LAPORAN PRAKTIKUM

MODUL 7

STACK

Disusun Oleh:

Dhiemas Tulus Ikhsan 2311104046

SE-07-02

Dosen :

Wahyu Andi Saputra, S.Pd., M.Eng

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

I. TUJUAN

a. Mampu memahami konsep stack pada struktur data dan algoritma

b. Mampu mengimplementasikan operasi-operasi pada stack

c. Mampu memecahkan permasalahan solusi stack

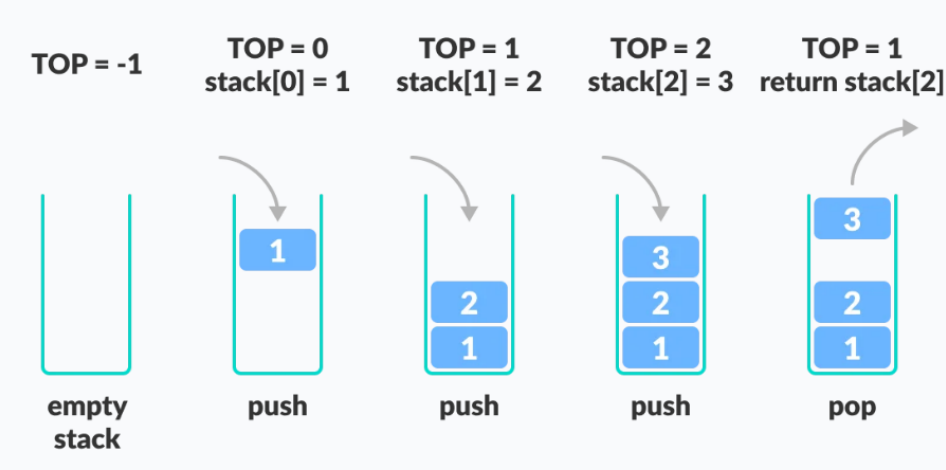
II. LANDASAN TEORI

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.



Working of Stack Data Structure

III. GUIDED

1. Guided_1

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

using namespace std;

class Stack {
private:
    int top;
    int arr[MAX];

public:
    Stack() { 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 elements: ";
    s.display();

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

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

    return 0;
}
```

Output:

```
Stack elements: 30 20 10
Top element: 30
After popping, stack elements: 10
```

2. Guided_2

```
#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()) { // Perbaikan kondisi Stack kosong
            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 != nullptr) {
            cout << current->data << " ";
            current = current->next;
        }
        cout << "\n";
    }
};

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(); // Pop pertama
    s.pop(); // Pop kedua
    cout << "After popping, stack elements: ";
    s.display();

    return 0;
}
```

Output:

```
Stack elements: 30 20 10
Top element: 30
After popping, stack elements: 10
```

IV. UNGUIDED

1. Task_1

   Buatlah program untuk menentukan apakah kalimat tersebut yang diinputkan dalam program stack adalah palindrom/tidak. Palindrom kalimat yang dibaca dari depan dan belakang sama. Jelaskan bagaimana cara kerja programnya.

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

bool isPalindrome(string str) {
    stack<char> s;
    string cleanedStr = "";

    // Membersihkan string dari spasi atau karakter non-alfabet
    for (char c : str) {
        if (isalnum(c)) {
            cleanedStr += tolower(c); // Ubah menjadi huruf kecil
        }
    }

    // Memasukkan karakter ke stack
    for (char c : cleanedStr) {
        s.push(c);
    }

    // Membandingkan karakter dari depan dengan yang diambil dari stack
    for (char c : cleanedStr) {
        if (c != s.top()) {
            return false; // Jika berbeda, bukan palindrom
        }
        s.pop();
    }

    return true; // Jika semua cocok, maka palindrom
}

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

    if (isPalindrome(input)) {
        cout << "Kalimat tersebut adalah: Palindrom" << endl;
    } else {
        cout << "Kalimat tersebut adalah: Bukan Palindrom" << endl;
    }

    return 0;
}
```

Penjelasan Program

1. Pembersihan Input:

- Kalimat yang dimasukkan dibersihkan dari spasi dan karakter non-alfabet menggunakan isalnum().
- Semua huruf diubah menjadi huruf kecil agar perbandingan tidak terganggu oleh huruf kapital.

