LAPORAN PRAKTIKUM
STRUKTUR DATA

MODUL VII
STACK

Disusun Oleh :
NAMA : DEVI YULIANA
NIM : 103112400151

Dosen
FAHRUDIN MUKTI WIBOWO
STRUKTUR DATA

PROGRAM STUDI S1 TEKNIK INFORMATIKA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025

A. Dasar Teori

Stack adalah salah satu bentuk struktur data yang cara kerjanya seperti tumpukan barang. Prinsipnya disebut LIFO (Last In First Out), yang artinya data yang terakhir dimasukkan akan menjadi data yang pertama diambil. Bayangkan seperti tumpukan piring — piring yang paling atas akan diambil duluan.

Dalam stack, ada satu penunjuk utama yang disebut TOP, yaitu bagian paling atas dari tumpukan tempat data bisa ditambah (push) atau diambil (pop). Dua operasi utama dalam stack adalah Push, untuk menambahkan elemen ke dalam stack dan Pop untuk mengambil elemen dari bagian atas stack.

Stack bisa dibuat dengan dua cara, yaitu menggunakan pointer (linked list) atau array (tabel).

Kalau menggunakan pointer, ukuran stack bisa fleksibel sesuai jumlah data yang dialokasikan. Sedangkan jika menggunakan array, ukuran stack sudah ditentukan dari awal (misalnya 10 atau 20 elemen saja).

Selain itu, ada beberapa fungsi dasar (primitif) seperti createStack(), isEmpty(), push(), pop(), dan viewStack(). Fungsi-fungsi ini memudahkan kita untuk mengatur data di dalam stack agar bisa dibuat, dicek, ditambah, dihapus, atau ditampilkan dengan mudah.

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

Guided 1

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

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

Node *ptr_first = NULL;
```

```
Node *ptr_last = NULL;

void add_first(int value)
{
    Node *newNode = new Node{value, NULL, ptr_first};

    if (ptr_first == NULL)
    {
        ptr_last = newNode;
    }
    else
    {
        ptr_first -> prev = newNode;
    }
    ptr_first = newNode;
}

void add_last(int value)
{
    Node *newNode = new Node{value, ptr_last, NULL};

    if (ptr_last == NULL)
    {
        ptr_first = newNode;
    }
    else
    {
        ptr_last -> next = newNode;
    }
    ptr_last = newNode;
}

void add_target(int targetValue, int newValue)
{
    Node *current = ptr_first;
    while(current != NULL && current -> data != targetValue)
    {
        current = current -> next;
    }
```

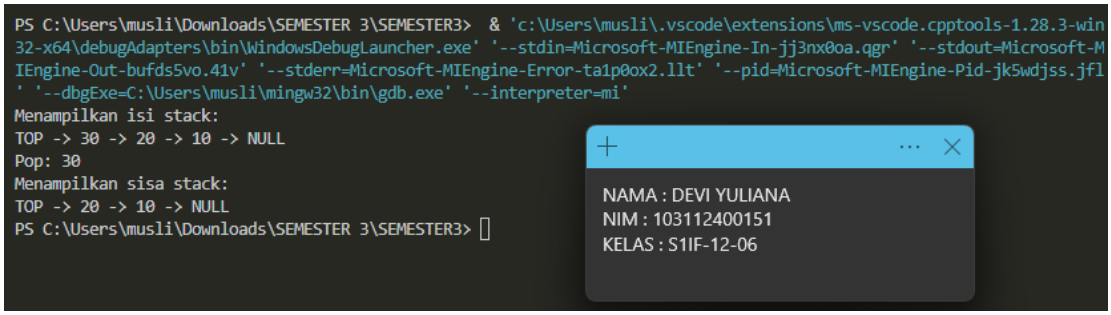
```
    if (current != NULL)
    {
        if(current == ptr_last)
        {
            add_last(newValue);
        }
        else
        {
            Node *newNode = new Node {newValue, current, current ->
next};
            current -> next -> prev = newNode;
            current -> next = newNode;
        }
    }
}

void view()
{
    Node *current = ptr_first;
    if (current == NULL)
    {
        cout << "List Kosong\n";
        return;
    }
    while (current != NULL)
    {
        cout << current -> data << (current -> next != NULL ? " <-> " :
"");
        current = current -> next;
    }
    cout << endl;
}

void delete_first()
{
    if (ptr_first == NULL)
        return;

