LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL VII
STACK

Disusun Oleh :

NAMA : Taufik Hafit Zakaria

NIM : 103112430093

Dosen

WAHYU ANDI SAPUTRA

PROGRAM STUDI STRUKTUR DATA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2025

A. Dasar Teori

tack (tumpukan) merupakan salah satu bentuk struktur data linear yang menerapkan prinsip operasi LIFO (Last In First Out). Sesuai dengan analogi sebuah tumpukan, elemen yang terakhir kali dimasukkan adalah elemen yang pertama kali akan dikeluarkan. Semua operasi pada stack, seperti penyisipan dan pengambilan data, hanya dapat dilakukan pada satu ujung yang disebut TOP (puncak). Stack dapat diimplementasikan menggunakan dua representasi utama: representasi pointer (menggunakan konsep linked list) yang dinamis, atau representasi tabel (menggunakan array) yang memiliki ukuran terbatas.

Karakteristik Stack

- Prinsip LIFO: Operasi didasarkan pada prinsip Last In First Out
- Akses Terbatas di TOP: Elemen hanya bisa diakses, ditambah, atau dihapus melalui puncak (TOP).
- Operasi Utama (Push & Pop): Memiliki dua operasi utama: Push (menambah elemen ke TOP) dan Pop (mengambil elemen dari TOP).

Keuntungan Stack

- Cepat dan Efisien: Operasi Push dan Pop sangat cepat (kompleksitas O(1)) dan efisien dalam manajemen memori, seperti yang digunakan pada pemanggilan fungsi di program.
- Pembalikan Data: Sangat ideal untuk membalikkan urutan data secara alami karena sifat LIFO-nya.

Keterbatasan Stack

- Akses Tidak Fleksibel: Elemen di tengah atau dasar tumpukan tidak bisa diakses tanpa mengeluarkan elemen di atasnya terlebih dahulu.
- Ukuran Statis (Array): Implementasi dengan array memiliki ukuran tetap, yang berisiko menyebabkan stack overflow (tumpukan penuh).
- Risiko Underflow: Berisiko mengalami underflow jika mencoba mengambil elemen dari stack yang sudah kosong.

B. Guided (berisi screenshot source code & output program disertai penjelasannya)

Guided 1

```cpp
//Taufik Hafit Zakaria
//103112430093_12 - IF 06
// File guided

#include <iostream>
using namespace std;

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

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

void push(Node *&top, int data){
    Node *newNode = new Node();
    newNode->data = data;
    newNode->next = top;
    top = newNode;
}
int pop(Node *&top){
    if(isEmpty(top)){
        cout << "Stack kosong, tidak bisa pop!" << endl;
        return 0;
    }
    int poppedData = top->data;
    Node *temp = top;
    top = top->next;

    delete temp;
    return poppedData;
}

```

```
36  void show(Node *top){
37      if(isEmpty(top)){
38          cout << "Stack kosong." << endl;
39          return;
40      }
41      cout << "TOP -> ";
42      Node *temp = top;
43      while(temp != nullptr){
44          cout << temp->data << " -> ";
45          temp = temp->next;
46      }
47      cout << "NULL" << endl;
48  }
49
50  int main() {
51      Node *stack = nullptr;
52      push(stack, 10);
53      push(stack, 20);
54      push(stack, 30);
55      cout << "menampilkan isi stack: " << endl;
56      show(stack);
57      cout << "POP: " << pop(stack) << endl;
58      cout << "menampilkan sisa stack: " << endl;
59      show(stack);
60      return 0;
61  }
62
```

Screenshots Output

```
PS C:\programing\Struktur Data\LAPRAK 7\unguided> cd "c:\programing\St
ruktur Data\LAPRAK 7\" ; if ($?) { g++ stack_1.cpp -o stack_1 } ; if (
$?) { .\stack_1 }
menampilkan isi stack:
TOP -> 30 -> 20 -> 10 -> NULL
POP: 30
menampilkan sisa stack:
TOP -> 20 -> 10 -> NULL
PS C:\programing\Struktur Data\LAPRAK 7>
```

Deskripsi:

Program ini adalah implementasi struktur data Stack menggunakan C++ dengan pendekatan pointer, yang pada dasarnya bekerja seperti Singly Linked List. Intinya, program ini menerapkan prinsip LIFO (Last In First Out), di mana data terakhir yang masuk akan menjadi yang pertama kali keluar. Seluruh kode, termasuk struktur Node, fungsi-fungsi inti seperti push (mirip insertFirst) dan pop (mirip deleteFirst), serta fungsi main, saya satukan dalam satu file agar lebih simpel.

