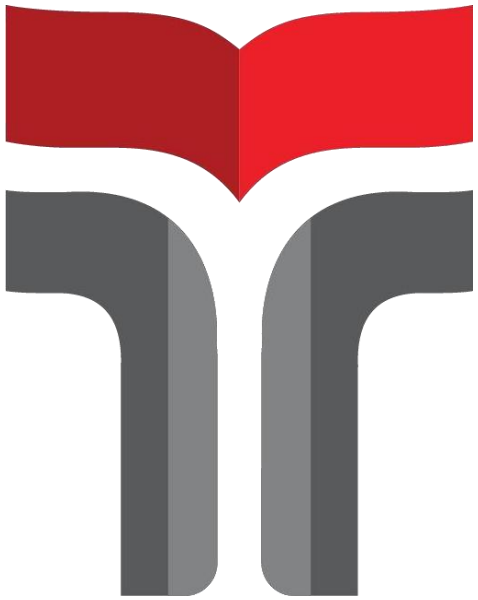
# LAPORAN PRAKTIKUM
# STRUKTUR DATA 3
# "ABSTRACT DATA TYPE (ADT)"




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

1. Memisahkan Antarmuka dan Implementasi
   - ADT memungkinkan pemisahan antara apa yang dapat dilakukan dengan tipe data dan bagaimana hal itu dilakukan. Ini memungkinkan pemrogram untuk fokus pada cara menggunakan data tanpa harus memahami detail internalnya.
   - Contoh: Saat menggunakan stack, pengguna cukup mengetahui operasi seperti push, pop, dan peek, tanpa perlu tahu bagaimana stack tersebut diimplementasikan (apakah menggunakan array, linked list, atau cara lainnya).
2. Meningkatkan Enkapsulasi
   - ADT membantu mengamankan data dengan menyembunyikan representasi internal dan hanya menyediakan antarmuka untuk memanipulasi data.
   - Dengan enkapsulasi, kita mengurangi kemungkinan kesalahan karena pengguna tidak dapat mengakses atau memodifikasi data internal secara langsung.
3. Mendukung Modularitas dan Reusabilitas
   - Dengan memisahkan antarmuka dan implementasi, kode yang menggunakan ADT menjadi modular dan lebih mudah untuk dikelola serta diperbarui.
   - Implementasi ADT yang baik dapat digunakan kembali di berbagai proyek tanpa harus menulis ulang algoritma yang sudah ada. Hal ini mendukung reusabilitas kode.
   - Contoh: Jika kita telah mengimplementasikan queue dengan ADT, kita dapat menggunakannya di berbagai aplikasi (seperti pemrograman jaringan, simulasi, dll.) tanpa mengubah antarmuka atau implementasinya.
4. Meningkatkan Fleksibilitas
   - Karena ADT terfokus pada antarmuka, kita dapat mengubah implementasi di belakang layar tanpa mempengaruhi kode yang menggunakan ADT tersebut. Misalnya, kita dapat mengubah implementasi stack dari array menjadi linked list, tanpa memengaruhi kode yang memanfaatkan stack.
   - ADT juga memberikan fleksibilitas dalam memilih struktur data yang paling efisien untuk kasus tertentu tanpa mengorbankan antarmuka yang konsisten.
5. Membantu Merancang Program yang Lebih Besar
   - ADT sangat berguna dalam desain sistem besar karena membantu pemrogram memecah masalah besar menjadi modul-modul yang lebih kecil, dapat dikelola, dan saling independen.
   - Setiap ADT dapat berfungsi sebagai modul tersendiri yang memiliki tanggung jawab khusus, sehingga memudahkan pemeliharaan dan pengembangan sistem.
6. Memudahkan Pemeliharaan dan Pengembangan Kode
   - Karena pengguna ADT hanya berinteraksi melalui antarmuka, perubahan pada implementasi internal ADT tidak akan berdampak pada kode yang menggunakannya. Hal ini membuat proses pemeliharaan kode menjadi lebih mudah.
   - Misalnya, kita dapat memperbaiki atau meningkatkan algoritma yang ada dalam ADT tanpa mengubah cara pengguna berinteraksi dengan tipe data tersebut.

