LAPORAN PRAKTIKUM

PERTEMUAN 13

Multi Linked List

Nama :

Resita Istania Purwanto (2311104037)

Dosen :

Yudha Islami Sulistya

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. GUIDED

a. multilist.h

```h
#ifndef MULTILIST_H
#define MULTILIST_H

#include "circularlist.h"  // Memanfaatkan struktur circularlist
#include <iostream>
#include <string>

using namespace std;

// Struktur untuk multi list
Codeium: Refactor | Explain
struct Child {
    string nama;  // Nama entitas child
    Child* next;
};

Codeium: Refactor | Explain
struct Parent {
    string nama;  // Nama entitas parent
    Parent* next;
    Child* firstChild;  // Pointer ke anak pertama
};

// Operasi pada parent dan child
Codeium: Refactor | Explain | ×
void createParent(Parent* &P, string nama);
void createChild(Child* &C, string nama);
void insertChild(Parent* &P, Child* C);
void printMultiList(Parent* P);

#endif
```

b. multilist.cpp

```cpp
#include "multilist.h"

Codeium: Refactor | Explain | Generate Function Comment | ×
void createParent(Parent* &P, string nama) {
    P = new Parent;
    P->nama = nama;
    P->next = nullptr;
    P->firstChild = nullptr;
}

Codeium: Refactor | Explain | Generate Function Comment | ×
void createChild(Child* &C, string nama) {
    C = new Child;
    C->nama = nama;
    C->next = nullptr;
}

Codeium: Refactor | Explain | Generate Function Comment | ×
void insertChild(Parent* &P, Child* C) {
    if (P->firstChild == nullptr) {
        P->firstChild = C;
    } else {
        Child* temp = P->firstChild;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = C;
    }
}

Codeium: Refactor | Explain | Generate Function Comment | ×
void printMultiList(Parent* P) {
    while (P != nullptr) {
        cout << "Parent: " << P->nama << endl;
```

```
        Child* C = P->firstChild;
        while (C != nullptr) {
            cout << "  Child: " << C->nama << endl;
            C = C->next;
        }

        P = P->next;
    }
}
```

c. circularlist.h

```
#ifndef CIRCULARLIST_H
#define CIRCULARLIST_H

#include <string>
using namespace std;

Codeium: Refactor | Explain
struct mahasiswa {
    string nama;
    string nim;
    char jenis_kelamin;
    float ipk;
};

typedef mahasiswa infotype;

Codeium: Refactor | Explain
struct ElmList {
    infotype info;
    ElmList *next;
};

typedef ElmList* address;
typedef struct {
    address first;
} List;

// Fungsi dan prosedur untuk Circular List
Codeium: Refactor | Explain | X
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);
    address createData(string nama, string nim, char jenis_kelamin, float ipk);

    #endif // CIRCULARLIST_H
```

d. circularlist.cpp

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

using namespace std;

Codeium: Refactor | Explain | Generate Function Comment | ×
void createList(List &L) {
    L.first = nullptr;
}

Codeium: Refactor | Explain | Generate Function Comment | ×
address alokasi(infotype x) {
    address P = new ElmList;
    if (P != nullptr) {
        P->info = x;
        P->next = nullptr;
    }
    return P;
}

Codeium: Refactor | Explain | Generate Function Comment | ×
void dealokasi(address &P) {
    delete P;
}

Codeium: Refactor | Explain | Generate Function Comment | ×
void insertFirst(List &L, address P) {
    if (L.first == nullptr) {
        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;
        L.first = P;
    }
}

Codeium: Refactor | Explain | Generate Function Comment | ×
void insertAfter(List &L, address Prec, address P) {
    P->next = Prec->next;
    Prec->next = P;
}

Codeium: Refactor | Explain | Generate Function Comment | ×
void insertLast(List &L, address P) {
    if (L.first == nullptr) {
        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;
    }
}

