# SISTEM PENGELOLAAN DATA MEKANIK DAN LAYANAN SERVIS BENGKEL


**Disusun oleh Kelompok :**

**Yosevin Hendraji / 103092400010**

**Naila Ayu Permatasari/ 103092400061**

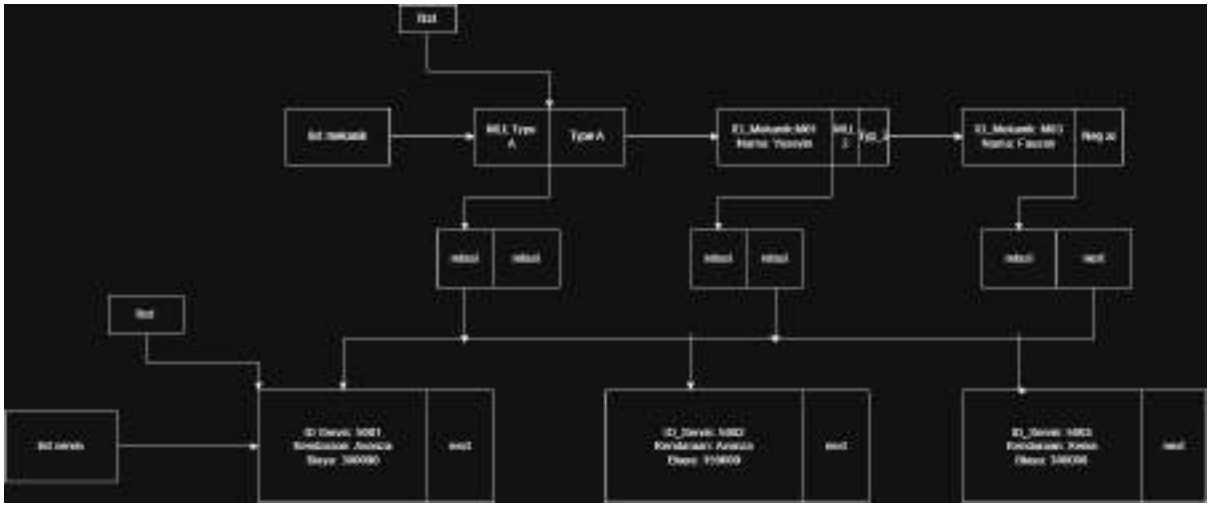
**Muhammad Fauzan Januar/ 103092400037**

**Mata Kuliah: Struktur Data**

**Prodi Teknologi Informasi, Fakultas Informatika, Universitas Telkom, 2025**

A. Type MLL : A
B. Jenis List Parent : Single Linked List Mekanik
C. Jenis List Child : Single  Linked List Servis
D. Model MLL
E. Data Mekanik (Parents)
    - ID Mekanik
    - Nama Mekanik
    - Jam Kerja per Hari  (int)
F. Data Servis (child)
    - ID servis
    - Nama Kendaraan
    - Biaya Servis (int)

G. Spesifikasi Data Multi Linked List Tipe A



1. Sebuah bengkel membutuhkan sistempendataan untuk mengelola informmasi mengenai mekanik dan layanan servis yang mereka kerjakan setiap hari. Saat ini, pencataan masih dilakukan secara manual sehingga sering terjadi kesalahan dalam pencatatan jam kerja mekanik, jenis servis yang ditangani, serta biaya servis yang harus dibayar pelanggan.

   Asistem yang akan dibuat berfungsi untuk menuimpan daftar mekanik (sebagai **parent**) dan daftar servis yang dikerjakan setiap mekanik (sebagai **child**). Setiap mekanik memiliki ID, nama, dan jumlah jam kerja per hari. Sementara itu, setiap layanan servis memiliki ID servis, nama kendaraan yang diservis, dan biaya servis.

   Sistem ini diharapkan membantu bengkel dalam:
- Melacak mekanik mana yang menangani servis tertentu.
- Menghitung total biaya servis.

- Mempermudah pelaporan aktivitas harian bengkel.
- Menghindari duplikasi data dan kehilangan catatan.

