LAPORAN PRAKTIKUM

PERTEMUAN 7

Disusun Oleh:

Naura Aisha Zahira (2311104078)

S1SE-07-02

Dosen :

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY

PURWOKERTO

2024

1. 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

2. 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.

3. Guided

1. Guided 1

Sourcecode:

```
#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()) {
            cout << "Stack is empty\n";
            return -1; // Returning -1 as an error code if the stack is empty
        }
        return arr[top];
    }

    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 stack;
    stack.push(10);
    stack.push(20);
    stack.push(30);

    cout << "Top element is: " << stack.peek() << "\n"; // Expected output: 30
    stack.display(); // Expected output: 30 20 10
```

```
    stack.pop();
    cout << "Top element after pop is: " << stack.peek() << "\n"; // Expected output: 20
    stack.display(); // Expected output: 20 10

    return 0;
}
```

Output:

```
Top element is: 30
30 20 10
Top element after pop is: 20
20 10
```

2. Guided 2

Sourcecode:

```
#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; // Return a sentinel value
    }

    void display() {
        if (isEmpty()) {
            cout << "Stack is empty\n";
            return;
        }
```

```
        Node* current = top;
        while (current) {
            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();
    cout << "After popping, stack elements: ";
    s.display();

    return 0;
}
```

Output:

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

4. Unguided

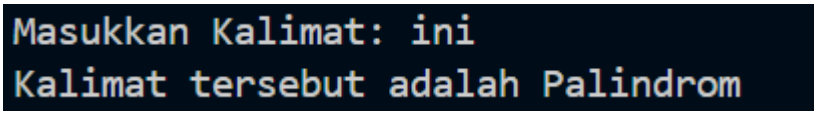
1. Buatlah program untuk menentukan apakah kalimat tersebut yang diinputkan dalam program stack adalah palindrom/tidak. Palindrom kalimat yang dibaca dari depan dan belakang sama. Jelaskan bagaimana cara kerja programnya.

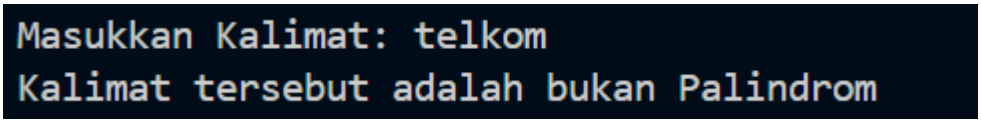
   Code:

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

bool isPalindrome(string str) {
    stack<char> s;
    string cleanStr = "", reversedStr = "";

    // Membersihkan string dari spasi dan membuat huruf kecil
    for (char c : str) {
        if (isalnum(c)) {
            cleanStr += tolower(c);
        }
    }

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

    // Membuat string terbalik dari stack
    while (!s.empty()) {
        reversedStr += s.top();
        s.pop();
    }

    // Membandingkan string asli dengan yang terbalik
    return cleanStr == reversedStr;
```

```
}

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;
}
```

Output:





Penjelasan:

Program ini memeriksa apakah sebuah kalimat adalah palindrom dengan langkah berikut:

- Membersihkan string: Menghapus spasi dan karakter non-alfanumerik, serta mengubah semua huruf menjadi kecil.
- Menggunakan stack: Memasukkan setiap karakter dari string yang telah dibersihkan ke dalam stack untuk membuat versi terbaliknya.
- Membandingkan: Membandingkan string asli (setelah dibersihkan) dengan versi terbalik. Jika sama, maka itu palindrom.

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?

   Code:

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

// Fungsi untuk membalik kata menggunakan stack
string reverseWord(string word) {
    stack<char> s;
    string reversedWord = "";

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

    // Pop setiap karakter dari stack untuk membalik urutannya
    while (!s.empty()) {
        reversedWord += s.top();
        s.pop();
    }

    return reversedWord;
}

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

    stringstream ss(input); // Untuk memisahkan kata-kata dalam kalimat

    // Proses setiap kata dalam kalimat
    while (ss >> word) {
        result += reverseWord(word) + " ";
    }
```

```
    cout << "Data stack Array:" << endl;
    cout << "Data: " << result << endl;

    return 0;
}
```

Output:

```
Masukkan kalimat (minimal 3 kata): Telkom University Purwokerto
Data stack Array:
Data: mokleT ytisrevinU otrekowruP
```

Penjelasan:

- Fungsi reverseWord: Menggunakan stack untuk membalikkan urutan karakter dalam sebuah kata.
- Operasi utama: push (menambah elemen ke stack) dan pop (mengambil elemen dari stack).
- Proses Input: Menggunakan stringstream untuk memisahkan kata-kata dalam kalimat berdasarkan spasi.
- Output:Menampilkan setiap kata yang telah dibalik dalam satu baris.

5. Kesimpulan

Kesimpulan dari laporan ini adalah bahwa praktikum berhasil memahami dan mengimplementasikan konsep stack (LIFO) untuk berbagai operasi seperti push, pop, dan peek. Stack digunakan secara efektif dalam aplikasi nyata seperti pembalikan kata dan pengecekan palindrom, menghasilkan output yang sesuai dengan spesifikasi tugas. Praktikum ini menunjukkan fleksibilitas dan efisiensi stack dalam menyelesaikan masalah pemrograman.