LAPORAN PRAKTIKUM

STRUKTUR DATA

MODUL 11 (13)

MULTI LINKED LIST

Disusun Oleh :

NAMA : LAHRA BUDI SAPUTRA

NIM : 103112430054

Dosen :

FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

Multi-Linked List (atau Multi List) merupakan struktur data yang terdiri dari sekumpulan *list* berbeda yang memiliki keterhubungan satu sama lain, biasanya membentuk hierarki antara *list* induk (*parent*) dan *list* anak (*child*). Dalam struktur ini, setiap elemen pada *list* induk dapat menunjuk ke sebuah *list* anak tersendiri. Karena adanya relasi ketergantungan ini, manipulasi data memerlukan penanganan khusus: penambahan elemen anak (*insert child*) harus didahului dengan mengetahui posisi induknya , dan penghapusan elemen induk (*delete parent*) mewajibkan seluruh elemen anak yang terkait untuk dihapus terlebih dahulu guna menjaga integritas data.

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

“main.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) {
```

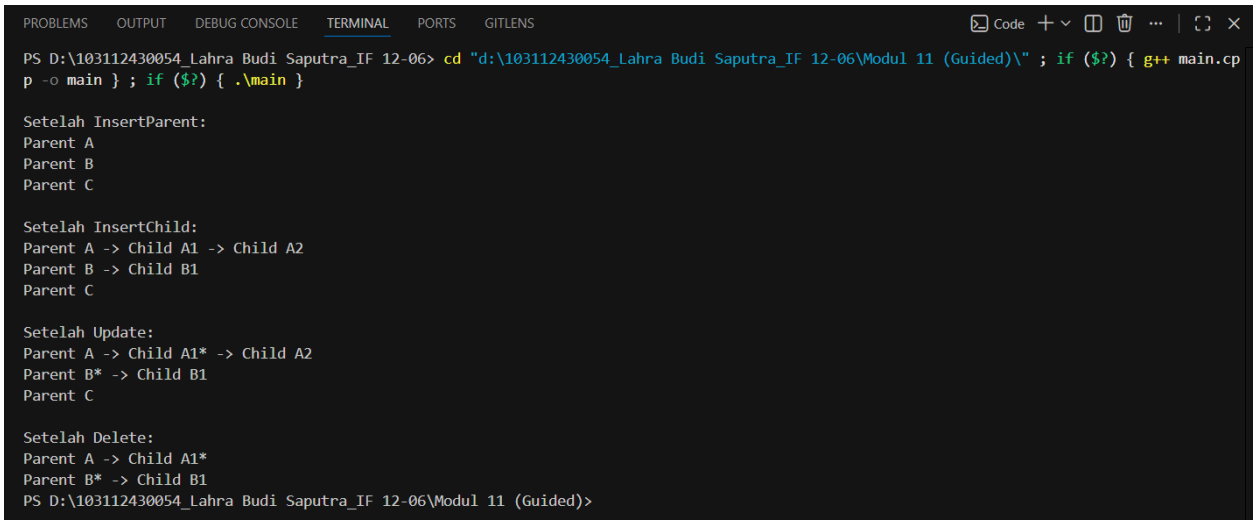
```
    ChildNode *newNode = new ChildNode;
    newNode->info = info;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}

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



Deskripsi:

Program ini mengimplementasikan struktur data **Multi Linked List** dengan model hierarki **Parent Child**, di mana kedua level senarai (baik Parent maupun Child) dibangun menggunakan struktur *Doubly Linked List*. Setiap node *Parent* tidak hanya terhubung satu sama lain, tetapi juga memiliki pointer khusus yang menunjuk ke daftar node *Child* miliknya sendiri, memungkinkan pengelompokan data yang terstruktur.

Fitur utama kode ini mencakup operasi **CRUD** yang lengkap: penambahan data (insert) dilakukan dengan menaruh simpul baru di akhir senarai, pembaruan data (update) dilakukan dengan mencari nilai string tertentu, dan penghapusan data (delete) yang secara otomatis menyesuaikan pointer next dan prev agar rantai data tidak putus. Secara khusus, logika penghapusan Parent dirancang agar aman dengan mekanisme yang membersihkan seluruh daftar anak (childHead) terlebih dahulu sebelum menghapus induknya, guna mencegah kebocoran memori (*memory leak*).

