LAPORAN PRAKTIKUM
STRUKTUR DATA
Modul
“Multi Linked List”

Disusun Oleh:
MUHAMMAD RALFI - 2211104054
SE-07-2

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO
2024

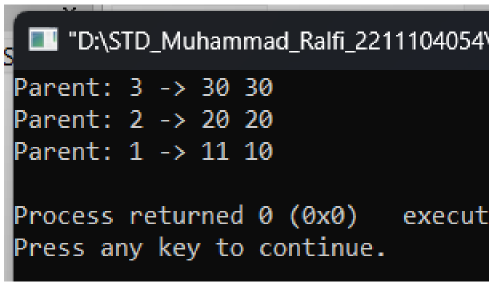
1. **Tujuan**
   a. Mahasiswa dapat memahami konsep Multi Linked List
   b. Mahasiswa mampu mendefinisikann tentang Multi Linked List pada pemrograman
   c. Mahasiswa dapat mengimplementasikan konsep Multi Linked List pada pemrograman
2. **Landasan Teori**
   Multi Linked List

   Multi List adalah struktur data yang terdiri dari beberapa list yang saling terhubung dan memiliki relasi satu sama lain. Setiap elemen dalam Multi List memiliki kemampuan untuk membentuk list independen, dimana struktur ini umumnya mengikuti hierarki dengan adanya list yang berperan sebagai induk (parent list) dan list yang berperan sebagai anak (child list), menciptakan suatu organisasi data yang terstruktur dan saling berelasi.

   Multi List memungkinkan pengelolaan data yang kompleks dimana setiap elemen dapat terhubung dengan multiple elemen lainnya, memfasilitasi pembentukan struktur data yang bercabang namun tetap terhubung secara sistematis melalui konsep parent-child relationship. Organisasi data seperti ini sangat efektif untuk merepresentasikan informasi yang memiliki hierarki atau ketergantungan bertingkat.
3. **Guided**
   a. Guided 1
   File 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



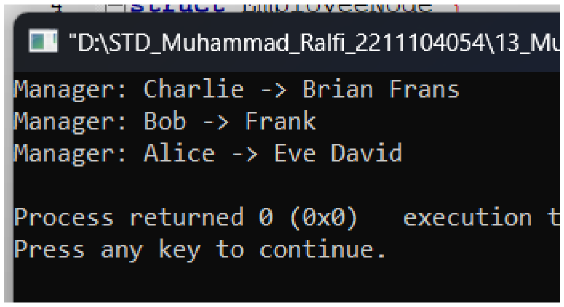
```
"D:\STD_Muhammad_Ralfi_2211104054\
Parent: 3 -> 30 30
Parent: 2 -> 20 20
Parent: 1 -> 11 10

Process returned 0 (0x0)   execut
Press any key to continue.
```

b. Guided 2

File 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



"D:\STD_Muhammad_Ralfi_2211104054\13_Mu

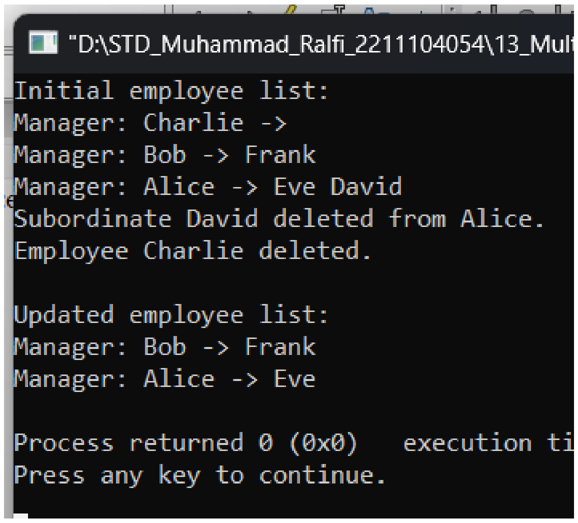
```
Manager: Charlie -> Brian Frans
Manager: Bob -> Frank
Manager: Alice -> Eve David

Process returned 0 (0x0)   execution t
Press any key to continue.
```

