LAPORAN PRAKTIKUM

MODUL 7

STACK

Nama :

Candra Dinata (2311104061)

Kelas :

S1SE 07 02

Dosen :

Wahyu Andi Saputra

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. TUJUAN

Tujuan pembelajaran praktikum pemrograman C++ adalah untuk mengembangkan pemahaman mendalam dan keterampilan praktis dalam pemrograman berbasis objek, struktur data, dan algoritma menggunakan bahasa C++. Mahasiswa diharapkan mampu memahami sintaks dasar, seperti penggunaan variabel, tipe data, operator, dan struktur kontrol, serta menerapkan konsep lanjutan seperti fungsi, kelas, pewarisan, dan polimorfisme. Praktikum ini juga bertujuan untuk meningkatkan kemampuan analisis dan pemecahan masalah melalui implementasi algoritma yang efisien, manipulasi data menggunakan array, pointer, dan file handling, serta penguasaan debugging untuk memperbaiki kode. Selain itu, mahasiswa didorong untuk membangun program yang modular, terstruktur, dan sesuai dengan prinsip rekayasa perangkat lunak, sehingga mampu menciptakan solusi berbasis teknologi yang dapat diaplikasikan dalam berbagai bidang industri dan penelitian.

II. DASAR 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..

III. GUIDED

1.

```cpp
#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;
}
```

Kode di atas merupakan implementasi sederhana dari struktur data stack menggunakan array di C++. Stack adalah struktur data yang mengikuti prinsip LIFO (Last In, First Out), di mana elemen terakhir yang masuk adalah elemen pertama yang keluar. Kelas stack memiliki variabel privat top untuk melacak indeks elemen teratas dan arr untuk menyimpan elemen stack dengan kapasitas maksimum MAX (100). Beberapa fungsi disediakan: push() untuk menambahkan elemen ke stack, pop() untuk menghapus elemen teratas, peek() untuk melihat elemen teratas tanpa menghapusnya, dan display() untuk menampilkan semua elemen stack dari atas ke bawah. Selain itu, terdapat fungsi isFull() dan isEmpty() untuk memeriksa apakah stack sudah penuh atau kosong.

Di dalam fungsi main(), objek stack dibuat, dan beberapa operasi dilakukan untuk mendemonstrasikan fungsionalitas stack. Elemen 10, 20, dan 30 ditambahkan ke stack menggunakan fungsi push(). Fungsi display() menampilkan elemen stack (30, 20, 10). Elemen teratas (30) diperiksa dengan peek(). Kemudian, dua elemen dihapus dari stack menggunakan pop(), dan kondisi stack diperiksa kembali dengan display(). Kode ini menunjukkan cara kerja dasar stack, termasuk penanganan kondisi khusus seperti stack overflow (penambahan elemen pada stack penuh) dan stack underflow (penghapusan elemen dari stack kosong), yang dilaporkan melalui pesan error.

2.

```cpp
#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";
    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();
    s.pop();

    cout << "after popping, stack elements: ";
    s.display();
    return 0;
}
```

Kode ini mengimplementasikan struktur data stack menggunakan linked list di C++. Dalam kelas Node, setiap elemen stack direpresentasikan sebagai node yang memiliki data (data) dan pointer ke elemen berikutnya (next). Kelas Stack menggunakan pointer top untuk menunjuk ke elemen teratas pada stack. Metode push() menambahkan elemen baru ke stack dengan membuat node baru yang menunjuk ke elemen sebelumnya, sedangkan pop() menghapus elemen teratas dengan memindahkan pointer top ke elemen berikutnya. Metode peek() digunakan untuk mendapatkan data dari elemen teratas tanpa menghapusnya, dan display() digunakan untuk mencetak semua elemen stack dari atas ke bawah.

Pada fungsi main(), objek Stack dibuat untuk mendemonstrasikan operasi stack. Elemen 10, 20, dan 30 ditambahkan ke stack menggunakan push(). Metode display() menampilkan elemen stack dalam urutan 30, 20, 10 (karena stack mengikuti prinsip LIFO). Elemen teratas (30) diperiksa menggunakan peek(). Kemudian, dua elemen dihapus menggunakan pop(), dan kondisi stack diperiksa kembali dengan display(). Implementasi ini dinamis, sehingga ukuran stack tidak dibatasi seperti pada versi array, dan elemen hanya dialokasikan saat diperlukan. Kode juga menangani kondisi khusus seperti stack underflow, yaitu ketika mencoba menghapus elemen dari stack kosong.

