

## Aturan Praktikum Struktur Data

1. **Akun GitHub:** Setiap praktikan wajib memiliki akun GitHub yang aktif dan digunakan selama praktikum berlangsung.
2. **Invite Collaborator:** Setiap praktikan diwajibkan untuk menambahkan collaborator di setiap repository
  - a. Asisten Praktikum: AndiniNH
  - b. Asisten Praktikum: 4ldiputra
3. **Repository Praktikum:** Setiap praktikan diwajibkan untuk membuat satu repository di GitHub yang akan digunakan untuk seluruh tugas dan laporan praktikum. Repository ini harus diatur dengan rapi dan sesuai dengan instruksi yang akan diberikan di lampiran.
4. **Penamaan Folder:** Penamaan folder dalam repository akan dibahas secara rinci di lampiran. Praktikan wajib mengikuti aturan penamaan yang telah ditentukan.

Nomor	Pertemuan	Penamaan
1	Pengalaman Bahasa C++ Bagian Pertama	01_Pengenalan_CPP_Bagian_1
2	Pengenalan Bahasa C++ Bagian Kedua	02_Pengenalan_CPP_Bagian_2
3	Abstract Data Type	03_Abstract_Data_Type
4	Single Linked List Bagian Pertama	04_Single_Linked_List_Bagian_1
5	Single Linked List Bagian Kedua	05_Single_Linked_List_Bagian_2
6	Double Linked List Bagian Pertama	06_Double_Linked_List_Bagian_1
7	Stack	07_Stack
8	Queue	08_Queue
9	Assessment Bagian Pertama	09_Assessment_Bagian_1
10	Tree Bagian Pertama	10_Tree_Bagian_1
11	Tree Bagian Kedua	11_Tree_Bagian_2
12	Asistensi Tugas Besar	12_Asistensi_Tugas_Besar

13	Multi Linked List	13_Multi_Linked_List
14	Graph	14_Graph
15	Assessment Bagian Kedua	15_Assessment_Bagian_2
16	Tugas Besar	16_Tugas_Besar

## 5. Jam Praktikum:

- Jam masuk praktikum adalah **1 jam lebih lambat** dari jadwal yang tercantum. Sebagai contoh, jika jadwal praktikum adalah pukul 06.30 - 09.30, maka aturan praktikum akan diatur sebagai berikut:
  - **06.30 - 07.30:** Waktu ini digunakan untuk **Tugas Praktikum dan Laporan Praktikum** yang dilakukan di luar laboratorium.
  - **07.30 - 08.30:** Sesi ini mencakup **tutorial, diskusi, dan kasus problem-solving**. Kegiatan ini berlangsung di dalam laboratorium dengan alokasi waktu sebagai berikut:
    - **60 menit pertama:** Tugas terbimbing.
    - **60 menit kedua:** Tugas mandiri.

6. **Pengumpulan Tugasn Pendahuluan:** Tugas Pendahuluan (TP) wajib dikumpulkan melalui GitHub sesuai dengan format berikut:

**nama\_repo/nama\_pertemuan/TP\_Pertemuan\_Ke.md**

Sebagai contoh:

**STD\_Yudha\_Islalmi\_Sulistya\_XXXXXXX/01\_Running\_Modul/TP\_01.md**

7. **Pengecekan Tugas Pendahuluan:** Pengumpulan laporan praktikum akan diperiksa **1 hari sebelum praktikum selanjutnya** dimulai. Pastikan tugas telah diunggah tepat waktu untuk menghindari sanksi.

**LAPORAN PRAKTIKUM**  
**MODUL 7**  
**STACK**



**Disusun Oleh :**

**Izzaty Zahara Br Barus – 2311104052**

**Kelas :**

**SE-07-02**

**Dosen :**

**Wahyu Andi Saputra, S.pd,M.Eng**

**PROGRAM STUDI SOFTWARE ENGINEERING**  
**FAKULTAS INFORMATIKA**  
**TELKOM UNIVERSITY**  
**PURWOKERTO**  
**2024**

## I. TUJUAN

1. Mampu memahami konsep stack pada struktur data dan algoritma
2. Mampu mengimplementasikan operasi-operasi pada stack
3. Mampu memecahkan permasalahan dengan solusi stack