Codeium: Refactor | Explain | Generate Function Comment | ×
void deleteFirst(List &L, address &P) {
    P = L.first;
```

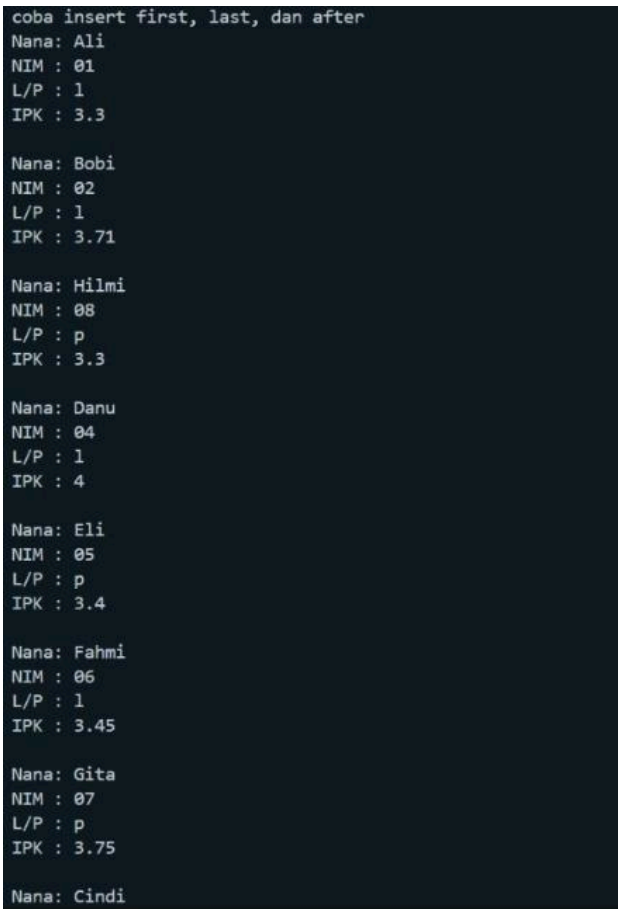
```
    if (P != nullptr) {
        if (P->next == L.first) {
            L.first = nullptr;
        } else {
            address last = L.first;
            while (last->next != L.first) {
                last = last->next;
            }
            L.first = P->next;
            last->next = L.first;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

Codeium: Refactor | Explain | Generate Function Comment | ×
void deleteAfter(List &L, address Prec, address &P) {
    P = Prec->next;
    if (P != nullptr) {
        Prec->next = P->next;
        P->next = nullptr;
    }
}

Codeium: Refactor | Explain | Generate Function Comment | ×
void deleteLast(List &L, address &P) {
    if (L.first != nullptr) {
        address last = L.first;
        while (last->next->next != L.first) {
            last = last->next;
        }
        P = last->next;
        last->next = L.first;
        P->next = nullptr;
    }
}

Codeium: Refactor | Explain | Generate Function Comment | ×
address findElm(List L, infotype x) {
    address P = L.first;
    if (P != nullptr) {
        do {
            if (P->info.nim == x.nim) {
                return P;
            }
            P = P->next;
        } while (P != L.first);
    }
    return nullptr;
}

Codeium: Refactor | Explain | Generate Function Comment | ×
void printInfo(List L) {
    address P = L.first;
    if (P != nullptr) {
        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 << endl;
            P = P->next;
        } while (P != L.first);
    }
}
```

```
Codeium: Refactor | Explain | Generate Function Comment | X
address createData(string nama, string nim, char jenis_kelamin, float ipk) {
    infotype x = {nama, nim, jenis_kelamin, ipk};
    return alokasi(x);
}
```

e. main.cpp

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

using namespace std;

Codeium: Refactor | Explain | Generate Function Comment | X
int main() {
    List L;
    address P1, parag2;
    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", '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;
}
```

Output:



Penjelasan:

- Multilist: Struktur data yang menghubungkan entitas induk dan anak. Tersedia kemampuan untuk membuat orang tua dan anak, serta menambahkan anak ke orang tua.
- Circular list: Mencantumkan struktur data yang elemennya terhubung secara melingkar. Berisi fungsionalitas untuk menambah, menghapus, dan mencari item dalam daftar.
- Dalam fungsi main(), data siswa dimasukkan ke dalam daftar melingkar menggunakan berbagai operasi (insertFirst, insertLast, insertAfter) dan informasi siswa ditampilkan menggunakan printInfo().

