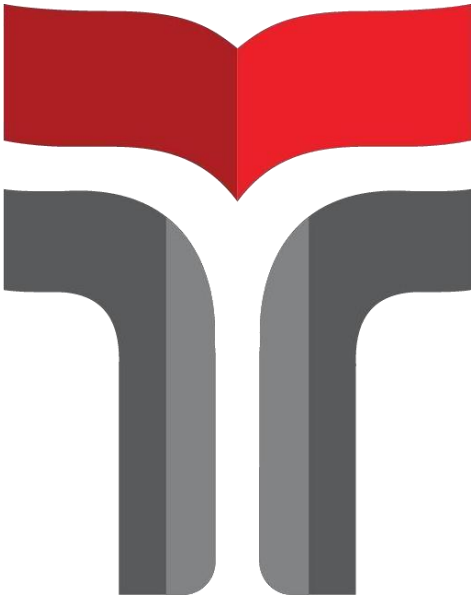
LAPORAN PRAKTIKUM
STRUKTUR DATA 4
"SINGLE LINKED LIST"



Oleh:

NAMA: Ammar Dzaki Nandana

NIM: 2311104071

KELAS: SE 07 B

DOSEN: Wahyu Andi Saputra

PRODI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

INSTITUT TEKNOLOGI TELKOM PURWOKERTO

2023/2024

I. TUJUAN

- Memahami konsep dasar Single Linked List.

- Mengimplementasikan operasi dasar pada Single Linked List, yaitu:

- Menambah node di depan dan belakang.

- Menghapus node berdasarkan nilai.

- Mencari node dengan nilai tertentu.

- Menghitung panjang linked list.

- Mempelajari cara mencetak isi linked list.

II. DASAR TEORI

A. Single Linked List

Single Linked List adalah salah satu jenis struktur data yang terdiri dari kumpulan elemen yang disebut node. Setiap node menyimpan dua bagian: data dan pointer (atau referensi) yang menunjuk ke node berikutnya dalam urutan. Struktur data ini bersifat dinamis, yang berarti ukuran linked list dapat bertambah atau berkurang sesuai dengan kebutuhan, tanpa memerlukan alokasi memori statis seperti array.

B. Struktur Node

Node adalah unit dasar dari linked list. Setiap node biasanya terdiri dari:

1. Data: Informasi atau nilai yang disimpan dalam node.
2. Pointer: Referensi ke node berikutnya. Node terakhir dalam linked list akan memiliki pointer yang menunjuk ke null, menandakan akhir dari list.

```
struct Node {
    int data;      // Menyimpan nilai node
    Node* next;   // Pointer ke node berikutnya
};
```

C. Operasi Dasar pada Single Linked List

1. Insert Node di Depan:

A. Menambah node baru di awal linked list.
B. Proses ini melibatkan pembuatan node baru, mengatur pointer next node baru ke node kepala yang ada, dan memperbarui kepala linked list ke node baru.

2. Insert Node di Belakang:

A. Menambah node baru di akhir linked list.
B. Proses ini melibatkan penelusuran linked list hingga menemukan node terakhir, kemudian mengatur pointer next node terakhir untuk menunjuk ke node baru.

3. Delete Node:

A. Menghapus node tertentu berdasarkan nilai yang diberikan.
B. Proses ini melibatkan penelusuran linked list untuk menemukan node yang ingin dihapus dan mengatur pointer dari node sebelumnya untuk melewati node yang dihapus.

4. Cari Node dengan Nilai Tertentu:

A. Mencari apakah suatu nilai ada dalam linked list.
B. Proses ini melibatkan penelusuran dari node kepala hingga akhir linked list, membandingkan nilai setiap node dengan nilai yang dicari.

Hitung Panjang Linked List:

A. Menghitung jumlah total node dalam linked list.
B. Proses ini melibatkan penelusuran dari node kepala hingga akhir linked list, menghitung setiap node yang dilalui.