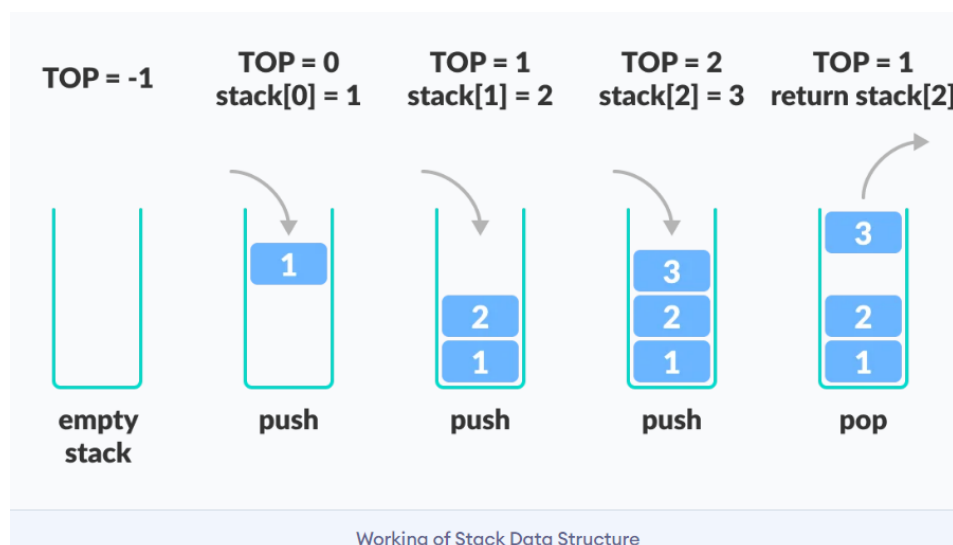
## II. LANDASAN TEORI

Sebuah stack atau tumpukan merupakan struktur data yang berfungsi untuk menyimpan dan mengelola kumpulan data dengan prinsip Last-In, First-Out (LIFO). Analogi yang sering digunakan adalah tumpukan piring di kafetaria, di mana piring terakhir yang ditambahkan akan menjadi yang pertama diambil.

Dalam implementasinya, stack dapat direpresentasikan sebagai struktur data terurut yang memiliki dua operasi utama: push dan pop. Operasi push digunakan untuk menambahkan elemen baru ke dalam stack, sementara operasi pop digunakan untuk menghapus elemen teratas dari stack.

Prinsip LIFO yang menjadi dasar stack membuatnya sangat bermanfaat dalam berbagai aplikasi, termasuk manajemen memori komputer, evaluasi ekspresi aritmatika, dan manajemen panggilan fungsi dalam pemrograman. Sebagai contoh, dalam manajemen memori, stack digunakan untuk menyimpan alamat-alamat memori yang dialokasikan untuk variabel dan fungsi.

Dengan prinsip LIFO ini, stack memungkinkan akses data dengan efisiensi, di mana elemen yang terakhir dimasukkan akan menjadi yang pertama diambil. Hal ini menjadikannya salah satu struktur data yang sangat penting dalam pengembangan perangkat lunak dan pemrograman komputer..



Operasi pada stack melibatkan beberapa fungsi dasar yang dapat dilakukan pada struktur

data ini. Berikut adalah beberapa operasi umum pada stack:

- a. Push (Masukkan): Menambahkan elemen ke dalam tumpukan pada posisi paling atas atau ujung.
- b. Pop (Keluarkan): Menghapus elemen dari posisi paling atas atau ujung tumpukan.
- c. Top (Atas): Mendapatkan nilai atau melihat elemen teratas pada tumpukan tanpa menghapusnya.
- d. IsEmpty (Kosong): Memeriksa apakah tumpukan kosong atau tidak.
- e. IsFull (Penuh): Memeriksa apakah tumpukan penuh atau tidak (terutama pada implementasi tumpukan dengan kapasitas terbatas).
- f. Size (Ukuran): Mengembalikan jumlah elemen yang ada dalam tumpukan.
- g. Peek (Lihat): Melihat nilai atau elemen pada posisi tertentu dalam tumpukan tanpa menghapusnya.
- h. Clear (Hapus Semua): Mengosongkan atau menghapus semua elemen dari tumpukan.
- i. Search (Cari): Mencari keberadaan elemen tertentu dalam tumpukan.

