LAPORAN PRAKTIKUM

Modul 13

Multi Linked List

Disusun Oleh :

Arzario Irsyad Al Fatih/2211104032

SE 07 2

Asisten Praktikum :

Aldi Putra

Andini Nur Hidayah

Dosen Pengampu :

Wahyu Andi Saputra

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

1. Tujuan

a. Mahasiswa mampu menjelaskan definisi dan konsep dari tree.
b. Mahasiswa mampu menerapkan operasi tambah, menghapus, pada tree.
c. Mahasiswa mampu menerapkan operasi tampil data pada tree.

2. Landasan Teori

a. Mullti Linked List

Multi Linked List adalah salah satu bentuk struktur data yang merupakan kombinasi dari beberapa linked list yang saling terhubung. Struktur ini memungkinkan elemen dari sebuah list induk untuk memiliki hubungan dengan elemen-elemen dari list lain yang disebut sebagai list anak. Dalam penerapannya, Multi Linked List digunakan untuk merepresentasikan data yang memiliki hubungan hierarkis, seperti data pegawai dengan daftar tugas mereka atau data pelanggan dengan pesanan mereka.

Operasi dasar dalam Multi Linked List meliputi:

- Insert (penyisipan): Proses menambahkan elemen baru ke dalam list induk atau list anak. Untuk menyisipkan elemen anak, diperlukan referensi elemen induknya.
- Delete (penghapusan): Proses menghapus elemen dari list, baik pada list induk maupun list anak. Dalam penghapusan elemen induk, seluruh elemen anak yang terkait juga akan dihapus.

Implementasi Multi Linked List

Multi Linked List diimplementasikan dengan memanfaatkan pointer untuk menghubungkan elemen-elemen dalam list induk dan anak. List ini digunakan pada berbagai kasus seperti manajemen hierarki data, pengelolaan data multidimensi, dan pembuatan representasi grafik berbasis node.

3. Guided

a. Guided 1

Source Code

```
#include <iostream>
#include <string>

using namespace std;

struct Node {
    int data;
    Node* next;
    Node* child;

    Node(int val) : data(val), next(nullptr), child(nullptr) {}
};

class MultiLinkedList {
private:
    Node* head;

public:
    MultiLinkedList() : head(nullptr) {}


    void addParent(int data) {
        Node* newNode = new Node(data);
        newNode->next = head;
        head = newNode;
    }


    void addChild(int parentData, int childData) {
        Node* parent = head;
        while (parent != nullptr && parent->data != parentData) {
            parent = parent->next;
        }
        if (parent != nullptr) {
            Node* newChild = new Node(childData);
            newChild->next = parent->child;
```

```
            parent->child = newChild;
        } else {
            cout << "Parent not found!" << endl;
        }
    }


    void display() {
        Node* current = head;
        while (current != nullptr) {
            cout << "Parent: " << current->data << " -> ";
            Node* child = current->child;
            while (child != nullptr) {
                cout << child->data << " ";
                child = child->next;
            }
            cout << endl;
            current = current->next;
        }
    }

    ~MultiLinkedList() {

        while (head != nullptr) {
            Node* temp = head;
            head = head->next;


            while (temp->child != nullptr) {
                Node* childTemp = temp->child;
                temp->child = temp->child->next;
                delete childTemp;
            }
            delete temp;
        }
    }
};

int main() {
    MultiLinkedList mList;
```

```
    mList.addParent(1);
    mList.addParent(2);
    mList.addParent(3);

    mList.addChild(1, 10);
    mList.addChild(1, 11);
    mList.addChild(2, 20);
    mList.addChild(2, 20);
    mList.addChild(3, 30);
    mList.addChild(3, 30);
    mList.display();

    return 0;
}
```

Output

```
PS C:\Users\toshiba\Documents\Tugas Kuliah\Semester 5\Praktikum\
rio_Irsyad_Al_Fatih_2211104032\10_Graf\GUIDED\" ; if ($?) { g++
Parent: 3 -> 30 30
Parent: 2 -> 20 20
Parent: 1 -> 11 10
```

Deskripsi

Kode diatas mengimplementasikan daftar berantai multi-level, di mana setiap node (parent) bisa memiliki anak. Kelas MultiLinkedList memungkinkan penambahan parent dan anak melalui fungsi addParent dan addChild, serta menampilkan struktur daftar dengan display. Setiap node memiliki pointer next untuk node berikutnya dan pointer child untuk anak pertama. Destruktor membersihkan memori dengan menghapus semua node parent dan child. Namun, terdapat sedikit kesalahan dalam cara menambahkan anak yang perlu diperbaiki agar struktur anak terhubung dengan benar.