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

Unguided (SOAL 1)

(“multilist.h”)

```
#ifndef MULTILIST_H_INCLUDED
#define MULTILIST_H_INCLUDED
#include <iostream>
#include <string>
```

```
#define Nil NULL

using namespace std;

typedef string infotypeinduk;
typedef string infotypeanak;
typedef struct elemen_list_induk *address;
typedef struct elemen_list_anak *address_anak;

// Struktur Anak
struct elemen_list_anak {
   infotypeanak info;
   address_anak next;
   address_anak prev;
};

struct listanak {
   address_anak first;
   address_anak last;
};

// Struktur Induk
struct elemen_list_induk {
   infotypeinduk info;
   listanak lanak;
   address next;
   address prev;
};

struct listinduk {
   address first;
   address last;
};

// PROTOTYPE FUNGSI (PROGRAM 46)

// Pengecekan List Kosong
bool ListEmpty(listinduk L);
bool ListEmptyAnak(listanak L);

// Pembuatan List
void CreateList(listinduk &L);
void CreateListAnak(listanak &L);
```

```
// Manajemen Memori
address alokasi(infotypeinduk X);
address_anak alokasiAnak(infotypeanak X);
void dealokasi(address P);
void dealokasiAnak(address_anak P);

// Pencarian
address findElm(listinduk L, infotypeinduk X);
address_anak findElm(listanak Lanak, infotypeanak X);

bool fFindElm(listinduk L, address P);
bool fFindElmanak(listanak Lanak, address_anak P);

address findBefore(listinduk L, address P);
address_anak findBeforeAnak(listanak Lanak, infotypeinduk X, address_anak P);

// Penambahan Elemen Induk
void insertFirst(listinduk &L, address P);
void insertLast(listinduk &L, address P);
void insertAfter(listinduk &L, address P, address Prec);

// Penambahan Elemen Anak
void insertFirstAnak(listanak &L, address_anak P);
void insertLastAnak(listanak &L, address_anak P);
void insertAfterAnak(listanak &L, address_anak P, address_anak Prec);

// Penghapusan Elemen Induk
void delFirst(listinduk &L, address &P);
void delLast(listinduk &L, address &P);
void delAfter(listinduk &L, address &P, address Prec);
void delP(listinduk &L, infotypeinduk X);

// Penghapusan Elemen Anak
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);

// Cetak & Hitung
void printInfo(listinduk L);
int nbList(listinduk L);
void printInfoAnak(listanak Lanak);
```

```
int nbListAnak(listanak Lanak);

// Proses Delete All
void delAll(listinduk &L);

#endif
```

