LAPORAN PRAKTIKUM STRUKTUR DATA

MODUL XIII
MULTI LINKED LIST

Disusun Oleh :

NAMA : ADIKA AUNURFIKRI NOVIYANTO

NIM : 103112400195

Dosen

FAHRUDIN MUKTI WIBOWO

PROGRAM STUDI STRUKTUR DATA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

C++ adalah pengembangan dari dari bahasa c yang dibuat oleh Bjarne Stroustrup sekitar tahun 1980-an. C++ disebut bahasa multi-paradigma, artinya bisa dipakai dengan gaya prosedural (pakai fungsi biasa), beriorientasi objek (pakai class dan object), atau bahkan gabungan keduanya. C++ punya dasar-dasar seperti variabel, operator percabangan (if, switch), perulangan (for, while), dan bisa memakai class untuk membuat objek.

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

Guided 2
multilist.h

```
#ifndef MULTILIST_H_INCLUDED
#define MULTILIST_H_INCLUDED

#include <iostream>
using namespace std;

#define Nil NULL
typedef bool boolean;
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;
};

boolean ListEmpty(listinduk L);
boolean ListEmptyAnak(listanak L);

void CreateList(listinduk &L);
void CreateListAnak(listanak &L);

address alokasi(infotypeinduk P);
address_anak alokasiAnak(infotypeanak P);

void dealokasi(address &P);
void dealokasiAnak(address_anak &P);

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

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

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

void insertFirst(listinduk &L, address P);
void insertAfter(listinduk &L, address P, address Prec);
void insertLast(listinduk &L, address P);

void insertFirstAnak(listanak &L, address_anak P);
void insertAfterAnak(listanak &L, address_anak P, address_anak Prec);
void insertLastAnak(listanak &L, address_anak P);

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);

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);

void printInfo(listinduk L);
int nbList(listinduk L);
```

```
void printInfoAnak(listanak Lanak);
int nbListAnak(listanak Lanak);

void delAll(listinduk &L);

#endif
```

multilist.cpp

```
#include "multilist.h"

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

boolean 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 P) {
   address newElm = new elemen_list_induk;
   if (newElm != Nil) {
      newElm->info = P;
      newElm->next = Nil;
      newElm->prev = Nil;
      CreateListAnak(newElm->lanak);
   }
```

```
    return newElm;
}

address_anak alokasiAnak(infotypeanak P) {
    address_anak newElm = new elemen_list_anak;
    if (newElm != Nil) {
        newElm->info = P;
        newElm->next = Nil;
        newElm->prev = Nil;
    }
    return newElm;
}

void dealokasi(address &P) {
    delete P;
    P = Nil;
}

void dealokasiAnak(address_anak &P) {
    delete P;
    P = Nil;
}

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 findElmAnak(listanak Lanak, infotypeanak X) {
```

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

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

boolean 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) {
    address Q = L.first;
    if (Q == P) {
```

```
        return Nil;
    }
    while (Q != Nil && Q->next != P) {
        Q = Q->next;
    }
    return Q;
}

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;
}

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 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;
    }
}

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 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 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;
    }
}

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 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;
        }
    }
}

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;
        }
    }
}

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;
    }
}

void delP(listinduk &L, infotypeinduk X) {
    address P = findElm(L, X);
    if (P != Nil) {
        if (P == L.first) {
            delFirst(L, P);
        } else if (P == L.last) {
            delLast(L, P);
        } else {
            address Prec = findBefore(L, P);
```

```
            delAfter(L, P, Prec);
        }

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

        dealokasi(P);
    }
}

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;
        }
    }
}

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;
        }
    }
}

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;
    }
}

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 = findBeforeAnak(L, P);
            delAfterAnak(L, P, Prec);
        }
        dealokasiAnak(P);
    }
}
```

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

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

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

```
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 Panak;
        while (!ListEmptyAnak(P->lanak)) {
            delFirstAnak(P->lanak, Panak);
            dealokasiAnak(Panak);
        }

        dealokasi(P);
    }
}
```

Main.cpp

```
#include "multilist.h"

int main() {
    listinduk Lpeg;
    address P1, P2, P3;
    address_anak A1, A2, A3, A4;

    CreateList(Lpeg);
```

```
cout << "=== MULTI LINKED LIST DEMO ===" << endl << endl;

cout << "Membuat data pegawai dan anak..." << endl << endl;

P1 = alokasi(1);
insertLast(Lpeg, P1);

P2 = alokasi(2);
insertLast(Lpeg, P2);

P3 = alokasi(3);
insertLast(Lpeg, P3);

A1 = alokasiAnak(101);
insertLastAnak(P1->lanak, A1);