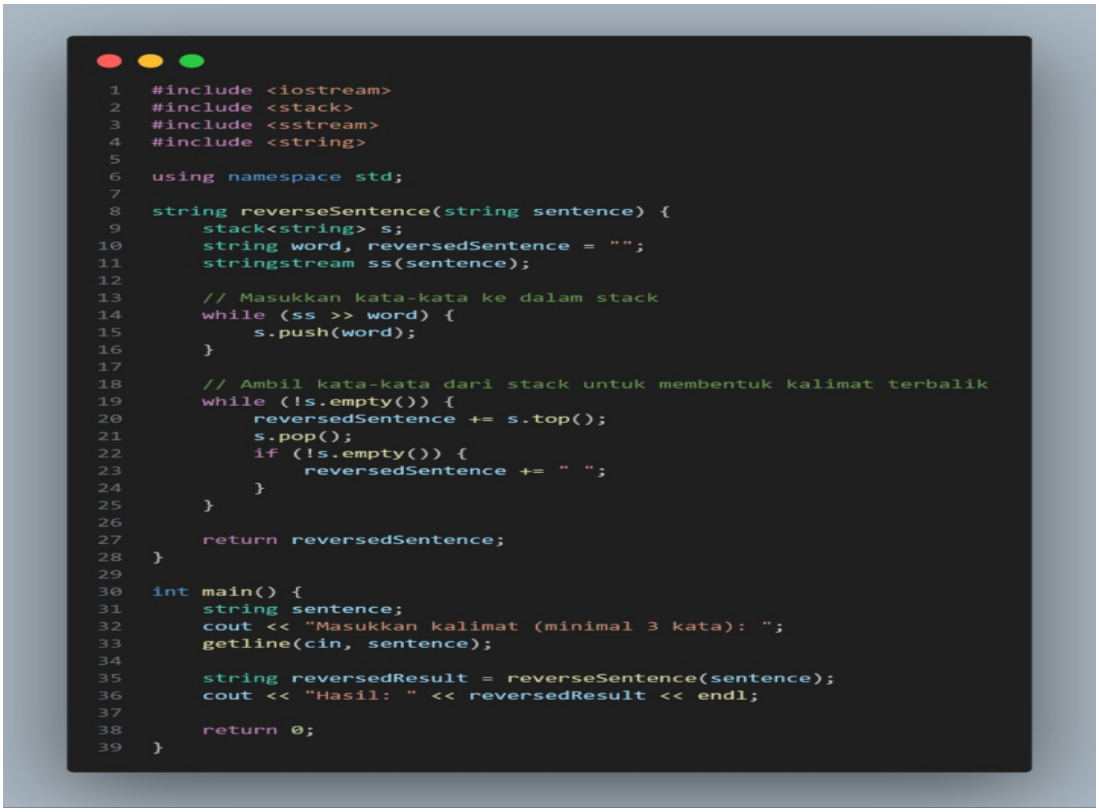
IV. UNGUIDED

1.

```cpp
#include <iostream>
#include <stack>
#include <string>
#include <algorithm>

using namespace std;

bool isPalindrome(string sentence) {
    // Hilangkan spasi dan ubah ke huruf kecil
    sentence.erase(remove(sentence.begin(), sentence.end(), ' '), sentence.end());
    transform(sentence.begin(), sentence.end(), sentence.begin(), ::tolower);

    stack<char> s;

    // Masukkan huruf-huruf ke dalam stack
    for (char c : sentence) {
        s.push(c);
    }

    string reversedSentence = "";
    while (!s.empty()) {
        reversedSentence += s.top();
        s.pop();
    }

    // Bandingkan string asli dan string terbalik
    return sentence == reversedSentence;
}

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

    if (isPalindrome(sentence)) {
        cout << "Kalimat tersebut adalah Palindrom" << endl;
    } else {
        cout << "Kalimat tersebut bukan Palindrom" << endl;
    }

    return 0;
}
```

