Ketentuan Tugas Pendahuluan

- Untuk soal teori **JAWABAN DIKETIK DENGAN RAPI** dan untuk soal algoritma **SERTAKAN SCREENSHOOT CODINGAN DAN HASIL OUTPUT.**
- Deadline pengumpulan TP Modul 6 adalah **Senin, 21 Oktober 2024 pukul 06.00 WIB.**
- **TIDAK ADA TOLERANSI KETERLAMBATAN, TERLAMBAT ATAU TIDAK MENGUMPULKAN TP ONLINE MAKA DIANGGAP TIDAK MENGERJAKAN.**
- **DILARANG PLAGIAT (PLAGIAT = E).**
- Kerjakan TP dengan jelas agar dapat dimengerti.
- Untuk setiap soal nama fungsi atau prosedur **WAJIB** menyertakan **NIM**, contoh: **insertFirst_130122xxxx**.
- File diupload di LMS menggunakan format **PDF** dengan ketentuan **:** **TP_MODX_NIM_KELAS.pdf**

```
Contoh:
int searchNode_130122xxxx (List L, int X);
```

CP:

- Raihan (089638482851)
- Kayyisa (085105303555)
- Abiya (082127180662)
- Rio (081210978384)

SELAMAT MENGERJAKAN^^

LAPORAN PRAKTIKUM

PERTEMUAN 6

DOUBLE LINKED LIST BAGIAN PERTAMA

Nama :

Tsaqif Hisyam Saputra (2311104024)

Dosen :

Yudha Islami Sulistya

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

A. Tujuan

- Memahami konsep modul linked list
- Mengaplikasikan konsep double linked list dengan menggunakan pointer dan dengan bahasa C++

B. Tools

Visual Studio Code dengan C++ Extension Pack

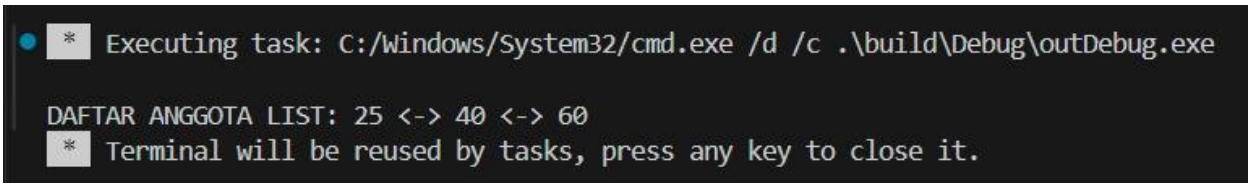
TUGAS PENDAHULUAN

1.

```cpp
#include <iostream>

using namespace std;

struct Node {
  int data;
  Node* prev;
  Node* next;
};

class DoublyLinkedList {
private:
  Node* head;
  Node* tail;

public:
  DoublyLinkedList() {
    head = NULL;
    tail = NULL;
  }

  void insertFirst(int data) {
    Node* newNode = new Node;
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = head;

    if (head != NULL) {
      head->prev = newNode;
    }
    head = newNode;

    if (tail == NULL) {
      tail = newNode;
    }
  }

  void insertLast(int data) {
    Node* newNode = new Node;
    newNode->data = data;
    newNode->next = NULL;
    newNode->prev = tail;

    if (tail != NULL) {
      tail->next = newNode;
    }
    tail = newNode;

    if (head == NULL) {
      head = newNode;
    }
  }

  void printList() {
    Node* current = head;
    cout << "DAFTAR ANGGOTA LIST: ";
    while (current != NULL) {
      cout << current->data;
      if (current->next != NULL) {
        cout << " <-> ";
      }
      current = current->next;
    }
    cout << endl;
  }
};

int main() {
  DoublyLinkedList dll;

  dll.insertFirst(40);  // Masukkan elemen pertama
  dll.insertFirst(25);  // Masukkan elemen kedua di awal
  dll.insertLast(60);   // Masukkan elemen ketiga di akhir

  dll.printList();

  return 0;
}
```