A2 = alokasiAnak(102);
insertLastAnak(P1->lanak, A2);

A3 = alokasiAnak(103);
insertLastAnak(P1->lanak, A3);

A1 = alokasiAnak(201);
insertLastAnak(P2->lanak, A1);

A2 = alokasiAnak(202);
insertLastAnak(P2->lanak, A2);

cout << "Data setelah insert:" << endl;
printInfo(Lpeg);

cout << "Jumlah pegawai: " << nbList(Lpeg) << endl << endl;

cout << "Menghapus anak terakhir dari Pegawai 1..." << endl;
```

```
address induk = findElm(Lpeg, 1);
if (induk != Nil) {
   address_anak Pdel;
   delLastAnak(induk->lanak, Pdel);
   if (Pdel != Nil) {
      cout << "Anak " << Pdel->info << " berhasil dihapus" << endl;
      dealokasiAnak(Pdel);
   }
}
cout << endl;

cout << "Data setelah delete anak:" << endl;
printInfo(Lpeg);

cout << "Menghapus Pegawai 2 (beserta semua anaknya)..." << endl;
delP(Lpeg, 2);
cout << endl;

cout << "Data setelah delete pegawai:" << endl;
printInfo(Lpeg);

cout << "Jumlah pegawai: " << nbList(Lpeg) << endl << endl;

cout << "Menambah pegawai baru di awal..." << endl;
address P4 = alokasi(4);
insertFirst(Lpeg, P4);

A1 = alokasiAnak(401);
insertLastAnak(P4->lanak, A1);

cout << endl;
cout << "Data akhir:" << endl;
printInfo(Lpeg);

cout << "Menghapus semua data..." << endl;
```

```
    delAll(Lpeg);

    if (ListEmpty(Lpeg)) {
        cout << "Semua data berhasil dihapus" << endl;
    }

    return 0;
}
```

Screenshots Output:

```
PS C:\Users\Adika Aunurfikri\Downloads\Latihan_Modul13_Lengkap> ./multilist
=== MULTI LINKED LIST DEMO ===

Membuat data pegawai dan anak...

Data setelah insert:
Pegawai 1 memiliki anak:
  - Anak 101
  - Anak 102
  - Anak 103

Pegawai 2 memiliki anak:
  - Anak 201
  - Anak 202

Pegawai 3 memiliki anak:
  Tidak ada anak

Jumlah pegawai: 3

Menghapus anak terakhir dari Pegawai 1...
Anak 103 berhasil dihapus

Data setelah delete anak:
Pegawai 1 memiliki anak:
  - Anak 101
  - Anak 102

Pegawai 2 memiliki anak:
  - Anak 201
  - Anak 202
```

```
Pegawai 3 memiliki anak:
  Tidak ada anak

Menghapus Pegawai 2 (beserta semua anaknya)...

Data setelah delete pegawai:
Pegawai 1 memiliki anak:
  - Anak 101
  - Anak 102

Pegawai 3 memiliki anak:
  Tidak ada anak

Jumlah pegawai: 2

Menambah pegawai baru di awal...

Pegawai 4 memiliki anak:
  - Anak 401

Pegawai 1 memiliki anak:
  - Anak 101
  - Anak 102

Pegawai 3 memiliki anak:
  Tidak ada anak

Menghapus semua data...
Semua data berhasil dihapus
PS C:\Users\Adika Aunurfikri\Downloads\Latihan_Modul13_Lengkap>
```

Guided 3

Circularlist.h

```
#ifndef CIRCULARLIST_H_INCLUDED
#define CIRCULARLIST_H_INCLUDED

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

#define Nil NULL

typedef struct {
   string nama;
   string nim;
   char jenis_kelamin;
   float ipk;
} 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 insertAfter(List &L, address Prec, address P);
void insertLast(List &L, address P);
void deleteFirst(List &L, address &P);
void deleteAfter(List &L, address Prec, address &P);
void deleteLast(List &L, address &P);
address findElm(List L, infotype x);
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;
   P = Nil;
}

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;
      L.first = P;
      last->next = L.first;
   }
}

void insertAfter(List &L, address Prec, address P) {
   if (Prec != Nil) {
      P->next = Prec->next;
      Prec->next = 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 deleteFirst(List &L, address &P) {
    if (L.first != Nil) {
        P = L.first;
        if (L.first->next == L.first) {
            L.first = Nil;
        } else {
            address last = L.first;
            while (last->next != L.first) {
                last = last->next;
            }
            L.first = L.first->next;
            last->next = L.first;
        }
        P->next = 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;
    }
}

void deleteLast(List &L, address &P) {
    if (L.first != Nil) {
        if (L.first->next == L.first) {
            P = L.first;
```