D. Kelebihan dan Kekurangan Single Linked List

Kelebihan:

- Dinamis: Ukuran linked list dapat berubah secara dinamis tanpa memerlukan alokasi memori statis.
- Efisien dalam Penambahan/Penghapusan: Operasi penambahan dan penghapusan dapat dilakukan dengan efisien tanpa perlu memindahkan elemen lain, berbeda dengan array.

Kekurangan:

- Akses Elemen yang Lambat: Akses langsung ke elemen tertentu lambat, karena harus menelusuri dari kepala hingga node yang dicari.
- Memori Tambahan: Setiap node memerlukan ruang memori tambahan untuk pointer.

III. GUIDED

I.

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

// Deklarasi Struct untuk mahasiswa
struct mahasiswa {
    char nama[30];
    char nim[10];
};

// Deklarasi Struct Node
struct Node {
    mahasiswa data;
    Node *next;
};

Node *head;
Node *tail;

// Inisialisasi List
void init() {
    head = nullptr;
    tail = nullptr;
}

// Pengecekan apakah list kosong
bool isEmpty() {
    return head == nullptr;
}

// Tambah Depan
void insertDepan(const mahasiswa &data) {
    Node *baru = new Node;
    baru->data = data;
    baru->next = nullptr;
    if (isEmpty()) {
        head = tail = baru;
    } else {
        baru->next = head;
        head = baru;
    }
}

// Tambah Belakang
void insertBelakang(const mahasiswa &data) {
    Node *baru = new Node;
    baru->data = data;
    baru->next = nullptr;
    if (isEmpty()) {
        head = tail = baru;
    } else {
        tail->next = baru;
        tail = baru;
    }
}

// Hitung Jumlah List
int hitungList() {
    Node *current = head;
    int jumlah = 0;
    while (current != nullptr) {
        jumlah++;
        current = current->next;
    }
    return jumlah;
}

// Hapus Depan
void hapusDepan() {
    if (!isEmpty()) {
        Node *hapus = head;
        head = head->next;
        delete hapus;
        if (head == nullptr) {
            tail = nullptr; // Jika list menjadi kosong
        }
    } else {
        cout << "List kosong!" << endl;
    }
}

// Hapus Belakang
void hapusBelakang() {
    if (!isEmpty()) {
        if (head == tail) {
            delete head;
            head = tail = nullptr; // List menjadi kosong
        } else {
            Node *bantu = head;
            while (bantu->next != tail) {
                bantu = bantu->next;
            }
            delete tail;
            tail = bantu;
            tail->next = nullptr;
        }
    } else {
        cout << "List kosong!" << endl;
    }
}

// Tampilkan List
void tampil() {
    Node *current = head;
    if (!isEmpty()) {
        while (current != nullptr) {
            cout << "Nama: " << current->data.nama << ", NIM: " << current->data.nim << endl;
            current = current->next;
        }
    } else {
        cout << "List masih kosong!" << endl;
    }
}

// Hapus List
void clearList() {
    Node *current = head;
    while (current != nullptr) {
        Node *hapus = current;
        current = current->next;
        delete hapus;
    }
    head = tail = nullptr;
    cout << "List berhasil terhapus!" << endl;
}

// Main function
int main() {
    init();

    // Contoh data mahasiswa
    mahasiswa m1 = {"Alice", "123456"};
    mahasiswa m2 = {"Bob", "654321"};
    mahasiswa m3 = {"Charlie", "112233"};

    // Menambahkan mahasiswa ke dalam list
    insertDepan(m1);
    tampil();
    insertBelakang(m2);
    tampil();
    insertDepan(m3);
    tampil();

    // Menghapus elemen dari list
    hapusDepan();
    tampil();
    hapusBelakang();
    tampil();