    Node *temp = ptr_first;
```

```
    if (ptr_first == ptr_last)
    {
        ptr_first = NULL;
        ptr_last = NULL;
    }
    else
    {
        ptr_first = ptr_first -> next;
        ptr_first -> prev = NULL;
    }
    delete temp;
}

void delete_last()
{
    if (ptr_last == NULL)
        return;

    Node *temp = ptr_last;

    if (ptr_first == ptr_last)
    {
        ptr_first = NULL;
        ptr_last = NULL;
    }
    else
    {
        ptr_last = ptr_last -> prev;
        ptr_last -> next = NULL;
    }
    delete temp;
}

void delete_target(int targetValue)
{
    Node *current = ptr_first;
    while(current != NULL && current -> data != targetValue)
    {
        current = current -> next;
    }
```

```
    if (current != NULL)
    {
        if(current == ptr_first)
        {
            delete_first();
            return;
        }
        if(current == ptr_last)
        {
            delete_last();
            return;
        }

        current -> prev -> next = current -> next;
        current -> next -> prev = current -> prev;
        delete current;
    }
}

void edit_node(int targetValue, int newValue)
{
    Node *current = ptr_first;
    while(current != NULL && current -> data != targetValue)
    {
        current = current -> next;
    }

    if (current != NULL)
    {
        current -> data = newValue;
    }
}

int main()
{
    add_first(10);
    add_first(5);
    add_last(20);
    cout << "Awal\t\t\t: ";
```

```
    view();

    delete_first();
    cout << "Setelah delete_first\t: ";
    view();
    delete_last();
    cout << "Setelah delete_last\t: ";
    view();

    add_last(30);
    add_last(40);
    cout << "Setelah tambah\t\t: ";
    view();

    delete_target(30);
    cout << "Setelah delete_target\t: ";
    view();
}
```

Screenshots Output



Deskripsi :

Struktur utama Node digunakan untuk menyimpan data dan penunjuk ke node berikutnya. Dengan begitu, setiap elemen bisa saling terhubung membentuk rantai data. Fungsi isEmpty() dipakai untuk mengecek apakah stack kosong, sedangkan push() digunakan untuk menambah data baru ke atas stack. Setiap kali push() dijalankan, program membuat node baru dan menjadikannya elemen teratas.

Fungsi pop() berfungsi untuk menghapus dan mengambil data paling

atas dari stack. Jika stack kosong, program akan menampilkan pesan error. Selain itu, ada juga fungsi show() yang menampilkan isi stack dari atas ke bawah agar pengguna bisa melihat urutannya.

Di bagian main(), program membuat stack kosong lalu menambahkan data 10, 20, dan 30. Setelah itu, isi stack ditampilkan, kemudian satu data paling atas (30) dihapus dengan pop(), dan hasil akhirnya ditampilkan kembali (menyisakan 20 dan 10).

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

Unguided 1

stack.h

```
#ifndef STACK_H
#define STACK_H
#include <iostream>
using namespace std;

const int maxEl = 20;
typedef int infotype;

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

void createStack(Stack &S) {
    S.top = -1;
}

bool isEmpty(Stack S) {
    return S.top == -1;
}

bool isFull(Stack S) {
    return S.top == maxEl - 1;
}

void push(Stack &S, infotype x) {
```

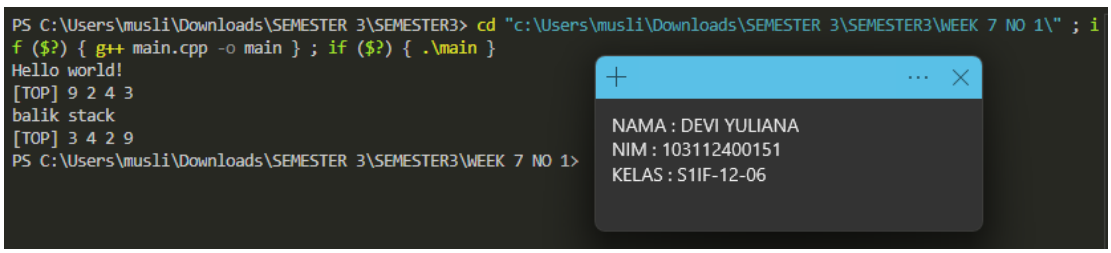
```
    if (isFull(S)) {
        cout << "Stack penuh!" << endl;
    } else {
        S.top++;
        S.info[S.top] = x;
    }
}

