LAPORAN PRAKTIKUM STRUKTUR DATA

MODUL 13

MULTI LINKED LIST

Disusun Oleh :
NAMA : DEVI YULIANA
NIM : 103112400151

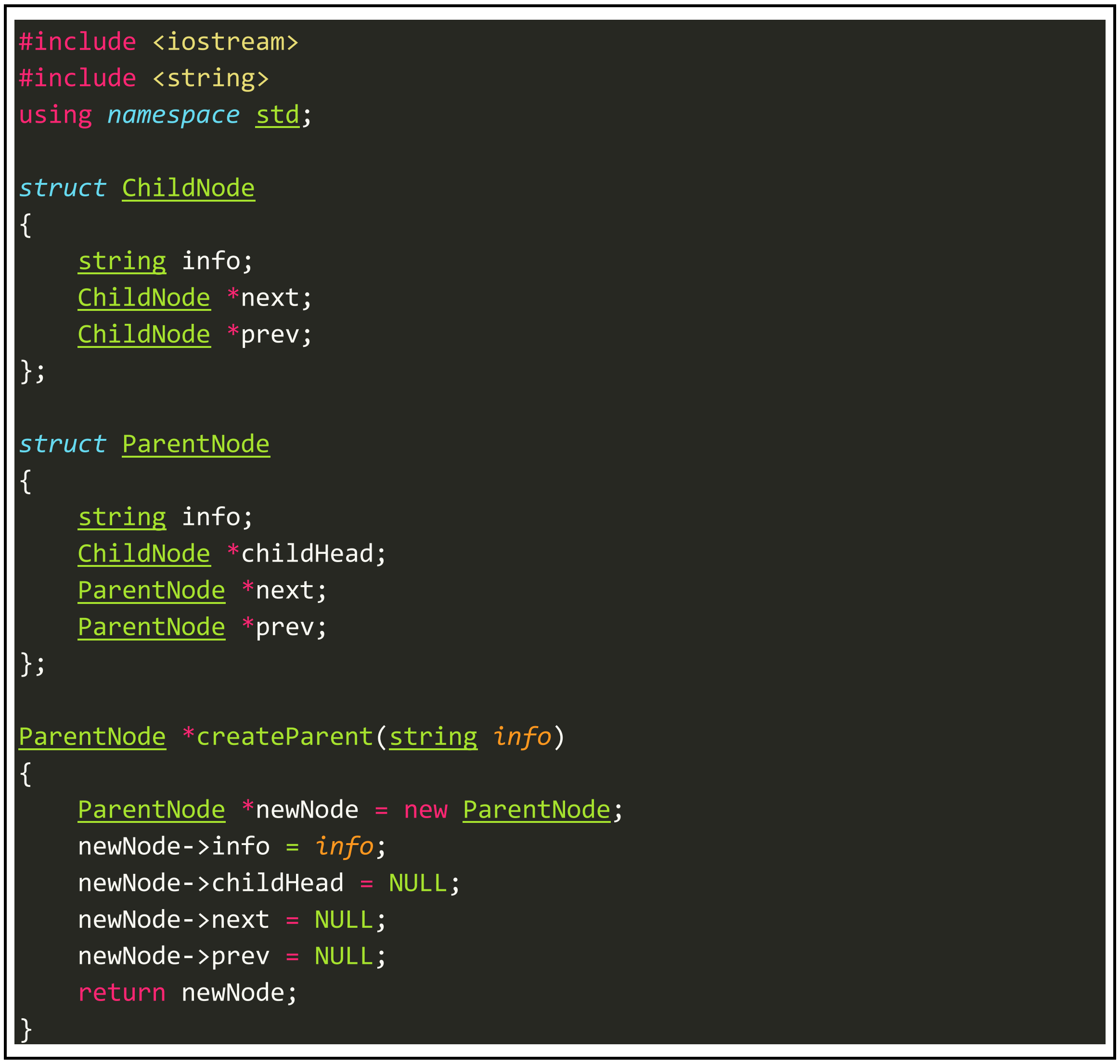
Dosen
FAHRUDIN MUKTI WIBOWO
STRUKTUR DATA