    // Menghapus seluruh list
    clearList();

    return 0;
}
```

II.

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

// Definisi struktur untuk elemen list
struct Node {
    int data;       // Menyimpan nilai elemen
    Node* next;     // Pointer ke elemen berikutnya
};

// Fungsi untuk mengalokasikan memori untuk node baru
Node* alokasi(int value) {
    Node* newNode = new Node; // Alokasi memori untuk elemen baru
    if (newNode != nullptr) { // Jika alokasi berhasil
        newNode->data = value; // Mengisi data node
        newNode->next = nullptr; // Set next ke nullptr
    }
    return newNode; // Mengembalikan pointer node baru
}

// Fungsi untuk dealokasi memori node
void dealokasi(Node* node) {
    delete node; // Mengembalikan memori yang digunakan oleh node
}

// Pengecekan apakah list kosong
bool isListEmpty(Node* head) {
    return head == nullptr; // List kosong jika head adalah nullptr
}

// Menambahkan elemen di awal list
void insertFirst(Node* &head, int value) {
    Node* newNode = alokasi(value); // Alokasi memori untuk elemen baru
    if (newNode != nullptr) {
        newNode->next = head;     // Menghubungkan elemen baru ke elemen pertama
        head = newNode;           // Menetapkan elemen baru sebagai elemen pertama
    }
}

// Menambahkan elemen di akhir list
void insertLast(Node* &head, int value) {
    Node* newNode = alokasi(value); // Alokasi memori untuk elemen baru
    if (newNode != nullptr) {
        if (isListEmpty(head)) {  // Jika list kosong
            head = newNode;       // Elemen baru menjadi elemen pertama
        } else {
            Node* temp = head;
            while (temp->next != nullptr) { // Mencari elemen terakhir
                temp = temp->next;
            }
            temp->next = newNode; // Menambahkan elemen baru di akhir list
        }
    }
}

// Menampilkan semua elemen dalam list
void printList(Node* head) {
    if (isListEmpty(head)) {
        cout << "List kosong!" << endl;
    } else {
        Node* temp = head;
        while (temp != nullptr) {  // Selama belum mencapai akhir list
            cout << temp->data << " "; // Menampilkan data elemen
            temp = temp->next; // Melanjutkan ke elemen berikutnya
        }
        cout << endl;
    }
}

// Menghitung jumlah elemen dalam list
int countElements(Node* head) {
    int count = 0;
    Node* temp = head;
    while (temp != nullptr) {
        count++;            // Menambah jumlah elemen
        temp = temp->next; // Melanjutkan ke elemen berikutnya
    }
    return count;          // Mengembalikan jumlah elemen
}

// Menghapus semua elemen dalam list dan dealokasi memori
void clearList(Node* &head) {
    while (head != nullptr) {
        Node* temp = head;  // Simpan pointer ke node saat ini
        head = head->next;  // Pindahkan ke node berikutnya
        dealokasi(temp);    // Dealokasi node
    }
}

int main() {
    Node* head = nullptr;  // Membuat list kosong

    // Menambahkan elemen ke dalam list
    insertFirst(head, 10); // Menambahkan elemen 10 di awal list
    insertLast(head, 20);  // Menambahkan elemen 20 di akhir list
    insertLast(head, 30);  // Menambahkan elemen 30 di akhir list

    // Menampilkan isi list
    cout << "Isi List: ";
    printList(head);

    // Menampilkan jumlah elemen
    cout << "Jumlah elemen: " << countElements(head) << endl;

    // Menghapus semua elemen dalam list
    clearList(head);

    // Menampilkan isi list setelah penghapusan
    cout << "Isi List setelah penghapusan: ";
    printList(head);

    return 0;
}
```

