LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL XIII
MULTI LINKED LIST

Disusun Oleh :
NAMA : Azzahra Farelika Esti Ning Tyas
NIM : 103112430023

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Multi linked list merupakan struktur data pengembangan dari linked list yang memungkinkan satu node memiliki lebih dari satu pointer untuk menghubungkan ke node lain. Struktur ini digunakan untuk merepresentasikan hubungan data yang kompleks seperti hubungan satu ke banyak atau banyak ke banyak. Dalam bahasa pemrograman C++, multi linked list bersifat dinamis karena menggunakan pointer sehingga tidak memerlukan alokasi memori secara berurutan. Struktur ini sering diterapkan pada sistem yang memiliki keterkaitan data, seperti sistem akademik, struktur organisasi, dan basis data sederhana, meskipun implementasinya lebih kompleks dibandingkan linked list biasa.

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

Guided 1

```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 *temp = c;
                c = c->next;
                delete temp;
            }

            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:\n";
    printAll(list);

    insertChild(list, "Parent A", "Child A1");
    insertChild(list, "Parent B", "Child B1");
    insertChild(list, "Parent B", "Child B2");

    cout << "\nSetelah insertChild:\n";
    printAll(list);

    updateParent(list, "Parent B", "Parent B*");
    updateChild(list, "Parent B", "Child B1", "Child B1*");

    cout << "\nSetelah update:\n";
    printAll(list);

    deleteChild(list, "Parent A", "Child A1");
    deleteParent(list, "Parent C");

    cout << "\nSetelah delete:\n";
    printAll(list);

    return 0;
}
```

Screenshots Output

```
PS D:\StrukturData> cd "d:\StrukturData\Modul13\GUIDED\" ; if ($?) { g++ multilist.cpp -o multilist } ; if ($?) { .\multilist }

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
```

Deskripsi:

Kode program tersebut mengimplementasikan struktur data multi linked list parentchild menggunakan bahasa C++. Setiap parent disusun dalam double linked list dan memiliki pointer ke linked list child yang juga bertipe double linked list. Program ini menyediakan operasi untuk menambah, menampilkan, memperbarui, dan menghapus data parent maupun child, di mana penghapusan parent juga menghapus seluruh child yang terkait. Pada fungsi main, seluruh operasi tersebut diuji untuk menunjukkan cara kerja multi linked list secara sederhana dan terstruktur.

