LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL XIII
MULTI LINKED LIST

Disusun Oleh :
NAMA : HILMI HAKIM RAMADANI
NIM : 103112430016

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Multi List merupakan sekumpulan list yang berbeda yang memiliki suatu keterhubungan satu sama lain. Tiap elemen dalam multi link list dapat membentuk listsendiri.Biasanya ada yang bersifat sebagai list induk dan list anak .

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

Guided

#main.cpp

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

using namespace std;

int main()
{
   // 1. inisialisasi list
   listInduk LInduk;
   createListInduk(LInduk);

   // 2. membuat data parent (kategori makanan)
   NodeParent K01 = alokasiNodeParent("K001", "Makanan Berat");
   insertLastParent(LInduk, K01);
   NodeParent K02 = alokasiNodeParent("K002", "Minuman");
   insertLastParent(LInduk, K02);
   NodeParent K03 = alokasiNodeParent("K003", "Dessert");
   insertLastParent(LInduk, K03);
   cout << endl;

   // 3. memasukkan data child  (menu makanan) ke kategori tertentu
   //  --> isi kategori makanan berat (K01)
   NodeChild M01 = alokasiNodeChild("M01", "Nasi Goreng Spesial");
   insertLastChild(K01->L_Anak, M01);
   NodeChild M02 = alokasiNodeChild("M02", "Ayam Bakar Madu");
   insertLastChild(K01->L_Anak, M02);
   // --> isi kategori minuman (K02)
   NodeChild D02 = alokasiNodeChild("D02", "Jus Alpukat");
   insertLastChild(K02->L_Anak, D02);
   NodeChild D01 = alokasiNodeChild("D01", "Es Teh Manis");
   insertLastChild(K02->L_Anak, D01);
   // --> isi kategori dessert (K03)
   NodeChild S01 = alokasiNodeChild("S01", "Puding Coklat");
   insertLastChild(K03->L_Anak, S01);
   cout << endl;

   // 4. print mll setelah insert-insert
   printStrukturMLL(LInduk);

   // 5. searching node child
   findChildByID(LInduk, "D01");
```

```
    cout << endl;

    // 6. delete node child
    deleteAfterChild(K01->L_Anak, M01); // hapus node ayam bakar
    cout << endl;

    // 7. delete node parent
    deleteAfterParent(LInduk, K02); // hapus node minuman
    cout << endl;

    // 8. print mll setelah delete
    printStrukturMLL(LInduk);
    cout << endl;