Kode di atas merupakan program C++ untuk memeriksa apakah sebuah kalimat merupakan palindrom menggunakan struktur data stack. Palindrom adalah teks yang terbaca sama baik dari depan maupun dari belakang (dengan mengabaikan spasi dan huruf besar/kecil). Fungsi isPalindrome() menerima masukan berupa string dan memprosesnya untuk menghapus semua spasi menggunakan erase() dan remove(), serta mengubah semua huruf menjadi huruf kecil menggunakan transform() agar perbandingan tidak peka terhadap huruf besar/kecil. Kemudian, huruf-huruf dari string tersebut dimasukkan ke dalam stack satu per satu. Dengan memanfaatkan sifat LIFO stack, string terbalik dibuat dengan mengeluarkan karakter dari stack ke variabel reversedSentence. Selanjutnya, string asli dibandingkan dengan string terbalik, dan fungsi mengembalikan true jika keduanya sama (palindrom) atau false jika berbeda. Pada fungsi main(), pengguna diminta untuk memasukkan kalimat, yang kemudian diperiksa oleh isPalindrome(). Jika kalimat adalah palindrom, program mencetak pesan bahwa kalimat tersebut adalah palindrom; jika tidak, program mencetak bahwa kalimat tersebut bukan palindrom. Program ini mengilustrasikan penggunaan stack untuk membalikkan string secara efisien.

2.



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

using namespace std;

string reverseSentence(string sentence) {
    stack<string> s;
    string word, reversedSentence = "";
    stringstream ss(sentence);

    // Masukkan kata-kata ke dalam stack
    while (ss >> word) {
        s.push(word);
    }

    // Ambil kata-kata dari stack untuk membentuk kalimat terbalik
    while (!s.empty()) {
        reversedSentence += s.top();
        s.pop();
        if (!s.empty()) {
            reversedSentence += " ";
        }
    }

    return reversedSentence;
}

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

    string reversedResult = reverseSentence(sentence);
    cout << "Hasil: " << reversedResult << endl;

    return 0;
}
```

Kode di atas adalah program C++ yang digunakan untuk membalik urutan kata dalam sebuah kalimat menggunakan struktur data stack. Fungsi reverseSentence() menerima masukan berupa sebuah string kalimat dan memprosesnya dengan bantuan kelas stringstream untuk memisahkan kata-kata dalam kalimat berdasarkan spasi. Setiap kata yang dipisahkan dimasukkan ke dalam stack menggunakan metode push(). Karena stack mengikuti prinsip LIFO (Last In, First Out), kata-kata yang terakhir dimasukkan akan dikeluarkan terlebih dahulu, sehingga urutannya terbalik. Setelah semua kata berada di stack, program mengeluarkannya satu per satu menggunakan metode pop() dan menyusunnya menjadi kalimat terbalik di variabel reversedSentence. Pada fungsi main(), pengguna diminta memasukkan sebuah kalimat minimal tiga kata, yang kemudian diproses oleh reverseSentence() untuk menghasilkan versi kalimat dengan urutan kata terbalik. Hasilnya ditampilkan pada layar. Program ini menunjukkan bagaimana stack dapat digunakan untuk membalikkan urutan elemen secara sederhana dan efisien.

V. KESIMPULAN

Berdasarkan hasil praktikum, dapat disimpulkan bahwa struktur data stack dalam C++ merupakan salah satu implementasi dari prinsip LIFO (Last In, First Out), di mana elemen yang terakhir dimasukkan akan menjadi elemen pertama yang dikeluarkan. Penggunaan stack dapat diimplementasikan dengan array, linked list, maupun library bawaan seperti <stack> di C++. Dalam praktikum ini, fungsi-fungsi utama stack seperti push(), pop(), dan peek() berhasil diuji dan berfungsi dengan baik untuk menambah, menghapus, dan melihat elemen teratas stack. Selain itu, stack juga dapat digunakan untuk menyelesaikan berbagai kasus, seperti membalikkan string, membalikkan urutan kata dalam kalimat, dan memeriksa apakah sebuah string merupakan palindrom.

Implementasi stack memerlukan perhatian terhadap kondisi khusus seperti stack overflow (pada stack berbasis array) atau stack underflow, yang terjadi ketika mencoba menambah elemen pada stack penuh atau menghapus elemen dari stack kosong.