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

Unguided 1

multilist.h

Modul13 > UNGUIDED > multilist.h > ...

```h
#ifndef MULTILIST_H_INCLUDED
#define MULTILIST_H_INCLUDED
#define Nil NULL

typedef int infotypeanak;
typedef int infotypeinduk;
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;
};

Tabnine | Edit | Test | Explain | Document
bool ListEmpty(listinduk L);
Tabnine | Edit | Test | Explain | Document
bool ListEmptyAnak(listanak L);

Tabnine | Edit | Test | Explain | Document
void CreateList(listinduk &L);
Tabnine | Edit | Test | Explain | Document
void CreateListAnak(listanak &L);

Tabnine | Edit | Test | Explain | Document
address alokasi(infotypeinduk P);
Tabnine | Edit | Test | Explain | Document
address_anak alokasiAnak(infotypeanak P);
Tabnine | Edit | Test | Explain | Document
void dealokasi(address P);
Tabnine | Edit | Test | Explain | Document
void dealokasiAnak(address_anak P);

Tabnine | Edit | Test | Explain | Document
address findElm(listinduk L, infotypeinduk X);
Tabnine | Edit | Test | Explain | Document
address_anak findElmAnak(listanak Lanak, infotypeanak X);
Tabnine | Edit | Test | Explain | Document
bool fFindElm(listinduk L, address P);
Tabnine | Edit | Test | Explain | Document
bool fFindElmanak(listanak Lanak, address_anak P);
Tabnine | Edit | Test | Explain | Document
address findBefore(listinduk L, address P);
Tabnine | Edit | Test | Explain | Document
address_anak findBeforeAnak(listanak Lanak, address_anak P);

Tabnine | Edit | Test | Explain | Document
void insertFirst(listinduk &L, address P);
Tabnine | Edit | Test | Explain | Document
void insertAfter(listinduk &L, address P, address Prec);
Tabnine | Edit | Test | Explain | Document
void insertLast(listinduk &L, address P);
Tabnine | Edit | Test | Explain | Document
void insertFirstAnak(listanak &L, address_anak P);
Tabnine | Edit | Test | Explain | Document
void insertAfterAnak(listanak &L, address_anak P, address_anak Prec);
Tabnine | Edit | Test | Explain | Document
void insertLastAnak(listanak &L, address_anak P);
Tabnine | Edit | Test | Explain | Document
void delFirst(listinduk &L, address &P);
Tabnine | Edit | Test | Explain | Document
void delLast(listinduk &L, address &P);
Tabnine | Edit | Test | Explain | Document
void delAfter(listinduk &L, address &P, address Prec);
Tabnine | Edit | Test | Explain | Document
void delP(listinduk &L, infotypeinduk X);
Tabnine | Edit | Test | Explain | Document
void delFirstAnak(listanak &L, address_anak &P);
Tabnine | Edit | Test | Explain | Document
void delLastAnak(listanak &L, address_anak &P);
Tabnine | Edit | Test | Explain | Document
void delAfterAnak(listanak &L, address_anak &P, address_anak Prec);
Tabnine | Edit | Test | Explain | Document
void delPAnak(listanak &L, infotypeanak X);
Tabnine | Edit | Test | Explain | Document
void printInfo(listinduk L);
Tabnine | Edit | Test | Explain | Document
int nbList(listinduk L);
Tabnine | Edit | Test | Explain | Document
void printInfoAnak(listanak Lanak);
Tabnine | Edit | Test | Explain | Document
int nbListAnak(listanak Lanak);
Tabnine | Edit | Test | Explain | Document
void delAll(listinduk &L);
Tabnine | Edit | Test | Explain | Document
void delAllAnak(listanak &L);

#endif
```

multilist.cpp

Modul13 > UNGUIDED > multilist.cpp > alokasi(infotypeinduk)

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

Tabnine | Edit | Test | Explain | Document
bool ListEmpty(listinduk L) {
    return L.first == Nil;
}

Tabnine | Edit | Test | Explain | Document
bool ListEmptyAnak(listanak L) {
    return L.first == Nil;
}

Tabnine | Edit | Test | Explain | Document
void CreateList(listinduk &L) {
    L.first = Nil;
    L.last = Nil;
}

Tabnine | Edit | Test | Explain | Document
void CreateListAnak(listanak &L) {
    L.first = Nil;
    L.last = Nil;
}

Tabnine | Edit | Test | Explain | Document
address alokasi(infotypeinduk X) {
    address P = new elemen_list_induk;
    if (P != Nil) {
        P->info = X;
        P->next = Nil;
        P->prev = Nil;
        CreateListAnak(P->lanak);
    }
    return P;
}

Tabnine | Edit | Test | Explain | Document
address_anak alokasiAnak(infotypeanak X) {
    address_anak P = new elemen_list_anak;
    if (P != Nil) {
        P->info = X;
        P->next = Nil;
        P->prev = Nil;
    }
    return P;
}

Tabnine | Edit | Test | Explain | Document
void dealokasi(address P) {
    delete P;
}

Tabnine | Edit | Test | Explain | Document
void dealokasiAnak(address_anak P) {
    delete P;
}

Tabnine | Edit | Test | Explain | Document
address findElm(listinduk L, infotypeinduk X) {
    address P = L.first;
    while (P != Nil) {
        if (P->info == X) {
            return P;
        }
        P = P->next;
    }
    return Nil;
}

Tabnine | Edit | Test | Explain | Document
address_anak findElmAnak(listanak Lanak, infotypeanak X) {
    address_anak P = Lanak.first;
    while (P != Nil) {
        if (P->info == X) {
            return P;
        }
        P = P->next;
    }
    return Nil;
}