    return 0;
}
```

#multilist.cpp

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

using namespace std;

// create list
void createListInduk(listInduk &LInduk)
{
    LInduk.first = LInduk.last = NULL;
}

void createListAnak(listAnak &LAnak)
{
    LAnak.first = LAnak.last = NULL;
}

// alokasi & dealokasi parent
NodeParent alokasiNodeParent(string idKategoriMakanan, string
namaKategoriMakanan)
{
    NodeParent nodeBaruParent = new nodeParent;
    nodeBaruParent->idKategoriMakanan = idKategoriMakanan;
    nodeBaruParent->namaKategoriMakanan = namaKategoriMakanan;
    nodeBaruParent->next = NULL;
    nodeBaruParent->prev = NULL;
    createListAnak(nodeBaruParent->L_Anak);
    return nodeBaruParent;
}

void dealokasiNodeParent(NodeParent &nodeInduk)
{
    if (nodeInduk != NULL)
    {
        nodeInduk->next = nodeInduk->prev = NULL;
        delete nodeInduk;
        nodeInduk = NULL;
```

```
    }
}

// alokasi & delaokasi child
NodeChild alokasiNodeChild(string idMakanan, string namaMakanan)
{
    NodeChild nodeBaruChild = new nodeChild;
    nodeBaruChild->idMakanan = idMakanan;
    nodeBaruChild->namaMakanan = namaMakanan;
    nodeBaruChild->next = NULL;
    nodeBaruChild->prev = NULL;
    return nodeBaruChild;
}

void dealokasiNodeChild(NodeChild &nodeAnak)
{
    if (nodeAnak != NULL)
    {
        nodeAnak->next = nodeAnak->prev = NULL;
        delete nodeAnak;
        nodeAnak = NULL;
    }
}

// operasi pada parent
void insertLastParent(listInduk &LInduk, NodeParent nodeBaruParent)
{
    if (LInduk.first == NULL)
    {
        LInduk.first = LInduk.last = nodeBaruParent;
    }
    else
    {
        nodeBaruParent->prev = LInduk.last;

        LInduk.last->next = nodeBaruParent;
        LInduk.last = nodeBaruParent;
    }
    cout << "Node parent " << nodeBaruParent->namaKategoriMakanan << " berhasil
ditambahkan kedalam urutan terakhir di list Induk!" << endl;
}

void hapusListAnak(listAnak &LAnak)
{
    NodeChild nodeBantu = LAnak.first;
    while (nodeBantu != NULL)
    {
        NodeChild nodeHapus = nodeBantu;
        nodeBantu = nodeBantu->next;
        dealokasiNodeChild(nodeHapus);
    }
    LAnak.first = LAnak.last = NULL;
}

void deleteAfterParent(listInduk &LInduk, NodeParent nodePrev)
```

```
{
   if (LInduk.first == NULL)
   {
      cout << "List induk kosong!" << endl;
   }
   else
   {
      if (nodePrev != NULL && nodePrev->next != NULL)
      {
         NodeParent nodeHapus = nodePrev->next;
         nodePrev->next = nodeHapus->next;
         if (nodeHapus->next != NULL)
         {
            (nodeHapus->next)->prev = nodePrev;
         }
         else
         {
            LInduk.last = nodePrev;
         }
         nodeHapus->next = NULL;
         if (nodeHapus->L_Anak.first != NULL)
         {
            hapusListAnak(nodeHapus->L_Anak);
         }
         dealokasiNodeParent(nodeHapus);
         cout << "Node parent setelah node " << nodePrev->namaKategoriMakanan <<
" berhasil dihapus beserta anak-anaknya!" << endl;
      }
      else
      {
         cout << "Node prev tidak valid!" << endl;
      }
   }
}

// operasi pada child
void insertLastChild(listAnak &LAnak, NodeChild nodeBaruChild)
{
   if (LAnak.first == NULL)
   {
      LAnak.first = LAnak.last = nodeBaruChild;
   }
   else
   {
      nodeBaruChild->prev = LAnak.last;
      LAnak.last->next = nodeBaruChild;
      LAnak.last = nodeBaruChild;
   }
   cout << "Node child " << nodeBaruChild->namaMakanan << " berhasil ditambahkan
kedalam urutan terakhir di list Anak!" << endl;
}

void deleteAfterChild(listAnak &LAnak, NodeChild nodePrev)
{
   if (LAnak.first == NULL)
```

```
    {
        cout << "List anak kosong!" << endl;
    }
    else
    {
        if (nodePrev != NULL && nodePrev->next != NULL)
        {
            NodeChild nodeHapus = nodePrev->next;
            nodePrev->next = nodeHapus->next;
            if (nodeHapus->next != NULL)
            {
                (nodeHapus->next)->prev = nodePrev;
            }
            else
            {
                LAnak.last = nodePrev;
            }
            nodeHapus->next = NULL;
            dealokasiNodeChild(nodeHapus);
            cout << "Node child setelah node " << nodePrev->namaMakanan << " berhasil
dihapus!" << endl;
        }
        else
        {
            cout << "Node prev tidak valid!" << endl;
        }
    }
}
void findChildByID(listInduk &LInduk, string IDCari)
{
    if (LInduk.first == NULL)
    {
        cout << "List induk kosong!" << endl;
    }
    else
    {
        NodeParent nodeBantuParent = LInduk.first;
        int indexParent = 1;
        int ketemu = false;
        while (nodeBantuParent != NULL)
        {
            NodeChild nodeBantuChild = nodeBantuParent->L_Anak.first;
            int indexChild = 1;
            while (nodeBantuChild != NULL)
            {
                if (nodeBantuChild->idMakanan == IDCari)
                {
                    cout << "Data ID child ditemukan pada list anak dari node parent " <<
nodeBantuParent->namaKategoriMakanan << " pada posisi ke-" << indexChild << "!"
<< endl;
                    cout << "--- Data Child ---" << endl;
                    cout << "ID Child (ID Makanan) : " << nodeBantuChild->idMakanan <<
endl;
                    cout << "Posisi dalam list anak : posisi ke-" << indexChild << endl;
                    cout << "Nama Makanan : " << nodeBantuChild->namaMakanan << endl;
```

```
                cout << "--------------------------" << endl;
                cout << "--- Data Parent ---" << endl;
                cout << "ID Parent (ID Kategori Makanan): " << nodeBantuParent-
>idKategoriMakanan << endl;
                cout << "Posisi dalam list induk : posisi ke-" << indexParent << endl;
                cout << "Nama Kategori Makanan : " << nodeBantuParent-
>namaKategoriMakanan << endl;
                ketemu = true;
                break;
            }
            else
            {
                nodeBantuChild = nodeBantuChild->next;
                indexChild++;
            }
        }
        if (ketemu)
        {
            break;
        }
        else
        {
            nodeBantuParent = nodeBantuParent->next;
            indexParent++;
        }
    }
    if (!ketemu)
    {
        cout << "Data ID child tidak ditemukan didalam list anak!" << endl;
    }
  }
}

// operasi print
void printStrukturMLL(listInduk &LInduk)
{
  if (LInduk.first == NULL)
  {
    cout << "List induk kosong!" << endl;
  }
  else
  {
    NodeParent nodeBantuParent = LInduk.first;
    int indexParent = 1;
    while (nodeBantuParent != NULL)
    {
        cout << "=== Parent " << indexParent << " ===" << endl;
        cout << "ID Kategori Makanan : " << nodeBantuParent->idKategoriMakanan
<< endl;
        cout << "Nama Kategori Makanan : " << nodeBantuParent-
>namaKategoriMakanan << endl;

        // print list anak dari node parentnya
        NodeChild nodeBantuChild = nodeBantuParent->L_Anak.first;
        if (nodeBantuChild == NULL)
```

```
        {
            cout << "  (tidak ada child)" << endl;
        }
        else
        {
            int indexChild = 1;
            while (nodeBantuChild != NULL)
            {
                cout << "  - Child " << indexChild << " :" << endl;
                cout << "      ID Makanan : " << nodeBantuChild->idMakanan << endl;
                cout << "      Nama Makanan : " << nodeBantuChild->namaMakanan <<
endl;
                nodeBantuChild = nodeBantuChild->next;
                indexChild++;
            }
        }
        cout << "----------------------------" << endl;
        nodeBantuParent = nodeBantuParent->next;
        indexParent++;
    }
  }
}
```