### III. GUIDE

#### 1. Guide

##### a. Syntax

```
#include <iostream>
#define MAX 100

using namespace std;

class Stack
{
private:
    int top;
    int arr[MAX];
    /* data */
public:
    Stack(/* args */) {top = -1;}
    bool isFull() { return top == MAX -1; }
    bool isEmpty(){ return top == -1; }

    void push(int x){
        if (isFull()){
```

```
        cout << "Stack Overflow\n";
        return;
    }
    arr[++top] = x;
}

void pop(){
    if (isEmpty()){
        cout << "Stack Underflow\n";
        return;
    }
    top --;
}

int peek(){
    if(!isEmpty()){
        return arr[top];
    }
    cout << "Stack is Empty\n";
    return -1;
}

void display(){
    if (isEmpty()){
        cout << "Stack is Empty\n";
        return;
    }
    for (int i = top; i >= 0; i--){
        cout << arr[i] << " ";
    }
    cout << "\n";
}

};

int main() {
    Stack s;
    s.push(10);
    s.push(20);
```

```
s.push(30);

cout << "Stack element: ";
s.display();

cout << "Top element: " << s.peek() << "\n";

s.pop();
s.pop();
cout << "After Popping, stack elements: ";
s.display();

return 0;
}
```

b. Penjelasan syntax

Definisi Kelas Stack:

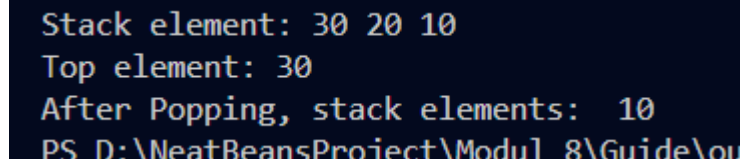
- Kelas Stack dibuat untuk merepresentasikan struktur data stack (LIFO - Last In, First Out).
- Atribut:
  - top: Menyimpan indeks elemen teratas dalam stack. Awalnya diinisialisasi dengan -1 (stack kosong).
  - arr[MAX]: Array statis berukuran MAX (dalam hal ini 100) untuk menyimpan elemen stack.
- Metode:
  - isFull(): Mengecek apakah stack penuh.
  - isEmpty(): Mengecek apakah stack kosong.
  - push(x): Menambahkan elemen x ke stack, dengan mengecek apakah stack penuh terlebih dahulu.
  - pop(): Menghapus elemen teratas dari stack, dengan mengecek apakah stack kosong.
  - peek(): Mengembalikan elemen teratas tanpa menghapusnya. Mengembalikan -1 jika stack kosong.
  - display(): Menampilkan elemen-elemen dalam stack dari atas ke bawah.

Bagian main():



- Membuat objek s dari kelas Stack.
- Menambahkan elemen ke stack menggunakan push():
  - Elemen 10, 20, dan 30 dimasukkan ke stack.
- Menampilkan elemen stack menggunakan display(), yang mencetak elemen dari atas ke bawah.
- Menggunakan peek() untuk mencetak elemen teratas (30).
- Menghapus dua elemen teratas dengan pop():
  - Setelah dua kali pop(), elemen yang tersisa hanya 10.
- Menampilkan kembali elemen stack dengan display().

c. Output



```
Stack element: 30 20 10
Top element: 30
After Popping, stack elements: 10
PS D:\MeatBeansProject\Modul 8\Guide\ou
```

2. Guide2

a. Code Program

```
#include <iostream>

using namespace std;

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

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

class Stack {
```

private:

Node\* top;

public:

Stack() { top = nullptr; }

bool isEmpty() { return top == nullptr; }

```
void push(int x) {  
    Node* newNode = new Node(x);  
    newNode->next = top;  
    top = newNode;  
}
```

```
void pop() {  
    if (isEmpty()) {  
        cout << "Stack Underflow\n";  
        return;  
    }  
    Node* temp = top;  
    top = top->next;  
    delete temp;  
}
```

```
int peek() {  
    if (!isEmpty()) {  
        return top->data;  
    }  
    cout << "Stack is empty\n";  
    return -1;  
}
```

```
void display() {
```

```
    if (isEmpty()) {
        cout << "Stack is empty\n";
        return;
    }
    Node* current = top;
    while (current) {
        cout << current->data << " ";
        current = current->next;
    }
    cout << endl;
}
};

int main() {
    Stack s;
    s.push(10);
    s.push(20);
    s.push(30);

    cout << "Stack elements: ";
    s.display();

    cout << "Top element: " << s.peek() << "\n";

    s.pop();
    s.pop();

    cout << "After popping, stack elements: ";
    s.display();

    return 0;
}
```

