

**LAPORAN PRAKTIKUM
STRUKTUR DATA**

**MODUL 13
MULTI LINKED LIST**



Disusun Oleh :
NAMA : Dealova Agta Syahlevi
NIM : 103112400124

Dosen
FAHRUDIN MUKTI WIBOWO

**PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025**

A. Dasar Teori

C++ adalah bahasa pemrograman yang dikembangkan oleh Bjarne Stroustrup sebagai pengembangan dari bahasa C. Bahasa ini mendukung pemrograman prosedural maupun berorientasi objek, jadi fleksibel digunakan dalam pembuatan aplikasi, game, sampai sistem operasi. Keunggulan C++ ada pada efisiennya yang tinggi serta kemampuannya menjembatani pemrograman tingkat rendah dan tinggi.

Dalam perkuliahan C++ sering digunakan karena strukturnya hampir sama dengan bahasa C namun memiliki fitur yang lebih modern. Jadi bisa lebih memudahkan kami mahasiswa memahami dasar-dasar pemrograman, struktur data, dan algoritma, sekaligus mengenalkan konsep berorientasi objek. Selain itu, bahasa C++ juga membiasakan kami mahasiswa dengan bahasa yang banyak dipakai dalam industri perangkat lunak dan pengembangan sistem.

B. Guided (berisi screenshot source code & output program disertai penjelasannya)

Guided.cpp

```
#include <iostream>
#include <string>
using namespace std;

struct ChildNode
{
    string info;
    ChildNode *next;
    ChildNode *prev;
};

struct ParentNode
{
    string info;
    ChildNode *childHead;
    ParentNode *next;
    ParentNode *prev;
};
```

```
ParentNode *createParent(string info)
{
    ParentNode *newNode = new ParentNode;
    newNode->info = info;
    newNode->childHead = NULL;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}

ChildNode *createChild(string info)
{
    ChildNode *newNode = new ChildNode;
    newNode->info = info;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}

void insertParent(ParentNode *&head, string info)
{
    ParentNode *newNode = createParent(info);
    if (head == NULL)
    {
        head = newNode;
    }
    else
    {
        ParentNode *temp = head;
        while (temp->next != NULL)
        {
```

```

        temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
}
}

void insertChild(ParentNode *head, string parentInfo, string childInfo)
{
    ParentNode *p = head;
    while (p != NULL && p->info != parentInfo)
    {
        p = p ->next;
    }

    if (p != NULL)
    {
        ChildNode *newChild = createChild(childInfo);
        if (p->childHead == NULL)
        {
            p->childHead = newChild;
        } else {
            ChildNode *C = p->childHead;
            while (C->next != NULL) {
                C = C ->next;
            }
            C->next = newChild;
            newChild->prev = C;
        }
    }
}

```

```

void printAll(ParentNode *head)
{
    while (head != NULL)
    {
        cout << head->info;
        ChildNode *c = head->childHead;
        while (c != NULL)
        {
            cout << " -> " << c->info;
            c = c->next;
        }
        cout << endl;
        head = head->next;
    }
}

void updateParent(ParentNode *head, string oldInfo, string newInfo)
{
    ParentNode *p = head;
    while (p != NULL)
    {
        if (p->info == oldInfo)
        {
            p->info = newInfo;
            return;
        }
        p = p->next;
    }
}

void updateChild(ParentNode *head, string parentInfo, string
oldchildInfo, string newchildInfo)
{
}

```

```

ParentNode *p = head;

while (p != NULL && p->info != parentInfo)
{
    p = p ->next;
}

if (p != NULL)
{
    ChildNode *c = p->childHead;
    while (c != NULL)
    {
        if (c->info == oldchildInfo)
        {
            c->info = newchildInfo;
            return;
        }
        c = c->next;
    }
}

void deleteChild(ParentNode *head, string parentInfo, string childInfo)
{
    ParentNode *p = head;
    while (p != NULL && p->info != parentInfo)
    {
        p = p ->next;
    }

    if (p != NULL)
    {
        ChildNode *c = p->childHead;

