Nama : Reihan Al Sya’Ban

NIM : 2109106051

Kelas : A2 2021

**Struktur Data**

**POSTTEST 3**

#include <iostream>

#include <string>

#include <conio.h>

#include <windows.h>

using namespace std;

struct tim\_liga{

string nama\_tim;

string kota\_asal;

string nama\_stadion;

string suporter;

int jumlah\_pemain;

};

tim\_liga tl;

struct Node{

tim\_liga data;

Node \*next = NULL;

};

bool isEmpty(Node \*head){

if (head == NULL){

return true;

}

return false;

}

int length(Node \*head){

int jumlah = 0;

while (head != NULL){

jumlah++;

head = head->next;

}

return jumlah;

}

Node \*newNode(){

Node \*nodeBaru = new Node;

cout<<"\n=============== MASUKKAN DATA ================"<<endl;

cout<<"Nama TIM : ";

cin>>nodeBaru->data.nama\_tim;

cout<<"Kota Asal : ";

cin>>nodeBaru->data.kota\_asal;

cout<<"Nama Stadion : ";

cin>>nodeBaru->data.nama\_stadion;

cout<<"Nama Suporter : ";

cin>>nodeBaru->data.suporter;

cout<<"Jumlah Pemain : ";

cin>>nodeBaru->data.jumlah\_pemain;

system("CLS");

return nodeBaru;

}

void addFirst(Node \*\*head){

Node \*nodeBaru = newNode();

nodeBaru->next = \*head;

\*head = nodeBaru;

}

void addLast(Node \*\*head){

Node \*nodeBaru = newNode();

if (isEmpty(\*head)){

\*head = nodeBaru;

}

else{

Node \*temp = \*head;

while (temp->next != NULL){

temp = temp->next;

}

temp->next = nodeBaru;

}

}

void addMiddle(Node \*\*head) {

if (isEmpty(\*head)) {

cout << "\n================= DATA KOSONG =================" << endl;

getch();

system("CLS");

return;

}

Node \*nodeBaru = \*head;

int i = 1;

while (nodeBaru != NULL) {

cout <<"\n=============== Data Ke-"<<i<<" ================"<<endl

<<"Nama TIM : "<<nodeBaru->data.nama\_tim<<endl

<<"Kota Asal : "<<nodeBaru->data.kota\_asal<<endl

<<"Nama Stadion : "<<nodeBaru->data.nama\_stadion<<endl

<<"Nama Suporter : "<<nodeBaru->data.suporter<<endl

<<"Jumlah Pemain : "<<nodeBaru->data.jumlah\_pemain<<endl;

i += 1;

nodeBaru = nodeBaru->next;

}

int index;

cout << "\n- Masukkan Data Sebelum Node Ke-";

cin >> index;

if (index > 0 && index <= length(\*head)){

Node \*nodeBaru = newNode();

int nomor = 1;

Node \*temp = (\*head);

while(nomor < index-1){

temp = temp->next;

nomor++;

}

nodeBaru->next = temp->next;

temp->next = nodeBaru;

}

else{

cout << "\n=============== DATA TIDAK TERSEDIA ===============" << endl;

}

}

void addMenu(Node\*& HEAD){

int pilih = 1;

cout << "\n================= TAMBAH MENU =================" << endl

<< "\n1. Add First" << endl

<< "2. Add Middle" << endl

<< "3. Add Last" << endl

<< "Pilih : ";

cin>>pilih;

system("CLS");

if(pilih == 1){

addFirst(&HEAD);

}

else if(pilih == 2){

addMiddle(&HEAD);

}

else{

addLast(&HEAD);

}

}

void deleteFirst(Node \*\*head){

if (isEmpty(\*head)){

cout << "\nLinked List Kosong" << endl;

return;

}

\*head = (\*head)->next;

cout << "\nDelete Node Berhasil" << endl;

}

void deleteLast(Node \*\*head){

if (isEmpty(\*head)){

cout << "\nLinked List Kosong" << endl;

return;

}

if ((\*head)->next == NULL){

\*head = NULL;

cout << "\nDelete Node Berhasil" << endl;

return;

