LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL XI
MULTI LINKED LIST

Disusun Oleh :
NAMA : Afief Amar Purnomo
NIM : 103112430067

Dosen
FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Multi linked list adalah pengembangan dari linked list di mana satu node tidak hanya terhubung ke node berikutnya, tetapi juga dapat memiliki list lain sebagai “anak”, sehingga membentuk hubungan induk–anak dalam satu struktur. Pada contoh modul, list induk menyimpan data pegawai dan setiap pegawai mempunyai pointer ke list anak yang berisi data anak pegawai tersebut. Operasi dasarnya tetap sama seperti linked list biasa (insert, delete, cari elemen), hanya saja untuk mengolah anak harus ditentukan dulu node induknya.

Dalam C++, multi linked list biasanya dibuat dengan beberapa struct: struct induk yang berisi data (misalnya pegawai), pointer ke node induk berikutnya/ sebelumnya, dan satu field list anak; kemudian struct anak yang berisi data anak dan pointer next/prev. Memori dialokasikan secara dinamis (misalnya lewat fungsi alokasi() atau new), sedangkan penghapusan memakai dealokasi() supaya node yang dihapus tidak lagi memakai memori. Konsep ini mengikuti prinsip ADT (Abstract Data Type): pengguna hanya memanggil prosedur/fungsi seperti CreateList, insertFirst, insertLast, delFirst, delLast, tanpa perlu memikirkan detail pointer di dalamnya.

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

Guided 1

multilist.cpp > main()

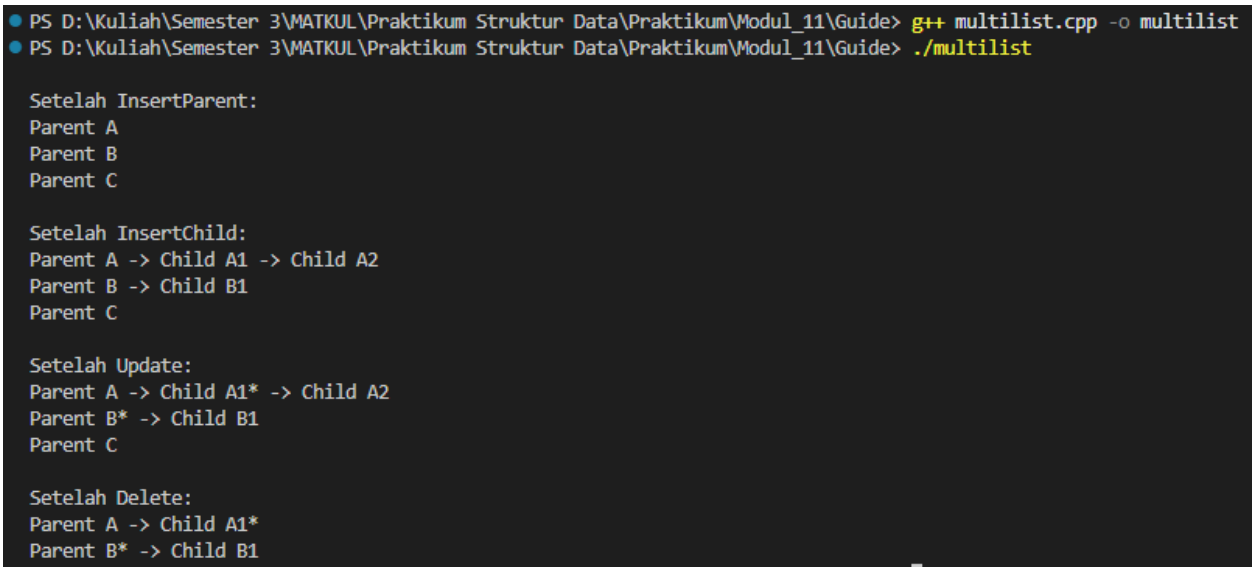
```cpp
#include <iostream>
#include <string>
using namespace std;

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

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

Tabnine | Edit | Test | Explain | Document
ParentNode *createParent(string info) {
    ParentNode *newNode = new ParentNode;
    newNode->info = info;
    newNode->childHead = NULL;
    newNode->next = NULL;
    newNode->prev = NULL;
    return newNode;
}

Tabnine | Edit | Test | Explain | Document
ChildNode *createChild(string info) {
    ChildNode *newNode = new ChildNode;
    newNode->info = info;
    newNode->next = NULL;
```

```
31      newNode->prev = NULL;
32      return newNode;
33  }
34
    Tabnine | Edit | Test | Explain | Document
35  void insertParent(ParentNode *&head, string info) {
36      ParentNode *newNode = createParent(info);
37      if (head == NULL) {
38          head = newNode;
39      } else {
40          ParentNode *temp = head;
41          while (temp->next != NULL) {
42              temp = temp->next;
43          }
44          temp->next = newNode;
45          newNode->prev = temp;
46      }
47  }
48
    Tabnine | Edit | Test | Explain | Document
49  void insertChild(ParentNode *head, string parentInfo, string childInfo) {
50      ParentNode *p = head;
51      while (p != NULL && p->info != parentInfo) {
52          p = p->next;
53      }
54
55      if (p != NULL) {
56          ChildNode *newChild = createChild(childInfo);
57          if (p->childHead == NULL) {
58              p->childHead = newChild;
59          } else {
60              ChildNode *c = p->childHead;
61              while (c->next != NULL) {
62                  c = c->next;
63              }
64              c->next = newChild;
65              newChild->prev = c;
66          }
67      }
68  }
69
    Tabnine | Edit | Test | Explain | Document
70  void printAll(ParentNode *head) {
71      while (head != NULL) {
72          cout << head->info;
73          ChildNode *c = head->childHead;
74          while (c != NULL) {
75              cout << " -> " << c->info;
76              c = c->next;
77          }
78          cout << endl;
79          head = head->next;
80      }
81  }
82
    Tabnine | Edit | Test | Explain | Document
83  void updateParent(ParentNode *head, string oldInfo, string newInfo) {
84      ParentNode *p = head;
85      while (p != NULL) {
```

```
    while (p != NULL) {
        if (p->info == oldInfo) {
            p->info = newInfo;
            return;
        }
        p = p->next;
    }
}

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

