LAPORAN PRAKTIKUM

Modul 13

Multi Linked List

Disusun Oleh:

Yogi Hafidh Maulana - 2211104061

SE06-02

Dosen :

Wahyu Andi Saputra, S.Pd.,M.Eng.

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

A. Tujuan

- Memahami konsep dasar Multi Linked List sebagai kumpulan list yang saling terhubung.
- Mampu mengimplementasikan penyisipan (insert) dan penghapusan (delete) elemen pada Multi Linked List.
- Mempraktikkan penggunaan Multi Linked List untuk menyelesaikan berbagai studi kasus dalam pengelolaan data terstruktur.

B. Landasan Teori

- Pengertian Multi 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 Multi Linked List

  Operasi dasar dalam Multi Linked List meliputi:

  a. Insert (penyisipan): Proses menambahkan elemen baru ke dalam list induk atau list anak. Untuk menyisipkan elemen anak, diperlukan referensi elemen induknya.
  b. 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.

C. Guided

1) Guided 1

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:

Deskripsi Program:

Code di atas adalah implementasi sederhana dari Multi Linked List, sebuah struktur data di mana setiap node induk (parent) dapat memiliki daftar node anak (child). Struktur ini dibuat dengan kelas MultiLinkedList yang berisi pointer ke node induk pertama (head). Metode addParent menambahkan node induk baru di awal list, sementara addChild mencari node induk berdasarkan data yang diberikan, lalu menambahkan node anak baru di awal daftar anak dari induk tersebut. Metode display digunakan untuk mencetak semua data induk beserta daftar anaknya, dimulai dari node induk pertama. Pada bagian main, tiga node induk (1, 2, 3) ditambahkan, masing-masing dengan beberapa anak, kemudian seluruh struktur dicetak ke layar. Kode ini juga memiliki destructor untuk memastikan semua node induk dan anak dihapus dari memori saat program selesai. Algoritmanya bekerja dengan traversing (menelusuri) list menggunakan pointer, baik untuk induk maupun anak, hingga elemen yang dicari ditemukan atau semua elemen diproses.

2) Guided 2

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:

```
Manager: Charlie -> Brian Frans
Manager: Bob -> Frank
Manager: Alice -> Eve David
PS D:\PROJECT\C++ Project\Modul 10>
```

Deskripsi Program:

Code di atas adalah implementasi sederhana dari struktur data Multi Linked List untuk manajemen hierarki karyawan, di mana setiap node induk (manager) dapat memiliki daftar anak (subordinate). Kelas EmployeeList digunakan untuk membuat daftar karyawan, dengan metode addEmployee untuk menambahkan karyawan baru sebagai manager di awal list. Metode addSubordinate memungkinkan menambahkan bawahan ke manager tertentu dengan cara mencari node manager berdasarkan namanya, lalu menyisipkan node bawahan di awal daftar bawahan manager tersebut. Metode display mencetak daftar semua manager beserta bawahannya. Program ini juga memiliki destructor untuk menghapus semua node manager dan bawahan dari memori. Dalam main, beberapa manager (Alice, Bob, Charlie) dan bawahan mereka ditambahkan, lalu hierarki karyawan dicetak. Algoritma bekerja dengan traversing (menelusuri) list menggunakan pointer, baik untuk mencari manager maupun menampilkan daftar bawahannya.

3) Guided 3

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:

```
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
PS D:\PROJECT\C++ Project\Modul 10>
```

Deskripsi Program:

Code di atas merupakan implementasi dari Multi Linked List untuk mengelola hierarki karyawan dan bawahan. Struktur data ini menggunakan EmployeeNode, di mana setiap node menyimpan nama karyawan, pointer ke karyawan berikutnya, dan pointer ke daftar bawahan. Kelas EmployeeList menyediakan metode untuk menambahkan karyawan baru (addEmployee) di awal daftar, menambahkan bawahan ke karyawan tertentu (addSubordinate), menghapus karyawan beserta semua bawahannya (deleteEmployee), dan menghapus bawahan spesifik dari karyawan tertentu (deleteSubordinate). Metode display menampilkan semua karyawan beserta bawahan mereka, sementara destructor memastikan semua memori yang digunakan dibebaskan. Di fungsi main, beberapa karyawan (Alice, Bob, Charlie) dan bawahannya ditambahkan, kemudian daftar karyawan ditampilkan. Selanjutnya, David dihapus sebagai bawahan Alice, dan Charlie dihapus sepenuhnya dari daftar. Akhirnya, daftar diperbarui dan ditampilkan kembali. Algoritmanya bekerja dengan mencari elemen yang sesuai menggunakan pointer, lalu memperbarui hubungan antar node dengan hati-hati saat menyisipkan atau menghapus data.

D. Unguided

1) Soal 1

Code:

```
#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;
    }
};

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:

```
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 Code:

Program ini menggunakan struktur data Pegawai dan Proyek untuk mengelola data pegawai beserta proyek yang mereka kelola.

Struktur Data:

- Pegawai: Struktur ini menyimpan nama dan ID pegawai, serta pointer menuju linked list proyek yang dikelola pegawai tersebut.
- Proyek: Struktur ini menyimpan nama proyek, durasi proyek (dalam bulan), dan pointer ke proyek berikutnya.

Operasi:

1. tambahPegawai: Fungsi ini menambahkan data pegawai baru ke dalam linked list pegawai. Pegawai baru ditempatkan di awal linked list.
2. tambahProyek: Fungsi ini menambahkan proyek baru ke pegawai berdasarkan ID. Proyek baru akan dimasukkan ke awal linked list proyek milik pegawai.
3. hapusProyek: Fungsi ini mencari proyek tertentu berdasarkan nama di linked list proyek pegawai, lalu menghapusnya jika ditemukan.
4. tampilkanData: Fungsi ini menampilkan seluruh data pegawai dan proyek mereka dalam format tabel yang rapi, dengan kolom nama pegawai, ID, dan

proyek beserta durasinya.

Hasil:
Setelah menjalankan program, daftar pegawai dan proyek mereka ditampilkan dalam bentuk tabel. Tabel tersebut memiliki struktur yang rapi, menampilkan nama pegawai, ID mereka, dan proyek-proyek yang sedang dikelola, termasuk durasi tiap proyek dalam bulan. Jika pegawai tidak memiliki proyek, akan ditampilkan keterangan "Tidak ada proyek."

2) **Soal 2**

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:

```
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
-----------------------
PS D:\PROJECT\C++ Project\Modul 10>
```

Deskripsi Code:

E. Kesimpulan

Konsep Multi Linked List memungkinkan pengelolaan data yang saling terkait dalam bentuk list induk dan list anak. Implementasi struktur data ini melibatkan beberapa operasi penting, seperti menambahkan, menghapus, dan menampilkan elemen baik pada level induk maupun anak. Operasi-operasi tersebut memastikan fleksibilitas dan efisiensi dalam pengelolaan data yang bersifat hierarkis.

Selain itu, penggunaan Multi Linked List mendemonstrasikan pentingnya manajemen memori, seperti alokasi dan dealokasi elemen, serta prosedur pencarian elemen untuk memastikan data dapat diakses dengan tepat. Implementasi yang disajikan dalam bentuk kode, seperti pada file `multilist.h`, memberikan pemahaman praktis tentang cara kerja struktur data ini dalam sistem pemrograman berbasis pointer.