}

Node \*temp = \*head;

while (temp->next->next != NULL){

temp = temp->next;

}

Node \*varDelete = temp->next;

temp->next = NULL;

delete varDelete;

cout << "\nDelete Node Berhasil" << endl;

}

void deleteMiddle(Node\*& head) {

if (head == NULL)

return;

if (head->next == NULL) {

delete head;

return;

}

struct Node\* copyHead = head;

int count = length(head);

int mid = count / 2;

while (mid-- > 1)

head = head->next;

head->next = head->next->next;

}

void deleteMenu(Node\*& HEAD){

int pilih = 1;

cout << "\n================= DELETE MENU =================" << endl

<< "\n1. Delete First" << endl

<< "2. Delete Middle" << endl

<< "3. Delete Last" << endl;

cout<<"Pilih : ";

cin>>pilih;

system("CLS");

if(pilih == 1){

deleteFirst(&HEAD);

}

else if(pilih == 2){

deleteMiddle(HEAD);

}

else{

deleteLast(&HEAD);

}

}

void display(Node \*head){

if (isEmpty(head)){

cout << "Linked List Kosong" << endl;

return;

}

cout << "\n=================== DATA TIM ==================" << endl;

Node \*temp = head;

while (temp != NULL){

cout<<"\nNama TIM : "<< temp->data.nama\_tim <<endl;

cout<<"Kota Asal : "<< temp->data.kota\_asal <<endl;

cout<<"Nama Stadion : "<< temp->data.nama\_stadion <<endl;

cout<<"Nama Suporter : "<< temp->data.suporter <<endl;

cout<<"Jumlah Pemain : "<< temp->data.jumlah\_pemain <<endl;

cout << "\n===============================================" << endl;

temp = temp->next;

}

getch();

system("CLS");

cout << endl;

}

void update(Node \*\*head){

if (isEmpty(\*head)){

cout << "\nLinked List Kosong" << endl;

getch();

system("CLS");

return;

}

int pilihan = 0;

cout << "Banyak node ada : " << length(\*head) << endl;

cout << "Pilih node yang ingin diupdate : ";

cin >> pilihan;

Node \*temp = \*head;

if (pilihan > 0 && pilihan <= length(\*head)){

for (int i = 1; i < pilihan; i++){

temp = temp->next;

}

cout<<"\nNama TIM : ";

cin>>temp->data.nama\_tim;

cout<<"Kota Asal : ";

cin>>temp->data.kota\_asal;

cout<<"Nama Stadion : ";

cin>>temp->data.nama\_stadion;

cout<<"Nama Suporter : ";

cin>>temp->data.suporter;

cout<<"Jumlah Pemain : ";

cin>>temp->data.jumlah\_pemain;

getch();

system("CLS");

}

else{

cout << "\nInputan melebihi jumlah node" << endl;

getch();

system("CLS");

}

}

Node \*SortedMerge(Node \*a, Node \*b, int attribute, int type);

void FrontBackSplit(Node \*source, Node \*\*frontRef, Node \*\*backRef);

void MergeSort(Node \*\*headRef, int attribute, int type)

{

Node \*head = \*headRef;

Node \*a;

Node \*b;

if ((head == NULL) || (head->next == NULL))

{

return;

}

FrontBackSplit(head, &a, &b);

MergeSort(&a, attribute, type);

MergeSort(&b, attribute, type);

\*headRef = SortedMerge(a, b, attribute, type);

}

Node \*SortedMerge(Node \*a, Node \*b, int attribute, int type)

{

Node \*result = NULL;

bool isAsc = type == 1;

bool condition = false;

if (a == NULL)

return (b);

else if (b == NULL)

return (a);

if(attribute == 1) {

condition = isAsc ? a->data.nama\_tim <= b->data.nama\_tim : a->data.nama\_tim >= b->data.nama\_tim;

} else if(attribute == 2) {

condition = isAsc ? a->data.suporter <= b->data.suporter : a->data.suporter >= b->data.suporter;

} else if(attribute == 3) {

condition = isAsc ? a->data.jumlah\_pemain <= b->data.jumlah\_pemain : a->data.jumlah\_pemain >= b->data.jumlah\_pemain;