#multilist.h

```
#ifndef MULTILIST_H
#define MULTILIST_H
#include <iostream>
using namespace std;

typedef struct nodeParent *NodeParent; // alias pointer ke struct nodeParent
typedef struct nodeChild *NodeChild;   // alias pointer ke struct nodeChild

struct nodeChild
{ // node child
   string idMakanan;
   string namaMakanan;
   NodeChild next;
   NodeChild prev;
};

struct listAnak
{ // list child
   NodeChild first;
   NodeChild last;
};

struct nodeParent
{ // node parent
   string idKategoriMakanan;
   string namaKategoriMakanan;
   NodeParent next;
   NodeParent prev;
   listAnak L_Anak;
```

```
};

struct listInduk
{ // list parent
    NodeParent first;
    NodeParent last;
};

// create list
void createListInduk(listInduk &LInduk);
void createListAnak(listAnak &LAnak);

// alokasi & dealokasi parent
NodeParent alokasiNodeParent(string idKategoriMakanan, string
namaKategoriMakanan);
void dealokasiNodeParent(NodeParent &nodeInduk);

// alokasi & dealokasi child
NodeChild alokasiNodeChild(string idMakanan, string namaMakanan);
void dealokasiNodeChild(NodeChild &nodeAnak);

void insertLastParent(listInduk &LInduk, NodeParent nodebaruParent);
void hapusListAnak(listAnak &LAnak);
void deleteAfterParent(listInduk &LInduk, NodeParent nodePrev);

void insertLastChild(listAnak &LAnak, NodeChild nodebaruChild);
void deleteAfterChild(listAnak &LAnak, NodeChild nodePrev);
void findChildByID(listInduk &LInduk, string IDCari);

void printStrukturMLL(listInduk &LInduk);

#endif
```