7. Mengembangkan Pola Pikir Abstrak

- Mempelajari ADT membantu mengembangkan pola pikir abstrak dalam merancang program. Alih-alih terjebak dalam detail teknis tentang cara menyimpan dan memanipulasi data, kita lebih fokus pada fungsi dan operasi yang tersedia.
- Ini juga memungkinkan pengembang untuk fokus pada logika bisnis dari aplikasi yang sedang dikembangkan.

8. Mengurangi Redundansi
   - Dengan menggunakan ADT, kita menghindari pengulangan kode. Sebagai contoh, implementasi sebuah queue dapat digunakan di berbagai konteks tanpa harus menulis ulang implementasinya.
   - ADT juga memudahkan untuk menyembunyikan detail implementasi, sehingga kode menjadi lebih terstruktur dan bersih.
9. Mendukung Implementasi yang Lebih Aman dan Konsisten
   - ADT memastikan bahwa operasi-operasi yang diizinkan pada tipe data tertentu dilakukan dengan cara yang benar dan sesuai dengan spesifikasi, sehingga mencegah modifikasi yang tidak diinginkan atau tidak sesuai.
   - Contoh: Operasi pada stack (seperti push dan pop) hanya bisa dilakukan sesuai dengan aturan LIFO (Last In, First Out), sehingga mencegah penggunaan yang salah.
10. Meningkatkan Pemahaman Tentang Struktur Data
    - Mempelajari ADT memungkinkan pemrogram untuk lebih memahami berbagai struktur data dan bagaimana mereka digunakan dalam pemrograman nyata.
    - Dengan ADT, pemrogram bisa lebih cepat beralih di antara implementasi yang berbeda dari struktur data yang sama (misalnya, queue yang diimplementasikan menggunakan array atau linked list).

## II. DASAR TEORI

Pengertian Abstract Data Type (ADT)
Abstract Data Type (ADT) adalah konsep dalam ilmu komputer yang merujuk pada tipe data yang mendefinisikan perilaku atau operasi yang dapat dilakukan terhadap data, tanpa memperhatikan bagaimana data tersebut diimplementasikan secara internal. ADT berfokus pada abstraksi, yaitu memberikan antarmuka kepada pengguna untuk berinteraksi dengan data tanpa mengungkapkan detail implementasi di baliknya.

Komponen Utama dalam ADT:

1. Definisi Tipe Data:

ADT mendefinisikan tipe data yang akan digunakan, yang berisi informasi tentang properti atau atribut data.
Contoh: Pada ADT Stack, tipe data yang didefinisikan adalah kumpulan elemen yang dapat diakses secara LIFO (Last In, First Out).

2. Operasi yang Didukung:

ADT juga mendefinisikan operasi yang dapat dilakukan terhadap tipe data tersebut. Operasi-operasi ini mendefinisikan bagaimana data dapat diubah atau digunakan.
Contoh operasi pada ADT Stack:
Push: Menambahkan elemen ke dalam stack.
Pop: Menghapus elemen paling atas dari stack.
Peek: Melihat elemen paling atas dari stack tanpa menghapusnya.

Pemisahan Antarmuka dan Implementasi:

Salah satu karakteristik utama dari ADT adalah pemisahan antara antarmuka (interface) dan implementasi. Antarmuka mendefinisikan operasi apa yang tersedia, sementara implementasi adalah cara internal di mana operasi tersebut dijalankan.
Pengguna hanya perlu mengetahui antarmuka (operasi yang dapat dilakukan) tanpa perlu mengetahui detail implementasi.