```

```

        while (c != NULL)
    {
        if (c->info == childInfo)
        {
            if (c == p->childHead)
            {
                p->childHead = c->next;
                if (p->childHead != NULL)
                {
                    p->childHead->prev = NULL;
                }
            }
            else
            {
                c->prev->next = c->next;
                if (c->next != NULL)
                {
                    c->next->prev = c->prev;
                }
            }
            delete c;
            return;
        }
        c = c->next;
    }
}

void deleteParent(ParentNode *&head, string info)
{
    ParentNode *p = head;
    while (p != NULL)

```

```
{  
    if (p->info == info)  
    {  
        ChildNode *c = p->childHead;  
        while (c != NULL)  
        {  
            ChildNode *tempC = c;  
            c = c->next;  
            delete tempC;  
        }  
        if (p == head)  
        {  
            head = p->next;  
            if (head != NULL)  
            {  
                head->prev = NULL;  
            }  
        }  
        else  
        {  
            p->prev->next = p->next;  
            if (p->next != NULL)  
            {  
                p->next->prev = p->prev;  
            }  
        }  
        delete p;  
        return;  
    }  
    p = p->next;  
}
```

```
int main()
{
    ParentNode *list = NULL;
    insertParent(list, "Parent A");
    insertParent(list, "Parent B");
    insertParent(list, "Parent C");

    cout << "\nSetelah InsertParent:" << endl;
    printAll(list);

    insertChild(list, "Parent A", "Child A1");
    insertChild(list, "Parent A", "Child A2");
    insertChild(list, "Parent B", "Child B1");

    cout << "\nSetelah InsertChild:" << endl;
    printAll(list);

    updateParent(list, "Parent B", "Parent B");
    updateChild(list, "Parent A", "Child A1", "Child A1");

    cout << "\nSetelah Update:" << endl;
    printAll(list);

    deleteChild(list, "Parent A", "Child A2");
    deleteParent(list, "Parent C");

    cout << "\nSetelah Delete:" << endl;
    printAll(list);

    return 0;
}
```

Screenshots Output

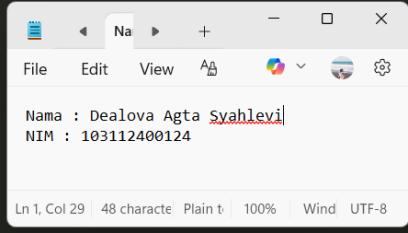
```
PS D:\StrukturDataSem3\103112400124_DealovaAgtaSyahlevi\Modul11-Strukdat> cd "d:\StrukturDataSem3\103112400124_DealovaAgtaSyahlevi\Modul13-Strukdat"
if ($?) { g++ guided.cpp -o guided } ; if ($?) { .\guided }

Setelah InsertParent:
Parent A
Parent B
Parent C

Setelah InsertChild:
Parent A -> Child A1 -> Child A2
Parent B -> Child B1
Parent C

Setelah Update:
Parent A -> Child A1 -> Child A2
Parent B -> Child B1
Parent C

Setelah Delete:
Parent A -> Child A1
Parent B -> Child B1
PS D:\StrukturDataSem3\103112400124_DealovaAgtaSyahlevi\Modul13-Strukdat
```



Deskripsi:

Program ini mengimplementasikan struktur multilist yang terdiri dari node Parent dan Child menggunakan linked list ganda. Setiap parent bisa memiliki daftar child yang saling terhubung. Program menyediakan fungsi untuk menambah, mengubah, menghapus, dan menampilkan parent maupun child. Pada fungsi main, program membuat tiga parent, menambahkan beberapa child, melakukan update data, lalu menghapus child dan parent tertentu, kemudian menampilkan hasil perubahan di setiap tahap.