b. Guided 2

Source Code

```
#include <iostream>
#include <string>
```

```
using namespace std;

struct EmployeeNode {
    string name;
    EmployeeNode* next;
    EmployeeNode* subordinate;

    EmployeeNode(string empName) : name(empName), next(nullptr),
subordinate(nullptr) {}
};

class EmployeeList {
private:
    EmployeeNode* head;

public:
    EmployeeList() : head(nullptr) {}


    void addEmployee(string name) {
        EmployeeNode* newEmployee = new EmployeeNode(name);
        newEmployee->next = head;
        head = newEmployee;
    }


    void addSubordinate(string managerName, string subordinateName)
{
        EmployeeNode* manager = head;
        while (manager != nullptr && manager->name != managerName) {
            manager = manager->next;
        }
        if (manager != nullptr) {
            EmployeeNode* newSubordinate = new
EmployeeNode(subordinateName);
            newSubordinate->next = manager->subordinate;
            manager->subordinate = newSubordinate;
        } else {
            cout << "Manager not found!" << endl;
        }
```

```
    }


    void display() {
        EmployeeNode* current = head;
        while (current != nullptr) {
            cout << "Manager: " << current->name << " -> ";
            EmployeeNode* sub = current->subordinate;
            while (sub != nullptr) {
                cout << sub->name << " ";
                sub = sub->next;
            }
            cout << endl;
            current = current->next;
        }
    }

    ~EmployeeList() {

        while (head != nullptr) {
            EmployeeNode* temp = head;
            head = head->next;


            while (temp->subordinate != nullptr) {
                EmployeeNode* subTemp = temp->subordinate;
                temp->subordinate = temp->subordinate->next;
                delete subTemp;
            }
            delete temp;
        }
    }
};

int main() {
    EmployeeList empList;

    empList.addEmployee("Alice");
    empList.addEmployee("Bob");
    empList.addEmployee("Charlie");
```

```
    empList.addSubordinate("Alice", "David");
    empList.addSubordinate("Alice", "Eve");
    empList.addSubordinate("Bob", "Frank");

    empList.addSubordinate("Charlie", "Frans");
    empList.addSubordinate("Charlie", "Brian");

    empList.display();

    return 0;
}
```

Output

```
PS C:\Users\toshiba\Documents\Tugas Kuliah\Semester 5\Praktikum
rio_Irsyad_Al_Fatih_2211104032\10_Graf\GUIDED\" ; if ($?) { g++
Manager: Charlie -> Brian Frans
Manager: Bob -> Frank
Manager: Alice -> Eve David
```

Deskripsi

Program C++ ini membuat daftar pegawai di mana setiap pegawai (manager) bisa memiliki bawahan. Dengan kelas EmployeeList, kita bisa menambah pegawai menggunakan addEmployee dan menambahkan bawahan ke manager dengan addSubordinate. Fungsi display menampilkan struktur manajer dan bawahannya. Destruktor membersihkan memori setelah program selesai. Program ini berguna untuk mengelola data pegawai dalam bentuk hirarki.

