else {//jika tidak head lama akan menjadi tail

tail = head;

head = newNode;// dan node baru akan menjadi head

}

} else if (NextNode == tail && head == NULL){

addHead(data); // Jika head null maka memanggil addHead

} else {

Node\* Temp = NextNode->prev;

Temp->next = newNode;

newNode->prev = Temp;

newNode->next = NextNode;

NextNode->prev = newNode;

Selanjutnya adalah Function *FiFo* & *LiFo* kedua function ini memiliki function yang sama dimana keduanya akan memulai membuat node baru dengan nama temp yang jika pada *FiFo* sebagai **head** dan jika pada *LiFo* sebagai **tail** lalu keduanya akan memulai iterasi sampai null menggunakan while loop dan menampilkan semua node ada yang sampai habis

void FiFo() {

Node \*temp = head;//Jika FiFo loop di mulai dari head

while (temp != NULL) {

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

temp = temp->next; //dan berpindah melalui next

} cout << endl;

}

void LiFo(){

Node \*temp = tail;//Sedangkan Lifo dari tail

while (temp != NULL){

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

temp = temp->prev;//dan berpindah melalui prex

} cout << endl;

Selanjutnya adalah function *deleteHead* & *deleteTail* yang memiliki fungsi kurang lebih mirip hanya berbeda **head tail** dan **next prev,** seperti contoh pada delete tail fungsi akan dimulai dengan mengecek apakah tail ada atau tidak

void deleteTail() {

if (tail != NULL) {

Jika iya tail akan di cek apakah memiliki node sebelumnya jika iya maka fungsi ini akan membuat **node temp** pada **tail lama** dan **tail** akan di pindah ke **node sebelum**nya dan **temp** akan di hapus

if(tail->prev != NULL){Node \*temp = tail;

tail = tail->prev;

tail->next = NULL;

delete temp;

Jika tidak memiliki node sebelumnya tail akan langsung di null & jika tail tidak ditemukan maka akan muncul pemberitahuan

} else {

tail = NULL;

}

} else {

cout << "Tail tidak ditemukan" << endl;

}

}

void deleteHead() {

if (head != NULL) {//Dimulai dengan mengecek head

if(head->next != NULL){Node \*temp = head;

head = head->next; //Jika head memiliki sambungan next

head->prev = NULL;//head akan di pindah ke next

delete temp; //dan di hapus

} else { // Jika tidak memiliki sambungan

head = NULL;//head akan di null

}

} else {//Jika head tidak ditemukan akan muncul pesan

cout << "Head tidak ditemukan" << endl;

Selanjutnya adalah Function *deleteNode* yang berfungi sebagai menghapus Node, function ini berjalan dengan membuat node baru dengan nama **current** di **head** dan mulai iterasi pada list hingga data pada habis/null dan jika data di **current** sama dengan data yang di panggil pada function *deleteNode* akan di mulai pengecekan lanjut, dan jika tidak maka **current** akan di next

void deleteNode(int data) {

Node \*current = head;

while (current != NULL) {

if (current->data == data) {

pengecekan lanjutnya adalah apakah **current** adalah **head** dan jika iya maka function ini akan memanggil function deleteHead

if (current == head) {

deleteHead();

return;

Jika **current** adalah **tail** maka function ini akan memanggil function deleteTail

**} else if (current == tail) {**

**deleteTail();**

**return;**

Jika bukan keduanya maka **node sebelum**nya akan di **next** kan dengan **node setelah**nya dan sebaliknya lalu current akan di hapus

} else {

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

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

delete current;

return;

Dan function terakhir adalah function yang menggunakan **node** yang berfungsi untuk mencari node dalam list dan me-returnnya, function ini bernama *cariNode* yang di awali dengan membuat node baru dengan nama **current** yang ada di **head**

Node\* cariNode(int data) {

Node\* current = head;

lalu akan dilakukan loop while hingga data pada **current** sama dengan **data** yang di panggil function dan jika tidak akan di next, lalu agar tidak terjadi infinte loop pada while harus ditambahkan dengan tidak null agar ketika nilai sudah menjadi null loop akan berakhir, dan ketika data **current** cocok maka nilai akan di return

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

current = current->next;

}

return current;

Pada baris selanjutnya hanyalah loop While yang akan menampilkan menu secara berulang ketika user menginputkan angka selain 11 tetapi sebelum itu kode nya harus mendeclare 2 variabel integer yang akan di gunakan selama berada di main yaitu **x** yang akan berperan sebagai placeholder input dan **pilihan** yang berperan sebagai memilih function mana yang akan di panggil oleh user

int main() {

int x, pilihan;

do {

cout << "\nMenu Operasi Linked List:\n";

cout << "1. Memasukan Data sebagai Head\n";

cout << "2. Memasukan Data sebagai Tail\n";

cout << "3. Mencari Node Tertentu\n";

cout << "4. Menampilkan Linked List dengan metode FiFo\n";

cout << "5. Menampilkan Linked List dengan metode LiFo\n";

cout << "6. Menghapus Node Tertentu\n";

cout << "7. Menghapus Head\n";

cout << "8. Menghapus Tail\n";

cout << "9. Menambahkan Data setelah Node tertentu\n";

cout << "10. Menambahkan Data sebelum Node tertentu\n";

cout << "11. Mengakhiri Program\n";

cout << "Masukkan pilihan: ";

cin >> pilihan;

switch (pilihan) {

case 1:

cout <<"Masukan Data yang ingin dimasukan: ";

cin >> x;

addHead(x);

cin.ignore();

break;

case 2: cout<<"Masukan Data yang ingin dimasukan:";

cin >> x;

addTail(x);

cin.ignore();

break;

case 3:

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

cin >> x;

cariNode(x);

if(cariNode(x) != NULL){

cout << "Data ditemukan: " << cariNode(x)->data << endl;

} else {

cout << "Data tidak ditemukan." << endl;

}

break;

case 4:cout<<"Data:";

FiFo();

break;

case 5:cout<<"Data:";

LiFo();

break;

case 6:cout<< "Masukan Data yang ingin dihapus: ";

cin >> x;

deleteNode(x);

break;

case 7:

deleteHead();

break;

case 8:

deleteTail();

break;

case 9:cout<<"Masukan Node yang berada di list: ";

cin >> x;

if (cariNode(x) != NULL){

Node \*target = cariNode(x);

cout << "Masukan Node yang akan di tambahkan setelah node "<< x <<": ";

cin >> x;

addNodeAfter(target, x);

break;}

else {

cout << "Node tidak ditemukan." << endl;

break;

}

case 10:cout<<"Masukan Node yang berada di list: ";

cin >> x;

if (cariNode(x) != NULL){

Node \*target = cariNode(x);

cout << "Masukan Node yang akan di tambahkan Sebelum node "<< x <<": ";

cin >> x;

addNodeBefore(target, x);

break;}

else {

cout << "Node tidak ditemukan." << endl;

break;

}

case 11:

cout << "Terima kasih telah menggunakan program ini.\n";

break;

default:

if (pilihan > 11) {

cout << "Pilihan tidak valid! Masukkan angka 1-11.\n";

}

else {

cout << "Pilihan tidak valid!\n";

}

break;

}

} while (pilihan != 11);

return 0;