2.

a) Insert elemen parents
b) Insert elemen child
c) Insert elemen relation
d) Delete elemen parents
e) Delete elemen child
f) Delete element relation
g) Find element parents
h) Find elemen child
i) Find element relation
j) Show all data di list parents
k) Show all data di list child
l) Show data child dari parent tertentu
m) Show setiap data parent beserta data child yang berelasi dengannya
n) Show data child beserta data parent yang masing-masing child miliki
o) Show data parent yang berelasi dengan child tertentu
p) Count relation dari setiap element parent
q) Count relation yang dimiliki oleh child tertentu
r) Count element child yang tidak memiliki relasi
s) Edit relasi/mengganti child dari parent tertentu

3.

a.

```
// a. Insert element parent (Mekanik)
void insertMekanik(listMekanik &LM, elmMekanik data) {
    adrMekanik M = new nodeMekanik;
    M->infoMekanik = data;
    M->nextMekanik = NULL;
    M->firstRelasi = NULL;

    if (LM.firstMekanik == NULL) {
        LM.firstMekanik = M;
    } else {
        adrMekanik curr = LM.firstMekanik;
        while (curr->nextMekanik != NULL) {
            curr = curr->nextMekanik;
        }
        curr->nextMekanik = M;
    }
    cout << "Mekanik berhasil ditambahkan.\n";
}
```

Penjelasan :
Data  mekanik baru  yang berisi ID Mekanik, nama mekanik, dan jam kerja per hari akan dimasukkan ke dalam list mekanik sebagai elemen baru. Jika list kosong, maka data mekanik tersebut akan menjadi elemen pertama.Jika list sudah terisi, proses akan menelusuri node mekanik hingga menemukan posisi yang sesuai, kemudia menambahkan data mekanik bar uke dalam list.

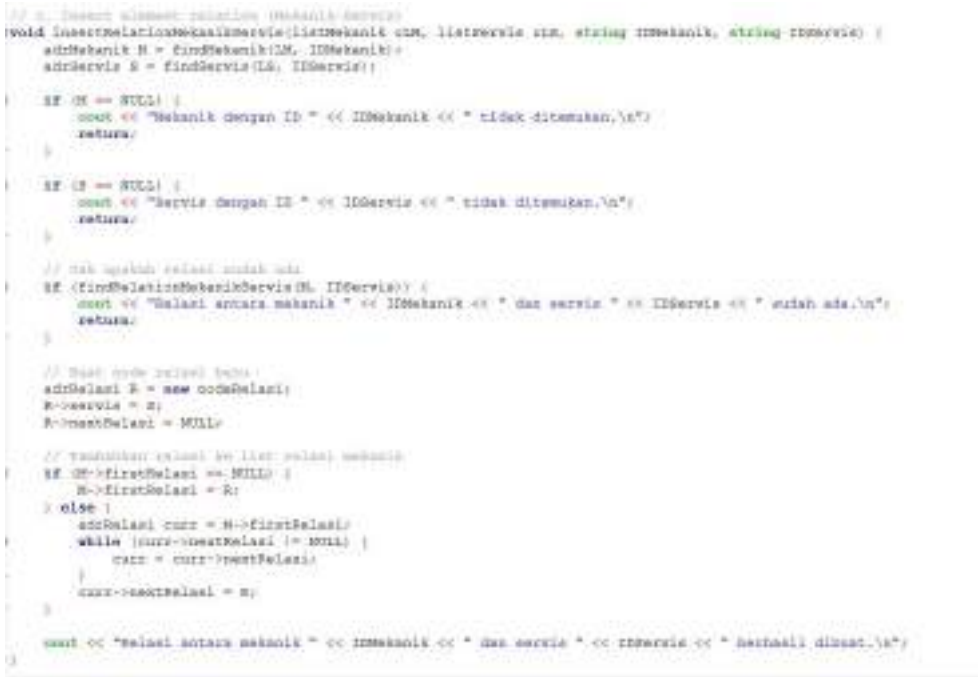
b.

```
// b. Insert element child (Servis)
void insertServis(listServis &LS, elmServis data) {
    adrServis S = new nodeServis;
    S->infoServis = data;
    S->nextServis = NULL;

    if (LS.firstServis == NULL) {
        LS.firstServis = S;
    } else {
        adrServis curr = LS.firstServis;
        while (curr->nextServis != NULL) {
            curr = curr->nextServis;
        }
        curr->nextServis = S;
    }
    cout << "Servis berhasil ditambahkan.\n";
}
```

Penjelasan:
Data Servis baru yang berisi ID  servis, Nama kendaraan, dan biaya servis akan di masukkan ke dalam list servis. Apabila list servis masih kosong, data servis akan menjadi elemen pertama. Jika list telah berisi elemenlain, maka data servis baru akan di tambahkan  ke dalam list sebagai elemen berikutnya.

c.