Kode tersebut mengimplementasikan Doubly Linked List dengan fungsi untuk menambahkan elemen di awal (**insertFirst**) dan di akhir (**insertLast**) list. Struktur **Node** memiliki data, pointer **next** ke node berikutnya, dan pointer **prev** ke node sebelumnya. Fungsi **insertFirst** menempatkan elemen baru di depan, memperbarui **head,** sementara **insertLast** menambah elemen di akhir dan memperbarui **tail**. Fungsi **printList** mencetak semua elemen dari depan ke belakang dengan format tertentu, memisahkan tiap elemen dengan tanda **<->**. Program menerima tiga input dari pengguna untuk menguji fungsi penambahan di awal dan akhir, lalu menampilkan seluruh isi list.

OUTPUT:

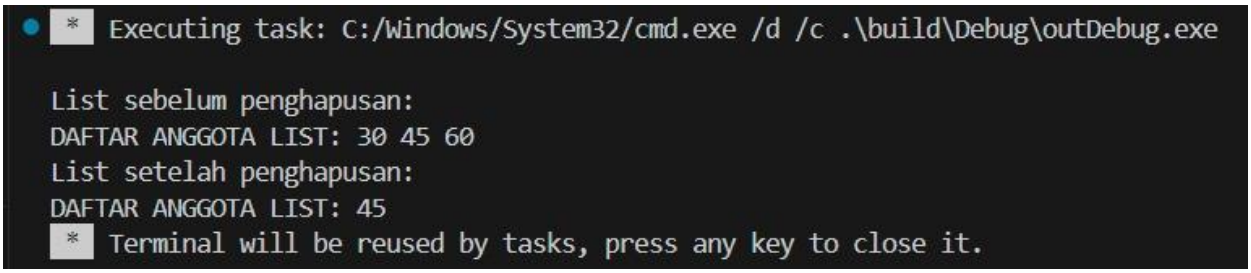


2.

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

// Struktur Node untuk Doubly Linked List
struct Node {
    int data;
    Node* next;
    Node* prev;
};

// Fungsi untuk menambahkan elemen di akhir list
void insertLast(Node*& head, int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;

    if (head == nullptr) {
        newNode->prev = nullptr;
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->prev = temp;
    }
}

// Fungsi untuk menghapus elemen pertama dalam list
void deleteFirst(Node*& head) {
    if (head == nullptr) {
        cout << "List kosong, tidak ada elemen untuk dihapus." << endl;
        return;
    }

    Node* temp = head;
    head = head->next;

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

    delete temp;
}

// Fungsi untuk menghapus elemen terakhir dalam list
void deleteLast(Node*& head) {
    if (head == nullptr) {
        cout << "List kosong, tidak ada elemen untuk dihapus." << endl;
        return;
    }

    if (head->next == nullptr) {
        // Jika hanya ada satu elemen
        delete head;
        head = nullptr;
    } else {
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }

        temp->prev->next = nullptr;
        delete temp;
    }
}

// Fungsi untuk menampilkan seluruh elemen dalam list
void displayList(Node* head) {
    if (head == nullptr) {
        cout << "List kosong." << endl;
        return;
    }

    Node* temp = head;
    cout << "DAFTAR ANGGOTA LIST: ";
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    Node* head = nullptr;

    // Input elemen elemen sesuai instruksi
    insertLast(head, 30); // Elemen pertama
    insertLast(head, 45); // Elemen kedua
    insertLast(head, 60); // Elemen ketiga

    cout << "List sebelum penghapusan:" << endl;
    displayList(head);