PROGRAM STUDI S1 TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

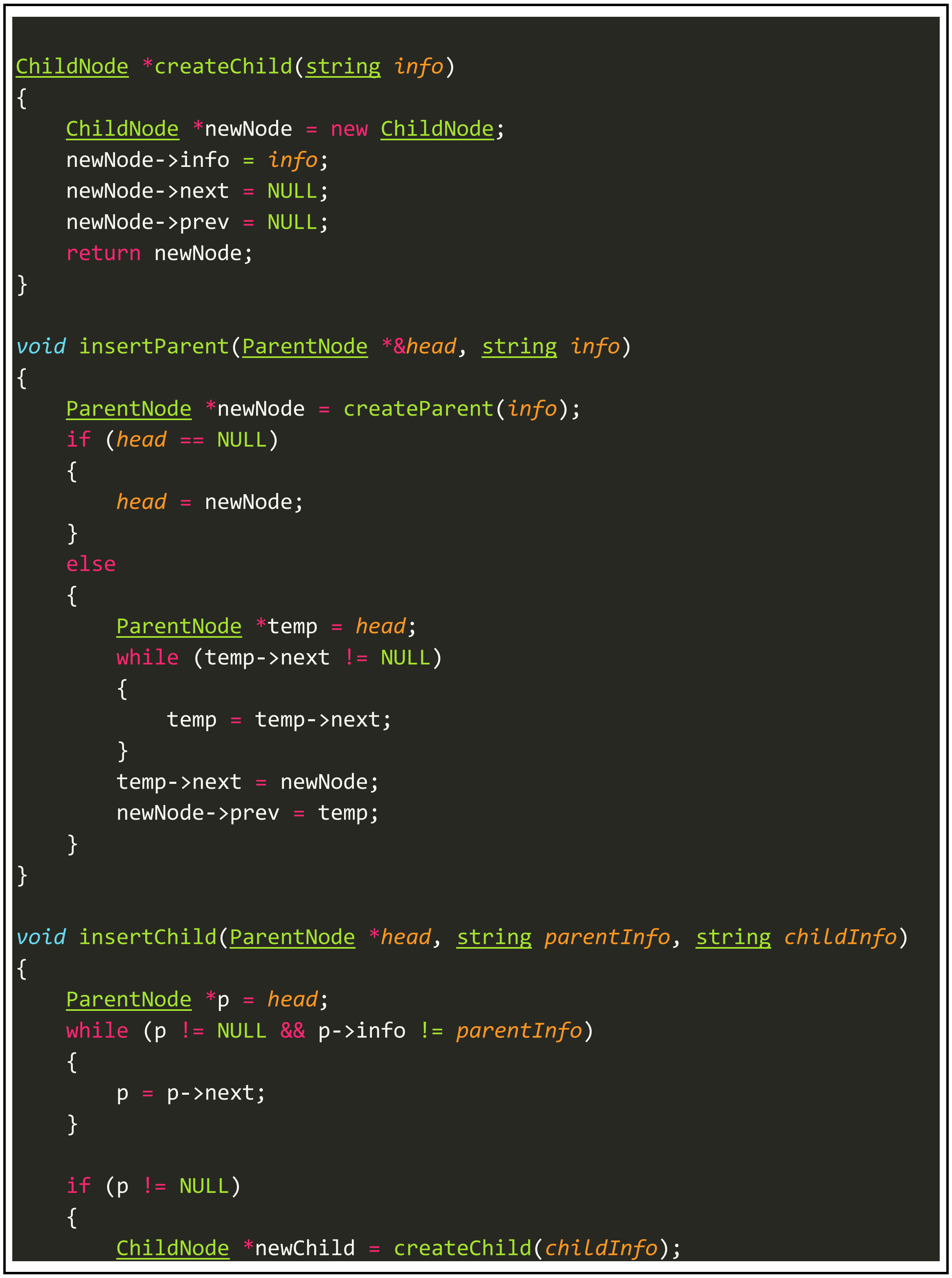
A. Dasar Teori

Circular linked list itu dasarnya sama seperti linked list biasa, tapi node terakhirnya tidak berhenti di NULL. Sebaliknya, pointer terakhir selalu kembali ke node pertama sehingga bentuk strukturnya melingkar. Karena tidak punya ujung, proses jalan dari satu node ke node berikutnya harus dihentikan saat pointer kembali ke awal lagi. Ini bikin operasi seperti traversing, insert, dan delete harus lebih hati-hati agar lingkarannya tidak rusak.