Contoh Abstract Data Type (ADT)

1. Stack:
Antarmuka (Operasi):

1. Push(x): Menambahkan elemen x ke dalam stack.
2. Pop(): Menghapus elemen teratas dari stack.
3. Top(): Mengembalikan elemen teratas tanpa menghapusnya.
4. IsEmpty(): Mengecek apakah stack kosong.
5. Size(): Mengembalikan ukuran stack.

Implementasi: Stack dapat diimplementasikan menggunakan array atau linked list. Namun, pengguna stack tidak perlu mengetahui bagaimana stack diimplementasikan; mereka hanya menggunakan antarmuka yang disediakan.

2. Queue:

- Antarmuka (Operasi):

1. Enqueue(x): Menambahkan elemen x ke dalam antrian.
1. Dequeue(): Menghapus elemen terdepan dari antrian.
2. Front(): Mengembalikan elemen terdepan tanpa menghapusnya.
3. IsEmpty(): Mengecek apakah antrian kosong.

- Implementasi: Queue dapat diimplementasikan menggunakan array melingkar (circular array) atau linked list.

Keuntungan Menggunakan ADT

Enkapsulasi: ADT membantu mengenkapsulasi data dengan menyembunyikan detail implementasi. Pengguna hanya dapat berinteraksi dengan data melalui antarmuka yang disediakan. Hal ini mencegah pengguna memodifikasi struktur data secara langsung dan membantu menjaga konsistensi.

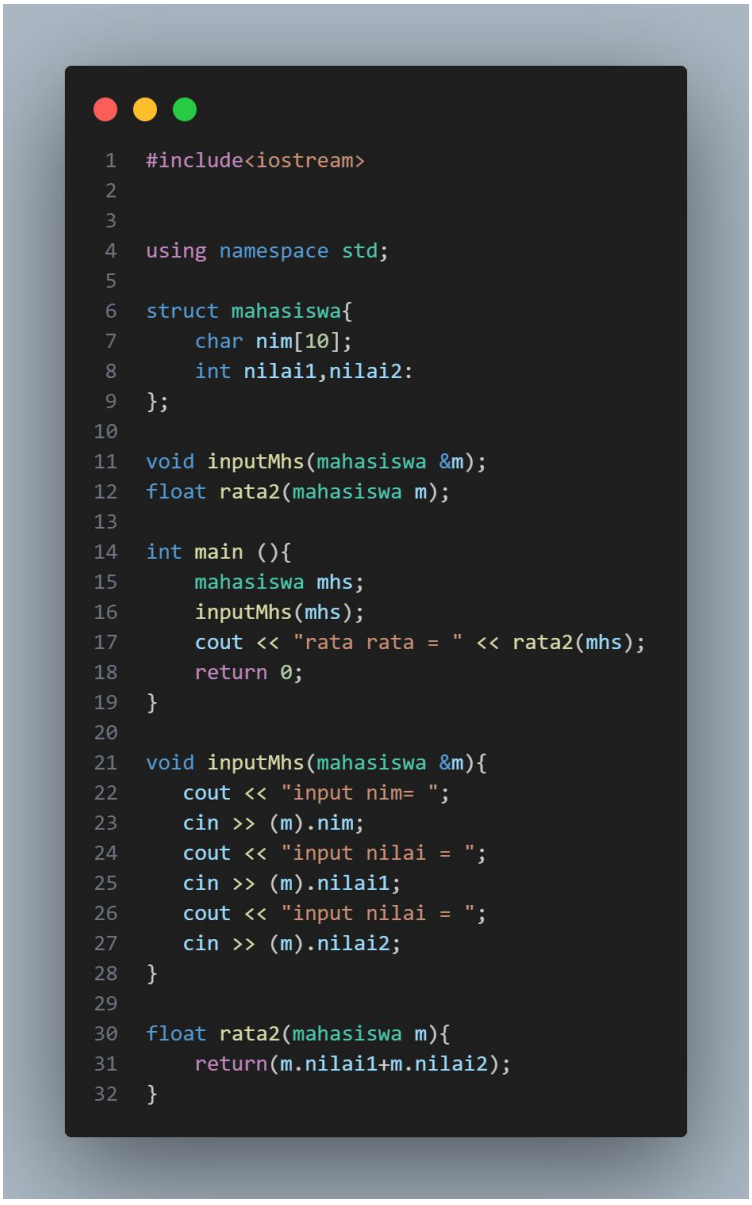
Abstraksi: ADT memberikan abstraksi yang memudahkan pengguna fokus pada cara menggunakan data daripada memikirkan bagaimana data tersebut diimplementasikan. Hal ini memungkinkan pengguna bekerja pada level yang lebih tinggi tanpa terjebak dalam detail implementasi teknis.

