LAPORAN PRAKTIKUM

MODUL XIII

“MULTI LINKED LIST”

Disusun Oleh:

Alya Rabani - 2311104076

S1SE-07-02

Dosen :

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

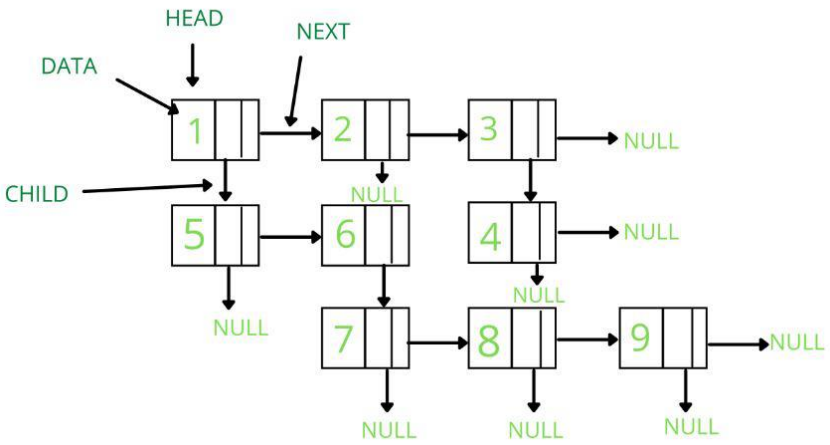
2024

1. Tujuan

- Memahami penggunaan Multi Linked list.
- Mengimplementasikan Multi Linked list dalam beberapa studi kasus.

2. Landasan Teori

Multi Linked List adalah struktur data yang menggunakan konsep linked list, tetapi dengan kemampuan untuk memiliki beberapa hubungan atau koneksi antar elemen, memungkinkan representasi data yang lebih kompleks dibandingkan single linked list



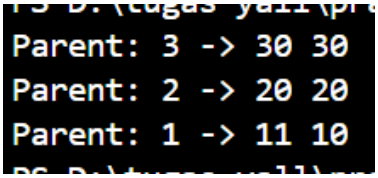
Ciri Utama Multi Linked List, yaitu terdiri dari node utama yang menyimpan data dan pointer ke node lain di level yang sama atau berbeda. Setiap node utama dapat memiliki sub-list yang menyimpan data terkait (contoh: data buku yang dipinjam oleh anggota perpustakaan). Multi linked list juga dapat digunakan untuk merepresentasikan hubungan antar entitas, misalnya anggota dan daftar buku yang dipinjam atau pegawai dan proyek yang dikelola. Terdapat dua tipe pointer, yaitu pointer horizontal yang menghubungkan node utama di tingkat yang sama (misalnya, daftar anggota). Pointer vertical untuk menghubungkan node utama ke sub-list (misalnya, daftar buku yang dipinjam oleh anggota).

Multi Linked List memiliki beberapa kelebihan misalnya, fleksibilitas representasi data dimana cocok untuk hubungan satu-ke-banyak (one-to-many) atau banyak-ke-banyak (many-to-many). Misalnya, satu anggota bisa meminjam banyak buku. Lalu, multi linked list dapat menghemat memori. dibandingkan array multidimensi, multi linked list menggunakan memori secara dinamis, hanya dialokasikan saat dibutuhkan. Kemudian, ia juga mudah dalam memanipulasi data, dapat menambahkan, menghapus, atau mengakses data dalam hubungan kompleks menjadi lebih mudah.

3. Guided

1. Program ini mengimplementasikan sebuah struktur data Multi Linked List. Multi linked list dalam program ini dikelola oleh kelas MultiLinkedList. Kelas ini menyediakan beberapa fungsi utama: addParent untuk menambahkan node baru sebagai parent, addChild untuk menambahkan node baru sebagai child dari suatu parent, dan display untuk menampilkan seluruh struktur data. Fungsi destructor memastikan bahwa semua node dihapus dari memori setelah tidak lagi digunakan. Program ini menciptakan beberapa node parent dengan data 1, 2, dan 3. Kemudian, node-node child dengan data 10, 11, 20, dan 30 ditambahkan sebagai anak dari node-node parent tersebut.