IV. UNGUIDED

1.

```cpp
#include <iostream>

using namespace std;

struct Node {
    int data;
    Node* next;

    Node(int nilai) {
        data = nilai;
        next = nullptr;
    }
};

class SingleLinkedList {
private:
    Node* head;

public:
    SingleLinkedList() {
        head = nullptr;

    void tambahNodeDiDepan(int nilai) {
        Node* nodeBaru = new Node(nilai);
        nodeBaru->next = head;
        head = nodeBaru;
    }

    void tambahNodeDiBelakang(int nilai) {
        Node* nodeBaru = new Node(nilai);
        if (head == nullptr) {
            head = nodeBaru;
            return;
        }
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = nodeBaru;
    }

    void cetakList() {
        Node* temp = head;
        while (temp != nullptr) {
            cout << temp->data;
            if (temp->next != nullptr) {
                cout << " -> ";
            }
            temp = temp->next;
        }
        cout << endl;
    }

    ~SingleLinkedList() {
        Node* current = head;
        Node* nextNode;
        while (current != nullptr) {
            nextNode = current->next;
            delete current;
            current = nextNode;
        }
    }
};

int main() {
    SingleLinkedList linkedList;

    linkedList.tambahNodeDiDepan(10);
    linkedList.tambahNodeDiBelakang(20);
    linkedList.tambahNodeDiDepan(5);
    cout << "Linked List setelah penambahan: ";
    linkedList.cetakList();

    return 0;
}
```

2.

```cpp
#include <iostream>

using namespace std;

struct Node {
    int data;
    Node* next;

    Node(int nilai) {
        data = nilai;
        next = nullptr;
    }
};

class SingleLinkedList {
private:
    Node* head;

public:
    SingleLinkedList() {
        head = nullptr;
    }

    void tambahNodeDiDepan(int nilai) {
        Node* nodeBaru = new Node(nilai);
        nodeBaru->next = head;
        head = nodeBaru;
    }

    void tambahNodeDiBelakang(int nilai) {
        Node* nodeBaru = new Node(nilai);
        if (head == nullptr) {
            head = nodeBaru;
            return;
        }
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = nodeBaru;
    }

    void hapusNode(int nilai) {
        if (head == nullptr) {
            return;
        }
        if (head->data == nilai) {
            Node* temp = head;
            head = head->next;
            delete temp;
            return;
        }
        Node* current = head;
        Node* previous = nullptr;
        while (current != nullptr && current->data != nilai) {
            previous = current;
            current = current->next;
        }
        if (current != nullptr) {
            previous->next = current->next;
            delete current;
        }
    }

    void cetakList() {
        Node* temp = head;
        while (temp != nullptr) {
            cout << temp->data;
            if (temp->next != nullptr) {
                cout << " -> ";
            }
            temp = temp->next;
        }
        cout << endl;
    }
    ~SingleLinkedList() {
        Node* current = head;
        Node* nextNode;
        while (current != nullptr) {
            nextNode = current->next;
            delete current;
            current = nextNode;
        }
    }
};

int main() {
    SingleLinkedList linkedList;

    linkedList.tambahNodeDiDepan(10);
    linkedList.tambahNodeDiBelakang(20);
    linkedList.tambahNodeDiDepan(5);

    cout << "Linked List sebelum penghapusan: ";
    linkedList.cetakList();

    linkedList.hapusNode(10);

    cout << "Linked List setelah penghapusan nilai 10: ";
    linkedList.cetakList();

    return 0;
}

```

3.

```cpp
#include <iostream>

using namespace std;

struct Node {
    int data;
    Node* next;

    Node(int nilai) {
        data = nilai;
        next = nullptr;
    }
};

class SingleLinkedList {
private:
    Node* head;

public:
    SingleLinkedList() {
        head = nullptr;
    }

    void tambahNodeDiDepan(int nilai) {
        Node* nodeBaru = new Node(nilai);
        nodeBaru->next = head;
        head = nodeBaru;
    }

    void tambahNodeDiBelakang(int nilai) {
        Node* nodeBaru = new Node(nilai);
        if (head == nullptr) {
            head = nodeBaru;
            return;
        }
        Node* temp = head;
        while (temp->next != nullptr) {
            temp = temp->next;
        }
        temp->next = nodeBaru;
    }

    bool cariNode(int nilai) {
        Node* temp = head;
        while (temp != nullptr) {
            if (temp->data == nilai) {
                return true;
            }
            temp = temp->next;
        }
        return false;
    }

    int hitungPanjang() {
        int panjang = 0;
        Node* temp = head;
        while (temp != nullptr) {
            panjang++;
            temp = temp->next;
        }
        return panjang;
    }

    void cetakList() {
        Node* temp = head;
        while (temp != nullptr) {
            cout << temp->data;
            if (temp->next != nullptr) {
                cout << " -> ";
            }
            temp = temp->next;
        }
        cout << endl;
    }

    ~SingleLinkedList() {
        Node* current = head;
        Node* nextNode;
        while (current != nullptr) {
            nextNode = current->next;
            delete current;
            current = nextNode;
        }
    }
};

int main() {
    SingleLinkedList linkedList;

    linkedList.tambahNodeDiDepan(10);
    linkedList.tambahNodeDiBelakang(20);
    linkedList.tambahNodeDiDepan(5);

    cout << "Linked List: ";
    linkedList.cetakList();

    int nilaiDicari = 20;
    if (linkedList.cariNode(nilaiDicari)) {
        cout << "Node dengan nilai " << nilaiDicari << " ditemukan." << endl;
    } else {
        cout << "Node dengan nilai " << nilaiDicari << " tidak ditemukan." << endl;
    }

    int panjang = linkedList.hitungPanjang();
    cout << "Panjang linked list: " << panjang << endl;

    return 0;
}
```