C. Unguided/Tugas (berisi screenshot source code & output program disertai penjelasannya)

Unguided 1

Multilist.h

```
#ifndef MULTILIST_H_INCLUDED
#define MULTILIST_H_INCLUDED

#include <iostream>
using namespace std;

#define Nil NULL

typedef bool boolean;
typedef int infotypeinduk;
typedef int infotypeanak;
```

```
typedef struct elemen list induk *address;
typedef struct elemen list anak *address anak;

struct elemen list anak {
    infotypeanak info;
    address anak next;
    address anak prev;
};

struct listanak {
    address anak first;
    address anak last;
};

struct elemen list induk {
    infotypeinduk info;
    listanak lanak;
    address next;
    address prev;
};

struct listinduk {
    address first;
    address last;
};

boolean ListEmpty(listinduk L);
boolean ListEmptyAnak(listanak L);
void CreateList(listinduk &L);
void CreateListAnak(listanak &L);
address alokasi(infotypeinduk X);
```

```

address anak alokasiAnak(infotypeanak X);

void dealokasi(address P);

void dealokasiAnak(address anak P);

address findElm(listinduk L, infotypeinduk X);

address_anak findElm(listanak L, infotypeanak X);

void insertFirst(listinduk &L, address P);

void insertLast(listinduk &L, address P);

void insertAfter(listinduk &L, address Prec, address P);

void insertFirstAnak(listanak &L, address_anak P);

void insertLastAnak(listanak &L, address_anak P);

void insertAfterAnak(listanak &L, address_anak P, address_anak Prec);

void delFirst(listinduk &L, address &P);

void delLast(listinduk &L, address &P);

void delP(listinduk &L, infotypeinduk X);

void delFirstAnak(listanak &L, address_anak &P);

void delLastAnak(listanak &L, address_anak &P);

void delAfterAnak(listanak &L, address_anak &P, address_anak Prec);

void delPAnak(listanak &L, infotypeanak X);

void printInfo(listinduk L);

void printInfoAnak(listanak L);

#endif

```

Multilist.cpp

```

#include <iostream>

#include "multilist.h"

using namespace std;

boolean ListEmpty(listinduk L) {

    return (L.first == Nil && L.last == Nil);

```

```
}

boolean ListEmptyAnak(listanak L) {
    return (L.first == Nil && L.last == Nil);
}

void CreateList(listinduk &L) {
    L.first = Nil;
    L.last = Nil;
}

void CreateListAnak(listanak &L) {
    L.first = Nil;
    L.last = Nil;
}

address alokasi(infotypeinduk X) {
    address P = new elemen list induk;

    P->info = X;
    CreateListAnak(P->lanak);
    P->next = Nil;
    P->prev = Nil;
    return P;
}

address anak alokasiAnak(infotypeanak X) {
    address anak P = new elemen list anak;

    P->info = X;
    P->next = Nil;
    P->prev = Nil;
```

```
        return P;
    }

void dealokasi(address P) {
    delete P;
}

void dealokasiAnak(address_anak P) {
    delete P;
}

address findElm(listinduk L, infotypeinduk X) {
    address P = L.first;

    while (P != Nil) {
        if (P->info == X)
            return P;
        P = P->next;
    }
    return Nil;
}

address_anak findElm(listanak L, infotypeanak X) {
    address_anak P = L.first;

    while (P != Nil) {
        if (P->info == X)
            return P;
        P = P->next;
    }
    return Nil;
}
```

```

void insertFirst(listinduk &L, address P) {
    if (ListEmpty(L)) {
        L.first = P;
        L.last = P;
    } else {
        P->next = L.first;
        L.first->prev = P;
        L.first = P;
    }
}

void insertLast(listinduk &L, address P) {
    if (ListEmpty(L)) {
        L.first = P;
        L.last = P;
    } else {
        L.last->next = P;
        P->prev = L.last;
        L.last = P;
    }
}

void insertAfter(listinduk &L, address Prec, address P) {
    if (Prec->next == Nil) {
        insertLast(L, P);
    } else {
        P->next = Prec->next;
        P->prev = Prec;
        Prec->next->prev = P;
        Prec->next = P;
    }
}

```

```

}

void insertFirstAnak(listanak &L, address_anak P) {
    if (ListEmptyAnak(L)) {
        L.first = P;
        L.last = P;
    } else {
        P->next = L.first;
        L.first->prev = P;
        L.first = P;
    }
}

void insertLastAnak(listanak &L, address_anak P) {
    if (ListEmptyAnak(L)) {
        L.first = P;
        L.last = P;
    } else {
        L.last->next = P;
        P->prev = L.last;
        L.last = P;
    }
}

void delFirst(listinduk &L, address &P) {
    P = L.first;
    if (L.first == L.last) {
        L.first = Nil;
        L.last = Nil;
    } else {
        L.first = P->next;
        L.first->prev = Nil;
    }
}