   Output program:



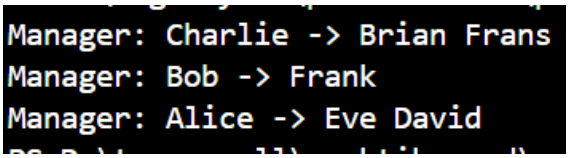
   Code program:

```cpp
#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;
}

```

2. Program ini dirancang untuk mengelola struktur organisasi perusahaan. Menggunakan konsep Multi Linked List, program ini merepresentasikan hubungan antara seorang karyawan sebagai manajer dan karyawan lain yang menjadi bawahannya. Setiap karyawan memiliki informasi nama dan dapat

memiliki beberapa bawahan. Fungsi-fungsi utama dalam program ini meliputi penambahan karyawan baru, penambahan bawahan ke seorang manajer, dan tampilan struktur organisasi secara keseluruhan. Ketika seorang karyawan baru ditambahkan, ia akan ditempatkan di awal daftar. Untuk menambahkan bawahan, program akan mencari manajer yang sesuai dan kemudian menambahkan bawahan tersebut ke daftar bawahan manajer tersebut.

Output program:



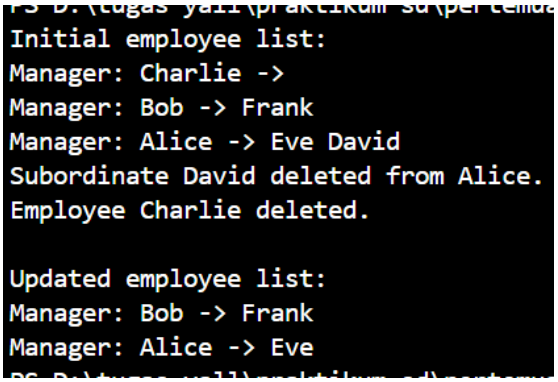
Code program:

```cpp
#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;
}

```

3. Program ini dirancang untuk mengelola struktur organisasi perusahaan secara efisien. Dengan memanfaatkan struktur data Multi-Linked List, program ini mampu merepresentasikan hubungan antara seorang manajer dengan bawahan-bawahannya. Setiap karyawan memiliki informasi nama, dan hubungan antara mereka diwakili oleh pointer. Program ini menyediakan

beberapa fungsi penting untuk mengelola data karyawan. Fungsi addEmployee digunakan untuk menambahkan karyawan baru ke dalam daftar. Fungsi addSubordinate digunakan untuk menambahkan bawahan baru kepada seorang manajer tertentu. Selain itu, program juga menyediakan fungsi untuk menghapus karyawan (deleteEmployee) dan menghapus bawahan (deleteSubordinate). Fungsi display digunakan untuk menampilkan struktur organisasi secara keseluruhan, mulai dari manajer hingga bawahannya.

Output program:



Code program:

```cpp
#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;
}
```

4. Unguided

1. Program ini menggunakan struktur Pegawai untuk menyimpan data pegawai, dan struktur Proyek untuk menyimpan data proyek yang dikelola oleh setiap pegawai. Dengan menggunakan Multi Linked List, setiap pegawai dapat memiliki daftar proyek yang fleksibel.
   Code program:

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

// Struktur data proyek
struct Proyek {
  string namaProyek;
  int durasi;
  Proyek* nextProyek;
};

// Struktur data pegawai
struct Pegawai {
  string namaPegawai;
  string idPegawai;
  Proyek* daftarProyek;
  Pegawai* nextPegawai;
};

// Fungsi untuk membuat pegawai baru
Pegawai* buatPegawai(string nama, string id) {
  Pegawai* pegawaiBaru = new Pegawai;
  pegawaiBaru->namaPegawai = nama;
  pegawaiBaru->idPegawai = id;
  pegawaiBaru->daftarProyek = nullptr;
  pegawaiBaru->nextPegawai = nullptr;
  return pegawaiBaru;
}