Penjelasan:

Relasi dibuat dengan cara menghubungkan satu data  mekanik dengan satu data servis. Proses dimulai dengan mencari mekanik dan servis  yang akan dihubungkan. Jika keduanya ditemukan, maka sebuah node relasi akan dibuat dan disimpan pada list relasi milik mekanik, sehingga  mekanik tersebut tercatat menangani  servis yang dipilih.

d.

```
// 8. Delete elemen parent (Mekanik)
void deleteMekanik(listMekanik &LM, string IDMekanik) {
    if (LM.firstMekanik == NULL) {
        cout << "List mekanik kosong.\n";
        return;
    }

    adrMekanik M = LM.firstMekanik;
    adrMekanik prev = NULL;

    // Cari mekanik yang akan dihapus
    while (M != NULL && M->infoMekanik.IDMekanik != IDMekanik) {
        prev = M;
        M = M->nextMekanik;
    }

    if (M == NULL) {
        cout << "Mekanik dengan ID " << IDMekanik << " tidak ditemukan.\n";
        return;
    }

    // Hapus semua relasi yang terkait
    adrRelasi R = M->firstRelasi;
    while (R != NULL) {
        adrRelasi temp = R;
        R = R->nextRelasi;
        delete temp;
    }

    // Hapus mekanik dari list
    if (prev == NULL) {
        LM.firstMekanik = M->nextMekanik;
    } else {
        prev->nextMekanik = M->nextMekanik;
    }

    delete M;
    cout << "Mekanik dengan ID " << IDMekanik << " berhasil dihapus.\n";
```

**Penjelasan elemen mekanik:**

Proses penghapusan mekanik dilakukan dengan mencari **ID Mekanik** pada **List Mekanik**.

Jika mekanik ditemukan, maka node mekanik tersebut akan dihapus dari list. Seluruh relasi yang dimiliki mekanik tersebut juga ikut terhapus karena relasi tersimpan di dalam node mekanik.

e.

**Penjelasan elemen child :**
Penghapusan data servis dilakukan dengan mencari **ID Servis** pada **List Servis**.
Jika servis ditemukan, maka node servis tersebut akan dihapus dari list.
Data servis yang dihapus tidak lagi dapat direlasikan dengan mekanik.

f.

**Penjelasan elemen relasi :**
Penghapusan relasi dilakukan dengan mencari **servis tertentu** pada daftar relasi milik **mekanik tertentu**.
Jika relasi ditemukan, maka hubungan antara mekanik dan servis tersebut akan dihapus tanpa menghapus data mekanik maupun servisnya.
.

g.

```
// g. Find element Parent (Mekanik)
adrMekanik findMekanik(listMekanik LM, string IDMekanik) {
    adrMekanik M = LM.firstMekanik;
    while (M != NULL) {
        if (M->infoMekanik.IDMekanik == IDMekanik) {
            return M;
        }
        M = M->nextMekanik;
    }
    return NULL;
}
```

**Penjelasan Find elemen parent :**
Penghapusan relasi dilakukan dengan mencari **servis tertentu** pada daftar relasi milik **mekanik tertentu**.
Jika relasi ditemukan, maka hubungan antara mekanik dan servis tersebut akan dihapus tanpa menghapus data mekanik maupun servisnya.

h.

```
// h. Find element child (Servis)
adrServis findServis(listServis LS, string IDServis) {
    adrServis S = LS.firstServis;
    while (S != NULL) {
        if (S->infoServis.IDServis == IDServis) {
            return S;
        }
        S = S->nextServis;
    }
    return NULL;
}
```

**Penjelasan find  elemen child :**
Proses pencarian servis dilakukan dengan menelusuri **List Servis** berdasarkan **ID Servis**.
Jika servis ditemukan, maka data tersebut dapat digunakan untuk proses relasi atau penampilan data.

i.