(“multilist.cpp”)

```
#include "multilist.h"
// PRIMITIF UMUM & MEMORI

bool ListEmpty(listinduk L) {
   return (L.first == Nil);
}

bool ListEmptyAnak(listanak L) {
   return (L.first == 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;
   if (P != Nil) {
      P->info = X;
      P->next = Nil;
      P->prev = Nil;
      CreateListAnak(P->lanak);
   }
   return P;
}

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;
}

void dealokasi(address P) {
    delete P;
}

void dealokasiAnak(address_anak P) {
    delete P;
}

// PENCARIAN (SEARCHING)

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 Lanak, infotypeanak X) {
    address_anak P = Lanak.first;
    while (P != Nil) {
        if (P->info == X) return P;
        P = P->next;
    }
    return Nil;
}

bool fFindElm(listinduk L, address P) {
    address Q = L.first;
    while (Q != Nil) {
        if (Q == P) return true;
        Q = Q->next;
    }
    return false;
}
```

```
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;
}

address findBefore(listinduk L, address P) {
    if (P == L.first) {
        return Nil;
    } else {
        return P->prev;
    }
}

address_anak findBeforeAnak(listanak Lanak, infotypeinduk X, address_anak P) {
    if (P == Lanak.first) {
        return Nil;
    } else {
        return P->prev;
    }
}

// INSERT (PENAMBAHAN)

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 {
        P->prev = L.last;
        L.last->next = P;
        L.last = P;
    }
}

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

void insertAfterAnak(listanak &L, address_anak P, address_anak Prec) {
    P->next = Prec->next;
    P->prev = Prec;
```

```
    if (Prec->next != Nil) {
        Prec->next->prev = P;
    } else {
        L.last = P;
    }
    Prec->next = P;
}

// DELETE (PENGHAPUSAN)

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;
        }
    } else {
        P = Nil;
    }
}

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;
        }
    } else {
        P = Nil;
    }
}

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;
    } else {
        P = Nil;
    }
}

void delP(listinduk &L, infotypeinduk X) {
    address P = findElm(L, X);
    if (P != Nil) {
        address_anak anakDel;
        while (!ListEmptyAnak(P->lanak)) {
            delFirstAnak(P->lanak, anakDel);
            dealokasiAnak(anakDel);
        }

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

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;
        }
    } else {
        P = Nil;
    }
}

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;
        }
    } else {
        P = Nil;
    }
}

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;
    } else {
        P = Nil;
    }
}

void delPAnak(listanak &L, infotypeanak X) {
    address_anak P = findElm(L, X);
```

```
    if (P != Nil) {
        address_anak Pdel;
        if (P == L.first) {
            delFirstAnak(L, Pdel);
        } else if (P == L.last) {
            delLastAnak(L, Pdel);
        } else {
            delAfterAnak(L, Pdel, P->prev);
        }
        dealokasiAnak(Pdel);
    }
}

// PROSES & INFO

void printInfo(listinduk L) {
    if (ListEmpty(L)) {
        cout << "[List Kosong]" << endl;
    } else {
        address P = L.first;
        while (P != Nil) {
            cout << "Pegawai: " << P->info << endl;
            printInfoAnak(P->lanak);
            P = P->next;
        }
    }
}

void printInfoAnak(listanak Lanak) {
    if (ListEmptyAnak(Lanak)) {
        cout << "   (Tidak ada anak)" << endl;
    } else {
        address_anak Q = Lanak.first;
        cout << "   Anak: ";
        while (Q != Nil) {
            cout << Q->info;
            if (Q->next != Nil) cout << ", ";
            Q = Q->next;
        }
        cout << endl;
    }
}
```

```
int nbList(listinduk L) {
    int count = 0;
    address P = L.first;
    while (P != Nil) {
        count++;
        P = P->next;
    }
    return count;
}

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

void delAll(listinduk &L) {
    address P;
    while (!ListEmpty(L)) {

        delFirst(L, P);

        address_anak anakDel;
        while (!ListEmptyAnak(P->lanak)) {
            delFirstAnak(P->lanak, anakDel);
            dealokasiAnak(anakDel);
        }

        dealokasi(P);
    }
}
```