c. Guided 3

File 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

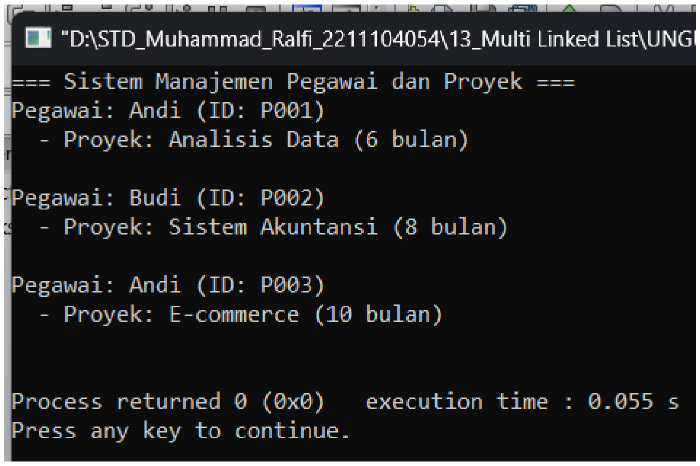


```
"D:\STD_Muhammad_Ralfi_2211104054\13_Mul
Initial employee list:
Manager: Charlie ->
Manager: Bob -> Frank
Manager: Alice -> Eve David
Subordinate David deleted from Alice.
Employee Charlie deleted.

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

Process returned 0 (0x0)   execution ti
Press any key to continue.
```

4. **Unguided**
   a. Unguided 1
      File Code

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

struct Project{
    string name;
    int duration;
    Project* next;

    Project(string n, int d) : name(n), duration(d), next(nullptr) {}
};

struct Employee {
    string name;
    string id;
    Project* projects;
    Employee* next;

    Employee(string n, string i) : name(n), id(i), projects(nullptr),
   next(nullptr) {}
};

// Kelas untuk mengelola sistem pegawai dan proyek
class EmployeeSystem {
private:
    Employee* head;

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

    void addEmployee(string name, string id) {
        Employee* newEmployee = new Employee(name, id);
        if (!head) {
            head = newEmployee;
        } else {
            Employee* current = head;
            while (current->next) {
                current = current->next;
            }
            current->next = newEmployee;
        }
    }

    void addProject(string employeeId, string projectName, int
duration) {
        Employee* emp = findEmployee(employeeId);
        if (emp) {
            Project* newProject = new Project(projectName, duration);
            if (!emp->projects) {
                emp->projects = newProject;
            } else {
                Project* current = emp->projects;
                while (current->next) {
                    current = current->next;
                }
                current->next = newProject;
            }
        }
    }

    void removeProject(string employeeId, string projectName) {
        Employee* emp = findEmployee(employeeId);
        if (emp && emp->projects) {
            Project* current = emp->projects;
            Project* prev = nullptr;

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

            if (current) {
                if (prev) {
                    prev->next = current->next;
                } else {
                    emp->projects = current->next;
                }
                delete current;
            }
        }
    }
    Employee* findEmployee(string id) {
        Employee* current = head;
        while (current && current->id != id) {
            current = current->next;
        }
        return current;
    }

    void displayAll() {
        Employee* current = head;
        while (current) {
            cout << "Pegawai: " << current->name << " (ID: " << current-
>id << ")" << endl;
            Project* proj = current->projects;
            while (proj) {
                cout << "  - Proyek: " << proj->name << " (" << proj-
>duration << " bulan)" << endl;
                proj = proj->next;
            }
            cout << endl;
            current = current->next;
        }
    }
};

int main(){
    cout << "=== Sistem Manajemen Pegawai dan Proyek ===" << endl;
    EmployeeSystem empSystem;
    empSystem.addEmployee("Andi", "P001");
    empSystem.addEmployee("Budi", "P002");
    empSystem.addEmployee("Andi", "P003");

    // Menambah proyek
    empSystem.addProject("P001", "Aplikasi Mobile", 12);
    empSystem.addProject("P002", "Sistem Akuntansi", 8);
    empSystem.addProject("P003", "E-commerce", 10);
    empSystem.addProject("P001", "Analisis Data", 6);

     // Menghapus proyek
    empSystem.removeProject("P001", "Aplikasi Mobile");

    // Menampilkan data
    empSystem.displayAll();
}
```

Output



```
"D:\STD_Muhammad_Ralfi_2211104054\13_Multi Linked List\UNG
=== Sistem Manajemen Pegawai dan Proyek ===
Pegawai: Andi (ID: P001)
  - Proyek: Analisis Data (6 bulan)

Pegawai: Budi (ID: P002)
  - Proyek: Sistem Akuntansi (8 bulan)

Pegawai: Andi (ID: P003)
  - Proyek: E-commerce (10 bulan)