```

```

    }

}

void delLast(listinduk &L, address &P) {
    P = L.last;
    if (L.first == L.last) {
        L.first = Nil;
        L.last = Nil;
    } else {
        L.last = P->prev;
        L.last->next = Nil;
    }
}

void delP(listinduk &L, infotypeinduk X) {
    address P = findElm(L, X);

    if (P != Nil) {
        address anak Q = P->lanak.first;
        while (Q != Nil) {
            address anak temp = Q;
            Q = Q->next;
            dealokasiAnak(temp);
        }

        if (P == L.first) {
            delFirst(L, P);
        } else if (P == L.last) {
            delLast(L, P);
        } else {
            P->prev->next = P->next;
            P->next->prev = P->prev;
        }
    }
}

```

```
    }

    dealokasi(P);

}

}

void delLastAnak(listanak &L, address anak &P) {

    P = L.last;

    if (L.first == L.last) {

        L.first = Nil;

        L.last = Nil;

    } else {

        L.last = P->prev;

        L.last->next = Nil;

    }

}

void printInfo(listinduk L) {

    address P = L.first;

    while (P != Nil) {

        cout << "Induk: " << P->info << endl;

        cout << " Anak: ";

        address anak Q = P->lanak.first;

        if (Q == Nil)

            cout << "(tidak ada anak)";

        else {

            while (Q != Nil) {

                cout << Q->info << " ";

                Q = Q->next;

            }

        }

    }

}
```

```
    }

    cout << endl;

    P = P->next;

}

}
```

Main1.cpp

```
#include <iostream>
#include "multilist.h"
using namespace std;

int main() {
    listinduk L;
    CreateList(L);

    cout << " INSERT INDUK " << endl;
    insertLast(L, alokasi(10));
    insertLast(L, alokasi(20));
    insertLast(L, alokasi(30));

    cout << "\n INSERT ANAK PADA INDUK " << endl;
    address induk20 = findElm(L, 20);

    insertLastAnak(induk20->lanak, alokasiAnak(101));
    insertLastAnak(induk20->lanak, alokasiAnak(102));
    insertLastAnak(induk20->lanak, alokasiAnak(103));

    cout << "\n INSERT ANAK PADA INDUK " << endl;
    address induk10 = findElm(L, 10);

    insertLastAnak(induk10->lanak, alokasiAnak(201));
```

```

        insertLastAnak(induk10->lanak, alokasiAnak(202));

        cout << "\n DATA MULTILIST " << endl;
        printInfo(L);

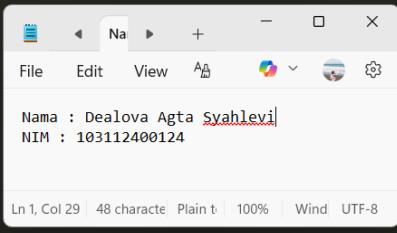
        cout << "\n DELETE INDUK " << endl;
        delP(L, 20);

        printInfo(L);

    return 0;
}

```

Screenshots Output



```

PS D:\StrukturDataSem3\103112400124_DealovaAgtaSyahlevi\Modul13-Strukdat> cd "d:\StrukturDataSem3\103112400124_DealovaAgtaSyahlevi\Modul13-Strukdat\" ;
if ($?) { g++ main1.cpp -o main1 } ; if ($?) { ./main1 }
INSERT INDUK

INSERT ANAK PADA INDUK
INSERT ANAK PADA INDUK

DATA MULTILIST
Induk: 10
    Anak: 201 202
Induk: 20
    Anak: 101 102 103
Induk: 30
    Anak: (tidak ada anak)

DELETE INDUK
Induk: 10
    Anak: 201 202
Induk: 30
    Anak: (tidak ada anak)
PS D:\StrukturDataSem3\103112400124_DealovaAgtaSyahlevi\Modul13-Strukdat>

```

Deskripsi:

Program ini membuat struktur multilist yang terdiri dari daftar induk dan setiap induk memiliki daftar anak berupa double linked list. Program menyediakan fitur untuk membuat list, menambah induk, menambah anak pada induk tertentu, mencari elemen, menghapus induk beserta seluruh anaknya, serta menampilkan keseluruhan data. Pada main, program menambahkan beberapa induk (10, 20, 30), menambahkan anak pada induk 20 dan 10, lalu menampilkan seluruh data, kemudian menghapus induk 20 beserta anak-anaknya, dan menampilkan hasil akhirnya.