Screenshots Output

```
Node parent Makanan Berat berhasil ditambahkan kedalam urutan terakhir di list Induk!
Node parent Minuman berhasil ditambahkan kedalam urutan terakhir di list Induk!
Node parent Dessert berhasil ditambahkan kedalam urutan terakhir di list Induk!

Node child Nasi Goreng Spesial berhasil ditambahkan kedalam urutan terakhir di list Anak!
Node child Ayam Bakar Madu berhasil ditambahkan kedalam urutan terakhir di list Anak!
Node child Jus Alpukat berhasil ditambahkan kedalam urutan terakhir di list Anak!
Node child Es Teh Manis berhasil ditambahkan kedalam urutan terakhir di list Anak!
Node child Puding Coklat berhasil ditambahkan kedalam urutan terakhir di list Anak!

=== Parent 1 ===
ID Kategori Makanan : K001
Nama Kategori Makanan : Makanan Berat
  - Child 1 :
      ID Makanan : M01
      Nama Makanan : Nasi Goreng Spesial
  - Child 2 :
      ID Makanan : M02
      Nama Makanan : Ayam Bakar Madu
--------------------------
=== Parent 2 ===
ID Kategori Makanan : K002
Nama Kategori Makanan : Minuman
  - Child 1 :
      ID Makanan : D02
      Nama Makanan : Jus Alpukat
  - Child 2 :
      ID Makanan : D01
      Nama Makanan : Es Teh Manis
--------------------------
=== Parent 3 ===
ID Kategori Makanan : K003
Nama Kategori Makanan : Dessert
  - Child 1 :
      ID Makanan : S01
      Nama Makanan : Puding Coklat
--------------------------
Data ID child ditemukan pada list anak dari node parent Minuman pada posisi ke-2!
--- Data Child ---
```

Deskripsi:

Program ini menerapkan Multi Linked List (MLL) untuk mengelola kategori makanan (parent) dan menu makanan (child). Setiap parent memiliki satu list anak yang berisi menu-menu terkait, dan baik list induk maupun list anak menggunakan double linked list.

Pada main.cpp, program melakukan inisialisasi list, menambahkan beberapa kategori makanan, memasukkan menu ke kategori tertentu, menampilkan struktur data, mencari menu berdasarkan ID, serta menghapus data parent maupun child.

File multilist.cpp berisi fungsi-fungsi untuk alokasi, insert, delete, pencarian, dan pencetakan data, sedangkan multilist.h mendefinisikan struktur dan prototipe fungsi. Program ini menunjukkan pengelolaan data hubungan satu-ke-banyak menggunakan struktur Multi Linked List.

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

Unguided

#main.cpp

```
#include "multilink.h"
#include "multilink.cpp"

int main()
```

```
{
   listParent LParent;
   createListParent(LParent);

   NodeParent G001 = allocNodeParent("G001", "Aves");
   NodeParent G002 = allocNodeParent("G002", "Mamalia");
   NodeParent G003 = allocNodeParent("G003", "Pisces");
   NodeParent G004 = allocNodeParent("G004", "Amfibi");
   NodeParent G005 = allocNodeParent("G005", "Reptil");

   insertLastParent(LParent, G001);
   insertLastParent(LParent, G002);
   insertLastParent(LParent, G003);
   insertLastParent(LParent, G004);
   insertLastParent(LParent, G005);

   insertLastChild(G001->L_Child, allocNodeChild("AV001", "Cendrawasih",
"Hutan", true, 0.3));
   insertLastChild(G001->L_Child, allocNodeChild("AV002", "Bebek", "Air", true,
2));

   insertLastChild(G002->L_Child, allocNodeChild("M001", "Harimau", "Hutan",
true, 200));
   insertLastChild(G002->L_Child, allocNodeChild("M003", "Gorila", "Hutan", false,
160));
   insertLastChild(G002->L_Child, allocNodeChild("M002", "Kucing", "Darat", true,
4));

   insertLastChild(G004->L_Child, allocNodeChild("AM001", "Kodok", "Sawah",
false, 0.2));

   printMLLStructure(LParent);

   searchHewanByEkor(LParent, false);

   deleteAfterParent(LParent, G003);

   printMLLStructure(LParent);

   return 0;
}
```