II. UNGUIDED

soal 1

a. pegawai.h

```h
#ifndef PEGAWAI_H
#define PEGAWAI_H

#include <string>
using namespace std;

// Struktur untuk Proyek
Codeium: Refactor | Explain
struct Proyek {
    string namaProyek;
    int durasi;
    Proyek* nextProyek;
};

// Struktur untuk Pegawai
Codeium: Refactor | Explain
struct Pegawai {
    string namaPegawai;
    string idPegawai;
    Proyek* proyek;
    Pegawai* nextPegawai;
};

// Fungsi untuk membuat pegawai baru
Codeium: Refactor | Explain | ×
Pegawai* buatPegawai(string nama, string id);

// Fungsi untuk menambahkan proyek ke pegawai
Codeium: Refactor | Explain | ×
void tambahProyek(Pegawai* pegawai, string namaProyek, int durasi);

// Fungsi untuk menghapus proyek dari pegawai
Codeium: Refactor | Explain | ×
void hapusProyek(Pegawai* pegawai, string namaProyek);
```

```h
28  // Fungsi untuk menghapus proyek dari pegawai
    Codeium: Refactor | Explain | ×
29  void hapusProyek(Pegawai* pegawai, string namaProyek);
30
31  // Fungsi untuk menampilkan data pegawai dan proyek mereka
    Codeium: Refactor | Explain | ×
32  void tampilkanPegawai(Pegawai* head);
33
34  #endif
```

b. pegawai.cpp

```cpp
#include "Pegawai.h"
#include <iostream>
using namespace std;


Codeium: Refactor | Explain | Generate Function Comment | ×
Pegawai* buatPegawai(string nama, string id) {
    Pegawai* pegawai = new Pegawai;
    pegawai->namaPegawai = nama;
    pegawai->idPegawai = id;
    pegawai->proyek = nullptr;
    pegawai->nextPegawai = nullptr;
    return pegawai;
}


Codeium: Refactor | Explain | Generate Function Comment | ×
void tambahProyek(Pegawai* pegawai, string namaProyek, int durasi) {
    Proyek* proyek = new Proyek;
    proyek->namaProyek = namaProyek;
    proyek->durasi = durasi;
    proyek->nextProyek = pegawai->proyek;
    pegawai->proyek = proyek;
}


Codeium: Refactor | Explain | Generate Function Comment | ×
void hapusProyek(Pegawai* pegawai, string namaProyek) {
    Proyek* proyek = pegawai->proyek;
    Proyek* prev = nullptr;

    while (proyek != nullptr && proyek->namaProyek != namaProyek) {
        prev = proyek;
        proyek = proyek->nextProyek;
    }

    if (proyek != nullptr) {
        if (prev == nullptr) {
            pegawai->proyek = proyek->nextProyek;
        } else {
            prev->nextProyek = proyek->nextProyek;
        }
        delete proyek;
    }
}


Codeium: Refactor | Explain | Generate Function Comment | ×
void tampilkanPegawai(Pegawai* head) {
    Pegawai* pegawai = head;
    while (pegawai != nullptr) {
        cout << "Nama Pegawai: " << pegawai->namaPegawai << ", ID: " << pegawai->idPegawai << endl;
        Proyek* proyek = pegawai->proyek;
        while (proyek != nullptr) {
            cout << "  - Proyek: " << proyek->namaProyek << ", Durasi: " << proyek->durasi << " bulan" << endl;
            proyek = proyek->nextProyek;
        }
        pegawai = pegawai->nextPegawai;
    }
}