Unguided 2

circularlist.h

```
#ifndef CIRCULARLIST_H_INCLUDED
#define CIRCULARLIST_H_INCLUDED
#define Nil NULL

#include <string>
using namespace std;

struct infotype {
    string nama;
    string nim;
    char jenis_kelamin;
    float ipk;
};

typedef struct ElmList *address;

struct ElmList {
    infotype info;
    address next;
};

struct List {
    address first;
};

void createList(List &L);
address alokasi(infotype x);
void dealokasi(address P);
void insertFirst(List &L, address P);
void insertAfter(List &L, address Prec, address P);
```

```

void insertLast(List &L, address P);
void deleteFirst(List &L, address &P);
void deleteAfter(List &L, address Prec, address &P);
void deleteLast(List &L, address &P);
address findElm(List L, infotype x);
void printInfo(List L);

#endif

```

circularlist.cpp

```

#include <iostream>
#include "circularlist.h"
using namespace std;

void createList(List &L){
    L.first = Nil;
}

address alokasi(infotype x){
    address P = new ElmList;
    P->info = x;
    P->next = P;
    return P;
}

void dealokasi(address P){
    delete P;
}

void insertFirst(List &L, address P){
    if(L.first == Nil){

```

```

    L.first = P;

} else {
    address Q = L.first;

    while(Q->next != L.first){
        Q = Q->next;
    }

    Q->next = P;
    P->next = L.first;
    L.first = P;
}

void insertAfter(List &L, address Prec, address P){

if(Prec != Nil){
    P->next = Prec->next;
    Prec->next = P;
}
}

void insertLast(List &L, address P){

if(L.first == Nil){
    insertFirst(L, P);
} else {
    address Q = L.first;

    while(Q->next != L.first){
        Q = Q->next;
    }

    Q->next = P;
}
}

```

```

    P->next = L.first;
}

}

void deleteFirst(List &L, address &P){
    if(L.first == Nil){
        P = Nil;
    }
    else if(L.first->next == L.first){
        P = L.first;
        L.first = Nil;
    }
    else {
        address last = L.first;

        while(last->next != L.first){
            last = last->next;
        }

        P = L.first;
        L.first = L.first->next;
        last->next = L.first;
    }
}

void deleteAfter(List &L, address Prec, address &P){
    if(Prec != Nil){
        P = Prec->next;
        Prec->next = P->next;
        P->next = Nil;
    }
}

```

```
void deleteLast(List &L, address &P){  
    if(L.first == Nil){  
        P = Nil;  
    }  
    else if(L.first->next == L.first){  
        P = L.first;  
        L.first = Nil;  
    }  
    else {  
        address Prev = Nil;  
        address Q = L.first;  
  
        while(Q->next != L.first){  
            Prev = Q;  
            Q = Q->next;  
        }  
  
        P = Q;  
        Prev->next = L.first;  
    }  
}  
  
address findElm(List L, infotype x){  
    if(L.first == Nil) return Nil;  
  
    address P = L.first;  
  
    do {  
        if(P->info.nim == x.nim){  
            return P;  
        }  
    }
```

```

    P = P->next;
} while(P != L.first);

return Nil;
}

void printInfo(List L){
if(L.first == Nil){
    cout << "List kosong" << endl;
    return;
}

address P = L.first;
do {
    cout << "Nama : " << P->info.nama << endl;
    cout << "NIM : " << P->info.nim << endl;
    cout << "L/P : " << P->info.jenis_kelamin << endl;
    cout << "IPK : " << P->info.ipk << endl;
    cout << endl;

    P = P->next;
} while(P != L.first);
}

```

Main2.cpp

```

#include <iostream>
#include "circularlist.h"
#include "circularlist.cpp"
using namespace std;

address createData(string nama, string nim, char jenis_kelamin,