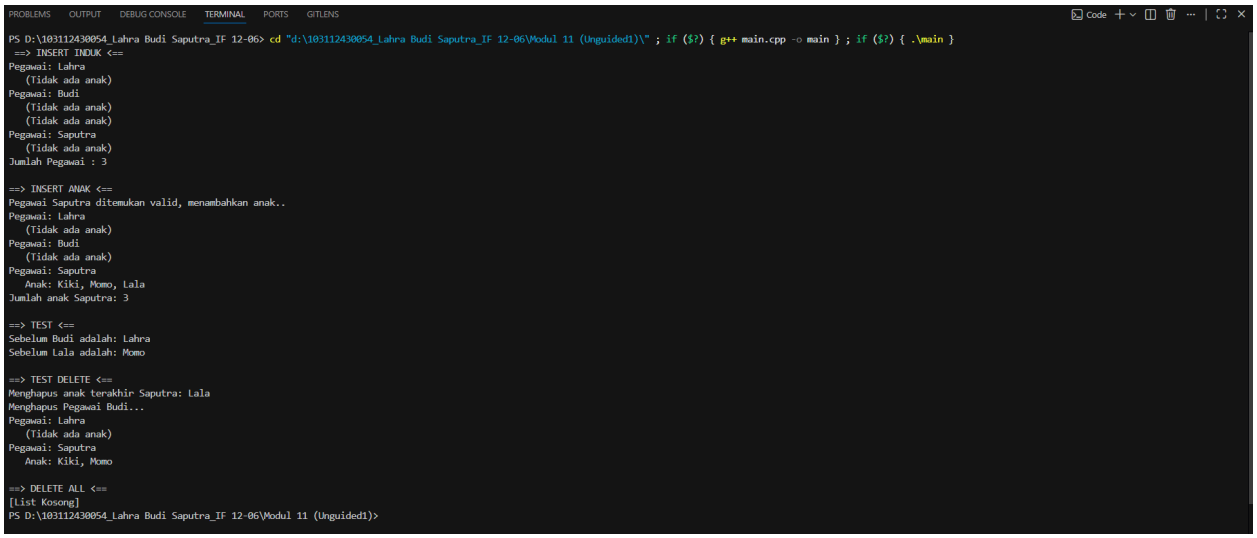
(“main.cpp”)

```
#include <iostream>
#include "multilist.h"
#include "multilist.cpp"

using namespace std;
```

```
int main() {
   listinduk L;
   CreateList(L);
   address P;
   address_anak Panak;

   cout << " ==> INSERT INDUK <==" << endl;

   P = alokasi("Saputra");
   insertLast(L, P);

   P = alokasi("Lahra");
   insertFirst(L, P);

   address Prec = findElm(L, "Lahra");
   if (Prec != Nil) {
      P = alokasi("Budi");
      insertAfter(L, P, Prec);
   }

   printInfo(L);
   cout << "Jumlah Pegawai : " << nbList(L) << endl << endl;

   cout << "==> INSERT ANAK <==" << endl;

   address ortu = findElm(L, "Saputra");

   if (fFindElm(L, ortu)) {
      cout << "Pegawai Saputra ditemukan valid, menambahkan anak.." << endl;

      Panak = alokasiAnak("Kiki");
      insertLastAnak(ortu->lanak, Panak);
      Panak = alokasiAnak("Lala");
      insertLastAnak(ortu->lanak, Panak);

      address_anak PrecAnak = findElm(ortu->lanak, "Kiki");
      if (PrecAnak != Nil) {
         Panak = alokasiAnak("Momo");
         insertAfterAnak(ortu->lanak, Panak, PrecAnak);
      }
   }
```

```
    printInfo(L);
    if (ortu != Nil) {
        cout << "Jumlah anak Saputra: " << nbListAnak(ortu->lanak) << endl;
    }
    cout << endl;

    cout << "==> TEST <==" << endl;

    address target = findElm(L, "Budi");
    address before = findBefore(L, target);
    if (before != Nil) {
        cout << "Sebelum Budi adalah: " << before->info << endl;
    }

    if (ortu != Nil) {
        address_anak targetAnak = findElm(ortu->lanak, "Lala");
        address_anak beforeAnak = findBeforeAnak(ortu->lanak, "XXX", targetAnak);
        if (beforeAnak != Nil) {
            cout << "Sebelum Lala adalah: " << beforeAnak->info << endl;
        }
    }
    cout << endl;

    cout << "==> TEST DELETE <==" << endl;

    if (ortu != Nil) {
        address_anak delAnak;
        delLastAnak(ortu->lanak, delAnak);
        cout << "Menghapus anak terakhir Saputra: " << delAnak->info << endl;
        dealokasiAnak(delAnak);
    }

    cout << "Menghapus Pegawai Budi..." << endl;
    delP(L, "Budi");
    printInfo(L);
    cout << endl;

    cout << "==> DELETE ALL <==" << endl;
    delAll(L);
    printInfo(L);

    return 0;
}
```

Screenshots Output:



Deskripsi:

Program di atas ini mengimplementasikan struktur data Multi Linked List dengan model Doubly Linked List untuk merepresentasikan relasi hierarkis satu-ke-banyak antara data Induk (Pegawai) dan Anak, sesuai dengan spesifikasi Modul 13. Struktur data ini didefinisikan dalam **multilist.h** menggunakan konsep nested list, di mana setiap node Induk memiliki pointer navigasi dua arah (next dan prev) serta memuat sebuah list anak tersendiri. Logika program yang tertuang dalam **multilist.cpp** mencakup operasi CRUD lengkap untuk kedua level list, dengan fitur manajemen memori krusial pada fungsi penghapusan (delP dan delAll) yang secara otomatis menghapus seluruh elemen anak yang terkait sebelum elemen induknya dihapus demi mencegah kebocoran memori. Keseluruhan fungsionalitas ini kemudian divalidasi melalui **main.cpp** yang mensimulasikan alur kerja nyata mulai dari inisialisasi, pencarian data induk, hingga manipulasi data anak.