#multilink.cpp

```
#include "multilink.h"

bool isEmptyParent(listParent LParent)
{
   return LParent.first == NULL;
```

```
}

bool isEmptyChild(listChild LChild)
{
   return LChild.first == NULL;
}

void createListParent(listParent &LParent)
{
   LParent.first = LParent.last = NULL;
}

void createListChild(listChild &LChild)
{
   LChild.first = LChild.last = NULL;
}

NodeParent allocNodeParent(string idGol, string namaGol)
{
   NodeParent P = new nodeParent;
   P->isidata.idGolongan = idGol;
   P->isidata.namaGolongan = namaGol;
   P->next = P->prev = NULL;
   createListChild(P->L_Child);
   return P;
}

NodeChild allocNodeChild(string idHwn, string namaHwn,
                         string habitat, bool tail, float weight)
{
   NodeChild C = new nodeChild;
   C->isidata.idHewan = idHwn;
   C->isidata.namaHewan = namaHwn;
   C->isidata.habitat = habitat;
   C->isidata.ekor = tail;
   C->isidata.bobot = weight;
   C->next = C->prev = NULL;
   return C;
}

void deallocNodeParent(NodeParent &NParent)
{
   delete NParent;
   NParent = NULL;
}

void deallocNodeChild(NodeChild &NChild)
{
   delete NChild;
   NChild = NULL;
```

```
}

void insertFirstParent(listParent &LParent, NodeParent newNParent)
{
   if (isEmptyParent(LParent))
   {
      LParent.first = LParent.last = newNParent;
   }
   else
   {
      newNParent->next = LParent.first;
      LParent.first->prev = newNParent;
      LParent.first = newNParent;
   }
}

void insertLastParent(listParent &LParent, NodeParent newNParent)
{
   if (isEmptyParent(LParent))
   {
      LParent.first = LParent.last = newNParent;
   }
   else
   {
      newNParent->prev = LParent.last;
      LParent.last->next = newNParent;
      LParent.last = newNParent;
   }
}

void deleteFirstParent(listParent &LParent)
{
   if (!isEmptyParent(LParent))
   {
      NodeParent del = LParent.first;
      LParent.first = del->next;
      if (LParent.first != NULL)
         LParent.first->prev = NULL;
      deleteListChild(del->L_Child);
      deallocNodeParent(del);
   }
}

void deleteAfterParent(listParent &LParent, NodeParent NPrev)
{
   if (NPrev != NULL && NPrev->next != NULL)
   {
      NodeParent del = NPrev->next;
      NPrev->next = del->next;
      if (del->next != NULL)
```

```
            del->next->prev = NPrev;
        deleteListChild(del->L_Child);
        deallocNodeParent(del);
    }
}

void insertFirstChild(listChild &LChild, NodeChild newNChild)
{
    if (isEmptyChild(LChild))
    {
        LChild.first = LChild.last = newNChild;
    }
    else
    {
        newNChild->next = LChild.first;
        LChild.first->prev = newNChild;
        LChild.first = newNChild;
    }
}

void insertLastChild(listChild &LChild, NodeChild newNChild)
{
    if (isEmptyChild(LChild))
    {
        LChild.first = LChild.last = newNChild;
    }
    else
    {
        newNChild->prev = LChild.last;
        LChild.last->next = newNChild;
        LChild.last = newNChild;
    }
}

void deleteFirstChild(listChild &LChild)
{
    if (!isEmptyChild(LChild))
    {
        NodeChild del = LChild.first;
        LChild.first = del->next;
        if (LChild.first != NULL)
            LChild.first->prev = NULL;
        deallocNodeChild(del);
    }
}

void deleteAfterChild(listChild &LChild, NodeChild NPrev)
{
    if (NPrev != NULL && NPrev->next != NULL)
    {
```

```
        NodeChild del = NPrev->next;
        NPrev->next = del->next;
        if (del->next != NULL)
            del->next->prev = NPrev;
        deallocNodeChild(del);
    }
}

void deleteListChild(listChild &LChild)
{
    while (!isEmptyChild(LChild))
    {
        deleteFirstChild(LChild);
    }
}

void printMLLStructure(listParent &LParent)
{
    NodeParent P = LParent.first;
    int noParent = 1;

    while (P != NULL)
    {
        cout << "=== Parent " << noParent << " ===" << endl;
        cout << "ID Golongan : " << P->isidata.idGolongan << endl;
        cout << "Nama Golongan : " << P->isidata.namaGolongan << endl;

        if (isEmptyChild(P->L_Child))
        {
            cout << "(tidak ada child)" << endl;
        }
        else
        {
            NodeChild C = P->L_Child.first;
            int noChild = 1;
            while (C != NULL)
            {
                cout << "- Child " << noChild << " :" << endl;
                cout << "  ID Hewan : " << C->isidata.idHewan << endl;
                cout << "  Nama Hewan : " << C->isidata.namaHewan << endl;
                cout << "  Habitat : " << C->isidata.habitat << endl;
                cout << "  Ekor : " << (C->isidata.ekor ? 1 : 0) << endl;
                cout << "  Bobot : " << C->isidata.bobot << endl;
                C = C->next;
                noChild++;
            }
        }
        cout << "------------------------" << endl;
        P = P->next;
        noParent++;
```

```
    }
}

void searchHewanByEkor(listParent &LParent, bool tail)
{
    if (isEmptyParent(LParent))
    {
        cout << "List parent kosong!" << endl;
        return;
    }

    NodeParent P = LParent.first;
    int indexParent = 1;
    bool ketemu = false;

    while (P != NULL)
    {
        NodeChild C = P->L_Child.first;
        int indexChild = 1;

        while (C != NULL)
        {
            if (C->isidata.ekor == tail)
            {
                cout << "Data ditemukan pada list anak dari node parent "
                     << P->isidata.namaGolongan
                     << " pada posisi ke-" << indexChild << endl;

                cout << "--- Data Child ---" << endl;
                cout << "ID Hewan : " << C->isidata.idHewan << endl;
                cout << "Nama Hewan : " << C->isidata.namaHewan << endl;
                cout << "Habitat : " << C->isidata.habitat << endl;
                cout << "Ekor : " << (C->isidata.ekor ? "TRUE" : "FALSE") << endl;
                cout << "Bobot : " << C->isidata.bobot << endl;

                cout << "--- Data Parent ---" << endl;
                cout << "ID Golongan : " << P->isidata.idGolongan << endl;
                cout << "Nama Golongan : " << P->isidata.namaGolongan << endl;
                cout << "Posisi dalam list parent : posisi ke-"
                     << indexParent << endl;
                cout << "--------------------------" << endl;

                ketemu = true;
            }
            C = C->next;
            indexChild++;
        }

        P = P->next;
        indexParent++;
```

```
    }

    if (!ketemu)
    {
        cout << "Data hewan dengan ekor = "
             << (tail ? "TRUE" : "FALSE")
             << " tidak ditemukan!" << endl;
    }
}
```