Tabnine | Edit | Test | Explain | Document
bool fFindElm(listinduk L, address P) {
    address Q = L.first;
    while (Q != Nil) {
        if (Q == P) {
            return true;
        }
        Q = Q->next;
    }
    return false;
}

Tabnine | Edit | Test | Explain | Document
bool fFindElmanak(listanak Lanak, address_anak P) {
    address_anak Q = Lanak.first;
    while (Q != Nil) {
        if (Q == P) {
            return true;
        }
        Q = Q->next;
    }
    return false;
}

Tabnine | Edit | Test | Explain | Document
address findBefore(listinduk L, address P) {
```

multilist.cpp

Modul13 > UNGUIDED > multilist.cpp > ...

```cpp

Tabnine | Edit | Test | Explain | Document
address findBefore(listinduk L, address P) {
    address Q = L.first;
    if (Q == P) {
        return Nil;
    }
    while (Q != Nil && Q->next != P) {
        Q = Q->next;
    }
    return Q;
}

Tabnine | Edit | Test | Explain | Document
address_anak findBeforeAnak(listanak Lanak, address_anak P) {
    address_anak Q = Lanak.first;
    if (Q == P) {
        return Nil;
    }
    while (Q != Nil && Q->next != P) {
        Q = Q->next;
    }
    return Q;
}

Tabnine | Edit | Test | Explain | Document
void insertFirst(listinduk &L, address P) {
    if (ListEmpty(L)) {
        L.first = P;
        L.last = P;
        P->next = Nil;
        P->prev = Nil;
    } else {
        P->next = L.first;
        P->prev = Nil;
        L.first->prev = P;
        L.first = P;
    }
}

Tabnine | Edit | Test | Explain | Document
void insertAfter(listinduk &L, address P, address Prec) {
    if (Prec != Nil) {
        P->next = Prec->next;
        P->prev = Prec;
        if (Prec->next != Nil) {
            Prec->next->prev = P;
        } else {
            L.last = P;
        }
        Prec->next = P;
    }
}

Tabnine | Edit | Test | Explain | Document
void insertLast(listinduk &L, address P) {
    if (ListEmpty(L)) {
        L.first = P;
        L.last = P;
        P->next = Nil;
        P->prev = Nil;
    } else {
        P->prev = L.last;
        P->next = Nil;
        L.last->next = P;
        L.last = P;
    }
}

Tabnine | Edit | Test | Explain | Document
void insertFirstAnak(listanak &L, address_anak P) {
    if (ListEmptyAnak(L)) {
        L.first = P;
        L.last = P;
        P->next = Nil;
        P->prev = Nil;
    } else {
        P->next = L.first;
        P->prev = Nil;
        L.first->prev = P;
        L.first = P;
    }
}

Tabnine | Edit | Test | Explain | Document
void insertAfterAnak(listanak &L, address_anak P, address_anak Prec) {
    if (Prec != Nil) {
        P->next = Prec->next;
        P->prev = Prec;
        if (Prec->next != Nil) {
            Prec->next->prev = P;
        } else {
            L.last = P;
        }
        Prec->next = P;
    }
}

Tabnine | Edit | Test | Explain | Document
void insertLastAnak(listanak &L, address_anak P) {
```

multilist.cpp

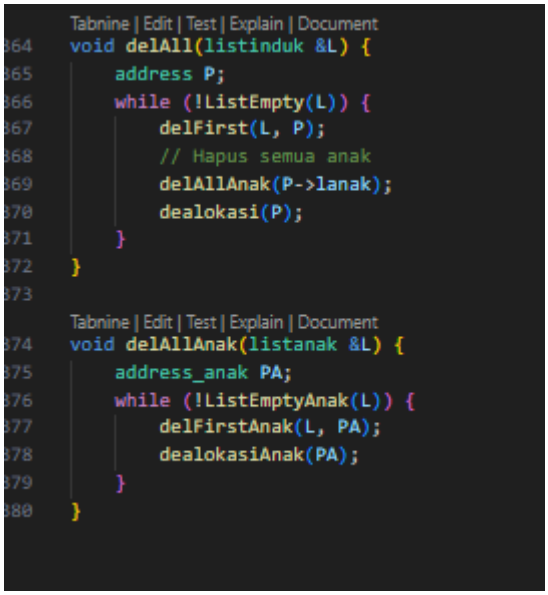
Modul13 > UNGUIDED > multilist.cpp > insertAfter(listinduk &, address, address)