V. KESIMPULAN

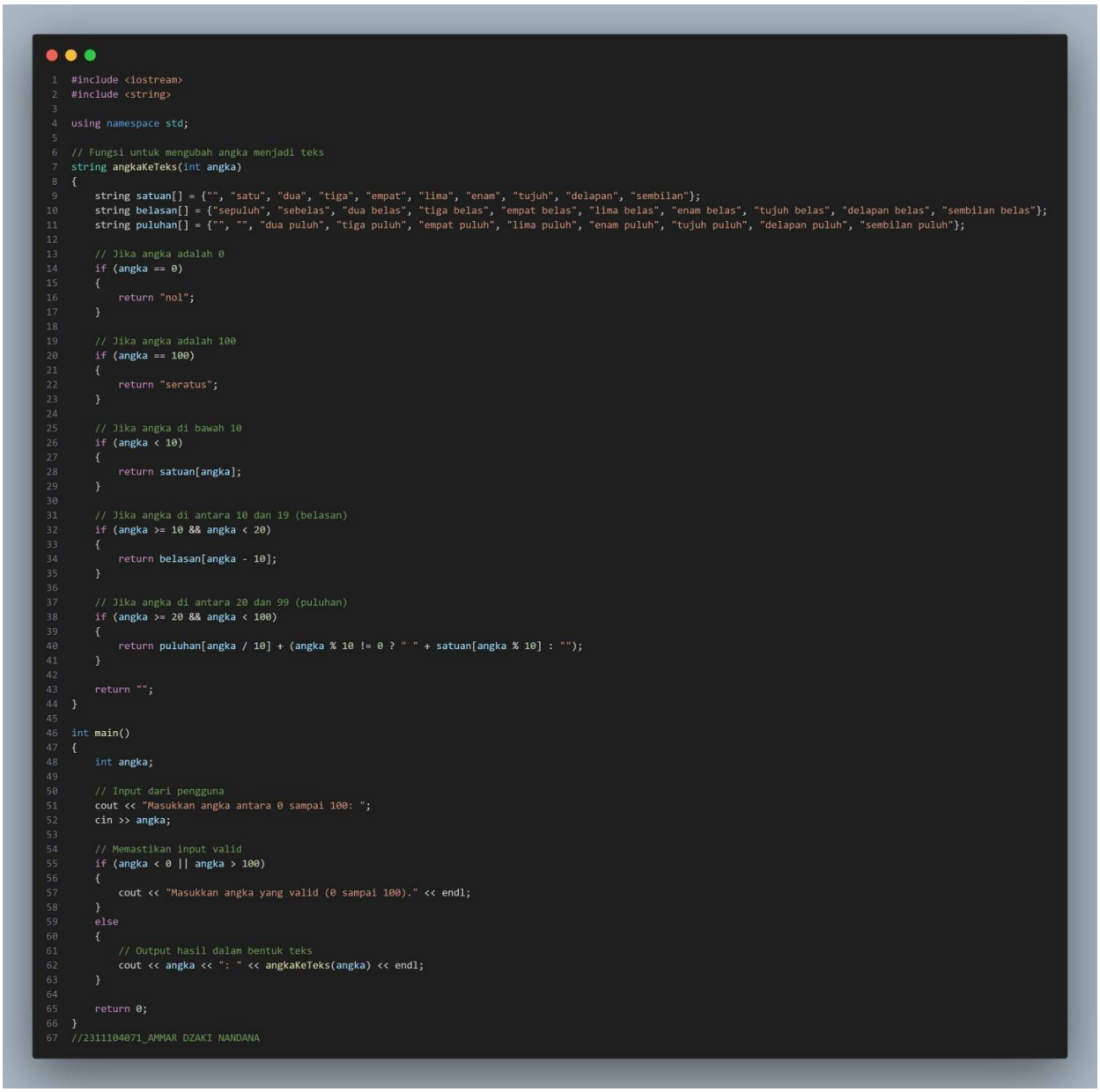
Praktikum ini berhasil menjelaskan dan mengimplementasikan konsep dasar Single Linked List dalam bahasa C++. Melalui berbagai operasi dasar seperti penambahan node di depan dan belakang, penghapusan node berdasarkan nilai, pencarian node tertentu, dan penghitungan panjang linked list, peserta praktikum dapat memahami struktur dan dinamika dari linked list. Implementasi ini menunjukkan bahwa linked list adalah struktur data yang fleksibel dan efisien, terutama dalam pengelolaan data yang sering berubah. Meskipun memiliki kelebihan dalam hal dinamisitas dan efisiensi operasi penambahan serta penghapusan, linked list juga memiliki kekurangan seperti akses elemen yang lambat dan kebutuhan memori tambahan untuk pointer. Dengan demikian, praktik ini memberikan wawasan penting mengenai cara kerja dan aplikasi Single Linked List dalam pemrograman.