#multilink.h

```
#ifndef MULTILL_H
#define MULTILL_H

#include <iostream>
using namespace std;

struct golonganHewan
{
    string idGolongan;
    string namaGolongan;
};

struct dataHewan
{
    string idHewan;
    string namaHewan;
    string habitat;
    bool ekor;
    float bobot;
};

typedef struct nodeParent *NodeParent;
typedef struct nodeChild *NodeChild;

struct nodeChild
{
    dataHewan isidata;
    NodeChild next;
    NodeChild prev;
};

struct listChild
{
    NodeChild first;
    NodeChild last;
};

struct nodeParent
```

```
{
   golonganHewan isidata;
   NodeParent next;
   NodeParent prev;
   listChild L_Child;
};

struct listParent
{
   NodeParent first;
   NodeParent last;
};

bool isEmptyParent(listParent LParent);
bool isEmptyChild(listChild LChild);

void createListParent(listParent &LParent);
void createListChild(listChild &LChild);

NodeParent allocNodeParent(string idGol, string namaGol);
NodeChild allocNodeChild(string idHwn, string namaHwn,
                  string habitat, bool tail, float weight);

void deallocNodeParent(NodeParent &NParent);
void deallocNodeChild(NodeChild &NChild);

void insertFirstParent(listParent &LParent, NodeParent newNParent);
void insertLastParent(listParent &LParent, NodeParent newNParent);
void deleteFirstParent(listParent &LParent);
void deleteAfterParent(listParent &LParent, NodeParent NPrev);

void insertFirstChild(listChild &LChild, NodeChild newNChild);
void insertLastChild(listChild &LChild, NodeChild newNChild);
void deleteFirstChild(listChild &LChild);
void deleteAfterChild(listChild &LChild, NodeChild NPrev);

void deleteListChild(listChild &LChild);

void printMLLStructure(listParent &LParent);
void searchHewanByEkor(listParent &LParent, bool tail);

#endif
```