Reusabilitas: Karena ADT mendefinisikan antarmuka yang umum dan standar, implementasinya dapat digunakan kembali di berbagai aplikasi. Misalnya, implementasi stack dapat digunakan dalam berbagai program tanpa memerlukan perubahan besar.

Modularitas: ADT membantu menciptakan kode yang modular. Antarmuka ADT memisahkan penggunaan dari implementasi, yang memungkinkan kita mengganti implementasi di masa depan tanpa mempengaruhi kode yang menggunakannya. Misalnya, kita dapat mengganti implementasi

stack dari array menjadi linked list tanpa mengubah cara pengguna berinteraksi dengan stack tersebut.

## III. GUIDED



```cpp
#include<iostream>


using namespace std;

struct mahasiswa{
    char nim[10];
    int nilai1,nilai2:
};

void inputMhs(mahasiswa &m);
float rata2(mahasiswa m);

int main (){
    mahasiswa mhs;
    inputMhs(mhs);
    cout << "rata rata = " << rata2(mhs);
    return 0;
}

void inputMhs(mahasiswa &m){
   cout << "input nim= ";
   cin >> (m).nim;
   cout << "input nilai = ";
   cin >> (m).nilai1;
   cout << "input nilai = ";
   cin >> (m).nilai2;
}

float rata2(mahasiswa m){
    return(m.nilai1+m.nilai2);
}
```

Kode yang Anda unggah adalah sebuah program C++ yang menggunakan **struct** untuk menyimpan data mahasiswa, dan memiliki fungsi untuk input serta menghitung rata-rata nilai mahasiswa. Berikut adalah penjelasan lengkap dari kode tersebut:

### 1. **Inklusi Header dan Namespace**
```
#include <iostream>
using namespace std;
```

- Program ini menggunakan **iostream** untuk input dan output.
- `using namespace std;` membuat kita tidak perlu menuliskan `std::` sebelum menggunakan fungsi dari

standar C++ seperti `cin` dan `cout`.

### 2. **Definisi Struct `mahasiswa`**
```
struct mahasiswa {
    char nim[10];
    int nilai1, nilai2;
};
```

- Struct `mahasiswa` digunakan untuk mendefinisikan tipe data baru yang menyimpan informasi seorang mahasiswa.
  - `nim`: Sebuah array `char` dengan ukuran 10 untuk menyimpan NIM mahasiswa.
  - `nilai1` dan `nilai2`: Dua variabel `int` untuk menyimpan nilai-nilai mahasiswa.

### 3. **Prototipe Fungsi**
```
void inputMhs(mahasiswa &m);
float rata2(mahasiswa m);
```

- **inputMhs**: Fungsi untuk mengambil input data mahasiswa (NIM dan dua nilai). Parameter yang digunakan adalah `mahasiswa &m`, yang artinya nilai yang dioper akan dimodifikasi secara **by reference**.
- **rata2**: Fungsi untuk menghitung rata-rata nilai dari dua nilai mahasiswa. Fungsi ini menerima parameter bertipe **mahasiswa** (pass by value) dan mengembalikan nilai bertipe `float`.