VI. UNGUIDED

NO. 1

```cpp
#include <iostream>

using namespace std;

int main() {
    float bil1, bil2;

    // Input dua buah bilangan dari user
    cout << "Masukkan bilangan pertama: ";
    cin >> bil1;
    cout << "Masukkan bilangan kedua: ";
    cin >> bil2;

    // Tampilkan hasil penjumlahan
    cout << "Hasil penjumlahan: " << bil1 + bil2 << endl;

    // Tampilkan hasil pengurangan
    cout << "Hasil pengurangan: " << bil1 - bil2 << endl;

    // Tampilkan hasil perkalian
    cout << "Hasil perkalian: " << bil1 * bil2 << endl;

    // Tampilkan hasil pembagian
    // Pengecekan untuk menghindari pembagian dengan nol
    if (bil2 != 0) {
        cout << "Hasil pembagian: " << bil1 / bil2 << endl;
    } else {
        cout << "Pembagian tidak bisa dilakukan karena bilangan kedua adalah nol." << endl;
    }

    return 0;
}


//2311104071_AMMAR DZAKI NANDANA

```

NO. 2



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

using namespace std;

// Fungsi untuk mengubah angka menjadi teks
string angkaKeTeks(int angka)
{
    string satuan[] = {"", "satu", "dua", "tiga", "empat", "lima", "enam", "tujuh", "delapan", "sembilan"};
    string belasan[] = {"sepuluh", "sebelas", "dua belas", "tiga belas", "empat belas", "lima belas", "enam belas", "tujuh belas", "delapan belas", "sembilan belas"};
    string puluhan[] = {"", "", "dua puluh", "tiga puluh", "empat puluh", "lima puluh", "enam puluh", "tujuh puluh", "delapan puluh", "sembilan puluh"};

    // Jika angka adalah 0
    if (angka == 0)
    {
        return "nol";
    }

    // Jika angka adalah 100
    if (angka == 100)
    {
        return "seratus";
    }

    // Jika angka di bawah 10
    if (angka < 10)
    {
        return satuan[angka];
    }