```cpp
void insertAfterAnak(listanak &L, address_anak P, address_anak Prec) {
    if (Prec != Nil) {
    }
}

Tabnine | Edit | Test | Explain | Document
void insertLastAnak(listanak &L, address_anak P) {
    if (ListEmptyAnak(L)) {
        L.first = P;
        L.last = P;
        P->next = Nil;
        P->prev = Nil;
    } else {
        P->prev = L.last;
        P->next = Nil;
        L.last->next = P;
        L.last = P;
    }
}

Tabnine | Edit | Test | Explain | Document
void delFirst(listinduk &L, address &P) {
    if (!ListEmpty(L)) {
        P = L.first;
        if (L.first == L.last) {
            L.first = Nil;
            L.last = Nil;
        } else {
            L.first = L.first->next;
            L.first->prev = Nil;
            P->next = Nil;
        }
    }
}

Tabnine | Edit | Test | Explain | Document
void delLast(listinduk &L, address &P) {
    if (!ListEmpty(L)) {
        P = L.last;
        if (L.first == L.last) {
            L.first = Nil;
            L.last = Nil;
        } else {
            L.last = L.last->prev;
            L.last->next = Nil;
            P->prev = Nil;
        }
    }
}

Tabnine | Edit | Test | Explain | Document
void delAfter(listinduk &L, address &P, address Prec) {
    if (Prec != Nil && Prec->next != Nil) {
        P = Prec->next;
        Prec->next = P->next;
        if (P->next != Nil) {
            P->next->prev = Prec;
        } else {
            L.last = Prec;
        }
        P->next = Nil;
        P->prev = Nil;
    }
}

Tabnine | Edit | Test | Explain | Document
void delP(listinduk &L, infotypeinduk X) {
    address P = findElm(L, X);
    if (P != Nil) {

        delAllAnak(P->lanak);

        if (P == L.first) {
            delFirst(L, P);
        } else if (P == L.last) {
            delLast(L, P);
        } else {
            address Prec = P->prev;
            delAfter(L, P, Prec);
        }
        dealokasi(P);
    }
}

Tabnine | Edit | Test | Explain | Document
void delFirstAnak(listanak &L, address_anak &P) {
    if (!ListEmptyAnak(L)) {
        P = L.first;
        if (L.first == L.last) {
            L.first = Nil;
            L.last = Nil;
        } else {
            L.first = L.first->next;
            L.first->prev = Nil;
            P->next = Nil;
        }
    }
}

Tabnine | Edit | Test | Explain | Document
void delLastAnak(listanak &L, address_anak &P) {
```

multilist.cpp

Modul13 > UNGUIDED > multilist.cpp > delP(listinduk &, infotypeinduk)

```cpp
void delFirstAnak(listanak &L, address_anak &P) {
}

Tabnine | Edit | Test | Explain | Document
void delLastAnak(listanak &L, address_anak &P) {
    if (!ListEmptyAnak(L)) {
        P = L.last;
        if (L.first == L.last) {
            L.first = Nil;
            L.last = Nil;
        } else {
            L.last = L.last->prev;
            L.last->next = Nil;
            P->prev = Nil;
        }
    }
}

Tabnine | Edit | Test | Explain | Document
void delAfterAnak(listanak &L, address_anak &P, address_anak Prec) {
    if (Prec != Nil && Prec->next != Nil) {
        P = Prec->next;
        Prec->next = P->next;
        if (P->next != Nil) {
            P->next->prev = Prec;
        } else {
            L.last = Prec;
        }
        P->next = Nil;
        P->prev = Nil;
    }
}