// Fungsi untuk menambahkan proyek ke pegawai
void tambahkanProyek(Pegawai* pegawai, string namaProyek, int durasi) {
  Proyek* proyekBaru = new Proyek;
  proyekBaru->namaProyek = namaProyek;
  proyekBaru->durasi = durasi;
  proyekBaru->nextProyek = pegawai->daftarProyek;
  pegawai->daftarProyek = proyekBaru;
}
```

```
// Fungsi untuk menghapus proyek dari pegawai
void hapusProyek(Pegawai* pegawai, string namaProyek) {
  Proyek* curr = pegawai->daftarProyek;
  Proyek* prev = nullptr;

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

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

// Fungsi untuk menampilkan data pegawai dan proyeknya
void tampilkanData(Pegawai* head) {
  Pegawai* currPegawai = head;
  while (currPegawai != nullptr) {
       cout << "Pegawai: " << currPegawai->namaPegawai << " (ID: " <<
currPegawai->idPegawai << ")\n";
    Proyek* currProyek = currPegawai->daftarProyek;
    if (currProyek == nullptr) {
      cout << "  Tidak ada proyek.\n";
    } else {
      while (currProyek != nullptr) {
           cout << "  Proyek: " << currProyek->namaProyek << ", Durasi: " <<
currProyek->durasi << " bulan\n";
        currProyek = currProyek->nextProyek;
      }
    }
    currPegawai = currPegawai->nextPegawai;
    cout << endl;
  }
}

int main() {
  // Membuat daftar pegawai
```

```
    Pegawai* head = buatPegawai("Andi", "P001");
    head->nextPegawai = buatPegawai("Budi", "P002");
    head->nextPegawai->nextPegawai = buatPegawai("Citra", "P003");

    // Menambahkan proyek ke pegawai
    tambahkanProyek(head, "Aplikasi Mobile", 12); // Untuk Andi
    tambahkanProyek(head->nextPegawai, "Sistem Akuntansi", 8); // Untuk Budi
    tambahkanProyek(head->nextPegawai->nextPegawai, "E-commerce", 10); //
Untuk Citra

    // Menambahkan proyek baru ke Andi
    tambahkanProyek(head, "Analisis Data", 6);

    // Menghapus proyek "Aplikasi Mobile" dari Andi
    hapusProyek(head, "Aplikasi Mobile");

    // Menampilkan data pegawai dan proyek mereka
    tampilkanData(head);

    return 0;
}
```

Output dari program:

```
Pegawai: Andi (ID: P001)
  Proyek: Analisis Data, Durasi: 6 bulan

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

Pegawai: Citra (ID: P003)
  Proyek: E-commerce, Durasi: 10 bulan
```

2. Program ini merupakan sistem untuk mengelola data anggota perpustakaan dan buku yang dipinjam. Dengan beberapa fungsi utama seperti, membuat data anggota baru, menambahkan buku yang dipinjam oleh anggota, menghapus anggota beserta buku yang dipinjam, dan mencetak data anggota dan buku yang dipinjam

   Code program:

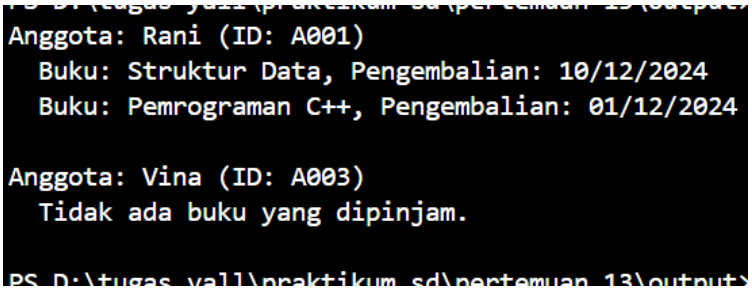
```
#include <iostream>
#include <string>
using namespace std;

// Struktur data buku
struct Buku {
```

```
    string judulBuku;
    string tanggalPengembalian;
    Buku* nextBuku;
};