Alur program di fungsi main secara langsung mendemonstrasikan cara kerjanya. Awalnya, saya memasukkan tiga nilai (10, 20, 30) ke dalam stack menggunakan push. Setelah menampilkan isinya, saya melakukan satu kali pop untuk mengambil elemen teratas (yaitu 30), dan terakhir menampilkan kembali sisa stack. Hasil akhir dengan jelas menunjukkan bahwa elemen 30 berhasil ditambahkan di puncak dan kemudian berhasil dikeluarkan, membuktikan bahwa prinsip LIFO berjalan dengan benar dan implementasi dinamis ini bekerja sesuai harapan.

C. Unguided/Tugas (berisi screenshot source code & output program disertai penjelasannya)

Unguided 1

```h
//Taufik Hafit Zakaria
//103112430093_12 - IF 06
// File: stack.h

#ifndef STACK_H
#define STACK_H

typedef int infotype;

struct Stack {
    infotype info[20];
    int top;
};

// Membuat stack kosong
void createStack(Stack &S);

// Menambah elemen ke stack
void push(Stack &S, infotype x);

// Menghapus elemen dari stack dan mengembalikannya
infotype pop(Stack &S);

// Menampilkan isi stack dari TOP ke bawah
void printInfo(Stack S);

// Membalik urutan elemen dalam stack
void balikStack(Stack &S);

// Menyisipkan elemen agar stack berurutan ascending
void pushAscending(Stack &S, infotype x);

// Membaca input deretan angka tanpa spasi hingga Enter
void getInputStream(Stack &S);

// Mengecek apakah stack kosong
bool isEmpty(Stack S);

#endif
```

```cpp
//Taufik Hafit Zakaria
//103112430093_12 - IF 06
// File: stack.cpp

#include <iostream>
#include "stack.h"
using namespace std;

// Membuat stack kosong
void createStack(Stack &S) {
    S.top = 0;
}

// Mengecek apakah stack kosong
bool isEmpty(Stack S) {
    return (S.top == 0);
}

// Menambah elemen ke stack
void push(Stack &S, infotype x) {
    if (S.top == 20) {
        cout << "Stack penuh!" << endl;
        return;
    }
    S.top++;
    S.info[S.top] = x;
}

// Mengambil dan menghapus elemen teratas
infotype pop(Stack &S) {
    if (isEmpty(S)) {
        cout << "Stack kosong!" << endl;
        return -1;
    }
    infotype temp = S.info[S.top];
    S.top--;
    return temp;
}