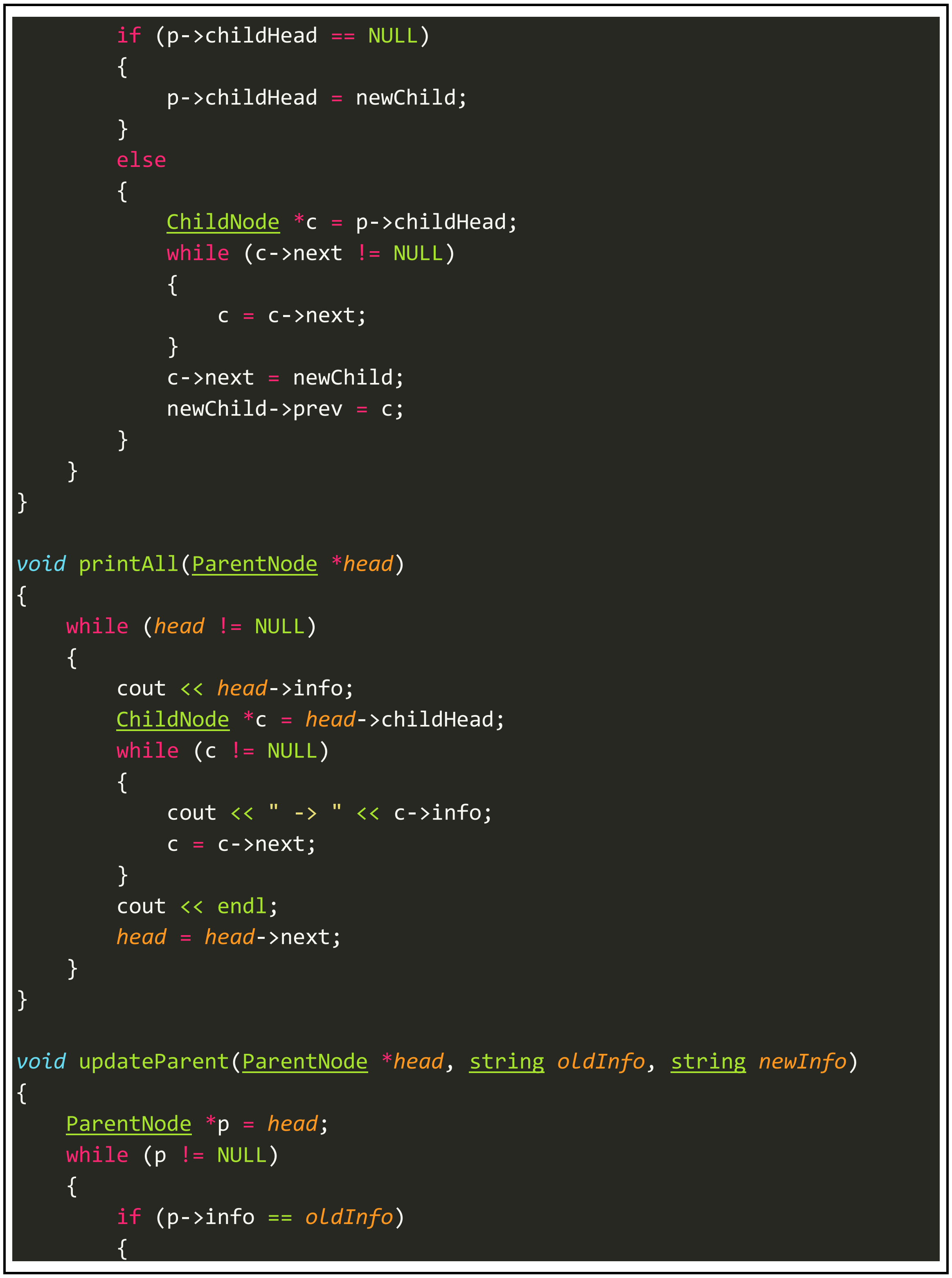
B. Guided (berisi screenshot source code & output program disertai penjelasannya)



```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;
}
```