infotype pop(Stack &S) {
    if (isEmpty(S)) {
        cout << "Stack kosong!" << endl;
        return -1;
    } else {
        infotype x = S.info[S.top];
        S.top--;
        return x;
    }
}

void printInfo(Stack S) {
    if (isEmpty(S)) {
        cout << "Stack kosong!" << endl;
    } else {
        cout << "[TOP] ";
        for (int i = S.top; i >= 0; i--) {
            cout << S.info[i] << " ";
        }
        cout << endl;
    }
}

void balikStack(Stack &S) {
    Stack temp;
    createStack(temp);

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

    S = temp;
}
```

```
#endif
```

main.cpp

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

int main() {
    cout << "Hello world!" << endl;

    Stack S;
    createStack(S);

    push(S, 3);
    push(S, 4);
    push(S, 2);
    push(S, 9);

    printInfo(S);

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

    return 0;
}
```

Screenshots Output



Deskripsi:

Program ini memiliki beberapa fungsi untuk mengelola data di dalam stack. Fungsi createStack() digunakan untuk menginisialisasi stack agar siap

digunakan, sedangkan isEmpty() dan isFull() masing-masing berfungsi untuk mengecek apakah stack sedang kosong atau sudah penuh. Fungsi push() digunakan untuk menambahkan data baru ke dalam stack. Jika stack belum penuh, data dimasukkan ke posisi paling atas dan nilai top akan bertambah satu. Sebaliknya, fungsi pop() digunakan untuk menghapus dan mengambil data paling atas. Jika stack kosong, program akan menampilkan pesan bahwa stack tidak bisa di-pop.

Selain itu, ada fungsi printInfo() yang berfungsi untuk menampilkan isi stack dari elemen paling atas ke bawah agar pengguna bisa melihat urutan datanya. Program ini juga memiliki fungsi tambahan bernama balikStack(), yang berguna untuk membalik urutan isi stack. Caranya dengan membuat stack sementara, lalu semua elemen dari stack utama dipindahkan ke stack sementara menggunakan operasi pop dan push, sehingga urutannya menjadi terbalik.

Unguided 2

stack.h

```
#ifndef STACK_H
#define STACK_H
#include <iostream>
using namespace std;

const int maxEl = 20;
typedef int infotype;

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

void createStack(Stack &S) {
    S.top = -1;
}

bool isEmpty(Stack S) {
    return S.top == -1;
}

bool isFull(Stack S) {
```

```
    return S.top == maxEl - 1;
}

void push(Stack &S, infotype x) {
    if (isFull(S)) {
        cout << "Stack penuh!" << endl;
    } else {
        S.top++;
        S.info[S.top] = x;
    }
}

infotype pop(Stack &S) {
    if (isEmpty(S)) {
        cout << "Stack kosong!" << endl;
        return -1;
    } else {
        infotype x = S.info[S.top];
        S.top--;
        return x;
    }
}

void printInfo(Stack S) {
    if (isEmpty(S)) {
        cout << "Stack kosong!" << endl;
    } else {
        cout << "[TOP] ";
        for (int i = S.top; i >= 0; i--) {
            cout << S.info[i] << " ";
        }
        cout << endl;
    }
}

void balikStack(Stack &S) {
    Stack temp;
    createStack(temp);

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

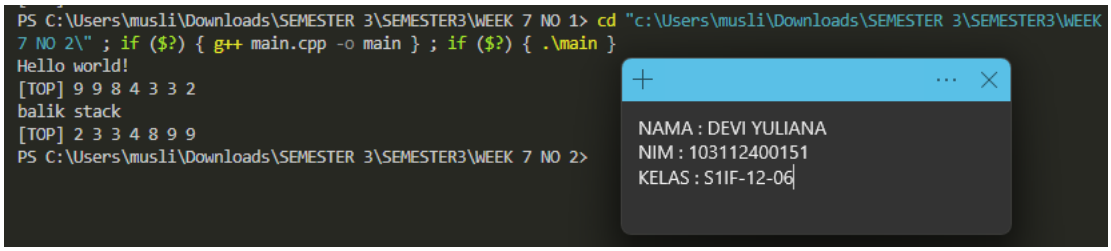
```
    }