```

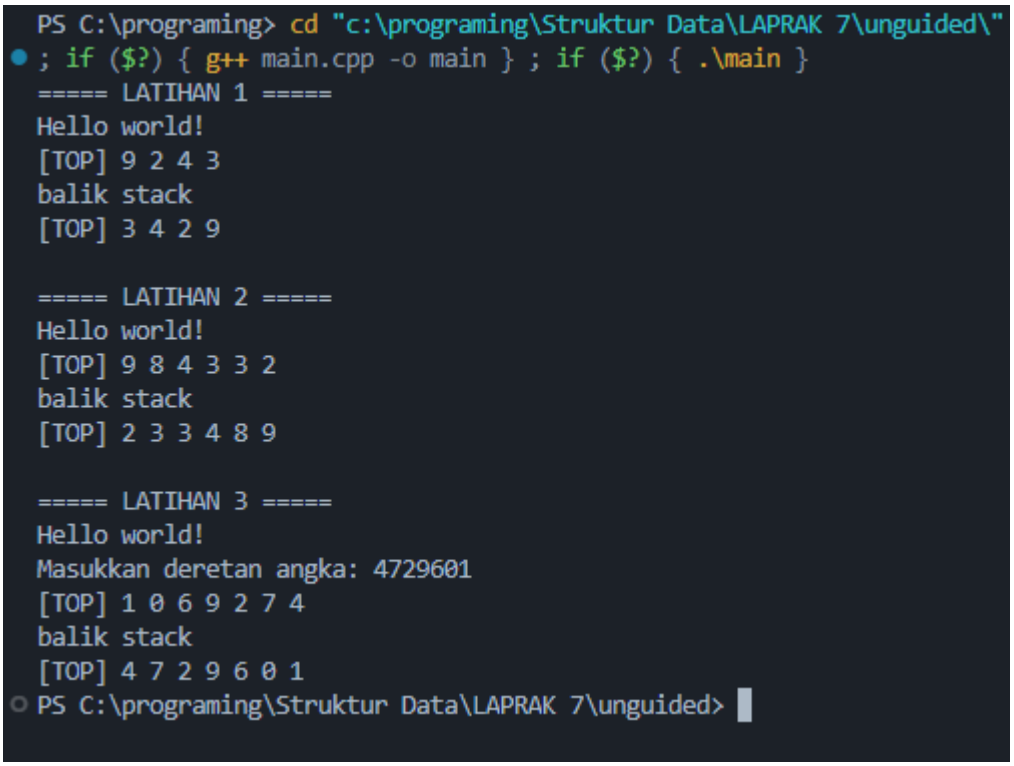
```cpp
// Menampilkan isi stack dari TOP ke bawah
void printInfo(Stack S) {
    if (isEmpty(S)) {
        cout << "Stack kosong!" << endl;
    } else {
        cout << "[TOP] ";
        for (int i = S.top; i >= 1; i--) {
            cout << S.info[i] << " ";
        }
        cout << endl;
    }
}

// Membalik isi stack
void balikStack(Stack &S) {
    Stack temp;
    createStack(temp);

    while (!isEmpty(S)) {
        push(temp, pop(S));
    }

    S = temp;
}

// Push ascending (urut dari bawah ke atas)
void pushAscending(Stack &S, infotype x) {
    Stack temp;
    createStack(temp);

    // Pindahkan elemen yang lebih besar dari x
    while (!isEmpty(S) && S.info[S.top] > x) {
        push(temp, pop(S));
    }

    // Tambahkan elemen baru
    push(S, x);

```

```
78      // Kembalikan elemen dari stack sementara
79      while (!isEmpty(temp)) {
80          push(S, pop(temp));
81      }
82  }
83
84  // Membaca input angka tanpa spasi (misal: 4729601)
85  void getInputStream(Stack &S) {
86      cout << "Masukkan deretan angka: ";
87
88      string input;
89      getline(cin, input);
90
91      for (char c : input) {
92          if (isdigit(c)) {
93              push(S, c - '0'); // ubah char ke int
94          }
95      }
96  }
97
```

```cpp
//Taufik Hafit Zakaria
//103112430093_12 - IF 06
// File: main.cpp

#include <iostream>
#include "stack.h"
#include "stack.cpp"
using namespace std;

int main() {
    // ========== LATIHAN 1 ==========
    cout << "===== LATIHAN 1 =====" << endl;
    cout << "Hello world!" << endl;

    Stack S1;
    createStack(S1);

    push(S1, 3);
    push(S1, 4);
    push(S1, 8);
    pop(S1);
    push(S1, 2);
    push(S1, 3);
    pop(S1);
    push(S1, 9);

    printInfo(S1);
    cout << "balik stack" << endl;
    balikStack(S1);
    printInfo(S1);