c. Guided 3

Source Code

```
#include <iostream>
#include <string>

using namespace std;

// Struktur untuk node karyawan
struct EmployeeNode {
    string name; // Nama karyawan
```

```
    EmployeeNode* next; // Pointer ke karyawan berikutnya
    EmployeeNode* subordinate; // Pointer ke subordinate pertama

    EmployeeNode(string empName) : name(empName), next(nullptr),
subordinate(nullptr) {}
};

// Kelas untuk Multi-Linked List Karyawan
class EmployeeList {
private:
    EmployeeNode* head; // Pointer ke kepala list

public:
    EmployeeList() : head(nullptr) {}

    // Menambahkan karyawan (induk)
    void addEmployee(string name) {
        EmployeeNode* newEmployee = new EmployeeNode(name);
        newEmployee->next = head; // Menyambungkan ke karyawan
sebelumnya
        head = newEmployee; // Memperbarui head
    }

    // Menambahkan subordinate ke karyawan tertentu
    void addSubordinate(string managerName, string subordinateName)
{
        EmployeeNode* manager = head;
        while (manager != nullptr && manager->name != managerName) {
            manager = manager->next;
        }
        if (manager != nullptr) { // Jika manajer ditemukan
            EmployeeNode* newSubordinate = new
EmployeeNode(subordinateName);
            newSubordinate->next = manager->subordinate; //
Menyambungkan ke subordinate sebelumnya
            manager->subordinate = newSubordinate; // Memperbarui
subordinate
        } else {
            cout << "Manager not found!" << endl;
        }
```

```
    }

    // Menghapus karyawan (induk)
    void deleteEmployee(string name) {
        EmployeeNode** current = &head;
        while (*current != nullptr && (*current)->name != name) {
            current = &((*current)->next);
        }

        if (*current != nullptr) { // Jika karyawan ditemukan
            EmployeeNode* toDelete = *current;
            *current = (*current)->next;

            // Hapus semua subordinate dari node ini
            while (toDelete->subordinate != nullptr) {
                EmployeeNode* subTemp = toDelete->subordinate;
                toDelete->subordinate = toDelete->subordinate->next;
                delete subTemp;
            }
            delete toDelete;
            cout << "Employee " << name << " deleted." << endl;
        } else {
            cout << "Employee not found!" << endl;
        }
    }

    // Menghapus subordinate dari karyawan tertentu
    void deleteSubordinate(string managerName, string
subordinateName) {
        EmployeeNode* manager = head;
        while (manager != nullptr && manager->name != managerName) {
            manager = manager->next;
        }

        if (manager != nullptr) { // Jika manajer ditemukan
            EmployeeNode** currentSub = &(manager->subordinate);
            while (*currentSub != nullptr && (*currentSub)->name !=
subordinateName) {
                currentSub = &((*currentSub)->next);
            }
```

```
            if (*currentSub != nullptr) { // Jika subordinate
ditemukan
                EmployeeNode* toDelete = *currentSub;
                *currentSub = (*currentSub)->next; // Menghapus dari
list

                delete toDelete; // Menghapus node subordinate
                cout << "Subordinate " << subordinateName << "
deleted from " << managerName << "." << endl;
            } else {
                cout << "Subordinate not found!" << endl;
            }
        } else {
            cout << "Manager not found!" << endl;
        }
    }

    // Menampilkan daftar karyawan dan subordinate mereka
    void display() {
        EmployeeNode* current = head;
        while (current != nullptr) {
            cout << "Manager: " << current->name << " -> ";
            EmployeeNode* sub = current->subordinate;
            while (sub != nullptr) {
                cout << sub->name << " ";
                sub = sub->next;
            }
            cout << endl;
            current = current->next;
        }
    }

    ~EmployeeList() {
        // Destructor untuk membersihkan memori
        while (head != nullptr) {
            EmployeeNode* temp = head;
            head = head->next;

            // Hapus semua subordinate dari node ini
```

```
            while (temp->subordinate != nullptr) {
                EmployeeNode* subTemp = temp->subordinate;
                temp->subordinate = temp->subordinate->next;
                delete subTemp;
            }
            delete temp;
        }
    }
};

int main() {
    EmployeeList empList;

    empList.addEmployee("Alice");
    empList.addEmployee("Bob");
    empList.addEmployee("Charlie");

    empList.addSubordinate("Alice", "David");
    empList.addSubordinate("Alice", "Eve");
    empList.addSubordinate("Bob", "Frank");

    cout << "Initial employee list:" << endl;
    empList.display(); // Menampilkan isi daftar karyawan

    empList.deleteSubordinate("Alice", "David"); // Menghapus David
dari Alice
    empList.deleteEmployee("Charlie"); // Menghapus Charlie

    cout << "\nUpdated employee list:" << endl;
    empList.display(); // Menampilkan isi daftar setelah penghapusan

    return 0;
}
```

Output

```
PS C:\Users\toshiba\Documents\Tugas Kuliah\Semester 5\Praktikum\
rio_Irsyad_Al_Fatih_2211104032\10_Graf\GUIDED\" ; if ($?) { g++
Initial employee list:
Manager: Charlie ->
Manager: Bob -> Frank
Manager: Alice -> Eve David
Subordinate David deleted from Alice.
Employee Charlie deleted.

Updated employee list:
Manager: Bob -> Frank
Manager: Alice -> Eve
```

Deskripsi

Program C++ ini mengelola daftar karyawan dengan menggunakan struktur multi-linked list. Setiap karyawan dapat memiliki bawahan yang disimpan dalam pointer subordinate. Fungsi-fungsi seperti addEmployee, addSubordinate, deleteEmployee, dan deleteSubordinate memungkinkan penambahan dan penghapusan karyawan serta bawahan. Fungsi display menampilkan daftar karyawan beserta bawahan mereka. Program ini juga memastikan pengelolaan memori yang baik melalui destruktor.