    // Menghapus elemen pertama dan terakhir
    deleteFirst(head);
    deleteLast(head);

    cout << "List setelah penghapusan:" << endl;
    displayList(head);

    return 0;
}
```

Dalam Doubly Linked List (DLL), setiap node memiliki data serta pointer **next** dan **prev** untuk menunjuk ke node berikutnya dan sebelumnya. Fungsi **insertLast** menambah elemen baru di akhir list; jika DLL kosong, elemen menjadi **head**. Fungsi **deleteFirst** menghapus elemen pertama, dan jika hanya ada satu elemen, **head** diset menjadi **nullptr**. Fungsi **deleteLast** menghapus elemen terakhir dan mengosongkan list jika elemen tersisa hanya satu. Fungsi **displayList** menampilkan semua elemen dalam DLL. Program utama menambahkan tiga elemen (30, 45, dan 60) lalu menampilkan DLL sebelum dan sesudah penghapusan elemen pertama dan terakhir.

OUTPUT:

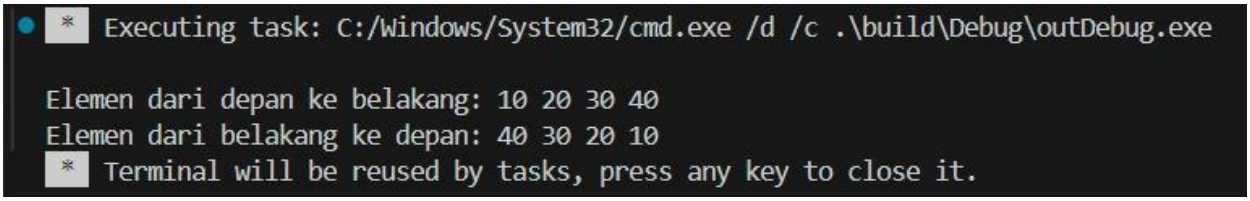


3.

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

// Struktur Node untuk Doubly Linked List
struct Node {
    int data;
    Node* next;
    Node* prev;
};

// Fungsi untuk menambahkan elemen di akhir list
void insertLast(Node*& head, int value) {
    Node* newNode = new Node();
    newNode->data = value;
    newNode->next = nullptr;

    if (head == nullptr) {
        newNode->prev = nullptr;
        head = newNode;
    } else {
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->prev = temp;
    }
}

// Fungsi untuk menampilkan elemen dari depan ke belakang
void displayForward(Node* head) {
    if (head == nullptr) {
        cout << "List kosong." << endl;
        return;
    }

    Node* temp = head;
    cout << "Elemen dari depan ke belakang: ";
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

// Fungsi untuk menampilkan elemen dari belakang ke depan
void displayBackward(Node* head) {
    if (head == nullptr) {
        cout << "List kosong." << endl;
        return;
    }

    Node* temp = head;

    // Bergerak ke elemen terakhir
    while (temp->next != nullptr) {
        temp = temp->next;
    }

    cout << "Elemen dari belakang ke depan: ";
    // Menampilkan dari elemen terakhir ke awal
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->prev;
    }
    cout << endl;
}

int main() {
    Node* head = nullptr;

    // Menambahkan 4 elemen ke dalam list
    insertLast(head, 10); // Elemen pertama
    insertLast(head, 20); // Elemen kedua
    insertLast(head, 30); // Elemen ketiga
    insertLast(head, 40); // Elemen keempat

    // Menampilkan elemen dari depan ke belakang
    displayForward(head);

    // Menampilkan elemen dari belakang ke depan
    displayBackward(head);

    return 0;
}
```

Program DLL ini memiliki node dengan tiga komponen: **data** untuk menyimpan nilai elemen, **next** sebagai pointer ke node berikutnya, dadn **prev** sebagai pointer ke node sebelumnya. Fungsi **insertLast** menambahka elemen baru di akhir list, di mana elemen pertama menjadi **head** jika list kosong, dan jika tidak, elemen baru ditambahkan di akhir. Fungsi **displayForward** menampilkan semua elemen dari depan ke belakang dengan memulai dari **head** dan melanjutkan ke elemen berikutnya melalui **next**, sedangkan **displayBackward** menampilkan elemen dari belakang ke depan, mencari node terakhir terlebih dahulu lalu bergerak mundur menggunakan **prev**. Program utama menambahkan empat elemen (10, 20, 30, dan 40) ke dalam list, lalu menampilkan elemen-elemen tersebut dalam dua arah: dari depan dke belakang dari belakang ke depan.