// Struktur data anggota
struct Anggota {
    string namaAnggota;
    string idAnggota;
    Buku* daftarBuku;
    Anggota* nextAnggota;
};

// Fungsi untuk membuat anggota baru
Anggota* buatAnggota(string nama, string id) {
    Anggota* anggotaBaru = new Anggota;
    anggotaBaru->namaAnggota = nama;
    anggotaBaru->idAnggota = id;
    anggotaBaru->daftarBuku = nullptr;
    anggotaBaru->nextAnggota = nullptr;
    return anggotaBaru;
}

// Fungsi untuk menambahkan buku yang dipinjam oleh anggota
void tambahkanBuku(Anggota* anggota, string judulBuku, string
tanggalPengembalian) {
    Buku* bukuBaru = new Buku;
    bukuBaru->judulBuku = judulBuku;
    bukuBaru->tanggalPengembalian = tanggalPengembalian;
    bukuBaru->nextBuku = anggota->daftarBuku;
    anggota->daftarBuku = bukuBaru;
}

// Fungsi untuk menghapus anggota beserta buku yang dipinjam
Anggota* hapusAnggota(Anggota* head, string idAnggota) {
    Anggota* curr = head;
    Anggota* prev = nullptr;

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

    if (curr != nullptr) {
```

```
    if (prev == nullptr) {
      head = curr->nextAnggota;
    } else {
      prev->nextAnggota = curr->nextAnggota;
    }

    Buku* currBuku = curr->daftarBuku;
    while (currBuku != nullptr) {
      Buku* temp = currBuku;
      currBuku = currBuku->nextBuku;
      delete temp;
    }
    delete curr;
  }
  return head;
}

// Fungsi untuk menampilkan data anggota dan buku yang dipinjam
void tampilkanData(Anggota* head) {
  Anggota* currAnggota = head;
  while (currAnggota != nullptr) {
       cout << "Anggota: " << currAnggota->namaAnggota << " (ID: " <<
currAnggota->idAnggota << ")\n";
    Buku* currBuku = currAnggota->daftarBuku;
    if (currBuku == nullptr) {
      cout << "  Tidak ada buku yang dipinjam.\n";
    } else {
      while (currBuku != nullptr) {
          cout << "  Buku: " << currBuku->judulBuku << ", Pengembalian: " <<
currBuku->tanggalPengembalian << "\n";
        currBuku = currBuku->nextBuku;
      }
    }
    currAnggota = currAnggota->nextAnggota;
    cout << endl;
  }
}

int main() {
  // Membuat daftar anggota
  Anggota* head = buatAnggota("Rani", "A001");
  head->nextAnggota = buatAnggota("Dito", "A002");
  head->nextAnggota->nextAnggota = buatAnggota("Vina", "A003");
```

```
    // Menambahkan buku yang dipinjam
    tambahkanBuku(head, "Pemrograman C++", "01/12/2024"); // Untuk Rani
    tambahkanBuku(head->nextAnggota, "Algoritma Pemrograman",
"15/12/2024"); // Untuk Dito

    // Menambahkan buku baru untuk Rani
    tambahkanBuku(head, "Struktur Data", "10/12/2024");

    // Menghapus anggota Dito beserta buku yang dipinjam
    head = hapusAnggota(head, "A002");

    // Menampilkan data anggota dan buku yang dipinjam
    tampilkanData(head);

    return 0;
}
```

Ouput dari program:



5. Kesimpulan

Multi Linked List adalah struktur data yang fleksibel dan efisien untuk merepresentasikan hubungan kompleks antara data. Struktur ini seperti pohon keluarga yang dapat berkembang dan berubah sesuai kebutuhan. Dengan menggunakan pointer horizontal dan vertikal, kita bisa menghubungkan satu node dengan banyak node lainnya. Hal ini membuat Multi Linked List sangat cocok untuk aplikasi seperti sistem manajemen proyek atau struktur organisasi, di mana hubungan antara data sering berubah dan perlu dikelola dengan baik.