    S = temp;
}

void pushAscending(Stack &S, infotype x) {
    Stack temp;
    createStack(temp);

    while (!isEmpty(S) && S.info[S.top] > x) {
        push(temp, pop(S));
    }

    push(S, x);

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

#endif
```

main.cpp

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

int main() {
    cout << "Hello world!" << endl;

    Stack S;
    createStack(S);

    pushAscending(S, 3);
    pushAscending(S, 9);
    pushAscending(S, 8);
    pushAscending(S, 4);
    pushAscending(S, 3);
    pushAscending(S, 2);
    pushAscending(S, 9);
```

```
    printInfo(S);

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

    return 0;
}
```

Screenshots Output



Deskripsi:

Struktur utama program terdapat pada stack.h, di mana didefinisikan sebuah struct Stack yang berisi array info[maxEl] untuk menyimpan data dengan kapasitas maksimum 20 elemen, serta variabel top untuk menandai posisi elemen paling atas di stack. Nilai awal top diatur ke -1 agar menandakan stack masih kosong.

Fungsi-fungsi dasar seperti createStack(), isEmpty(), dan isFull() digunakan untuk membuat stack baru dan memeriksa apakah stack kosong atau sudah penuh. Fungsi push() berfungsi untuk menambahkan data baru ke stack, sedangkan pop() digunakan untuk mengambil sekaligus menghapus data paling atas dari stack. Fungsi printInfo() menampilkan isi stack dari atas ke bawah, lengkap dengan tanda [TOP] di bagian awal agar urutan data terlihat jelas.

Fungsi tambahan balikStack() digunakan untuk membalik urutan elemen di dalam stack. Prosesnya dilakukan dengan membuat stack sementara (temp) yang menampung hasil pop dari stack utama, lalu hasilnya dimasukkan kembali ke stack utama sehingga urutannya menjadi terbalik. Sementara itu, fungsi pushAscending() digunakan untuk menambahkan elemen baru ke stack agar tetap dalam urutan menaik (ascending order). Cara kerjanya, program

akan memindahkan sementara elemen-elemen yang lebih besar dari nilai baru ke stack lain (temp), lalu memasukkan elemen baru, dan mengembalikan elemen-elemen sebelumnya sehingga urutan tetap terjaga.

Pada file main.cpp, program dimulai dengan membuat stack baru menggunakan createStack(S). Setelah itu, beberapa nilai dimasukkan ke stack dengan fungsi pushAscending(), yaitu 3, 9, 8, 4, 3, 2, dan 9. Karena menggunakan metode ascending, elemen-elemen akan tersimpan secara berurutan dari kecil ke besar. Setelah semua data dimasukkan, isi stack ditampilkan menggunakan printInfo(). Kemudian, program membalik urutan stack dengan memanggil balikStack(S) dan menampilkannya kembali.

Unguided 3

stack.h

```
#ifndef STACK_H
#define STACK_H
#include <iostream>
#include <string>
using namespace std;

const int maxEl = 20;
typedef int infotype;

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

void createStack(Stack &S) {
    S.top = -1;
}

bool isEmpty(Stack S) {
    return S.top == -1;
}

bool isFull(Stack S) {
    return S.top == maxEl - 1;
}
```