```
141  void deleteParent(ParentNode *&head, string info) {
142      ParentNode *p = head;
143      while (p != NULL) {
144          if (p->info == info) {
145              ChildNode *c = p->childHead;
146              while (c != NULL) {
147                  ChildNode *tempC = c;
148                  c = c->next;
149                  delete tempC;
150              }
151
152              if (p == head) {
153                  head = p->next;
154                  if (head != NULL) {
155                      head->prev = NULL;
156                  }
157              } else {
158                  p->prev->next = p->next;
159                  if (p->next != NULL) {
160                      p->next->prev = p->prev;
161                  }
162              }
163              delete p;
164              return;
165          }
166          p = p->next;
167      }
168  }
169
170  int main() {
171      ParentNode *list = NULL;
172
173      insertParent(list, "Parent A");
174      insertParent(list, "Parent B");
175      insertParent(list, "Parent C");
176
177      cout << "\nSetelah InsertParent: " << endl;
178      printAll(list);
179
180      insertChild(list, "Parent A", "Child A1");
181      insertChild(list, "Parent A", "Child A2");
182      insertChild(list, "Parent B", "Child B1");
183
184      cout << "\nSetelah InsertChild: " << endl;
185      printAll(list);
186
187      updateParent(list, "Parent B", "Parent B*");
188      updateChild(list, "Parent A", "Child A1", "Child A1*");
189
190      cout << "\nSetelah Update: " << endl;
191      printAll(list);
192
193      deleteChild(list, "Parent A", "Child A2");
194      deleteParent(list, "Parent C");
195
196      cout << "\nSetelah Delete: " << endl;
197      printAll(list);
198
199      return 0;
200  }
```

Screenshots Output



Deskripsi:

Program di atas merupakan implementasi struktur data multilist parent–child menggunakan doubly linked list, di mana setiap ParentNode dapat memiliki banyak ChildNode. Setiap parent disimpan dalam linked list utama, sedangkan setiap parent juga memiliki linked list anaknya sendiri. Program menyediakan berbagai operasi seperti insertParent untuk menambahkan parent baru, insertChild untuk menambahkan anak ke parent tertentu, updateParent dan updateChild untuk mengubah data parent atau child, serta deleteChild dan deleteParent untuk menghapus child tertentu atau menghapus seluruh parent beserta semua anaknya. Fungsi printAll digunakan untuk menampilkan seluruh parent beserta daftar child-nya. Pada fungsi main, program menambahkan tiga parent (“Parent A”, “Parent B”, “Parent C”), kemudian menambahkan beberapa child seperti “Child A1”, “Child A2”, dan “Child B1”. Selanjutnya, program melakukan update pada “Parent B” menjadi “Parent B*” dan mengubah “Child A1” menjadi “Child A1*”. Setelah itu, program menghapus “Child A2” dan menghapus “Parent C” beserta seluruh anaknya. Semua operasi tersebut kemudian ditampilkan melalui printAll sehingga memperlihatkan hasil perubahan dari setiap langkah.

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

Unguided 1 (soal 1)

(file circularlist.h)

circularlist.h > ...

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

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

typedef struct ElmList *address;

struct ElmList {
    infotype info;
    address next;
    address prev;
};

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);