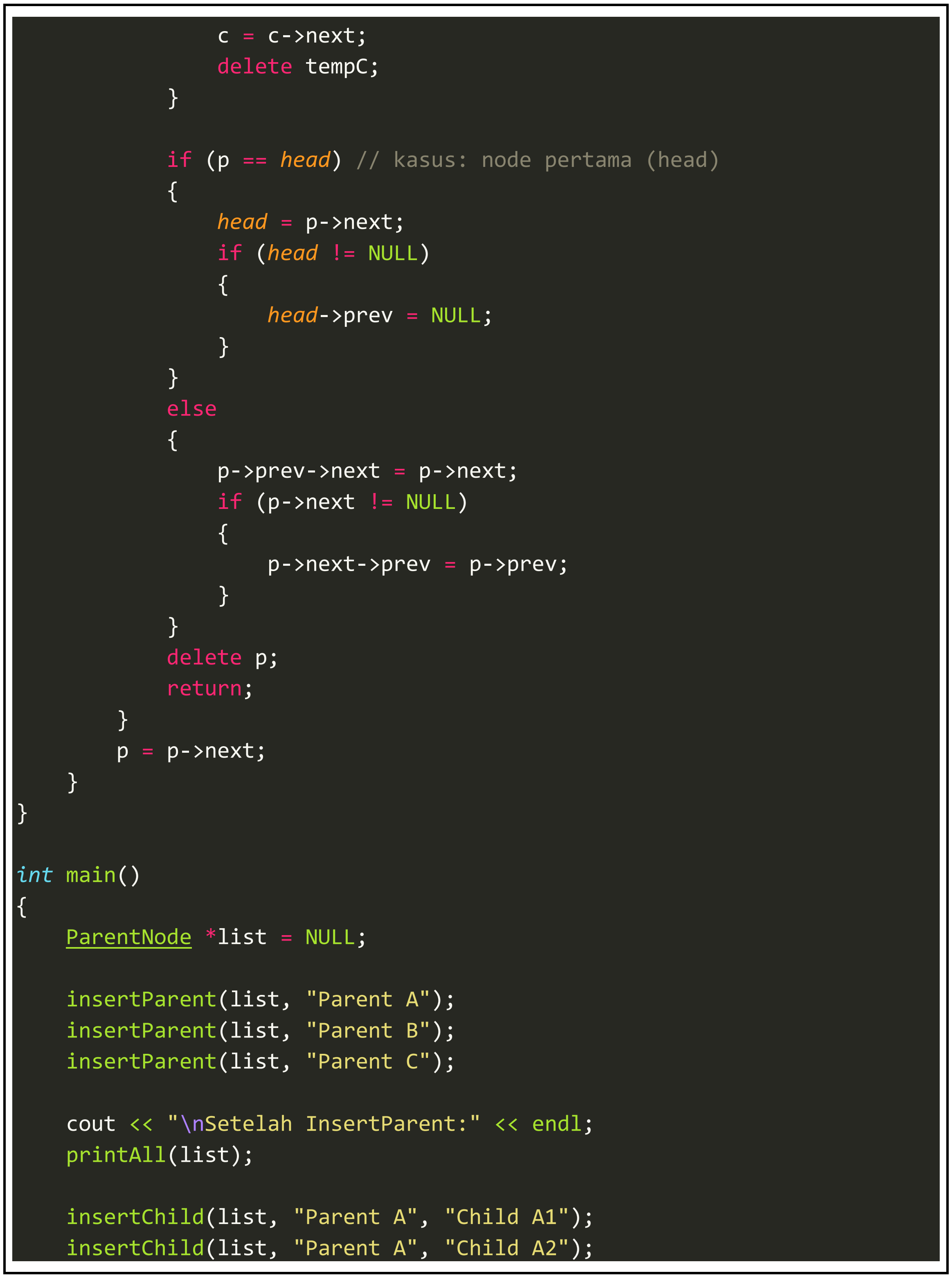


```
ChildNode *createChild(string info)
{
    ChildNode *newNode = new ChildNode;
    newNode->info = info;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}

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);
```