Tabnine | Edit | Test | Explain | Document
void delPAnak(listanak &L, infotypeanak X) {
    address_anak P = findElmAnak(L, X);
    if (P != Nil) {
        if (P == L.first) {
            delFirstAnak(L, P);
        } else if (P == L.last) {
            delLastAnak(L, P);
        } else {
            address_anak Prec = P->prev;
            delAfterAnak(L, P, Prec);
        }
        dealokasiAnak(P);
    }
}

Tabnine | Edit | Test | Explain | Document
void printInfo(listinduk L) {
    address P = L.first;
    cout << "=== Data Induk dan Anak ===" << endl;
    while (P != Nil) {
        cout << "Induk: " << P->info << endl;
        cout << "  Anak: ";
        printInfoAnak(P->lanak);
        cout << endl;
        P = P->next;
    }
}

Tabnine | Edit | Test | Explain | Document
void printInfoAnak(listanak Lanak) {
    address_anak P = Lanak.first;
    if (ListEmptyAnak(Lanak)) {
        cout << "(kosong)";
    } else {
        while (P != Nil) {
            cout << P->info;
            if (P->next != Nil) {
                cout << ", ";
            }
            P = P->next;
        }
    }
}

Tabnine | Edit | Test | Explain | Document
int nbList(listinduk L) {
    int count = 0;
    address P = L.first;
    while (P != Nil) {
        count++;
        P = P->next;
    }
    return count;
}

Tabnine | Edit | Test | Explain | Document
int nbListAnak(listanak Lanak) {
    int count = 0;
    address_anak P = Lanak.first;
    while (P != Nil) {
        count++;
        P = P->next;
    }
    return count;
}

Tabnine | Edit | Test | Explain | Document
void delAll(listinduk &L) {
    address P;
    while (!ListEmpty(L)) {
        delFirst(L, P);
        // Hapus semua anak
        delAllAnak(P->lanak);
        dealokasi(P);
```



```
Tabnine | Edit | Test | Explain | Document
void delAll(listinduk &L) {
    address P;
    while (!ListEmpty(L)) {
        delFirst(L, P);
        // Hapus semua anak
        delAllAnak(P->lanak);
        dealokasi(P);
    }
}

Tabnine | Edit | Test | Explain | Document
void delAllAnak(listanak &L) {
    address_anak PA;
    while (!ListEmptyAnak(L)) {
        delFirstAnak(L, PA);
        dealokasiAnak(PA);
    }
}
```

main.cpp

Modul13 > UNGUIDED > main.cpp > main()

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

Tabnine | Edit | Test | Explain | Document
int main() {
    listinduk L;
    address P_induk;
    address_anak P_anak;

    CreateList(L);

    cout << "Menambah data Parent" << endl;
    P_induk = alokasi(1);
    insertFirst(L, P_induk);

    P_induk = alokasi(2);
    insertLast(L, P_induk);

    P_induk = alokasi(3);
    insertLast(L, P_induk);

    cout << "Menambah anak untuk parent 1" << endl;
    P_induk = findElm(L, 1);
    if (P_induk != Nil) {
        P_anak = alokasiAnak(101);
        insertFirstAnak(P_induk->lanak, P_anak);

        P_anak = alokasiAnak(102);
        insertLastAnak(P_induk->lanak, P_anak);

        P_anak = alokasiAnak(103);
        insertLastAnak(P_induk->lanak, P_anak);
    }

    cout << "Menambah anak untuk parent 2" << endl;
    P_induk = findElm(L, 2);
    if (P_induk != Nil) {
        P_anak = alokasiAnak(201);
        insertFirstAnak(P_induk->lanak, P_anak);

        P_anak = alokasiAnak(202);
        insertLastAnak(P_induk->lanak, P_anak);
    }

    cout << "Menambah anak untuk parent 3" << endl;
    P_induk = findElm(L, 3);
    if (P_induk != Nil) {
        P_anak = alokasiAnak(301);
        insertFirstAnak(P_induk->lanak, P_anak);
    }

    cout << endl;
    printInfo(L);

    cout << endl << "Menghapus anak 102 dari parent 1..." << endl;
    P_induk = findElm(L, 1);
    if (P_induk != Nil) {
        delPAnak(P_induk->lanak, 102);
    }

    printInfo(L);

    cout << endl << "Menghapus parent 2 beserta semua anaknya..." << endl;
    delP(L, 2);

    printInfo(L);

    cout << endl << "Jumlah parent: " << nbList(L) << endl;

    delAll(L);

    return 0;
}
```