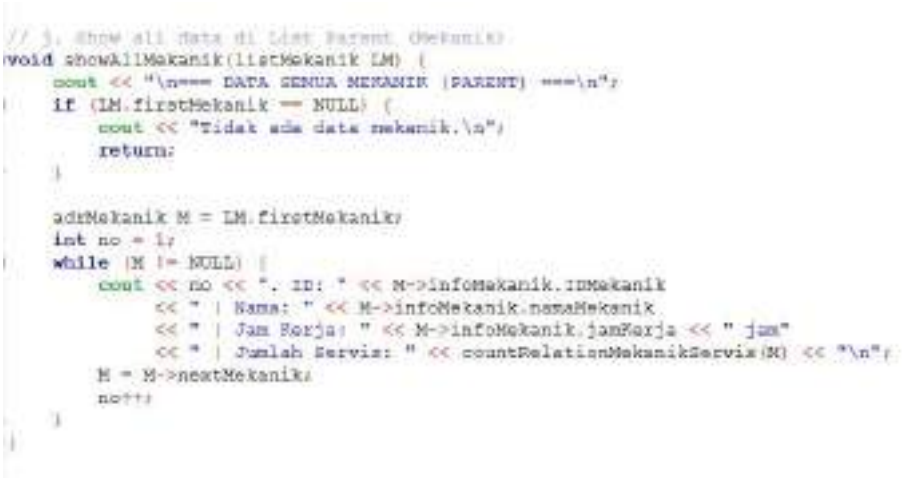
```
// i. Find element relation (Mekanik-Servis)
bool findRelationMekanikServis(adrMekanik mekanik, string IDServis) {
    if (mekanik == NULL) return false;

    adrRelasi R = mekanik->firstRelasi;
    while (R != NULL) {
        if (R->servis->infoServis.IDServis == IDServis) {
            return true;
        }
        R = R->nextRelasi;
    }
    return false;
}
```

**Penjelasan Find element relation:**

Fungsi ini digunakan untuk mengecek apakah seorang mekanik memiliki relasi dengan servis tertentu berdasarkan, **true** jika relasi ditemukan, **false** jika relasi tidak ditemukan

j.

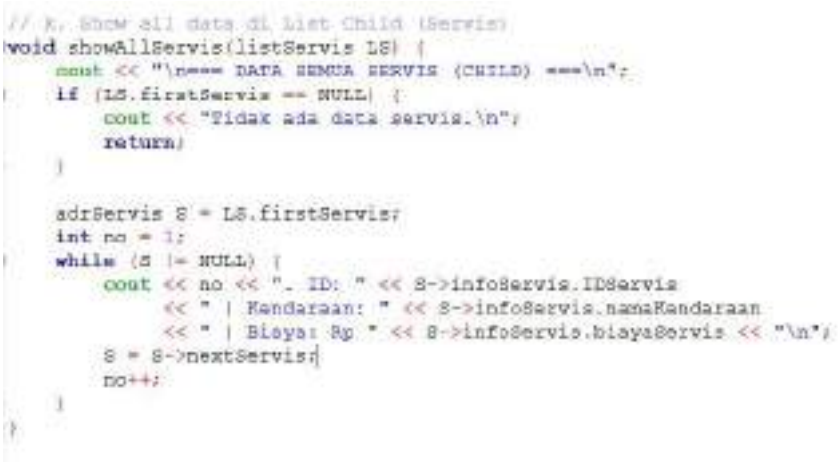


```
// j. Show all data di List Parent (Mekanik)
void showAllMekanik(listMekanik LM) {
    cout << "\n=== DATA SEMUA MEKANIK (PARENT) ===\n";
    if (LM.firstMekanik == NULL) {
        cout << "Tidak ada data mekanik.\n";
        return;
    }

    adrMekanik M = LM.firstMekanik;
    int no = 1;
    while (M != NULL) {
        cout << no << ". ID: " << M->infoMekanik.IDMekanik
             << " | Nama: " << M->infoMekanik.namaMekanik
             << " | Jam Kerja: " << M->infoMekanik.jamKerja << " jam"
             << " | Jumlah Servis: " << countRelasiMekanikServis(M) << "\n";
        M = M->nextMekanik;
        no++;
    }
}
```

**Penjelasan Show data Parent:**
Proses ini menampilkan seluruh data mekanik yang terdapat di dalam **List Mekanik**, termasuk **ID Mekanik, Nama Mekanik, dan Jam Kerja**.

k.