```

```
float ipk)
{
    infotype x;
    x.nama = nama;
    x.nim = nim;
    x.jenis_kelamin = jenis_kelamin;
    x.ipk = ipk;
    return alokasi(x);
}

int main()
{
    List L;
    address P1 = Nil;
    address P2 = Nil;
    infotype x;
    createList(L);

    cout<<"coba insert first, last, dan after"<<endl;
    P1 = createData("Dani", "04", 'l', 4.0);
    insertFirst(L,P1);

    P1 = createData("FBagus", "06", 'l', 3.45);
    insertLast(L,P1);

    P1 = createData("Dina", "02", 'l', 3.71);
    insertFirst(L,P1);

    P1 = createData("Alif", "01", 'l', 3.3);
    insertFirst(L,P1);

    P1 = createData("Gita", "07", 'p', 3.75);
    insertLast(L,P1);
```

```
x.nim = "07";
P1 = findElm(L,x);
P2 = createData("Indi", "03", 'p', 3.5);
insertAfter(L, P1, P2);

x.nim = "02";
P1 = findElm(L,x);
P2 = createData("Ilham", "08", 'p', 3.3);
insertAfter(L, P1, P2);

x.nim = "04";
P1 = findElm(L,x);
P2 = createData("Ela", "05", 'p', 3.4);
insertAfter(L, P1, P2);
printInfo(L);
return 0;
}
```

Screenshots Output

```

PS D:\StrukturDataSem3\103112400124_DealovaAgtaSyahlevi\Modul13-Strukdat> cd "d:\StrukturDataSem3\103112400124_DealovaAgtaSyahlevi\Modul13-Strukdat"
if ($?) { g++ main2.cpp -o main2 } ; if ($?) { .\main2 }

coba insert first, last, dan after
Nama : Alif
NIM : 01
L/P : l
IPK : 3.3

Nama : Dina
NIM : 02
L/P : l
IPK : 3.71

Nama : Ilham
NIM : 08
L/P : p
IPK : 3.3

Nama : Dani
NIM : 04
L/P : l
IPK : 4

Nama : Ela
NIM : 05
L/P : p
IPK : 3.4

Nama : FBagus
NIM : 06
L/P : l
IPK : 3.45

Nama : Gita
NIM : 07
L/P : p
IPK : 3.75

Nama : Indi
NIM : 03
L/P : p

```

Deskripsi:

Program ini mengimplementasikan circular singly linked list untuk menyimpan data mahasiswa berupa nama, NIM, jenis kelamin, dan IPK. List berbentuk melingkar sehingga elemen terakhir selalu menunjuk kembali ke elemen pertama. Program menyediakan operasi dasar seperti membuat list, menambah elemen di awal, akhir, maupun setelah elemen tertentu, mencari data berdasarkan NIM, serta menampilkan seluruh isi list secara berputar. Pada fungsi main, program membuat beberapa data mahasiswa, memasukkannya ke list dengan berbagai metode insert, menambahkan data baru setelah node tertentu menggunakan findElm, lalu menampilkan seluruh data mahasiswa dalam urutan circular.

D. Kesimpulan

Secara keseluruhan, tiga program yang kamu berikan sama-sama membahas struktur data linked list, tetapi dalam bentuk dan tujuan yang berbeda. Program pertama menggunakan multilist sederhana yang menghubungkan parent dan child menggunakan double linked list. Program kedua memperluas konsep tersebut menjadi multilist yang lebih terstruktur, lengkap dengan operasi insert, delete, search, dan print untuk induk maupun anak. Program ketiga menggunakan circular singly linked list untuk menyimpan

data mahasiswa yang saling terhubung dalam bentuk melingkar. Ketiga program ini menunjukkan bagaimana linked list dapat dimodifikasi menjadi berbagai bentuk—multilist, double linked list, dan circular list—untuk mengelola data dinamis dengan hubungan berbeda sesuai kebutuhan aplikasi.

E. Referensi

GeeksforGeeks. "C++ Programming Language – Introduction." <https://www.geeksforgeeks.org/c-plus-plus/>

Wikipedia. "C++." <https://id.wikipedia.org/wiki/C%2B%2B>