b. Penjelasan

Kode di atas adalah implementasi struktur data stack menggunakan linked list dalam bahasa C++. Stack adalah struktur data yang mengikuti prinsip LIFO (Last In, First Out), di mana elemen yang terakhir dimasukkan adalah elemen pertama yang dikeluarkan. Pada implementasi ini, setiap elemen stack direpresentasikan oleh sebuah node dalam linked list. Setiap node memiliki dua bagian: satu untuk menyimpan nilai data, dan satu lagi untuk menunjuk ke elemen berikutnya.

Kelas Node digunakan untuk membuat elemen stack, dengan atribut data untuk menyimpan nilai dan next untuk menunjuk ke node berikutnya. Kelas Stack memiliki pointer top yang menunjuk ke elemen teratas dalam stack. Beberapa operasi dasar yang diimplementasikan adalah:

- Push: Menambahkan elemen baru ke stack dengan membuat node baru dan menjadikannya elemen teratas.
- Pop: Menghapus elemen teratas dengan memindahkan pointer top ke elemen berikutnya dan membebaskan memori node yang dihapus.
- Peek: Mengambil nilai elemen teratas tanpa menghapusnya.
- Display: Menampilkan semua elemen dalam stack dari atas ke bawah.

Dalam fungsi main(), stack diuji dengan menambahkan elemen 10, 20, dan 30, yang kemudian ditampilkan menggunakan fungsi display. Elemen teratas (30) diperiksa menggunakan peek, lalu dua elemen teratas dihapus menggunakan pop. Setelah penghapusan, sisa elemen dalam stack ditampilkan kembali, yaitu hanya 10. Implementasi ini fleksibel karena menggunakan linked list, sehingga tidak dibatasi oleh ukuran tetap seperti array, tetapi memiliki overhead tambahan karena menggunakan pointer dan alokasi dinamis.

#### c. Output

```
PS D:\NeatBeansProject\Modul 8\Guide\output> & .\Guide
Stack elements: 30 20 10
Top element: 30
After popping, stack elements: 10
```

## IV. UNGUIDED

### 1. Task1

#### a. Code Program

```
#include <iostream>
```

```
#include <stack>
#include <cctype>
#include <string>

using namespace std;

bool isPalindrome(string str) {
    stack<char> s;
    string cleanStr;

    for (char c : str) {
        if (isalnum(c)) {
            cleanStr += tolower(c);
            s.push(tolower(c));
        }
    }

    for (char c : cleanStr) {
        if (c != s.top()) {
            return false;
        }
        s.pop();
    }
    return true;
}

int main() {
    string input;
    cout << "Masukkan kalimat: ";
    getline(cin, input);

    if (isPalindrome(input)) {
        cout << "Kalimat tersebut adalah palindrom." <<
endl;
    } else {
        cout << "Kalimat tersebut bukan palindrom." << endl;
    }
}
```

```
    return 0;  
}
```

b. Penjelasan

Kode di atas adalah program untuk memeriksa apakah sebuah kalimat adalah **palindrom** menggunakan struktur data stack. Palindrom adalah kalimat atau kata yang sama jika dibaca dari depan maupun dari belakang, seperti "madam" atau "A man a plan a canal Panama". Program ini pertama-tama membersihkan kalimat input dari karakter non-alfanumerik (seperti spasi dan tanda baca) dan mengonversinya menjadi huruf kecil agar perbandingan tidak peka huruf besar/kecil. Karakter-karakter bersih ini kemudian dimasukkan ke dalam stack, sebuah struktur data LIFO (Last In, First Out). Setelah semua karakter dimasukkan ke stack, program membandingkan karakter asli (dari string yang telah dibersihkan) dengan karakter yang dikeluarkan dari stack satu per satu. Jika ada karakter yang tidak cocok, kalimat tersebut bukan palindrom, dan program mengembalikan `false`. Jika seluruh karakter cocok, kalimat dianggap palindrom. Pada fungsi `main()`, program meminta pengguna untuk memasukkan sebuah kalimat. Kalimat tersebut akan diperiksa oleh fungsi `isPalindrome`, dan hasilnya ditampilkan kepada pengguna apakah kalimat tersebut adalah palindrom atau bukan. Program ini fleksibel karena dapat memeriksa kalimat yang mengandung spasi, tanda baca, atau campuran huruf besar dan kecil. Contohnya, jika pengguna memasukkan "A man a plan a canal Panama", program akan menganggapnya sebagai palindrom karena karakter aslinya tetap sama meskipun dibaca dari belakang.