```

c. Main.cpp

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

Codeium: Refactor | Explain
struct Proyek {
    string namaProyek;
    int durasi;
    Proyek* nextProyek;
};

Codeium: Refactor | Explain
struct Pegawai {
    string namaPegawai;
    string idPegawai;
    Proyek* proyek;
    Pegawai* nextPegawai;
};

Codeium: Refactor | Explain | Generate Function Comment | ×
Pegawai* buatPegawai(string nama, string id) {
    Pegawai* pegawai = new Pegawai;
    pegawai->namaPegawai = nama;
    pegawai->idPegawai = id;
    pegawai->proyek = nullptr;
    pegawai->nextPegawai = nullptr;
    return pegawai;
}

Codeium: Refactor | Explain | Generate Function Comment | ×
void tambahProyek(Pegawai* pegawai, string namaProyek, int durasi) {
    Proyek* proyek = new Proyek;
    proyek->namaProyek = namaProyek;
    proyek->durasi = durasi;
```

```cpp
    proyek->nextProyek = pegawai->proyek;
    pegawai->proyek = proyek;
}

Codeium: Refactor | Explain | Generate Function Comment | ×
void hapusProyek(Pegawai* pegawai, string namaProyek) {
    Proyek* proyek = pegawai->proyek;
    Proyek* prev = nullptr;

    while (proyek != nullptr && proyek->namaProyek != namaProyek) {
        prev = proyek;
        proyek = proyek->nextProyek;
    }

    if (proyek != nullptr) {
        if (prev == nullptr) {
            pegawai->proyek = proyek->nextProyek;
        } else {
            prev->nextProyek = proyek->nextProyek;
        }
        delete proyek;
    }
}

Codeium: Refactor | Explain | Generate Function Comment | ×
void tampilkanPegawai(Pegawai* head) {
    Pegawai* pegawai = head;
    while (pegawai != nullptr) {
        cout << "Nama Pegawai: " << pegawai->namaPegawai << ", ID: " << pegawai->idPegawai << endl;
        Proyek* proyek = pegawai->proyek;
        while (proyek != nullptr) {
            cout << "  - Proyek: " << proyek->namaProyek << ", Durasi: " << proyek->durasi << " bulan" << endl;
            proyek = proyek->nextProyek;
```

```
        }
        pegawai = pegawai->nextPegawai;
    }
}

Codeium: Refactor | Explain | Generate Function Comment | ×
int main() {
    Pegawai* head = nullptr;
    Pegawai* andi = buatPegawai("Andi", "P001");
    Pegawai* budi = buatPegawai("Budi", "P002");
    Pegawai* citra = buatPegawai("Citra", "P003");

    head = andi;
    andi->nextPegawai = budi;
    budi->nextPegawai = citra;


    tambahProyek(andi, "Aplikasi Mobile", 12);
    tambahProyek(budi, "Sistem Akuntansi", 8);
    tambahProyek(citra, "E-commerce", 10);


    tambahProyek(andi, "Analisis Data", 6);


    hapusProyek(andi, "Aplikasi Mobile");

    tampilkanPegawai(head);

    return 0;
}
```

Output:

```
Nama Pegawai: Andi, ID: P001
  - Proyek: Analisis Data, Durasi: 6 bulan
Nama Pegawai: Budi, ID: P002
  - Proyek: Sistem Akuntansi, Durasi: 8 bulan
Nama Pegawai: Citra, ID: P003
  - Proyek: E-commerce, Durasi: 10 bulan
```

Penjeasan:

- Pegawai.h (header untuk struktur data pegawai dan proyek)
- Pegawai.cpp (implementasi fungsi-fungsi terkait pegawai dan proyek)
- Main.cpp (file utama untuk menjalankan program)

Soal 2

a. Anggota.h

```
#ifndef ANGOTA_H
#define ANGOTA_H

#include <string>
using namespace std;

Codeium: Refactor | Explain
struct Buku {
    string judulBuku;
    string tanggalPengembalian;
    Buku* nextBuku;
};