}

if (condition) {

result = a;

result->next = SortedMerge(a->next, b, attribute, type);

} else {

result = b;

result->next = SortedMerge(a, b->next, attribute, type);

}

return (result);

}

void FrontBackSplit(Node \*source, Node \*\*frontRef, Node \*\*backRef)

{

Node \*fast;

Node \*slow;

slow = source;

fast = source->next;

while (fast != NULL)

{

fast = fast->next;

if (fast != NULL)

{

slow = slow->next;

fast = fast->next;

}

}

\*frontRef = source;

\*backRef = slow->next;

slow->next = NULL;

}

void sort(Node \*\*head)

{

int attribute = 1;

int type = 1;

cout<< "=================== KATEGORI ==================" << endl

<< "1. Nama Tim" << endl

<< "2. Nama Suporter" << endl

<< "3. Jumlah Pemain" << endl

<< "Pilih : ";

cin >> attribute;

cout<< "==================== METODE ==================" << endl

<< "1. Ascending" << endl

<< "2. Descending" << endl

<< "Pilih : ";

cin >> type;

MergeSort(head, attribute, type);

cout << "Data Berhasil Disorting" << endl;

}

int main()

{

Node \*HEAD = NULL;

int pilihan = 0;

while (pilihan != 6)

{

cout << "\n================= LINKED LIST =================" << endl;

cout << "\n1. Create" << endl;

cout << "2. Read" << endl;

cout << "3. Update" << endl;

cout << "4. Delete" << endl;

cout << "5. Sorting" << endl;

cout << "6. Exit Program" << endl;

cout << "Masukan pilihan : ";

cin >> pilihan;

system("CLS");

switch (pilihan)

{

case 1:

addMenu(HEAD);

break;

case 2:

display(HEAD);

break;

case 3:

update(&HEAD);

break;

case 4:

deleteMenu(HEAD);

break;

case 5:

sort(&HEAD);

break;

case 6:

break;

default:

break;

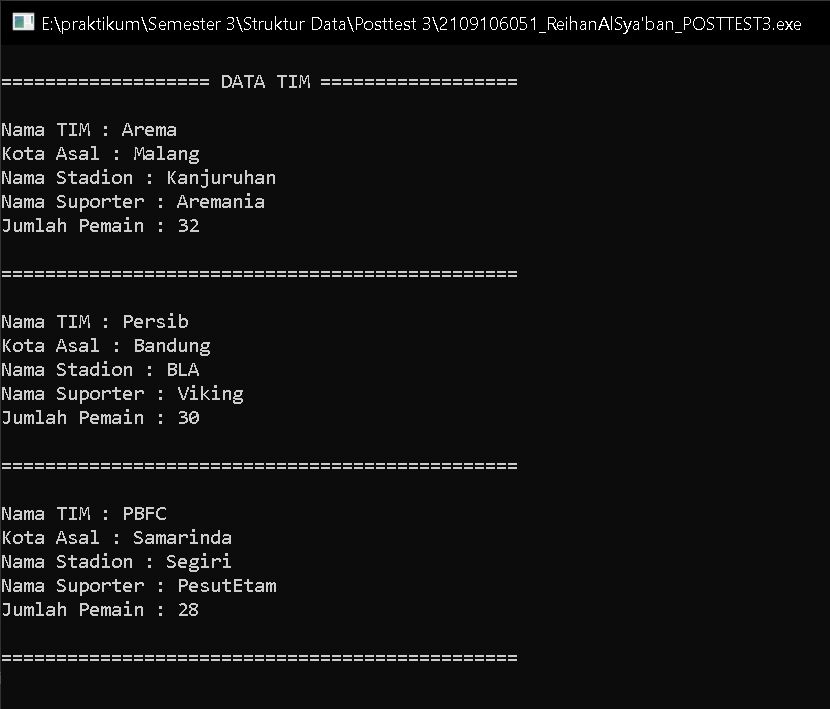
}

}

return 0;

}

1. Data Awal



1. Setelah di sorting berdasarkan jumlah pemain secara descending