```

```
29  void insertFirst(List &L, address P);
    Tabnine | Edit | Test | Explain | Document
30  void insertLast(List &L, address P);
    Tabnine | Edit | Test | Explain | Document
31  void insertAfter(List &L, address Prec, address P);
32
    Tabnine | Edit | Test | Explain | Document
33  void deleteFirst(List &L, address &P);
    Tabnine | Edit | Test | Explain | Document
34  void deleteLast(List &L, address &P);
    Tabnine | Edit | Test | Explain | Document
35  void deleteAfter(List &L, address Prec, address &P);
36
    Tabnine | Edit | Test | Explain | Document
37  address findElm(List L, infotype x);
    Tabnine | Edit | Test | Explain | Document
38  void printInfo(List L);
39
    Tabnine | Edit | Test | Explain | Document
40  address createData(string nama, string nim, char jenis_kelamin, float ipk);
41
42  #endif
43
```

(file circularlist.cpp)

circularlist.cpp > ...

```cpp
#include "circularlist.h"

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

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

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

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

```
    }
}

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

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

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

Tabnine | Edit | Test | Explain | Document
void deleteLast(List &L, address &P) {
    address last = L.first->prev;
    P = last;

    if (last == L.first) {
        L.first = NULL;
    } else {
        address beforeLast = last->prev;
        beforeLast->next = L.first;
        L.first->prev = beforeLast;
    }
}