OUTPUT:

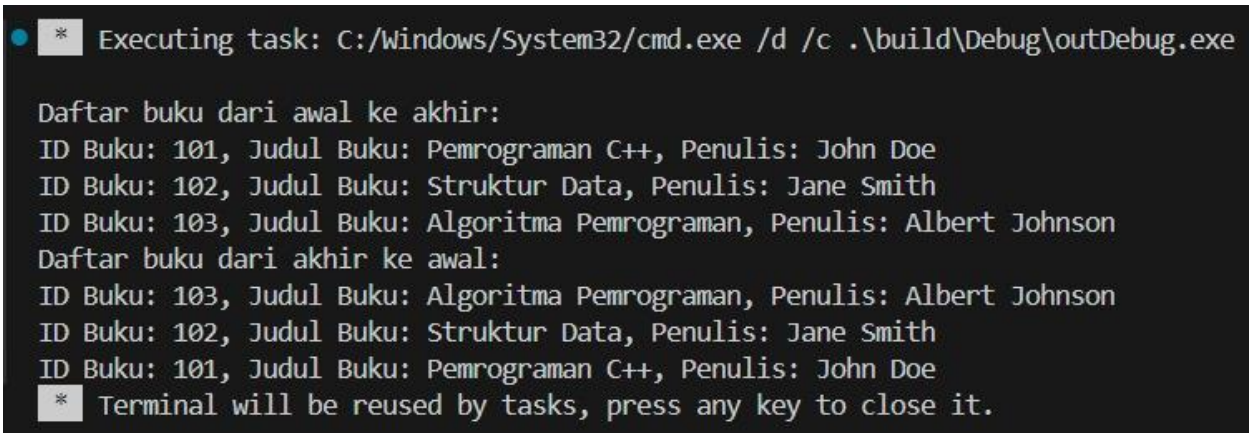


UNGUIDED

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

using namespace std;

// Struktur Node untuk menyimpan data buku
struct Node {
  int IDBuku;
  string JudulBuku;
  string PenulisBuku;
  Node* next;
  Node* prev;
};

// Fungsi untuk membuat node baru
Node* newNode(int IDBuku, string JudulBuku, string PenulisBuku) {
  Node* node = new Node;
  node->IDBuku = IDBuku;
  node->JudulBuku = JudulBuku;
  node->PenulisBuku = PenulisBuku;
  node->next = nullptr;
  node->prev = nullptr;
  return node;
}

// Fungsi untuk menambahkan buku di akhir linked list
void append(Node** head, Node** tail, int IDBuku, string JudulBuku, string PenulisBuku) {
  Node* new_node = newNode(IDBuku, JudulBuku, PenulisBuku);

  // Jika list kosong
  if (*head == nullptr) {
    *head = new_node;
    *tail = new_node;
    return;
  }

  // Tambahkan node baru di akhir list
  (*tail)->next = new_node;
  new_node->prev = *tail;
  *tail = new_node;
}

// Fungsi untuk menampilkan semua buku dari awal ke akhir
void printList(Node* node) {
  while (node != nullptr) {
    cout << "ID Buku: " << node->IDBuku << endl;
    cout << "Judul Buku: " << node->JudulBuku << endl;
    cout << "Penulis Buku: " << node->PenulisBuku << endl;
    cout << endl;
    node = node->next;
  }
}

// Fungsi untuk menampilkan semua buku dari akhir ke awal
void printReverseList(Node* node) {
  while (node != nullptr) {
    cout << "ID Buku: " << node->IDBuku << endl;
    cout << "Judul Buku: " << node->JudulBuku << endl;
    cout << "Penulis Buku: " << node->PenulisBuku << endl;
    cout << endl;
    node = node->prev;
  }
}

int main() {
  Node* head = nullptr;
  Node* tail = nullptr;