```

```
32      // ========== LATIHAN 2 ==========
33      cout << "\n===== LATIHAN 2 =====" << endl;
34      cout << "Hello world!" << endl;
35
36      Stack S2;
37      createStack(S2);
38
39      pushAscending(S2, 3);
40      pushAscending(S2, 4);
41      pushAscending(S2, 8);
42      pushAscending(S2, 2);
43      pushAscending(S2, 3);
44      pushAscending(S2, 9);
45
46      printInfo(S2);
47      cout << "balik stack" << endl;
48      balikStack(S2);
49      printInfo(S2);
50
51      // ========== LATIHAN 3 ==========
52      cout << "\n===== LATIHAN 3 =====" << endl;
53      cout << "Hello world!" << endl;
54
55      Stack S3;
56      createStack(S3);
57
58      getInputStream(S3);
59      printInfo(S3);
60      cout << "balik stack" << endl;
61      balikStack(S3);
62      printInfo(S3);
63
64      return 0;
65  }
66
```

Screenshots Output



Deskripsi:

Program ini adalah implementasi struktur data Stack menggunakan C++ dengan pendekatan representasi tabel (array). Intinya, program ini menerapkan prinsip LIFO (Last In First Out), di mana data terakhir yang masuk akan menjadi yang pertama kali keluar. Struktur kodenya saya pecah menjadi tiga file: stack.h untuk deklarasi struktur dan fungsi, stack.cpp untuk implementasi detailnya, dan main.cpp sebagai program utama. Selain fungsi-fungsi dasar seperti createStack, push, dan pop, saya juga membuat fungsi tambahan yang lebih kompleks seperti balikStack untuk membalik urutan, pushAscending untuk menyisipkan data secara terurut, dan getInputStream untuk membaca deretan angka dari input pengguna.

Alur program di fungsi main secara langsung mendemonstrasikan cara kerja semua fungsi tersebut melalui tiga bagian latihan. Pada Latihan 1, saya menguji operasi dasar push dan pop, lalu menggunakan balikStack untuk membalik isinya. Pada Latihan 2, saya mendemonstrasikan pushAscending dengan memasukkan angka secara acak, yang hasilnya menunjukkan stack tetap tersusun secara menaik. Terakhir, pada Latihan 3, saya menggunakan getInputStream untuk mengisi stack dari input pengguna. Setiap latihan diakhiri dengan menampilkan kondisi stack sebelum dan sesudah dibalik, yang membuktikan bahwa semua fungsi berjalan sesuai harapan.

D. Kesimpulan

Dari hasil praktikum yang telah dilakukan, dapat disimpulkan bahwa struktur data Stack memungkinkan pengelolaan data secara efisien berdasarkan prinsip LIFO (Last In First Out). Melalui implementasi program latihan ini, konsep dasar operasi stack seperti createStack, push, dan pop, serta algoritma yang lebih kompleks seperti balikStack untuk membalik urutan dan pushAscending untuk penyisipan terurut, dapat dipahami dan diterapkan dengan baik. Praktikum ini juga mendemonstrasikan implementasi stack menggunakan representasi tabel (array) yang bersifat statis, di mana akses data dibatasi hanya pada elemen teratas (TOP).

Selain itu, program ini membuktikan bahwa penggunaan Stack sangat berguna dalam kasus-kasus yang secara alami memerlukan pemrosesan data secara terbalik, seperti pada mekanisme undo, pemanggilan fungsi, atau evaluasi ekspresi. Dengan memahami prinsip dasar LIFO dan cara kerjanya, mahasiswa dapat secara efektif menerapkan Stack untuk memecahkan berbagai masalah algoritmik. Pemahaman ini juga menjadi fondasi penting untuk mempelajari struktur data lain seperti Queue serta algoritma yang lebih lanjut yang memanfaatkan Stack dalam implementasinya, seperti Depth-First Search (DFS) pada graph.

E. Referensi

GeeksforGeeks. (2024). *Stack Data Structure*. Diakses pada tahun 2025 dari https://www.geeksforgeeks.org/stack-data-structure/

Programiz. (2024). *Stack Data Structure*. Diakses pada tahun 2025 dari https://www.programiz.com/dsa/stack

Kelas Terbuka. (2023). *Stack dan Queue | Struktur Data*. Diakses pada tahun 2025 dari https://www.kelasterbuka.id/materi/stack-dan-queue-struktur-data/