4. Unguided

a. Unguided 1

Source Code

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

struct Proyek
{
    string namaProyek;
    int durasi;
    Proyek *next;
};

struct Pegawai
{
    string namaPegawai;
```

```
    string idPegawai;
    Proyek *proyekHead;
    Pegawai *next;
};

class ManajemenData
{
private:
    Pegawai *head;

public:
    ManajemenData() : head(nullptr) {}

    void tambahPegawai(const string &nama, const string &id)
    {
        Pegawai *pegawaiBaru = new Pegawai{nama, id, nullptr, head};
        head = pegawaiBaru;
    }

    void tambahProyek(const string &idPegawai, const string
&namaProyek, int durasi)
    {
        Pegawai *pegawai = cariPegawai(idPegawai);
        if (pegawai)
        {
            Proyek *proyekBaru = new Proyek{namaProyek, durasi,
pegawai->proyekHead};
            pegawai->proyekHead = proyekBaru;
        }
        else
        {
            cout << "Pegawai dengan ID " << idPegawai << " tidak
ditemukan.\n";
        }
    }

    void hapusProyek(const string &idPegawai, const string
&namaProyek)
    {
        Pegawai *pegawai = cariPegawai(idPegawai);
```

```
        if (pegawai)
        {
            Proyek *prev = nullptr;
            Proyek *current = pegawai->proyekHead;

            while (current && current->namaProyek != namaProyek)
            {
                prev = current;
                current = current->next;
            }

            if (current)
            {
                if (prev)
                {
                    prev->next = current->next;
                }
                else
                {
                    pegawai->proyekHead = current->next;
                }
                delete current;
                cout << "Proyek " << namaProyek << " berhasil
dihapus dari pegawai " << idPegawai << ".\n";
            }
            else
            {
                cout << "Proyek " << namaProyek << " tidak ditemukan
pada pegawai " << idPegawai << ".\n";
            }
        }
        else
        {
            cout << "Pegawai dengan ID " << idPegawai << " tidak
ditemukan.\n";
        }
    }

    void tampilkanData()
    {
```

```
        Pegawai *currentPegawai = head;
        cout << left << setw(20) << "Nama Pegawai" << setw(10) <<
"ID" << "Proyek dan Durasi" << endl;
        cout << string(50, '-') << endl;
        while (currentPegawai)
        {
            cout << left << setw(20) << currentPegawai->namaPegawai
<< setw(10) << currentPegawai->idPegawai;
            Proyek *currentProyek = currentPegawai->proyekHead;
            if (!currentProyek)
            {
                cout << "Tidak ada proyek";
            }
            cout << endl;
            while (currentProyek)
            {
                cout << right << setw(30) << "- " << currentProyek-
>namaProyek << " (" << currentProyek->durasi << " bulan)\n";
                currentProyek = currentProyek->next;
            }
            currentPegawai = currentPegawai->next;
            cout << endl;
        }
    }

private:
    Pegawai *cariPegawai(const string &idPegawai)
    {
        Pegawai *current = head;
        while (current)
        {
            if (current->idPegawai == idPegawai)
            {
                return current;
            }
            current = current->next;
        }
        return nullptr;
    }
};
```

```cpp
int main()
{
    ManajemenData manajemen;

    // Tambahkan data pegawai
    manajemen.tambahPegawai("Andi", "P001");
    manajemen.tambahPegawai("Budi", "P002");
    manajemen.tambahPegawai("Citra", "P003");

    // Tambahkan proyek ke pegawai
    manajemen.tambahProyek("P001", "Aplikasi Mobile", 12);
    manajemen.tambahProyek("P002", "Sistem Akuntansi", 8);
    manajemen.tambahProyek("P003", "E-commerce", 10);

    // Tambahkan proyek baru
    manajemen.tambahProyek("P001", "Analisis Data", 6);

    // Hapus proyek "Aplikasi Mobile" dari Andi
    manajemen.hapusProyek("P001", "Aplikasi Mobile");

    // Tampilkan data pegawai dan proyek mereka
    manajemen.tampilkanData();

    return 0;
}
```

Output

```
PS C:\Users\toshiba\Documents\Tugas Kuliah\Semester 5\Praktikum\ST
rio_Irsyad_Al_Fatih_2211104032\10_Graf\UNGUIDED\" ; if ($?) { g++
Proyek Aplikasi Mobile berhasil dihapus dari pegawai P001.
Nama Pegawai        ID        Proyek dan Durasi
-----------------------------------------------
Citra               P003
                            - E-commerce (10 bulan)

Budi                P002
                            - Sistem Akuntansi (8 bulan)