Unguided (SOAL 2)

("circularlist.h")

```
#ifndef CIRCULARLIST_H_INCLUDED
#define CIRCULARLIST_H_INCLUDED

#include <iostream>
#include <string>

#define Nil NULL

using namespace std;

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

typedef mahasiswa infotype;
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 insertLast(List &L, address P);
void insertAfter(List &L, address Prec, address P);

void deleteFirst(List &L, address &P);
void deleteLast(List &L, address &P);
void deleteAfter(List &L, address Prec, address &P);
```

```
address findElm(List L, infotype x);
void printInfo(List L);

#endif
```

(“circularlist.cpp”)

```
#include "circularlist.h"

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 (L.First == Nil) {
      L.First = P;
      P->next = P;
   } else {
      address Last = L.First;
      while (Last->next != L.First) {
         Last = Last->next;
      }
      P->next = L.First;
      Last->next = P;
      L.First = P;
   }
}

void insertLast(List &L, address P) {
```

```
    if (L.First == Nil) {
        L.First = P;
        P->next = P;
    } else {
        address Last = L.First;

        while (Last->next != L.First) {
            Last = Last->next;
        }
        Last->next = P;
        P->next = L.First;
    }
}

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

void deleteFirst(List &L, address &P) {
    if (L.First != Nil) {
        P = L.First;
        if (P->next == L.First) {
            L.First = Nil;
        } else {
            address Last = L.First;
            while (Last->next != L.First) {
                Last = Last->next;
            }
            L.First = P->next;
            Last->next = L.First;
        }
        P->next = Nil;
    } else {
        P = Nil;
    }
}

void deleteLast(List &L, address &P) {
    if (L.First != Nil) {
        address Last = L.First;
```

```
        address PrecLast = Nil;

        while (Last->next != L.First) {
            PrecLast = Last;
            Last = Last->next;
        }

        P = Last;
        if (PrecLast == Nil) {
            L.First = Nil;
        } else {
            PrecLast->next = L.First;
        }
        P->next = Nil;
    } else {
        P = Nil;
    }
}

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

address findElm(List L, infotype x) {
    address P = L.First;
    if (P != Nil) {
        do {
            if (P->info.nim == x.nim) {
                return P;
            }
            P = P->next;
        } while (P != L.First);
    }
    return Nil;
}

void printInfo(List L) {
```

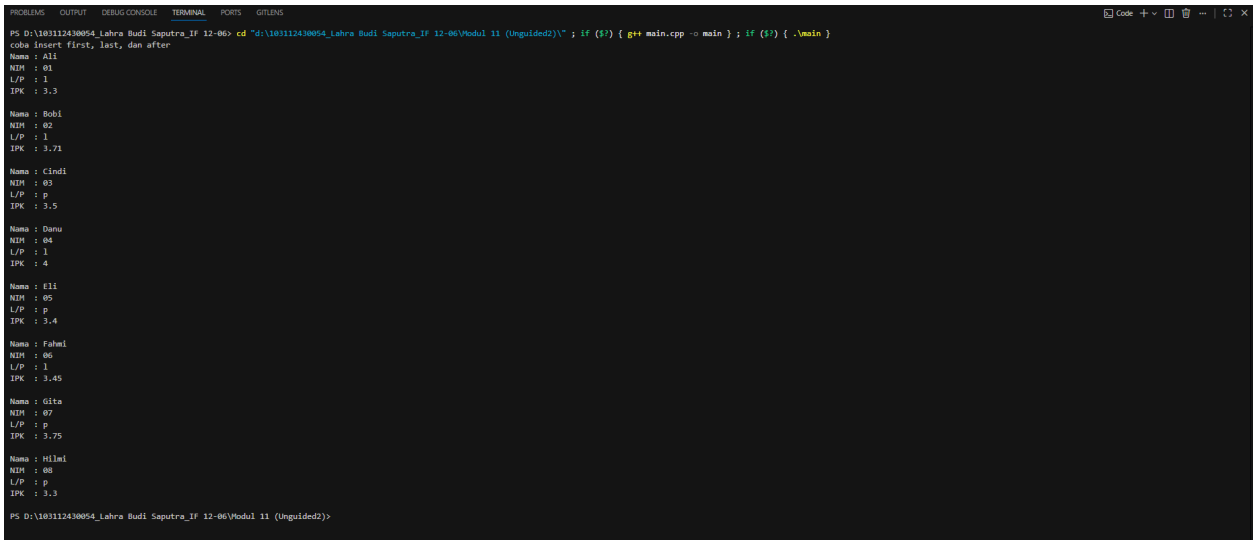
```
    if (L.First == Nil) {
        cout << "List Kosong" << endl;
        return;
    }

    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);
}
```