Screenshots Output

```
PS D:\StrukturData> cd "d:\StrukturData\Modul13\UNGUIDED\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
Menambah anak untuk parent 1
Menambah anak untuk parent 2
Menambah anak untuk parent 3

=== Data Induk dan Anak ===
Induk: 1
  Anak: 101, 102, 103
Induk: 2
  Anak: 201, 202
Induk: 3
  Anak: 301

Menghapus anak 102 dari parent 1...
=== Data Induk dan Anak ===
Induk: 1
  Anak: 101, 103
Induk: 2
  Anak: 201, 202
Induk: 3
  Anak: 301

Menghapus parent 2 beserta semua anaknya...
=== Data Induk dan Anak ===
Induk: 1
  Anak: 101, 103
Induk: 3
  Anak: 301

Jumlah parent: 2
```

Deskripsi:

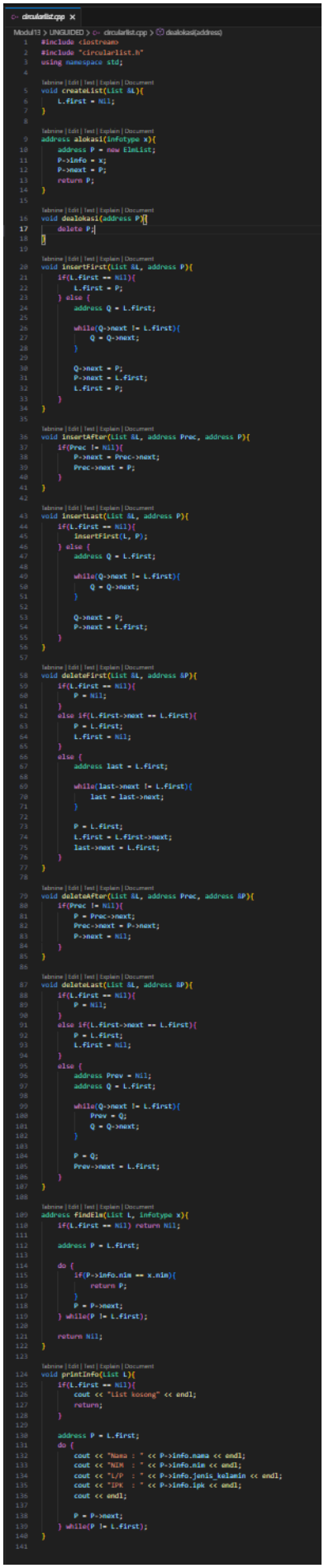
Kode ADT tersebut mengimplementasikan circular singly linked list menggunakan bahasa C++. File circularlist.h mendefinisikan struktur data berupa infotype untuk menyimpan data mahasiswa, node ElmList, serta struktur List yang memiliki pointer first. File circularlist.cpp berisi implementasi operasi dasar circular linked list seperti inisialisasi list, alokasi dan dealokasi node, penyisipan data di awal, tengah, dan akhir list, penghapusan node, pencarian data berdasarkan NIM, serta penampilan seluruh isi list secara melingkar. Pada mainn.cpp, program digunakan untuk menguji ADT dengan membuat data mahasiswa, melakukan berbagai operasi insert dan pencarian, lalu menampilkan hasilnya untuk menunjukkan bahwa list bersifat melingkar, yaitu node terakhir selalu menunjuk kembali ke node pertama.