### 4. **Fungsi `main`**
```
int main() {
    mahasiswa mhs;
    inputMhs(mhs);
    cout << "rata rata = " << rata2(mhs);
    return 0;
}
```

- Dalam fungsi `main`:
  - **mahasiswa mhs**: Variabel `mhs` dideklarasikan sebagai sebuah objek `mahasiswa`.
  - **inputMhs(mhs)**: Memanggil fungsi `inputMhs` untuk mengisi data mahasiswa.
  - **rata2(mhs)**: Setelah data mahasiswa diisi, fungsi ini menghitung rata-rata nilai dan mencetak hasilnya menggunakan `cout`.

### 5. **Fungsi `inputMhs`**
```
void inputMhs(mahasiswa &m) {
    cout << "input nim= ";
    cin >> (m).nim;
    cout << "input nilai = ";
    cin >> (m).nilai1;
    cout << "input nilai = ";
    cin >> (m).nilai2;
}
```

- Fungsi ini bertanggung jawab untuk mengambil input data dari pengguna, yaitu NIM dan dua nilai.

- **cin >> m.nim**: Meminta pengguna memasukkan NIM mahasiswa.
- **cin >> m.nilai1**: Meminta pengguna memasukkan nilai pertama.
- **cin >> m.nilai2**: Meminta pengguna memasukkan nilai kedua.

## IV. UNGUIDED

**1.**

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

struct Mahasiswa {
    string nama;
    string nim;
    float uts;
    float uas;
    float tugas;
    float nilaiAkhir;
};

float hitungNilaiAkhir(float uts, float uas, float tugas) {
    return (0.3 * uts) + (0.4 * uas) + (0.3 * tugas);
}

int main() {
    const int MAX_MAHASISWA = 10;
    Mahasiswa mahasiswa[MAX_MAHASISWA];
    int jumlahMahasiswa;

    cout << "Masukkan jumlah mahasiswa (max 10): ";
    cin >> jumlahMahasiswa;

    if (jumlahMahasiswa > MAX_MAHASISWA) {
        cout << "Jumlah mahasiswa melebihi batas maksimal!" << endl;
        return 1;
    }

    for (int i = 0; i < jumlahMahasiswa; i++) {
        cout << "\nMahasiswa ke-" << i + 1 << ":\n";
        cout << "Nama: ";
        cin >> mahasiswa[i].nama;
        cout << "NIM: ";
        cin >> mahasiswa[i].nim;
        cout << "UTS: ";
        cin >> mahasiswa[i].uts;
        cout << "UAS: ";
        cin >> mahasiswa[i].uas;
        cout << "Tugas: ";
        cin >> mahasiswa[i].tugas;

        mahasiswa[i].nilaiAkhir = hitungNilaiAkhir(mahasiswa[i].uts, mahasiswa[i].uas, mahasiswa[i].tugas);
    }

    cout << "\nData Mahasiswa:\n";
    for (int i = 0; i < jumlahMahasiswa; i++) {
        cout << "\nMahasiswa ke-" << i + 1 << ":\n";
        cout << "Nama: " << mahasiswa[i].nama << endl;
        cout << "NIM: " << mahasiswa[i].nim << endl;
        cout << "UTS: " << mahasiswa[i].uts << endl;
        cout << "UAS: " << mahasiswa[i].uas << endl;
        cout << "Tugas: " << mahasiswa[i].tugas << endl;
        cout << "Nilai Akhir: " << mahasiswa[i].nilaiAkhir << endl;
    }

    return 0;
}
```

**2.**

```cpp
// PELAJARAN.H
#ifndef PELAJARAN_H
#define PELAJARAN_H

#include <string>
using namespace std;

struct pelajaran {
    string namaMapel;
    string kodeMapel;
};

pelajaran create_pelajaran(string namapel, string kodepel);

void tampil_pelajaran(pelajaran pel);