  // Menambahkan beberapa buku ke dalam linked list
  append(&head, &tail, 1, "Art of War", "Sun Tzu");
  append(&head, &tail, 2, "Shrek", "William Steig");
  append(&head, &tail, 3, "I Have No Mouth, and I Must Scream", "Harlan Ellison");

  cout << "Daftar Buku dari awal ke akhir:" << endl;
  printList(head);

  cout << "Daftar Buku dari akhir ke awal:" << endl;
  printReverseList(tail);

  return 0;
}
```

Kode di atas adalah implementasi linked list gande untuk menyimpan data buku, di mana setiap node menyimpan ID, judul, dan penulis buku. Fungsi **newNode** membuat node baru, sefangkan **append** menambahkan node di akhir list dengan memperbarui poiter **next** dan **prev**. Fungsi **printList** mencetak daftar buku dari awal hingga akhir, sementara **printReverseList** mencetaknya dari akhir ke awal. Di dalam **main**, list diinisialisasi dengan **head** dan **tail** bernilai **nullptr**, kemudian beberapa buku ditambahkan menggunakan **append**, dan akhirnya daftar buku ditambahkan dalam kedua arah dengan memanggil **printList** dan **printRverseList**.

OUTPUT:



GUIDED

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

class Node {
public:
    int data;
    Node* prev;
    Node* next;
};

class DoublyLinkedList {
public:
    Node* head;
    Node* tail;

    // Constructor untuk inisialisasi head dan tail
    DoublyLinkedList() {
        head = nullptr;
        tail = nullptr;
    }

    // Fungsi untuk menambahkan elemen di depan list (push)
    void push(int data) {
        Node* newNode = new Node;
        newNode->data = data;
        newNode->prev = nullptr;
        newNode->next = head;

        if (head != nullptr) {
            head->prev = newNode;
        } else {
            tail = newNode; // Jika list kosong, tail juga mengarah ke node baru
        }
        head = newNode;
    }

    // Fungsi untuk menghapus elemen dari depan list (pop)
    void pop() {
        if (head == nullptr) {
            return; // Jika list kosong
        }
        Node* temp = head;
        head = head->next;
        if (head != nullptr) {
            head->prev = nullptr;
        } else {
            tail = nullptr; // Jika hanya satu elemen di list
        }
        delete temp; // Hapus elemen
    }

    // Fungsi untuk mengupdate data di list
    bool update(int oldData, int newData) {
        Node* current = head;
        while (current != nullptr) {
            if (current->data == oldData) {
                current->data = newData;
                return true; // Jika data ditemukan dan diupdate
            }
            current = current->next;
        }
        return false; // Jika data tidak ditemukan
    }

    // Fungsi untuk menghapus semua elemen di list
    void deleteAll() {
        Node* current = head;
        while (current != nullptr) {
            Node* temp = current;
            current = current->next;
            delete temp;
        }
        head = nullptr;
        tail = nullptr;
    }

    // Fungsi untuk menampilkan semua elemen di list
    void display() {
        Node* current = head;
        while (current != nullptr) {
            cout << current->data << " ";
            current = current->next;
        }
        cout << endl;
    }
};

int main() {
    DoublyLinkedList list;
    while (true) {
        cout << "1. Add data" << endl;
        cout << "2. Delete data" << endl;
        cout << "3. Update data" << endl;
        cout << "4. Clear data" << endl;
        cout << "5. Display data" << endl;
        cout << "6. Exit" << endl;

        int choice;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1: {
                int data;
                cout << "Enter data to add: ";
                cin >> data;
                list.push(data);
                break;
            }
            case 2: {
                list.pop();
                break;
            }
            case 3: {
                int oldData, newData;
                cout << "Enter old data: ";
                cin >> oldData;
                cout << "Enter new data: ";
                cin >> newData;
                bool updated = list.update(oldData, newData);
                if (!updated) {
                    cout << "Data not found" << endl;
                }
                break;
            }
            case 4: {
                list.deleteAll();
                break;
            }
            case 5: {
                list.display();
                break;
            }
            case 6: {
                return 0;
            }
            default: {
                cout << "Invalid choice" << endl;
                break;
            }
        }
    }
    return 0;
}
```