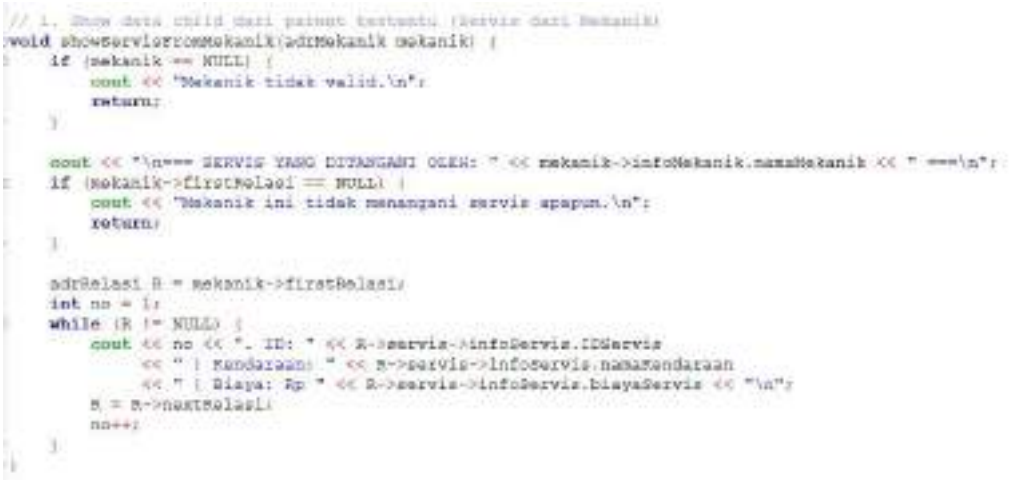


```
// k. Show all data di List Child (Servis)
void showAllServis(listServis LS) {
    cout << "\n=== DATA SEMUA SERVIS (CHILD) ===\n";
    if (LS.firstServis == NULL) {
        cout << "Tidak ada data servis.\n";
        return;
    }

    adrServis S = LS.firstServis;
    int no = 1;
    while (S != NULL) {
        cout << no << ". ID: " << S->infoServis.IDServis
             << " | Kendaraan: " << S->infoServis.namaKendaraan
             << " | Biaya: Rp " << S->infoServis.biayaServis << "\n";
        S = S->nextServis;
        no++;
    }
}
```

**Penjelasan Show all Child:**
Proses ini menampilkan seluruh data servis yang terdapat di dalam **List Servis**, termasuk **ID Servis, Nama Kendaraan, dan Biaya Servis**.

l.




**Penjelasan Show child dari parent:**
Proses ini menampilkan seluruh data servis yang ditangani oleh **mekanik tertentu** dengan cara menelusuri daftar relasi yang dimiliki mekanik tersebut.

m.


**Penjelasan Show setiap data parent beserta data child yang berelasi dengannya:**

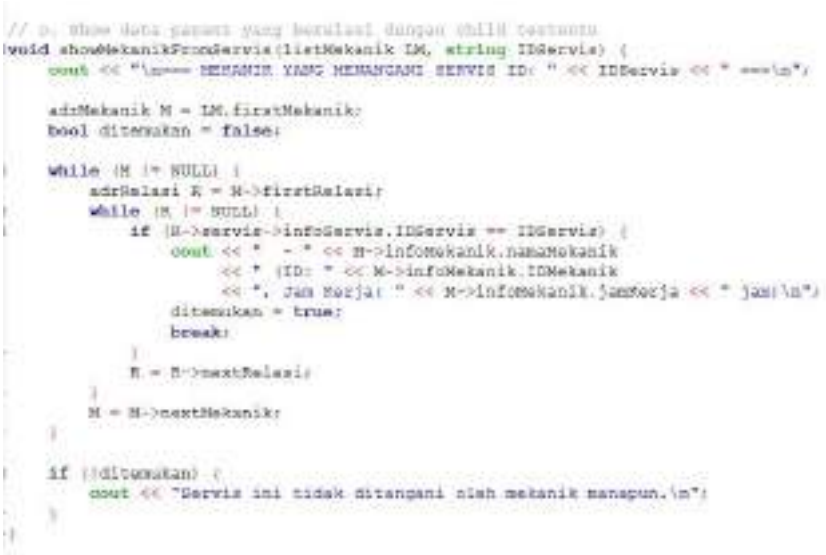
Proses ini menampilkan data mekanik yang memiliki relasi dengan **servis tertentu**.

n.

**Penjelasan Show data child beserta data parent yang masing-masing child miliki:**

Fungsi ini digunakan untuk menampilkan seluruh data servis **(child**) beserta data mekanik (**paren**t) yang menangani masing-masing servis.

o.