#endif


// PELAJARAN CPP
#include <iostream>
#include "pelajaran.h"

using namespace std;

pelajaran create_pelajaran(string namapel, string kodepel) {
    pelajaran pel;
    pel.namaMapel = namapel;
    pel.kodeMapel = kodepel;
    return pel;
}

void tampil_pelajaran(pelajaran pel) {
    cout << "nama pelajaran : " << pel.namaMapel << endl;
    cout << "nilai : " << pel.kodeMapel << endl;
}


// MAIN CPP
#include <iostream>
#include "pelajaran.h"

using namespace std;

int main() {
    string namapel = "Struktur Data";
    string kodepel = "STD";

    pelajaran pel = create_pelajaran(namapel, kodepel);

    tampil_pelajaran(pel);

    return 0;
}

```

**3.**

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

const int SIZE = 3;

void tampilkanArray(int arr[SIZE][SIZE]) {
    for (int i = 0; i < SIZE; i++) {
        for (int j = 0; j < SIZE; j++) {
            cout << arr[i][j] << " ";
        }
        cout << endl;
    }
}

void tukarElemenArray(int arr1[SIZE][SIZE], int arr2[SIZE][SIZE], int x, int y) {
    int temp = arr1[x][y];
    arr1[x][y] = arr2[x][y];
    arr2[x][y] = temp;
}

void tukarPointer(int *ptr1, int *ptr2) {
    int temp = *ptr1;
    *ptr1 = *ptr2;
    *ptr2 = temp;
}

int main() {
    int array1[SIZE][SIZE] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
    int array2[SIZE][SIZE] = {{9, 8, 7}, {6, 5, 4}, {3, 2, 1}};

    int *pointer1, *pointer2;
    int var1 = 10, var2 = 20;
    pointer1 = &var1;
    pointer2 = &var2;

    cout << "Array 1 sebelum pertukaran:" << endl;
    tampilkanArray(array1);

    cout << "\nArray 2 sebelum pertukaran:" << endl;
    tampilkanArray(array2);

    cout << "\nMenukar elemen array pada posisi (1,1):" << endl;
    tukarElemenArray(array1, array2, 1, 1);

    cout << "\nArray 1 setelah pertukaran:" << endl;
    tampilkanArray(array1);

    cout << "\nArray 2 setelah pertukaran:" << endl;
    tampilkanArray(array2);

    cout << "\nNilai sebelum pertukaran pointer:" << endl;
    cout << "Var1: " << var1 << ", Var2: " << var2 << endl;

    tukarPointer(pointer1, pointer2);

    cout << "\nNilai setelah pertukaran pointer:" << endl;
    cout << "Var1: " << var1 << ", Var2: " << var2 << endl;

    return 0;
}
```

## V. KESIMPULAN

ADT adalah konsep penting dalam pemrograman berorientasi objek yang memungkinkan pemrogram untuk mendefinisikan tipe data baru dengan antarmuka yang jelas, terpisah dari implementasi internalnya. Dalam C++, ADT biasanya diimplementasikan menggunakan **kelas** dengan **enkapsulasi** melalui penggunaan private dan public untuk melindungi data internal dan mendefinisikan operasi-operasi yang dapat dilakukan pada data tersebut.