Unguided 2

circularlist.h

Modul13 > UNGUIDED > circularlist.h > ...

```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;
};

Tabnine | Edit | Test | Explain | Document
void createList(List &L);
Tabnine | Edit | Test | Explain | Document
address alokasi(infotype x);
Tabnine | Edit | Test | Explain | Document
void dealokasi(address P);
Tabnine | Edit | Test | Explain | Document
void insertFirst(List &L, address P);
Tabnine | Edit | Test | Explain | Document
void insertAfter(List &L, address Prec, address P);
Tabnine | Edit | Test | Explain | Document
void insertLast(List &L, address P);
Tabnine | Edit | Test | Explain | Document
void deleteFirst(List &L, address &P);
Tabnine | Edit | Test | Explain | Document
void deleteAfter(List &L, address Prec, address &P);
Tabnine | Edit | Test | Explain | Document
void deleteLast(List &L, address &P);
Tabnine | Edit | Test | Explain | Document
address findElm(List L, infotype x);
Tabnine | Edit | Test | Explain | Document
void printInfo(List L);

#endif
```



circularlist.cpp

Modul13 > UNGUIDED > circularlist.cpp > dealokasi(address)

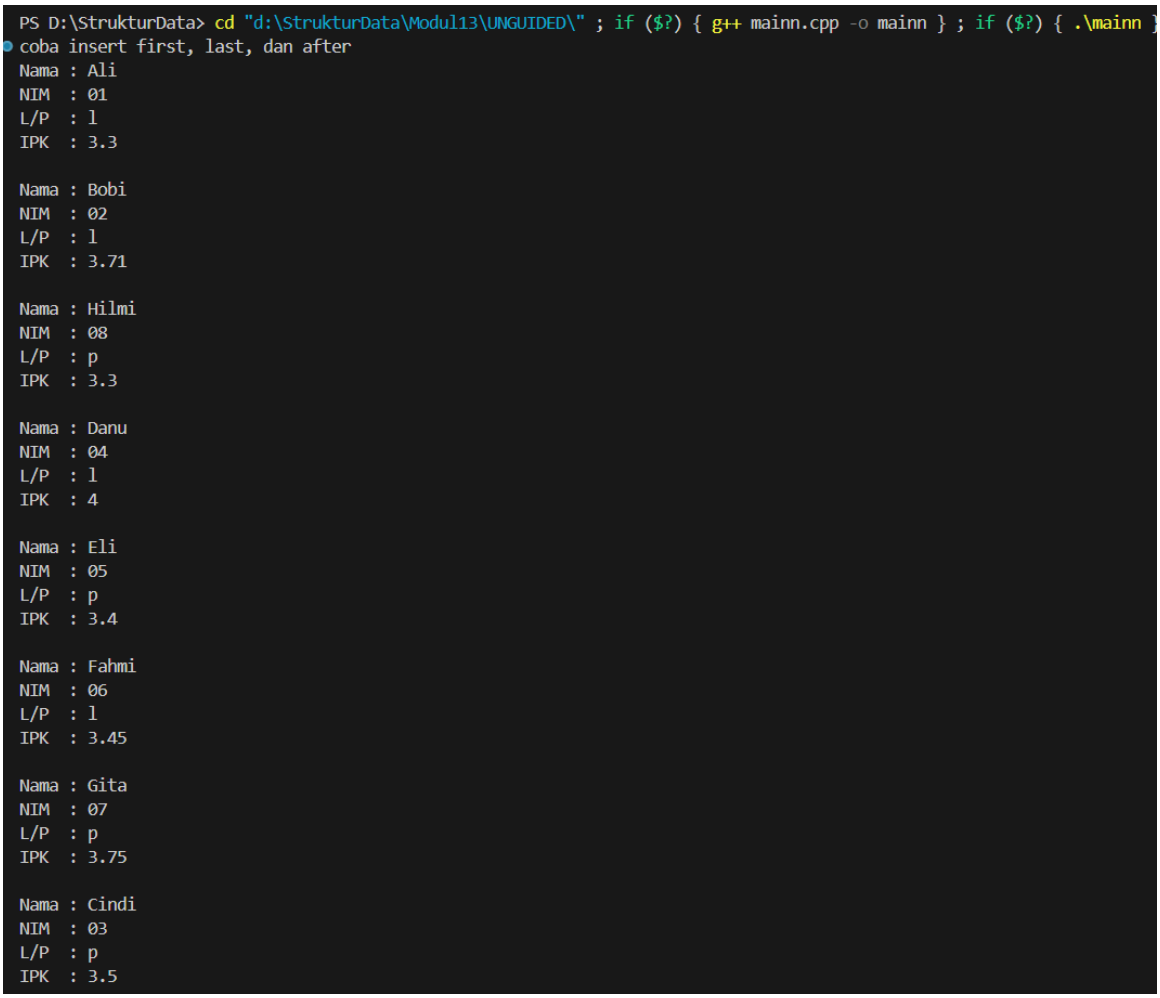
```cpp
#include <iostream>
#include "circularlist.h"
using namespace std;

labmine | Edit | Test | Explain | Document
void createList(List &L){
    L.first = Nil;
}

labmine | Edit | Test | Explain | Document
address alokasi(infotype x){
    address P = new ElmList;
    P->info = x;
    P->next = P;
    return P;
}

labmine | Edit | Test | Explain | Document
void dealokasi(address P){
    delete P;
}

labmine | Edit | Test | Explain | Document
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;
    }
}

labmine | Edit | Test | Explain | Document
void insertAfter(List &L, address Prec, address P){
    if(Prec != Nil){
        P->next = Prec->next;
        Prec->next = P;
    }
}

labmine | Edit | Test | Explain | Document
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;
    }
}

labmine | Edit | Test | Explain | Document
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;
    }
}

labmine | Edit | Test | Explain | Document
void deleteAfter(List &L, address Prec, address &P){
    if(Prec != Nil){
        P = Prec->next;
        Prec->next = P->next;
        P->next = Nil;
    }
}

labmine | Edit | Test | Explain | Document
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;
    }
}

labmine | Edit | Test | Explain | Document
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;
}

labmine | Edit | Test | Explain | Document
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);
}
```