Screenshots Output

```
=== Parent 1 ===
ID Golongan : G001
Nama Golongan : Aves
- Child 1 :
  ID Hewan : AV001
  Nama Hewan : Cendrawasih
  Habitat : Hutan
  Ekor : 1
  Bobot : 0.3
- Child 2 :
  ID Hewan : AV002
  Nama Hewan : Bebek
  Habitat : Air
  Ekor : 1
  Bobot : 2
-----------------------
=== Parent 2 ===
ID Golongan : G002
Nama Golongan : Mamalia
- Child 1 :
  ID Hewan : M001
  Nama Hewan : Harimau
  Habitat : Hutan
  Ekor : 1
  Bobot : 200
- Child 2 :
  ID Hewan : M003
  Nama Hewan : Gorila
  Habitat : Hutan
  Ekor : 0
  Bobot : 160
- Child 3 :
  ID Hewan : M002
  Nama Hewan : Kucing
  Habitat : Darat
  Ekor : 1
  Bobot : 4
```

Deskripsi:

Program ini mengimplementasikan Multi Linked List (MLL) untuk mengelola golongan hewan sebagai parent dan data hewan sebagai child, di mana setiap parent memiliki satu list child dan seluruh struktur menggunakan double linked list. Program melakukan inisialisasi list, penambahan golongan hewan, pemasukan data hewan ke golongan tertentu, penampilan struktur MLL, pencarian hewan berdasarkan atribut ekor (true/false), penghapusan satu golongan beserta seluruh data hewannya, serta penampilan ulang struktur setelah penghapusan, sehingga menunjukkan pengelolaan data hubungan satu-ke-banyak secara terstruktur dan efisien.

D. Kesimpulan

Berdasarkan praktikum Modul Multi Linked List yang telah dilakukan, dapat disimpulkan bahwa struktur data Multi Linked List sangat efektif untuk merepresentasikan hubungan satu-ke-banyak antara data parent dan data child. Melalui

implementasi program, mahasiswa dapat memahami cara membuat, menambah, mencari, menampilkan, dan menghapus data pada struktur Multi Linked List yang menggunakan double linked list baik pada list induk maupun list anak. Praktikum ini juga melatih pemahaman tentang manajemen memori, keterkaitan antar node, serta penerapan konsep struktur data dalam pemodelan kasus nyata, seperti pengelompokan kategori makanan dan golongan hewan. Dengan demikian, tujuan praktikum untuk memahami konsep dan implementasi Multi Linked List dapat tercapai dengan baik.

E. Referensi

Modul 13 Multi Linked List