    // Jika angka di antara 10 dan 19 (belasan)
    if (angka >= 10 && angka < 20)
    {
        return belasan[angka - 10];
    }

    // Jika angka di antara 20 dan 99 (puluhan)
    if (angka >= 20 && angka < 100)
    {
        return puluhan[angka / 10] + (angka % 10 != 0 ? " " + satuan[angka % 10] : "");
    }

    return "";
}

int main()
{
    int angka;

    // Input dari pengguna
    cout << "Masukkan angka antara 0 sampai 100: ";
    cin >> angka;

    // Memastikan input valid
    if (angka < 0 || angka > 100)
    {
        cout << "Masukkan angka yang valid (0 sampai 100)." << endl;
    }
    else
    {
        // Output hasil dalam bentuk teks
        cout << angka << ": " << angkaKeTeks(angka) << endl;
    }

    return 0;
}
//2311104071_AMMAR DZAKI NANDANA
```

NO. 3

```cpp
#include <iostream>

using namespace std;

int main() {
    int n;

    // Input dari pengguna
    cout << "Masukkan angka: ";
    cin >> n;

    // Loop untuk mencetak pola
    for (int i = n; i >= 1; i--) {
        // Bagian kiri: menurun dari i hingga 1
        for (int j = i; j >= 1; j--) {
            cout << j << " ";
        }

        // Cetak tanda bintang *
        cout << "*";

        // Bagian kanan: menaik dari 1 hingga i
        for (int j = 1; j <= i; j++) {
            cout << " " << j;
        }

        // Baris baru untuk pola berikutnya
        cout << endl;
    }

    // Output terakhir hanya tanda *
    cout << "*" << endl;

    return 0;
}
#include <iostream>
#include <string>

using namespace std;

// Fungsi untuk mengubah angka menjadi teks
string angkaKeTeks(int angka)
{
    string satuan[] = {"", "satu", "dua", "tiga", "empat", "lima", "enam", "tujuh", "delapan", "sembilan"};
    string belasan[] = {"sepuluh", "sebelas", "dua belas", "tiga belas", "empat belas", "lima belas", "enam belas", "tujuh belas", "delapan belas", "sembilan belas"};
    string puluhan[] = {"", "", "dua puluh", "tiga puluh", "empat puluh", "lima puluh", "enam puluh", "tujuh puluh", "delapan puluh", "sembilan puluh"};

    // Jika angka adalah 0
    if (angka == 0)
    {
        return "nol";
    }

    // Jika angka adalah 100
    if (angka == 100)
    {
        return "seratus";
    }

    // Jika angka di bawah 10
    if (angka < 10)
    {
        return satuan[angka];
    }

    // Jika angka di antara 10 dan 19 (belasan)
    if (angka >= 10 && angka < 20)
    {
        return belasan[angka - 10];
    }

    // Jika angka di antara 20 dan 99 (puluhan)
    if (angka >= 20 && angka < 100)
    {
        return puluhan[angka / 10] + (angka % 10 != 0 ? " " + satuan[angka % 10] : "");
    }

    return "";
}

int main()
{
    int angka;

    // Input dari pengguna
    cout << "Masukkan angka antara 0 sampai 100: ";
    cin >> angka;

    // Memastikan input valid
    if (angka < 0 || angka > 100)
    {
        cout << "Masukkan angka yang valid (0 sampai 100)." << endl;
    }
    else
    {
        // Output hasil dalam bentuk teks
        cout << angka << ": " << angkaKeTeks(angka) << endl;
    }

    return 0;
}
//2311104071_AMMAR DZAKI NANDANA
```

VII. KESIMPULAN

Penginstalan Code::Blocks beserta kompiler MinGW (untuk pengguna Windows) telah berhasil dilakukan. IDE ini siap digunakan untuk pengembangan program dalam bahasa C atau C++. Dengan menggunakan Code::Blocks, proses pengembangan menjadi lebih mudah karena tersedianya fitur debugging dan kompilasi otomatis.