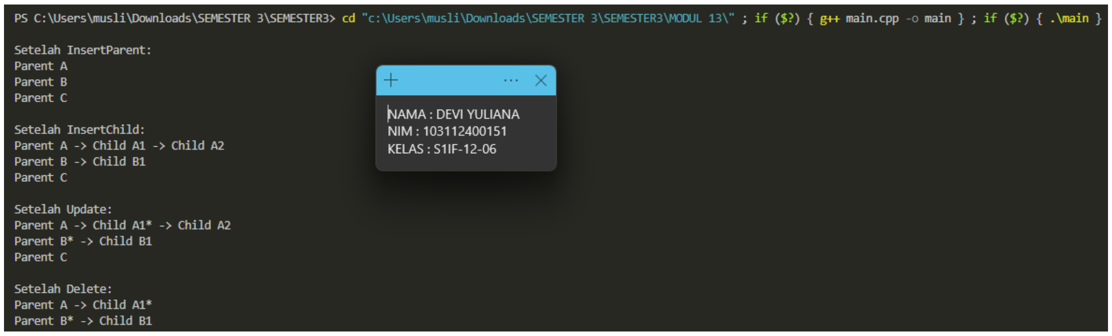
```
        if (p->childHead == NULL)
        {
            p->childHead = newChild;
        }
        else
        {
            ChildNode *c = p->childHead;
            while (c->next != NULL)
            {
                c = c->next;
            }
            c->next = newChild;
            newChild->prev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

```cpp
        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; // hapus seluruh child
            while (c != NULL)
            {
                ChildNode *tempC = c;
```



```cpp
                c = c->next;
                delete tempC;
            }

            if (p == head) // kasus: node pertama (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 C:\Users\musli\Downloads\SEMESTER 3\SEMESTER3> cd "c:\Users\musli\Downloads\SEMESTER 3\SEMESTER3\MODUL 13\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }

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
```

NAMA : DEVI YULIANA
NIM : 103112400151
KELAS : S1IF-12-06

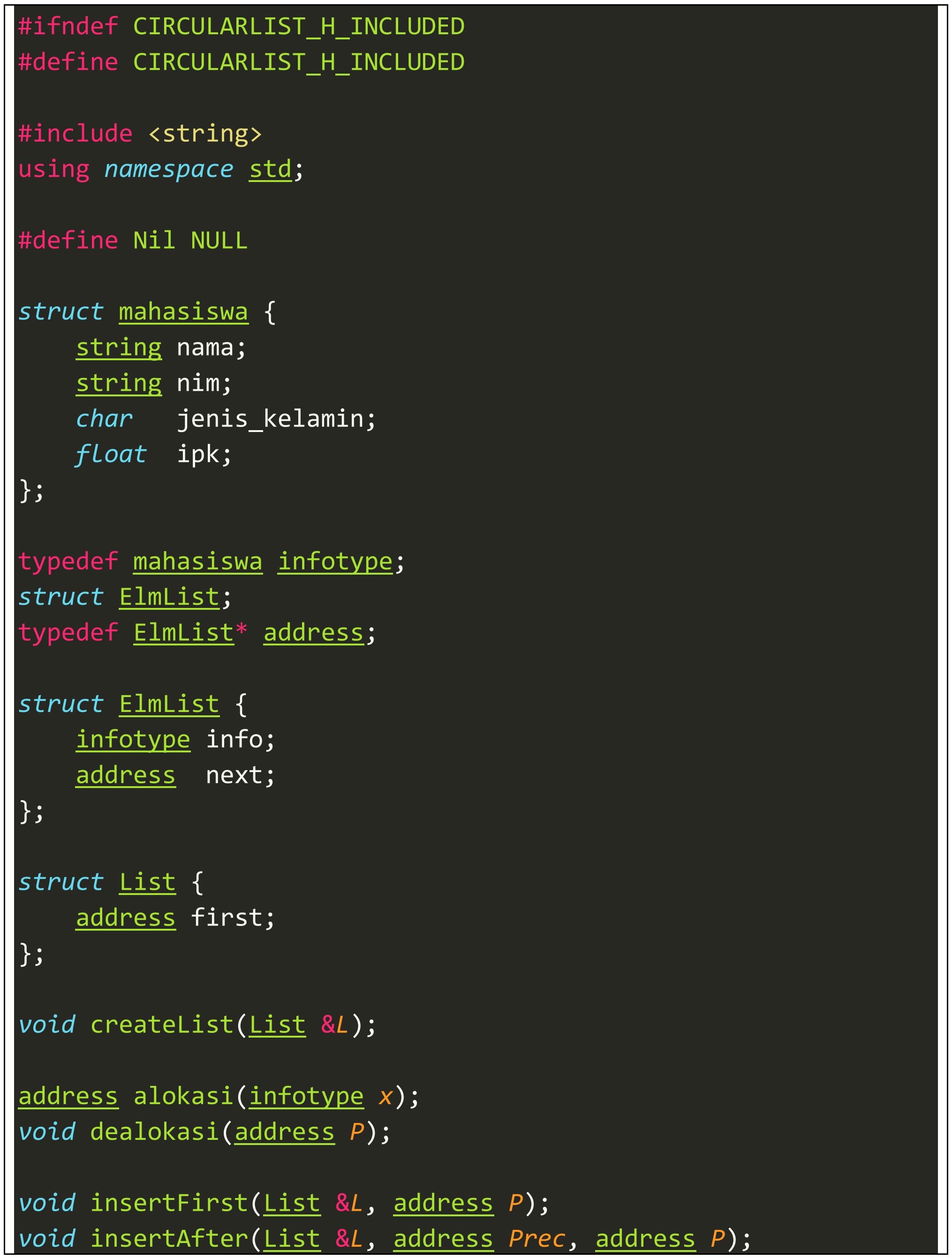
Deskripsi :

Program ini membuat dan mengelola struktur parent–child menggunakan linked list ganda. Setiap parent bisa memiliki beberapa child, dan program menyediakan operasi dasar seperti menambah parent, menambah child, mengubah data, menampilkan seluruh isi, serta menghapus parent atau child tertentu. Di fungsi main, program membentuk beberapa parent, menambahkan child ke parent tertentu, melakukan update, lalu menghapus sebagian data. Setelah setiap operasi, program menampilkan hasilnya untuk menunjukkan bagaimana struktur list berubah secara dinamis.

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

Unguided 1

circularlist.h