Tabnine | Edit | Test | Explain | Document
void deleteAfter(List &L, address Prec, address &P) {
    P = Prec->next;
    if (P == L.first) return;
```

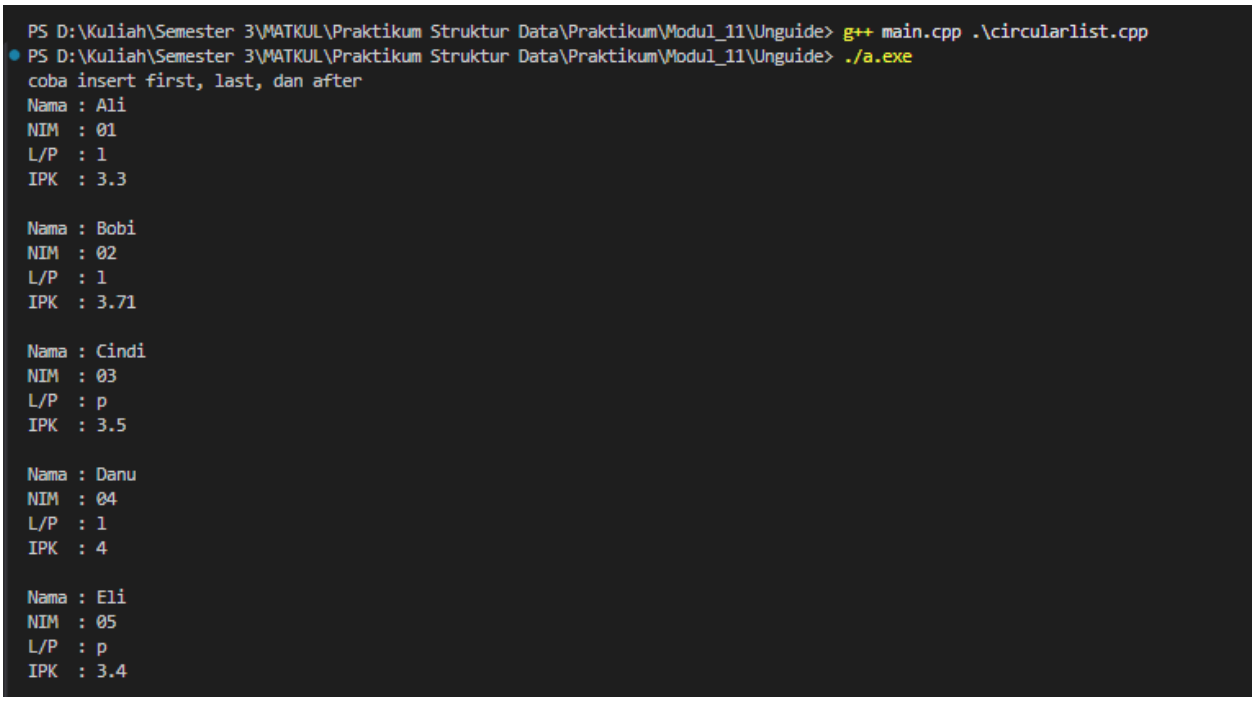
```
81
82      Prec->next = P->next;
83      P->next->prev = Prec;
84  }
85
    Tabnine | Edit | Test | Explain | Document
86  address findElm(List L, infotype x) {
87      if (L.first == NULL) return NULL;
88      address P = L.first;
89
90      do {
91          if (P->info.nim == x.nim)
92              return P;
93          P = P->next;
94      } while (P != L.first);
95
96      return NULL;
97  }
98
    Tabnine | Edit | Test | Explain | Document
99  void printInfo(List L) {
100     if (L.first == NULL) return;
101
102     address P = L.first;
103     do {
104         cout << "Nama : " << P->info.nama << endl;
105         cout << "NIM  : " << P->info.nim << endl;
106         cout << "L/P  : " << P->info.jenis_kelamin << endl;
107         cout << "IPK  : " << P->info.ipk << endl << endl;
108         P = P->next;
109     } while (P != L.first);
110 }
111
    Tabnine | Edit | Test | Explain | Document
112 address createData(string nama, string nim, char jenis_kelamin, float ipk) {
113     infotype x;
114     x.nama = nama;
115     x.nim = nim;
116     x.jenis_kelamin = jenis_kelamin;
117     x.ipk = ipk;
118     return alokasi(x);
119 }
120
```

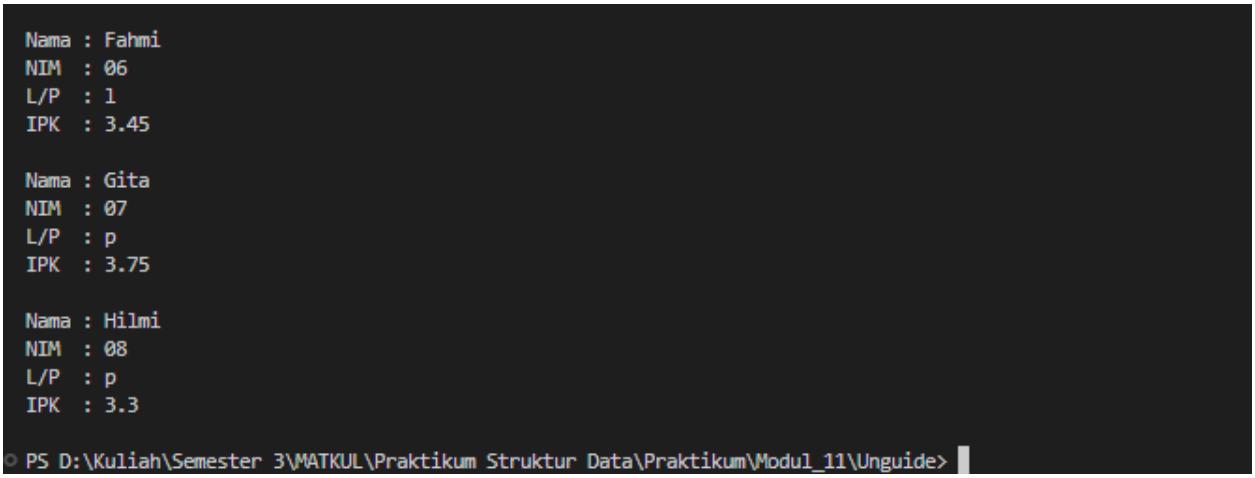
(file main.cpp)

main.cpp > main()

```cpp
#include "circularlist.h"

Tabnine | Edit | Test | Explain | Document
int main() {
    List L, A, B, L2;
    address P1 = NULL;
    address P2 = NULL;
    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);
    P2 = createData("Cindi", "03", 'p', 3.5);
    insertAfter(L, P1, P2);

    x.nim = "07";
    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:

Program ini merupakan implementasi struktur data Circular Doubly Linked List yang digunakan untuk menyimpan data mahasiswa berupa nama, NIM, jenis kelamin, dan IPK. Setiap elemen list bertipe ElmList memiliki pointer next dan prev yang saling terhubung secara melingkar, sehingga elemen terakhir menunjuk kembali ke elemen pertama. Program menyediakan berbagai operasi, seperti insertFirst, insertLast, dan insertAfter untuk menambahkan data di awal, akhir, atau setelah elemen tertentu; deleteFirst, deleteLast, dan deleteAfter untuk menghapus elemen; findElm untuk mencari mahasiswa berdasarkan NIM; serta printInfo untuk menampilkan seluruh isi list. Pada fungsi main, program membuat list baru lalu menambahkan beberapa data mahasiswa seperti Ali, Bobi, Danu, Fahmi, dan Gita menggunakan kombinasi insert first, last, dan after. Data tambahan seperti Cindi, Hilmi, dan Eli juga dimasukkan setelah elemen tertentu

berdasarkan pencarian NIM. Hasil akhir kemudian ditampilkan melalui fungsi printInfo, sehingga semua node dalam circular doubly linked list tercetak sesuai urutan penyisipan.

D. Kesimpulan

Berdasarkan hasil praktikum pada Modul XI tentang Multi Linked List, dapat disimpulkan bahwa multi linked list merupakan pengembangan dari struktur data linked list di mana setiap elemen (node induk) dapat memiliki list lain sebagai anak, sehingga membentuk hubungan parent–child dalam satu struktur terorganisasi. Melalui praktikum ini, mahasiswa memahami konsep pembentukan list induk dan list anak, proses alokasi serta dealokasi memori, dan berbagai operasi dasar seperti insert, delete, pencarian elemen, dan penelusuran data pada kedua tingkat list. Implementasi dilakukan dengan membangun ADT menggunakan bahasa C++, mencakup operasi seperti insertParent, insertChild, deleteParent, deleteChild, dan printAll untuk menampilkan seluruh data induk beserta anaknya.

Pada bagian Guided, mahasiswa mempraktikkan bagaimana setiap parent dapat memiliki banyak child, misalnya ketika “Parent A” memiliki “Child A1” dan “Child A2”. Operasi update juga dilakukan, contohnya mengubah “Parent B” menjadi “Parent B*” dan “Child A1” menjadi “Child A1*”. Sedangkan pada bagian Unguided, mahasiswa mengimplementasikan Circular Doubly Linked List untuk menyimpan data mahasiswa berupa nama, NIM, jenis kelamin, dan IPK, kemudian mengelola penambahan dan penghapusan node secara melingkar. Sebagai contoh, ketika program menambahkan parent “Parent A” dan memasukkan dua anak “Child A1” dan “Child A2”, struktur data akan menyimpan daftar parent dan menautkan list anak pada parent tersebut. Jika kemudian “Child A2” dihapus, maka list anak Parent A hanya menyisakan “Child A1”.

Contoh lain terlihat pada Circular Double Linked List, misalnya ketika data mahasiswa “Ali → Bobi → Danu” dimasukkan, pointer terakhir akan kembali menunjuk ke “Ali”, sehingga traversal dapat dilakukan tanpa henti dan tetap mempertahankan keterhubungan melingkar. Secara keseluruhan, praktikum ini memberikan pemahaman mendalam terkait bagaimana multi linked list bekerja untuk merepresentasikan data yang bersifat hierarkis, serta bagaimana circular doubly linked list mengelola data yang saling terhubung dalam bentuk melingkar. Praktikum ini juga memperkuat kemampuan mahasiswa dalam mengimplementasikan ADT, manajemen pointer, dan struktur data kompleks yang sering digunakan dalam aplikasi nyata seperti sistem data pegawai, data keluarga, manajemen kategori–subkategori, serta berbagai kasus yang melibatkan hubungan induk–anak.

E. Referensi

FIKTI UMSU. (2023). Pengertian Linked List: Struktur Data dalam Pemrograman. Fakultas Ilmu Komputer dan Teknologi Informasi UMSU. Di bagian jenis-jenis linked list dibahas juga “Multiple Linked List”. Diakses dari: https://fikti.umsu.ac.id/pengertian-linked-list-struktur-data-dalam-pemrograman/

Jurnal Struktur Data. (2025, Mei 19). Jurnal Modul 10 - Multi Linked List. Scribd. Implementasi multi linked list untuk kasus sales-penjualan dengan struct sales dan penjualan. Diakses dari https://id.scribd.com/document/618400689/Jurnal-Modul-10-Multi-Linked-List

Modul Struktur Data Revisi 2022. (2025). Circular Double Linked List dan Multiple / Multi Linked List. Dokumen modul (PDF) yang membahas implementasi multi linked list dalam konteks struktur data. Diakses dari: https://id.scribd.com/document/564327807/Modul-Struktur-Data-Revisi-2022