mainn.cpp

Modul13 > UNGUIDED > mainn.cpp > main()

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

Tabnine | Edit | Test | Explain | Document
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);
}

Tabnine | Edit | Test | Explain | Document
int main()
{
    List L;
    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



Deskripsi:

Kode ADT tersebut mengimplementasikan struktur data multi linked list bertipe parent child menggunakan bahasa C++ dengan konsep double linked list pada data induk maupun data anak. File multilist.h mendefinisikan struktur node induk yang memiliki pointer ke list anak, sedangkan multilist.cpp berisi implementasi operasi dasar seperti pembuatan list, alokasi dan dealokasi memori, pencarian, penyisipan, penghapusan parent dan anak, perhitungan jumlah elemen, serta penampilan seluruh data. Pada file main.cpp, ADT diuji dengan menambahkan beberapa parent dan anak, menampilkan hubungan parent–child, menghapus anak maupun parent beserta seluruh anaknya, sehingga menunjukkan bahwa struktur multi linked list bekerja secara terorganisir dan aman dalam pengelolaan data relasional.

D. Kesimpulan

Dari hasil pembuatan dan pengujian program, dapat disimpulkan bahwa struktur data linked list, baik circular linked list maupun multi linked list, dapat digunakan untuk mengelola data secara dinamis menggunakan bahasa C++. Multi linked list cocok untuk menyimpan data yang memiliki hubungan parent dan child karena setiap data induk dapat memiliki beberapa data anak. Sementara itu, circular linked list memungkinkan data saling terhubung secara melingkar sehingga dapat ditelusuri terus menerus tanpa batas akhir.

Penerapan ADT dengan pemisahan file header, source, dan main membuat program lebih rapi dan mudah dipahami. Operasi dasar seperti insert, delete, search, dan print dapat berjalan dengan baik sesuai dengan konsep struktur data yang digunakan. Dengan demikian, penggunaan linked list membantu memahami cara kerja pointer, manajemen memori, dan pengolahan data yang lebih fleksibel dibandingkan struktur data statis.

E. Referensi

https://firmaninformatika.blogspot.com/2014/11/referensi-multiple-linked-list-multi.html

https://www.geeksforgeeks.org/multilevel-linked-list/

https://www.geeksforgeeks.org/data-structures/linked-list/

https://www.programiz.com/dsa/linked-list