```
#ifndef CIRCULARLIST_H_INCLUDED
#define CIRCULARLIST_H_INCLUDED

#include <string>
using namespace std;

#define Nil NULL

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

typedef mahasiswa infotype;
struct ElmList;
typedef 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);   // cari berdasarkan
x.nim
void printInfo(List L);

#endif
```

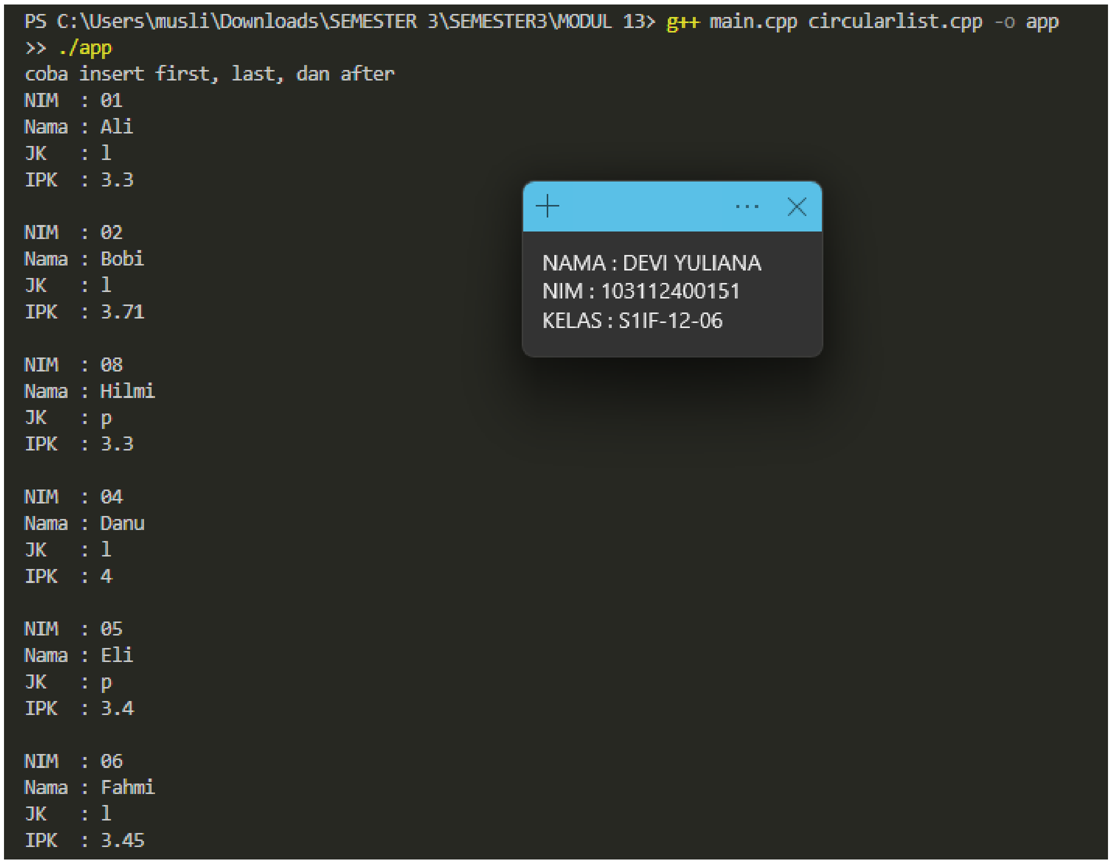
circularlist.cpp

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

static address getLast(List &L) {
    if (L.first == Nil) return Nil;
    address p = L.first;
    while (p->next != L.first) {
        p = p->next;
    }
    return p;
}

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

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

void dealokasi(address P) {
    delete P;
}

void insertFirst(List &L, address P) {
    if (P == Nil) return;

    if (L.first == Nil) {
        // list kosong
        L.first = P;
        P->next = P;      // circular
    } else {
        address last = getLast(L);
        P->next = L.first;
        last->next = P;
        L.first = P;
    }
}

void insertAfter(List &L, address Prec, address P) {
    if (L.first == Nil || Prec == Nil || P == Nil) return;

    P->next = Prec->next;
    Prec->next = P;
}

void insertLast(List &L, address P) {
    if (P == Nil) return;

    if (L.first == Nil) {
        L.first = P;
        P->next = P;
    } else {
        address last = getLast(L);
        last->next = P;
        P->next = L.first;
    }
}

void deleteFirst(List &L, address &P) {
    P = Nil;
    if (L.first == Nil) return;
    if (L.first->next == L.first) {
        // hanya satu elemen
        P = L.first;
        L.first = Nil;
    } else {
        address last = getLast(L);
        P = L.first;
        L.first = P->next;
        last->next = L.first;
    }
    P->next = Nil;
}

void deleteLast(List &L, address &P) {
    P = Nil;
    if (L.first == Nil) return;

    if (L.first->next == L.first) {
        // satu elemen
        P = L.first;
        L.first = Nil;
        P->next = Nil;
        return;
    }

    address prev = Nil;
    address curr = L.first;
    while (curr->next != L.first) {
        prev = curr;
        curr = curr->next;
    }
    // curr = last, prev = before last
    P = curr;
    prev->next = L.first;
    P->next = Nil;
}

void deleteAfter(List &L, address Prec, address &P) {
    P = Nil;
    if (L.first == Nil || Prec == Nil || Prec->next == Nil)
return;

    P = Prec->next;

    // jika yang dihapus adalah first
    if (P == L.first) {
        address last = getLast(L);
        L.first = P->next;
        last->next = L.first;
    }

    Prec->next = P->next;
    P->next = Nil;

    // kalau setelah ini list cuma satu elemen, tetap circular
    if (L.first != Nil && L.first->next == Nil)
        L.first->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) return;

    address p = L.first;
    do {
        cout << "NIM  : " << p->info.nim  << endl;
        cout << "Nama : " << p->info.nama << endl;
        cout << "JK   : " << p->info.jenis_kelamin << endl;
        cout << "IPK  : " << p->info.ipk << endl;
        cout << endl;
        p = p->next;
    } while (p != L.first);
}
```