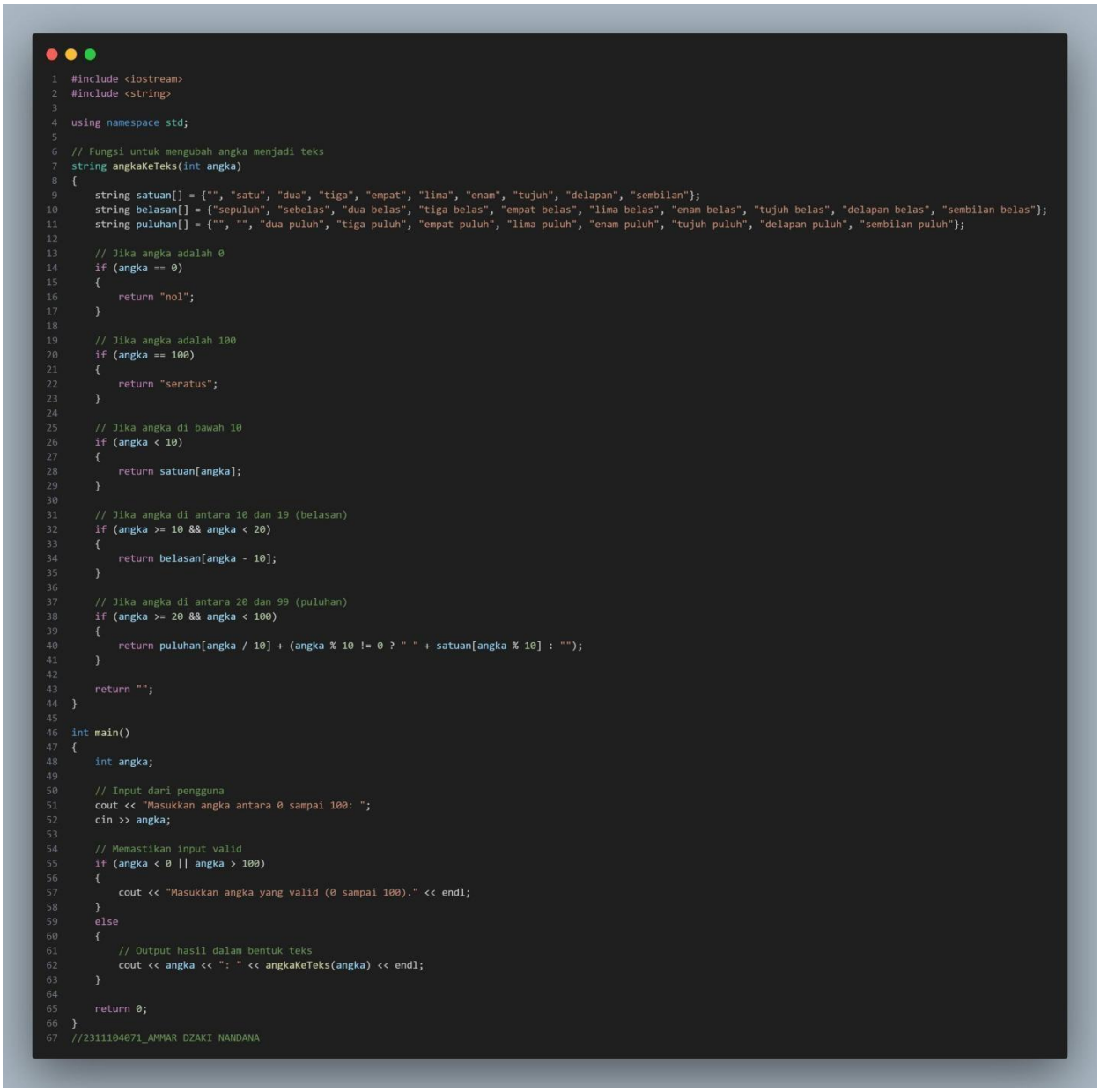
ADT adalah cara untuk mendefinisikan tipe data berdasarkan antarmuka atau operasi yang dapat dilakukan terhadapnya tanpa peduli bagaimana data tersebut diimplementasikan.
Tujuan utama dari ADT adalah abstraksi dan enkapsulasi, yang membantu menyederhanakan desain dan penggunaan struktur data serta memungkinkan pengembangan program yang lebih modular, aman, dan terstruktur.
ADT diterapkan dengan memisahkan antarmuka dari implementasi, yang memungkinkan pengguna bekerja pada level yang lebih tinggi tanpa harus memahami detail teknis dari bagaimana data dikelola di balik layar.

## 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.