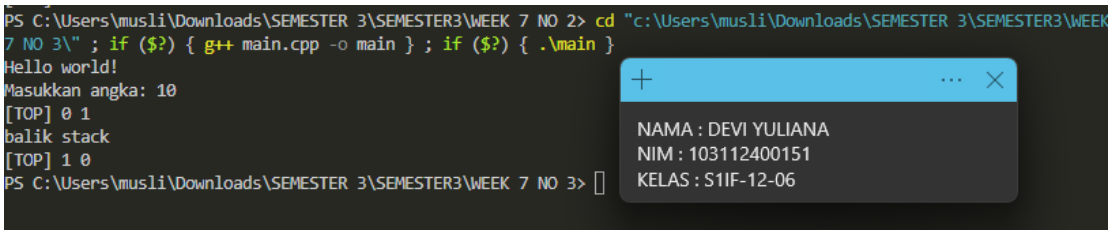
```cpp
void push(Stack &S, infotype x) {
    if (isFull(S)) {
        cout << "Stack penuh!" << endl;
    } else {
        S.top++;
        S.info[S.top] = x;
    }
}

infotype pop(Stack &S) {
    if (isEmpty(S)) {
        cout << "Stack kosong!" << endl;
        return -1;
    } else {
        infotype x = S.info[S.top];
        S.top--;
        return x;
    }
}

void printInfo(Stack S) {
    if (isEmpty(S)) {
        cout << "Stack kosong!" << endl;
    } else {
        cout << "[TOP] ";
        for (int i = S.top; i >= 0; i--) {
            cout << S.info[i] << " ";
        }
        cout << endl;
    }
}

void balikStack(Stack &S) {
    Stack temp;
    createStack(temp);

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

```
void getInputStream(Stack &S) {
    string input;
    cout << "Masukkan angka: ";
    getline(cin, input);

    for (char c : input) {
        if (isdigit(c)) {
            int x = c - '0';
            push(S, x);
        }
    }
}
#endif
```

main.cpp

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

int main() {
    cout << "Hello world!" << endl;

    Stack S;
    createStack(S);

    getInputStream(S);
    printInfo(S);

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

    return 0;
}
```

Screenshots Output



Deskripsi:

Struktur Stack memiliki dua bagian utama: array info[maxEl] sebagai tempat menyimpan data dengan kapasitas maksimal 20 elemen, dan variabel top untuk menandai posisi elemen paling atas. Saat stack dibuat dengan createStack(), nilai top diatur ke -1 sebagai tanda bahwa stack masih kosong. Fungsi isEmpty() dan isFull() digunakan untuk memeriksa kondisi stack, sedangkan push() dan pop() masing-masing berfungsi menambah dan menghapus elemen dari tumpukan.

Fungsi getInputStream() memungkinkan pengguna mengetikkan deretan angka. Setiap karakter angka dari input dikonversi menjadi bilangan bulat dan langsung dimasukkan ke dalam stack menggunakan push(). Setelah itu, isi stack dapat ditampilkan melalui printInfo(). Fungsi tambahan balikStack() digunakan untuk membalik urutan elemen di dalam stack dengan bantuan stack sementara.

Pada main.cpp, program membuat stack baru, meminta input angka dari pengguna, lalu menampilkan isi stack. Setelah itu, program membalik urutan stack dan menampilkannya kembali. Secara singkat, program ini menunjukkan cara menggunakan stack untuk menyimpan, menampilkan, dan membalik data yang dimasukkan pengguna dengan cara yang sederhana dan efisien.

D. Kesimpulan

Stack adalah struktur data yang bekerja seperti tumpukan barang dengan prinsip LIFO — data yang terakhir masuk akan keluar pertama kali. Dengan hanya bisa mengakses bagian atas (TOP), stack cocok digunakan untuk situasi di mana urutan pemrosesan harus dibalik, seperti pada proses undo-redo, back-forward browser, atau pemanggilan fungsi (call stack) di program Stack bisa diimplementasikan baik dengan pointer maupun array, dan keduanya memiliki kelebihan masing-masing. Pointer lebih fleksibel karena tidak terbatas ukuran, sedangkan array lebih mudah karena strukturnya

sederhana.

E. Referensi

Anita Sindar, R. M. S. (2019). *Struktur Data Dan Algoritma Dengan C++ (Vol. 1)*. CV. AA. RIZKY.

Juliansyah, N., Sari, S. Y., & Dristyan, F. (2024). *Optimasi Struktur Data Stack dan Queue Menggunakan Array Dinamis.* Fusion: Journal of Research in Engineering, Technology and Applied Sciences, 1(2), 90-97.

Selamet, R. (2016). *Implementasi struktur data list, queue dan stack dalam java.* Media Informatika, 15(3), 18-25.