main.cpp

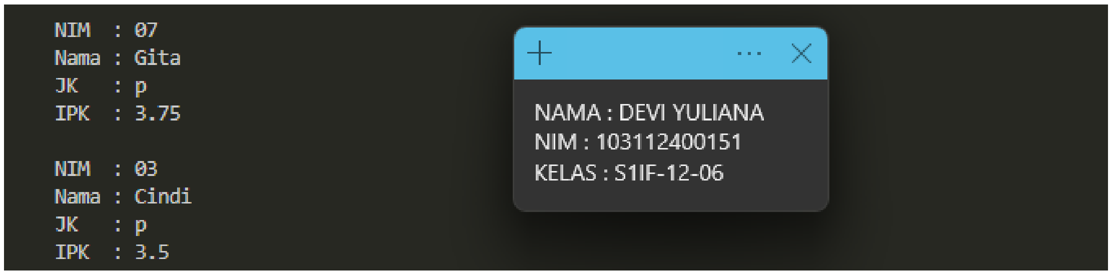
```
#include <iostream>
#include "circularlist.h"
using namespace std;

address createData(string nama, string nim, char jenis_kelamin,
float ipk)
{
    infotype x;
    address  P;

    x.nama          = nama;
    x.nim           = nim;
    x.jenis_kelamin = jenis_kelamin;
    x.ipk           = ipk;

    P = alokasi(x);
    return P;
}

int main()
{
    List L, A, B, L2;      // A, B, L2 tidak dipakai di contoh,
biarkan saja
    address P1 = Nil;
    address P2 = Nil;
    infotype x;

    createList(L);

    cout << "coba insert first, last, dan after" << endl;

    P1 = createData("Danu", "04", 'l', 4.0);
    insertFirst(L, P1);

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

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

    P1 = createData("Ali", "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("Cindi", "03", 'p', 3.5);
    insertAfter(L, P1, P2);

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

    x.nim = "04";
    P1 = findElm(L, x);
    P2 = createData("Eli", "05", 'p', 3.4);
    insertAfter(L, P1, P2);

    printInfo(L);

    return 0;
}
```

Screenshots Output



```
PS C:\Users\musli\Downloads\SEMESTER 3\SEMESTER3\MODUL 13> g++ main.cpp circularlist.cpp -o app
>> ./app
coba insert first, last, dan after
NIM  : 01
Nama : Ali
JK   : l
IPK  : 3.3

NIM  : 02
Nama : Bobi
JK   : l
IPK  : 3.71

NIM  : 08
Nama : Hilmi
JK   : p
IPK  : 3.3

NIM  : 04
Nama : Danu
JK   : l
IPK  : 4

NIM  : 05
Nama : Eli
JK   : p
IPK  : 3.4

NIM  : 06
Nama : Fahmi
JK   : l
IPK  : 3.45
```

NAMA : DEVI YULIANA
NIM : 103112400151
KELAS : S1IF-12-06



```
NIM  : 07
Nama : Gita
JK   : p
IPK  : 3.75

NIM  : 03
Nama : Cindi
JK   : p
IPK  : 3.5
```

NAMA : DEVI YULIANA
NIM : 103112400151
KELAS : S1IF-12-06

Deskripsi :

Program ini membangun dan mengelola **circular linked list** yang berisi data mahasiswa. Struktur list dibuat melingkar sehingga elemen terakhir selalu menunjuk kembali ke elemen pertama. Program menyediakan operasi dasar seperti membuat list, menambah data di awal/akhir/tengah, menghapus data, mencari elemen berdasarkan NIM, dan menampilkan seluruh isi list. Melalui fungsi-fungsi ini, program menunjukkan bagaimana circular linked list bekerja sebagai struktur data dinamis yang bisa bertambah, berkurang, dan ditelusuri tanpa henti karena bentuknya yang melingkar.

D. Kesimpulan

Circular linked list memberikan cara penyimpanan data yang fleksibel tanpa batas akhir, sedangkan multi linked list membuat setiap elemen lebih kaya informasi. Jika semua operasi dasar (insert, delete, find, print) dijalankan sesuai aturan, list akan stabil dan mudah digunakan untuk berbagai kebutuhan pemrosesan data.

E. Referensi

Anita Sindar, R. M. S. (2019). *Struktur Data Dan Algoritma Dengan C++ (Vol. 1). CV. AA. RIZKY.*