(“main.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;
    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;
    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 = "02";
    P1 = findElm(L, x);
    if (P1 != Nil) {
       P2 = createData("Cindi", "03", 'p', 3.5);
       insertAfter(L, P1, P2);
    }

    x.nim = "07";
    P1 = findElm(L, x);
    if (P1 != Nil) {
       P2 = createData("Hilmi", "08", 'p', 3.3);
       insertAfter(L, P1, P2);
    }

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

    printInfo(L);

    return 0;
}
```

Screenshots Output:



Deskripsi:

Program di atas ini mengimplementasikan struktur data Circular Singly Linked List untuk mengelola data mahasiswa yang terdiri dari atribut Nama, NIM, Jenis Kelamin, dan IPK . Sesuai dengan karakteristik sirkular, elemen terakhir dalam list ini memiliki pointer next yang menunjuk kembali ke elemen pertama (First), sehingga membentuk rangkaian data tertutup tanpa akhir NULL . Kode program terbagi menjadi tiga file utama: **circularlist.h** sebagai definisi ADT, **circularlist.cpp** yang berisi logika manipulasi pointer untuk menjaga integritas sirkular, dan **main.cpp** sebagai penggerak utama. Dalam fungsi main, program memanfaatkan fungsi bantuan createData untuk mengalokasikan elemen baru, kemudian menyusun urutan data mahasiswa secara spesifik (dari Ali hingga Hilmi) menggunakan kombinasi operasi penyisipan di awal, akhir, dan tengah list berdasarkan pencarian NIM .

D. Kesimpulan

Modul 13 ini membahas secara mendalam mengenai struktur data Multi Linked List, yaitu sekumpulan list berbeda yang memiliki keterhubungan satu sama lain, di mana setiap elemen di dalamnya dapat membentuk list tersendiri. Fokus utama pembelajaran terletak pada implementasi relasi hierarkis antara list induk (sebagai data utama seperti pegawai) dan list anak (sebagai data turunan), yang keduanya dibangun menggunakan struktur Doubly Linked List untuk memungkinkan navigasi data dua arah. Aspek teknis yang sangat ditekankan adalah integritas manajemen memori,

Khususnya aturan bahwa penghapusan elemen induk mewajibkan penghapusan seluruh elemen anak yang terkait terlebih dahulu. Selain itu, modul ini juga memperluas pemahaman praktis melalui studi kasus tambahan mengenai Circular Linked List untuk data mahasiswa, yang melatih logika penanganan struktur data dengan siklus tertutup di mana elemen terakhir kembali menunjuk ke elemen awal

E. Referensi

Juliansyah, N., Sari, S. Y., & Dristyan, F. (2024). Optimasi Struktur Data Stack dan Queue Menggunakan Array Dinamis. Fusion: Journal of Research in Engineering, Technology and Applied Sciences, 1(2), 90-97. https://ejurnal.faaslibsmedia.com/index.php/fusion/article/view/264

Anita Sindar, R. M. S. (2019). Struktur Data Dan Algoritma Dengan C++ (Vol. 1). CV. AA. RIZKY. https://books.google.com/books?hl=en&lr=&id=GP_ADwAAQBAJ&oi=fnd&pg=PA23&dq=stack+pada+c%2B%2B&ots=86k4Nl2OhV&sig=0KNR8rE2WYaLliEAZmi71x2eU7k

Santoso, L. E. (2004). STANDARD TEMPLATE LIBRARY C++ UNTUK MENGAJARKAN STRUKTUR DATA. Jurnal FASILKOM Vol, 2(2). https://www.academia.edu/download/56411324/standard-template-library-c__-untuk-mengajarkan-struktur-data.pdf