```
            L.first = Nil;
        } else {
            address last = L.first;
            address prevLast = Nil;
            while (last->next != L.first) {
                prevLast = last;
                last = last->next;
            }
            P = last;
            prevLast->next = L.first;
        }
        P->next = Nil;
    }
}

address findElm(List L, infotype x) {
    if (L.first == Nil) {
        return Nil;
    }

    address P = L.first;
    do {
        if (P->info.nim == x.nim) {
            return P;
        }
        P = P->next;
    } while (P != L.first);

    return Nil;
}

void printInfo(List L) {
    if (L.first == Nil) {
        cout << "List kosong" << endl;
    } else {
```

```
    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 "circularlist.h"

address createData(string nama, string nim, char jenis_kelamin, float ipk) {
  infotype x;
  address P;
  x.nama = nama;
  x.nim = nim;
  x.jenis_kelamin = jenis_kelamin;
  x.ipk = ipk;
  P = alokasi(x);
  return P;
}

int main() {
  List L, A, B, L2;
  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", 'l', 3.3);
    insertAfter(L, P1, P2);

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

    printInfo(L);

    return 0;
}
```

Screenshots Output:

```
PS C:\Users\Adika Aunurfikri\Downloads\Latihan_Modul13_Lengkap> g++ -o program m
pp circularlist.cpp
PS C:\Users\Adika Aunurfikri\Downloads\Latihan_Modul13_Lengkap> ./program
coba insert first, last, dan after
Nama : Ali
NIM  : 01
L/P  : l
IPK  : 3.3

Nama : Bobi
NIM  : 02
L/P  : l
IPK  : 3.71

Nama : Hilmi
NIM  : 08
L/P  : l
IPK  : 3.3

Nama : Danu
NIM  : 04
L/P  : l
IPK  : 4

Nama : Eli
NIM  : 05
L/P  : p
IPK  : 3.4
```

```
Nama : Fahmi
NIM  : 06
L/P  : l
IPK  : 3.45

Nama : Gita
NIM  : 07
L/P  : p
IPK  : 3.75

Nama : Cindi
NIM  : 03
L/P  : p
IPK  : 3.5
```

Deskripsi:

Program ini merupakan implementasi dari dua jenis struktur data linked list yang berbeda, yaitu Multi Linked List dan Circular Linked List. Multi Linked List digunakan untuk merepresentasikan hubungan hierarki antara data pegawai sebagai list induk dan data anak sebagai list anak, dimana setiap pegawai dapat memiliki beberapa anak yang tersimpan dalam list tersendiri. Sedangkan Circular Linked List diimplementasikan untuk menyimpan data mahasiswa dengan struktur list yang melingkar, dimana elemen terakhir menunjuk kembali ke elemen pertama. Kedua program ini dilengkapi dengan operasi-operasi dasar seperti insert (first, after, last), delete (first, after, last, delP), pencarian elemen (findElm), dan fungsi-fungsi pendukung lainnya seperti alokasi memori, dealokasi, serta print untuk menampilkan isi list. Program Multi Linked List mendemonstrasikan kemampuan untuk mengelola data yang memiliki keterhubungan kompleks antar elemen, sementara Circular Linked List menunjukkan implementasi list dengan konsep sirkuler yang efisien untuk navigasi data yang berulang.

C. Kesimpulan

Implementasi Multi Linked List dan Circular Linked List ini berhasil menunjukkan fleksibilitas dan keunggulan struktur data linked list dalam mengelola data yang kompleks. Multi Linked List terbukti efektif untuk merepresentasikan relasi hierarki one-to-many antara pegawai dan anak-anaknya, dimana penghapusan elemen induk secara otomatis akan menghapus seluruh elemen anak yang terkait, sehingga menjaga konsistensi data. Sementara itu, Circular Linked List memberikan kemudahan dalam traversal data secara berulang tanpa perlu pengecekan kondisi akhir list yang rumit. Kedua implementasi ini menggunakan pointer untuk mengatur hubungan antar elemen, yang memungkinkan operasi insert dan delete dilakukan dengan efisien tanpa perlu menggeser elemen-elemen lain seperti pada array. Program-program ini telah diuji dan berjalan dengan baik, menunjukkan bahwa semua fungsi bekerja sesuai spesifikasi yang diharapkan dan dapat menjadi dasar untuk pengembangan aplikasi yang membutuhkan struktur data dengan relasi yang lebih kompleks.

D. Referensi

- W3Resource. (2020). *C++ String Exercises: Convert digit/number to words*.
- GeeksforGeeks. (2020). *Loops in C++ (for, while, do-while).*