Process returned 0 (0x0)   execution time : 0.055 s
Press any key to continue.
```

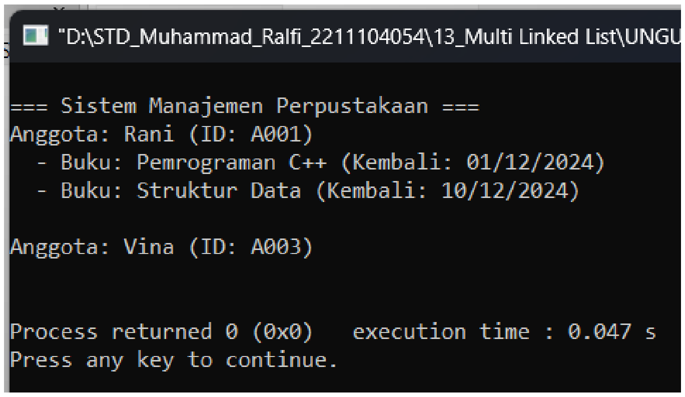
b. Unguided 2

File Code

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

struct Book {
    string title;
    string returnDate;
    Book* next;

    Book(string t, string rd) : title(t), returnDate(rd), next(nullptr)
   {}
};

struct Member {
    string name;
    string id;
    Book* books;
    Member* next;

    Member(string  n,  string  i)  :  name(n),  id(i),  books(nullptr),
   next(nullptr) {}
};

class LibrarySystem {
private:
    Member* head;

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

    void addMember(string name, string id) {
        Member* newMember = new Member(name, id);
        if (!head) {
            head = newMember;
        } else {
            Member* current = head;
            while (current->next) {
                current = current->next;
            }
            current->next = newMember;
        }
    }

    void addBook(string memberId, string title, string returnDate) {
        Member* mem = findMember(memberId);
        if (mem) {
            Book* newBook = new Book(title, returnDate);
            if (!mem->books) {
                mem->books = newBook;
            } else {
                Book* current = mem->books;
                while (current->next) {
                    current = current->next;
                }
                current->next = newBook;
            }
        }
    }

    void removeMember(string id) {
        if (!head) return;

        if (head->id == id) {
            Member* temp = head;
            head = head->next;
            deleteBooks(temp->books);
            delete temp;
            return;
        }

        Member* current = head;
        while (current->next && current->next->id != id) {
            current = current->next;
        }

        if (current->next) {
            Member* temp = current->next;
            current->next = temp->next;
            deleteBooks(temp->books);
            delete temp;
        }
    }
    void deleteBooks(Book* book) {
        while (book) {
            Book* temp = book;
            book = book->next;
            delete temp;
        }
    }

    Member* findMember(string id) {
        Member* current = head;
        while (current && current->id != id) {
            current = current->next;
        }
        return current;
    }

    void displayAll() {
        Member* current = head;
        while (current) {
            cout << "Anggota: " << current->name << " (ID: " << current-
   >id << ")" << endl;
            Book* book = current->books;
            while (book) {
                cout << "  - Buku: " << book->title << " (Kembali: " <<
   book->returnDate << ")" << endl;
                book = book->next;
            }
            cout << endl;
            current = current->next;
        }
    }
};

int main() {
    cout << "\n=== Sistem Manajemen Perpustakaan ===" << endl;
    LibrarySystem libSystem;

    libSystem.addMember("Rani", "A001");
    libSystem.addMember("Dito", "A002");
    libSystem.addMember("Vina", "A003");

    libSystem.addBook("A001", "Pemrograman C++", "01/12/2024");
    libSystem.addBook("A002", "Algoritma Pemrograman", "15/12/2024");
    libSystem.addBook("A001", "Struktur Data", "10/12/2024");

    libSystem.removeMember("A002");

    libSystem.displayAll();

    return 0;
```

```
}
```

Output



"D:\STD_Muhammad_Ralfi_2211104054\13_Multi Linked List\UNGU

```
=== Sistem Manajemen Perpustakaan ===
Anggota: Rani (ID: A001)
  - Buku: Pemrograman C++ (Kembali: 01/12/2024)
  - Buku: Struktur Data (Kembali: 10/12/2024)

Anggota: Vina (ID: A003)


Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

5. Kesimpulan

Implementasi Multi Linked List dalam pengembangan sistem manajemen pegawai dan perpustakaan menunjukkan efektivitas struktur data ini dalam mengelola data yang memiliki relasi kompleks. Melalui penggunaan pointer yang saling terhubung, sistem dapat dengan mudah mengelola data induk (pegawai/anggota) dan data anak (proyek/buku) secara terstruktur, memungkinkan operasi penambahan, penghapusan, dan penelusuran data dilakukan secara efisien tanpa perlu mengubah struktur keseluruhan data

Program yang dikembangkan menggunakan C++ ini membuktikan bahwa Multi Linked List sangat cocok untuk implementasi sistem yang membutuhkan hierarki data, dimana setiap node induk dapat memiliki beberapa node anak yang terkait. Hal ini terlihat dari kemampuan sistem dalam mengelola multiple proyek untuk setiap pegawai dan multiple buku untuk setiap anggota perpustakaan, serta kemudahan dalam melakukan modifikasi data seperti penghapusan proyek atau anggota beserta seluruh data terkaitnya, yang menunjukkan fleksibilitas dan efisiensi struktur data ini dalam pengelolaan data yang saling berhubungan.

Output:

a. Menambahkan node

b. Mengecek BST apakah valid atau tidak

c. Mencari simpul daun

6. Kesimpulan