Andi                P001
                            - Analisis Data (6 bulan)
```

Deskripsi

Program C++ ini mengelola data pegawai dan proyek menggunakan linked list. Setiap pegawai memiliki daftar proyek yang dikerjakan. Fungsi utama mencakup penambahan pegawai dan proyek, penghapusan proyek, serta menampilkan seluruh data pegawai dan proyek mereka. Program ini memungkinkan pengelolaan data pegawai secara dinamis dengan menambahkan atau menghapus proyek sesuai kebutuhan.

b. Unguided 2

Source Code

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

struct Book
{
    string title;
    string returnDate;
    Book *nextBook;
};

struct Member
{
    string name;
    string id;
    Book *borrowedBooks;
    Member *nextMember;
};

void addMember(Member *&head, const string &name, const string &id)
{
    Member *newMember = new Member{name, id, nullptr, nullptr};
    if (!head)
    {
        head = newMember;
    }
    else
    {
```

```
        Member *temp = head;
        while (temp->nextMember)
        {
            temp = temp->nextMember;
        }
        temp->nextMember = newMember;
    }
}

void addBook(Member *head, const string &memberId, const string
&title, const string &returnDate)
{
    Member *temp = head;
    while (temp)
    {
        if (temp->id == memberId)
        {
            Book *newBook = new Book{title, returnDate, nullptr};
            if (!temp->borrowedBooks)
            {
                temp->borrowedBooks = newBook;
            }
            else
            {
                Book *bookTemp = temp->borrowedBooks;
                while (bookTemp->nextBook)
                {
                    bookTemp = bookTemp->nextBook;
                }
                bookTemp->nextBook = newBook;
            }
            return;
        }
        temp = temp->nextMember;
    }
    cout << "Member with ID " << memberId << " not found.\n";
}

void removeMember(Member *&head, const string &memberId)
{
```

```
    Member *temp = head;
    Member *prev = nullptr;

    while (temp)
    {
        if (temp->id == memberId)
        {
            if (prev)
            {
                prev->nextMember = temp->nextMember;
            }
            else
            {
                head = temp->nextMember;
            }

            Book *bookTemp = temp->borrowedBooks;
            while (bookTemp)
            {
                Book *toDelete = bookTemp;
                bookTemp = bookTemp->nextBook;
                delete toDelete;
            }

            delete temp;
            cout << "Member with ID " << memberId << " has been
removed.\n";
            return;
        }
        prev = temp;
        temp = temp->nextMember;
    }
    cout << "Member with ID " << memberId << " not found.\n";
}

void displayAllMembers(Member *head)
{
    Member *temp = head;
    while (temp)
    {
```

```
        cout << "Name: " << temp->name << ", ID: " << temp->id <<
"\n";
        Book *bookTemp = temp->borrowedBooks;
        while (bookTemp)
        {
            cout << "  Book Title: " << bookTemp->title << ", Return
Date: " << bookTemp->returnDate << "\n";
            bookTemp = bookTemp->nextBook;
        }
        temp = temp->nextMember;
        cout << "----------------------\n";
    }
}

int main()
{
    Member *library = nullptr;

    // 1. Masukkan data anggota
    addMember(library, "Rani", "A001");
    addMember(library, "Dito", "A002");
    addMember(library, "Vina", "A003");

    // 2. Tambahkan buku yang dipinjam
    addBook(library, "A001", "Pemrograman C++", "01/12/2024");
    addBook(library, "A002", "Algoritma Pemrograman", "15/12/2024");

    // 3. Tambahkan buku baru
    addBook(library, "A001", "Struktur Data", "10/12/2024");

    // 4. Hapus anggota Dito beserta buku yang dipinjam
    removeMember(library, "A002");

    // 5. Tampilkan seluruh data anggota dan buku yang dipinjam
    displayAllMembers(library);

    return 0;
}
```

Output

```
PS C:\Users\toshiba\Documents\Tugas Kuliah\Semester 5\Praktikum\ST
rio_Irsyad_Al_Fatih_2211104032\10_Graf\UNGUIDED\" ; if ($?) { g++
Member with ID A002 has been removed.
Name: Rani, ID: A001
  Book Title: Pemrograman C++, Return Date: 01/12/2024
  Book Title: Struktur Data, Return Date: 10/12/2024
----------------------
Name: Vina, ID: A003
----------------------
```

Deskripsi

Program C++ ini mengelola data anggota perpustakaan dan buku yang dipinjam menggunakan linked list. Setiap anggota memiliki nama, ID, dan daftar buku yang dipinjam. Fungsi-fungsi utama yang ada antara lain untuk menambah anggota dengan addMember, menambahkan buku yang dipinjam dengan addBook, menghapus anggota beserta buku yang dipinjamnya dengan removeMember, dan menampilkan seluruh data anggota beserta buku yang dipinjam melalui displayAllMembers. Program ini memungkinkan pengelolaan data anggota perpustakaan secara dinamis dan fleksibel.