Codeium: Refactor | Explain
struct Anggota {
    string namaAnggota;
    string idAnggota;
    Buku* buku;
    Anggota* nextAnggota;
};

Anggota* buatAnggota(string nama, string id);

void tambahBuku(Anggota* anggota, string judulBuku, string tanggalPengembalian);

void hapusAnggota(Anggota*& head, string idAnggota);

void tampilkanAnggota(Anggota* head);

#endif
```

b. Anggota.cpp

```cpp
#include "Anggota.h"
#include <iostream>
using namespace std;
Codeium: Refactor | Explain | Generate Function Comment | X
Anggota* buatAnggota(string nama, string id) {
    Anggota* anggota = new Anggota;
    anggota->namaAnggota = nama;
    anggota->idAnggota = id;
    anggota->buku = nullptr;
    anggota->nextAnggota = nullptr;
    return anggota;
}

Codeium: Refactor | Explain | Generate Function Comment | X
void tambahBuku(Anggota* anggota, string judulBuku, string tanggalPengembalian) {
    Buku* buku = new Buku;
    buku->judulBuku = judulBuku;
    buku->tanggalPengembalian = tanggalPengembalian;
    buku->nextBuku = anggota->buku;
    anggota->buku = buku;
}

Codeium: Refactor | Explain | Generate Function Comment | X
void hapusAnggota(Anggota*& head, string idAnggota) {
    Anggota* anggota = head;
    Anggota* prev = nullptr;

    while (anggota != nullptr && anggota->idAnggota != idAnggota) {
        prev = anggota;
        anggota = anggota->nextAnggota;
    }

    if (anggota != nullptr) {
        if (prev == nullptr) {
            head = anggota->nextAnggota;
        } else {
            prev->nextAnggota = anggota->nextAnggota;
        }

        Buku* buku = anggota->buku;
        while (buku != nullptr) {
            Buku* temp = buku;
            buku = buku->nextBuku;
            delete temp;
        }

        delete anggota;
    }
}
Codeium: Refactor | Explain | Generate Function Comment | X
void tampilkanAnggota(Anggota* head) {
    Anggota* anggota = head;
    while (anggota != nullptr) {
        cout << "Nama Anggota: " << anggota->namaAnggota << ", ID: " << anggota->idAnggota << endl;
        Buku* buku = anggota->buku;
        while (buku != nullptr) {
            cout << "  - Buku: " << buku->judulBuku << ", Tanggal Pengembalian: " << buku->tanggalPengembalian << endl;
            buku = buku->nextBuku;
        }
        anggota = anggota->nextAnggota;
    }
}
```

c. Main.cpp

```cpp
#include <iostream>
#include "Anggota.h"
using namespace std;

Codeium: Refactor | Explain | Generate Function Comment | ×
int main() {
    // Membuat anggota
    Anggota* head = nullptr;
    Anggota* rani = buatAnggota("Rani", "A001");
    Anggota* dito = buatAnggota("Dito", "A002");
    Anggota* vina = buatAnggota("Vina", "A003");

    head = rani;
    rani->nextAnggota = dito;
    dito->nextAnggota = vina;

    tambahBuku(rani, "Pemrograman C++", "01/12/2024");
    tambahBuku(dito, "Algoritma Pemrograman", "15/12/2024");

    tambahBuku(rani, "Struktur Data", "10/12/2024");

    hapusAnggota(head, "A002");

    tampilkanAnggota(head);

    return 0;
}
```

Output:

```
Nama Anggota: Rani, ID: A001
  - Buku: Struktur Data, Tanggal Pengembalian: 10/12/2024
  - Buku: Pemrograman C++, Tanggal Pengembalian: 01/12/2024
Nama Anggota: Vina, ID: A003
```

Penjelasan:
Anggota.h (header untuk struktur data anggota dan buku)
Anggota.cpp (implementasi fungsi-fungsi terkait anggota dan buku)
Main.cpp (file utama untuk menjalankan program)