**Penjelasan Show data parent yang berelasi dengan child tertentu:**

Fungsi ini digunakan untuk menampilkan data mekanik (**parent**) yang menangani satu servis tertentu (**child**) berdasarkan **IDServis.**

p.

```
// p. Count relation dari setiap element parent
int countRelationMekanikServis(adrMekanik mekanik) {
    if (mekanik == NULL) return 0;

    int count = 0;
    adrRelasi R = mekanik->firstRelasi;
    while (R != NULL) {
        count++;
        R = R->nextRelasi;
    }
    return count;
}
```

**Penjelasan Count relation dari setiap element parent:**

Fungsi ini digunakan untuk menghitung jumlah relasi servis yang dimiliki oleh satu mekanik tertentu.

q.

```
// q. Count relation yang dimiliki oleh child tertentu
int countRelationFromServis(listMekanik LM, string IDServis) {
    int count = 0;

    adrMekanik M = LM.firstMekanik;
    while (M != NULL) {
        adrRelasi R = M->firstRelasi;
        while (R != NULL) {
            if (R->servis->infoServis.IDServis == IDServis) {
                count++;
                break; // break karena sudah ditemukan di mekanik ini
            }
            R = R->nextRelasi;
        }
        M = M->nextMekanik;
    }

    return count;
}
```

**Penjelasan Count relation yang dimiliki oleh child tertentu:**

Fungsi ini digunakan untuk menghitung jumlah relasi mekanik yang berhubungan dengan satu servis tertentu berdasarkan IDServis.

r.

```
// r. Count element child yang tidak memiliki relasi
int countServisWithoutRelation(listMekanik LM, listServis LS) {
    int count = 0;

    adrServis S = LS.firstServis;
    while (S != NULL) {
        if (countRelationFromServis(LM, S->infoServis.IDServis) == 0) {
            count++;
        }
        S = S->nextServis;
    }

    return count;
}
```

**Penjelasan Count element child yang tidak memiliki relasi:**
Tujuannya untuk menghitung jumlah data servis (child) yang tidak memiliki relasi dengan mekanik mana pun.

s.

```
// s. Edit relasi / mengganti child dari parent tertentu
void editRelasiMekanikServis(adrMekanik mekanik, string IDServisLama, string IDServisBaru, listServis LS) {
    if (mekanik == NULL) {
        cout << "Mekanik tidak valid.\n";
        return;
    }

    // Cari apakah servis lama ada di relasi mekanik
    if (!findRelasiMekanikServis(mekanik, IDServisLama)) {
        cout << "Servis lama (" << IDServisLama << ") tidak ditemukan dalam relasi mekanik ini.\n";
        return;
    }

    // Cari apakah servis baru ada di list servis
    adrServis SBaru = findServis(LS, IDServisBaru);
    if (SBaru == NULL) {
        cout << "Servis baru (" << IDServisBaru << ") tidak ditemukan.\n";
        return;
    }

    // Hapus relasi lama
    deleteRelasiMekanikServis(mekanik, IDServisLama);

    // Tambahkan relasi baru
    adrRelasi R = new nodeRelasi;
    R->servis = SBaru;
    R->nextRelasi = NULL;

    if (mekanik->firstRelasi == NULL) {
        mekanik->firstRelasi = R;
    } else {
        adrRelasi curr = mekanik->firstRelasi;
        while (curr->nextRelasi != NULL) {
            curr = curr->nextRelasi;
        }
        curr->nextRelasi = R;
    }

    cout << "Relasi berhasil diedit: " << IDServisLama << " -> " << IDServisBaru << "\n";
}
```

**Penjelasan Edit relasi/mengganti child dari parent tertentu:**
Fungsinya digunakan untuk mengedit relasi servis pada mekanik tertentu, yaitu mengganti servis lama dengan servis baru.

H. **O X : 35%  Yosevin**

- Membuat struktur data Mekanik sebagai parent pada multi linked list.
- Membuat fungsi insert data mekanik ke dalam List Mekanik.
- Membuat fungsi edit dan delete data mekanik.
- Melakukan pengujian fungsi-fungsi mekanik untuk memastikan data dapat ditambahkan, diubah, dan dihapus dengan benar.

**o Y : 35% Naila**

- Membuat struktur data Servis sebagai child pada multi linked list
- Membuat fungsi insert data servis ke dalam List Servis.
- Membuat fungsi edit dan delete data servis.
- Membuat dan mengelola relasi antara mekanik dan servis

**o Z : 30% Fauzan**

- Membuat struktur data Sparepart sebagai child tambahan yang disimpan dalam linked list terpisah.
- Membuat fungsi insert, delete, dan tampil data sparepart.
- Membuat fungsi menampilkan seluruh data mekanik beserta servis yang berelasi.
- Mengintegrasikan seluruh fitur ke dalam menu utama program.

I. Bukti responsi tugas besar bersama asdos, asprak, ataupun dosen

( Muhammad Fauzan Januar tidak hadir dikarenakan sakit)

Link GitHub : https://github.com/yhyosee95/tubes-strukturData