c. Output

```
PS D:\NeatBeansProject\Modul 8\UNGUIDED\output> & .\
Masukkan kalimat: ini
Kalimat tersebut adalah palindrom.
PS D:\NeatBeansProject\Modul 8\UNGUIDED\output> cd .
PS D:\NeatBeansProject\Modul 8\UNGUIDED\output> & .\
Masukkan kalimat: disana
Kalimat tersebut bukan palindrom.
```

## 2. Task2

### a. Syntax

```
#include <iostream>
#include <stack>
#include <sstream>

using namespace std;

void reverseSentence(string str) {
    stack<string> s;
    string word;

    stringstream ss(str);
    while (ss >> word) {
        s.push(word);
    }

    cout << "Kalimat setelah dibalik: ";
    while (!s.empty()) {
        cout << s.top() << " ";
        s.pop();
    }
    cout << endl;
}

int main() {
    string input;
    cout << "Masukkan kalimat (minimal 3 kata): ";
    getline(cin, input);

    reverseSentence(input);

    return 0;
}
```

### b. Penjelasan Syntax

Kode di atas adalah program untuk membalikkan urutan kata dalam sebuah kalimat menggunakan struktur data stack. Stack digunakan karena sifatnya yang LIFO (Last In, First Out), sehingga kata yang terakhir dimasukkan akan menjadi yang pertama dikeluarkan. Program bekerja dengan cara membaca kalimat yang dimasukkan oleh pengguna, lalu memisahkan setiap kata menggunakan objek `stringstream`. Kata-kata yang berhasil dipisahkan dimasukkan ke dalam stack satu per satu. Setelah semua kata berada di dalam stack, program mulai mengeluarkan kata dari stack (menggunakan operasi `pop()`) dan mencetaknya ke layar. Karena stack bekerja dengan urutan LIFO, urutan kata dalam kalimat akan terbalik. Contohnya, jika pengguna memasukkan kalimat "Belajar pemrograman itu menyenangkan", program akan membalikkan kalimat menjadi "menyenangkan itu pemrograman Belajar". Program ini memastikan setiap kata diproses secara individual dan tidak bergantung pada jumlah kata, selama input berisi minimal tiga kata seperti yang diminta. Secara keseluruhan, program ini adalah cara sederhana dan efektif untuk memanfaatkan stack untuk manipulasi string berbasis kata.

c. Output

```
PS D:\NeatBeansProject\Modul 8\UNGUIDED\output> & .\'TASK2.exe'  
Masukkan kalimat (minimal 3 kata): institut telkom purwokerto broo  
Kalimat setelah dibalik: broo purwokerto telkom institut  
PS D:\NeatBeansProject\Modul 8\UNGUIDED\output>
```

## V. KESIMPULAN

Stack merupakan struktur data yang mengikuti prinsip Last In, First Out (LIFO), di mana elemen terakhir yang dimasukkan adalah elemen pertama yang dikeluarkan. Struktur ini sangat berguna dalam berbagai konteks pemrograman, seperti manajemen memori, evaluasi ekspresi aritmatika, serta dalam pengelolaan panggilan fungsi dalam program. Stack dapat diimplementasikan menggunakan array atau linked list, dengan operasi dasar seperti push, pop, peek, isEmpty, dan display. Melalui implementasi stack dengan array dan linked list, kita dapat menyelesaikan berbagai masalah praktis, seperti memeriksa apakah suatu kalimat adalah palindrom dan membalikkan urutan kata dalam kalimat. Penggunaan stack untuk memecahkan masalah-masalah ini menunjukkan fleksibilitasnya dalam



menangani data secara efisien. Dengan operasi yang sederhana namun kuat, stack memungkinkan kita untuk mengelola data dengan cara yang sangat terstruktur dan efektif. Hal ini menjadikannya alat yang sangat penting dalam pemrograman, baik untuk tugas sehari-hari maupun untuk aplikasi yang lebih kompleks.