2. Penyimpanan ke Stack:
   - Setiap karakter dari kalimat yang sudah dibersihkan dimasukkan ke dalam stack.
   - Stack menyimpan karakter secara LIFO (Last In, First Out), sehingga saat diambil, karakter terakhir keluar lebih dulu.
3. Perbandingan:
   - Program membaca karakter dari awal string yang dibersihkan.
   - Setiap karakter dibandingkan dengan karakter yang di-*pop* dari stack (mewakili urutan dari belakang).
   - Jika ada karakter yang tidak cocok, program langsung menyatakan bukan palindrom.
4. Hasil Akhir:
   - Jika semua karakter cocok antara depan dan belakang, maka kalimat tersebut adalah palindrom.
   - Jika tidak cocok, kalimat bukan palindrom.

Output:

```
Masukkan Kalimat: pandora
Kalimat tersebut adalah: Bukan Palindrom
```

2. Task_2

Buatlah program untuk melakukan pembalikan terhadap kalimat menggunakan stack dengan minimal 3 kata. Jelaskan output program dan source codenya beserta operasi/fungsi yang dibuat?

```
#include <iostream>
#include <stack>
#include <sstream> // Untuk memisahkan kata-kata dalam kalimat
using namespace std;

// Fungsi untuk membalikkan string menggunakan stack
string reverseString(string word) {
    stack<char> charStack; // Stack untuk karakter
    string reversedWord = "";

    // Memasukkan setiap karakter ke dalam stack
    for (char c : word) {
        charStack.push(c);
    }

    // Mengambil karakter dari stack untuk membalikkan string
    while (!charStack.empty()) {
        reversedWord += charStack.top();
```

```
        charStack.pop();
    }

    return reversedWord;
}

// Fungsi untuk membalikkan kalimat per kata
string reverseSentence(string sentence) {
    stringstream ss(sentence); // Untuk memisahkan kata dalam kalimat
    string word;
    string result = "";

    // Membaca setiap kata dari kalimat
    while (ss >> word) {
        result += reverseString(word) + " "; // Membalikkan setiap kata
    }

    return result;
}

int main() {
    string sentence;

    // Input kalimat dari pengguna
    cout << "Masukkan kalimat (minimal 3 kata): ";
    getline(cin, sentence);

    // Validasi jumlah kata
    stringstream ss(sentence);
    int wordCount = 0;
    string temp;
    while (ss >> temp) {
        wordCount++;
    }

    if (wordCount < 3) {
        cout << "Kalimat harus memiliki minimal 3 kata!" << endl;
    } else {
        // Membalikkan kalimat
        string result = reverseSentence(sentence);
        cout << "Hasil: " << result << endl;
    }

    return 0;
}
```

Penjelasan Program:

1. Fungsi reverseString:
   - Membalikkan string dengan menggunakan stack:
     - Setiap karakter dari kata dimasukkan ke dalam stack.
     - Karakter dikeluarkan dari stack menggunakan pop() untuk membuat string terbalik.
2. Fungsi reverseSentence:
   - Menggunakan stringstream untuk memisahkan kalimat menjadi kata-kata.
   - Setiap kata diproses melalui fungsi reverseString untuk dibalik.
   - Kata-kata yang sudah dibalik digabungkan kembali menjadi kalimat.

3. Program Utama:
    - Membaca input dari pengguna.
    - Memvalidasi apakah kalimat memiliki minimal 3 kata.
    - Jika valid, fungsi reverseSentence dipanggil untuk membalikkan setiap kata.

Output :

```
Masukkan kalimat (minimal 3 kata): terbang ke langit
Hasil: gnabret ek tignal
```

V. KESIMPULAN

Praktikum ini memberikan pemahaman mendalam tentang konsep dasar Stack, yaitu struktur data yang menerapkan prinsip *Last In, First Out* (LIFO). Struktur ini memungkinkan elemen terakhir yang dimasukkan akan menjadi elemen pertama yang dikeluarkan. Melalui implementasi operasi dasar seperti *push* (menambah elemen ke dalam stack), *pop* (menghapus elemen teratas), dan *peek* (melihat elemen teratas tanpa menghapusnya), kita memahami cara data dikelola dalam urutan tertentu. Selain itu, penggunaan Stack dalam berbagai aplikasi, seperti pengelolaan undo-redo, evaluasi ekspresi matematika, konversi notasi, hingga sistem navigasi, menunjukkan betapa pentingnya konsep ini dalam pengembangan sistem yang efisien. Praktikum ini juga melatih kemampuan berpikir logis untuk mengelola data sesuai dengan kebutuhan aplikasi yang spesifik, sekaligus memberikan dasar yang kuat untuk mempelajari struktur data dan algoritma yang lebih kompleks.