LAPORAN PRAKTIKUM
MODUL VII
“STACK”

Disusun Oleh:

Alya Rabani - 2311104076

S1SE-07-B

Dosen :

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

1. Tujuan

- Dapat memahami konsep stack pada struktur data dan algoritma
- Dapat mengimplementasikan operasi-operasi pada stack
- Dapat memecahkan permasalahan dengan solusi stack

2. Dasar Teori

Stack

Istilah "stack" yang kita kenal dalam dunia pemrograman sebenarnya berasal dari kata "tumpukan" dalam bahasa sehari-hari. Struktur data ini bekerja seperti tumpukan benda fisik, di mana elemen yang terakhir dimasukkan akan menjadi yang pertama dikeluarkan. Konsep stack pertama kali diperkenalkan oleh John Backus pada tahun 1945 untuk menyederhanakan perhitungan dalam kalkulator. Seiring perkembangan teknologi, stack telah menjadi komponen integral dalam bahasa pemrograman seperti Forth dan LISP pada tahun 1950-an, dan kini menjadi salah satu struktur data paling dasar yang digunakan dalam berbagai bahasa pemrograman modern.

Konsep LIFO (Last In, First Out)

Stack mengadopsi mekanisme LIFO (Last In, First Out). Artinya, elemen yang terakhir ditambahkan ke dalam stack akan menjadi elemen pertama yang diambil. Analogikan seperti tumpukan piring, piring terakhir yang ditumpuk akan diambil terlebih dahulu. Proses penambahan data ke dalam stack disebut push, sedangkan penghapusan data disebut pop. Urutan penghapusan data ini mengikuti prinsip LIFO. Misalnya, jika dalam stack terdapat angka 1, 2, 3, dan 4, maka saat dilakukan pop secara berulang, angka yang akan keluar adalah 4, 3, 2, dan terakhir 1.

Operasi dasar stack

- Top (Puncak): Posisi atau indeks teratas dari stack. Top menunjukkan elemen terbaru yang dimasukkan ke dalam stack dan tempat di mana operasi-operasi stack dilakukan.
- Push: operasi untuk menambahkan elemen baru ke puncak stack.
- Pop: operasi untuk menghapus elemen dari puncak stack.
- Peek: melihat nilai atau elemen pada posisi tertentu dalam tumpukan
- isEmpty: operasi untuk memeriksa apakah stack kosong
- isFull: operasi untuk memeriksa apakah stack penuh
- Destroy: menghapus semua elemen dalam stack

3. Guided

1. Program ini mendefinisikan kelas Stack yang memiliki metode untuk menambah elemen (push), menghapus elemen (pop), melihat elemen teratas (peek), dan menampilkan semua elemen dalam stack (display). Kelas ini juga memiliki pemeriksaan untuk memastikan stack tidak penuh sebelum menambahkan elemen dan tidak kosong sebelum menghapus elemen. Dalam fungsi main, program mendemonstrasikan penggunaan kelas Stack dengan menambahkan beberapa elemen, menampilkan isi stack, dan mengeluarkan elemen dari stack, sehingga memberikan gambaran tentang cara kerja dan fungsi dari struktur data stack.

```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; // Return a sentinel value
    }

    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 program:

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

2. Dalam implementasi pada program tersebut, setiap elemen stack diwakili oleh sebuah objek Node, yang memiliki dua atribut: data untuk menyimpan nilai dan next untuk menunjuk ke node berikutnya dalam stack. Kelas Stack memiliki pointer top yang menunjuk ke elemen teratas stack.
   Program ini menyediakan beberapa operasi, yaitu:
   - isEmpty(): untuk memeriksa apakah stack kosong.
   - push(int x): untuk menambahkan elemen baru ke atas stack dengan membuat node baru dan menghubungkannya ke node teratas saat ini.
   - pop(): untuk menghapus elemen teratas dari stack, dengan memindahkan pointer top ke node berikutnya dan menghapus node yang dihapus dari memori.
   - peek(): untuk melihat nilai dari elemen teratas tanpa menghapusnya.
   - display(): untuk menampilkan semua elemen dalam stack dari teratas ke bawah.

   Dalam fungsi main, program mendemonstrasikan penggunaan kelas Stack dengan menambahkan beberapa elemen, menampilkan isi stack, dan mengeluarkan elemen dari stack. Program ini menunjukkan bagaimana stack dapat diimplementasikan dengan efisien menggunakan linked list, sehingga dapat mengatasi batasan ukuran yang ada pada implementasi stack berbasis array.

```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; // 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 program:

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

4. Unguided

1. Program untuk mengecek apakah sebuah string adalah palindrom bekerja dengan cara yang efisien menggunakan STL stack(Standard Template Library) agar lebih ringkas untuk implementasi stack.

   Pertama, program menerima input string dari pengguna menggunakan fungsi `getline()`. Setelah itu, string dibersihkan dari spasi dan karakter non-alfabet, lalu diubah menjadi huruf kecil untuk konsistensi. Hasil pembersihan disimpan dalam string baru bernama `cleanStr`. Program kemudian membagi string menjadi dua bagian, setengah pertama string dimasukkan ke dalam stack, sementara setengah kedua dibandingkan dengan karakter yang diambil dari stack. Untuk string dengan panjang ganjil, karakter tengah dilewati. Selama proses, karakter pada setengah kedua dibandingkan dengan elemen teratas stack; jika cocok, elemen dihapus dari stack, dan pengecekan dilanjutkan ke karakter berikutnya. Jika ada yang tidak sesuai, program segera mengembalikan nilai false, menunjukkan string bukan palindrom. Sebaliknya, jika semua karakter cocok dan stack menjadi kosong, program mengembalikan nilai *true*, menandakan string adalah palindrom.

   Contoh outputnya akan menjadi seperti berikut:

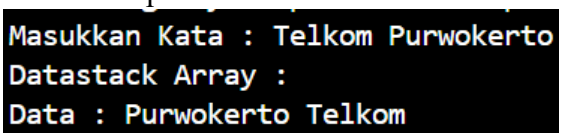
```
Masukan Kalimat : ini
Kalimat tersebut adalah : Palindrom
PS D:\tugas yall\praktikum sd\pertemuan 8\
PS D:\tugas yall\praktikum sd\pertemuan 8\
Masukan Kalimat : telkom
Kalimat tersebut adalah : Bukan Palindrom
```

Kode program:

```cpp
#include <iostream>
#include <stack> // import STL stack
#include <string>
using namespace std;

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

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

    int length = cleanStr.length();
    int midPoint = length / 2;

    // Memasukkan setengah karakter pertama ke dalam stack
    for(int i = 0; i < midPoint; i++) {
        s.push(cleanStr[i]);
    }

    // Jika panjang string ganjil, skip karakter tengah
    int startIndex = (length % 2 == 0) ? midPoint : midPoint + 1;

    // Membandingkan setengah karakter terakhir dengan isi stack
    for(int i = startIndex; i < length; i++) {
        if(s.empty() || s.top() != cleanStr[i]) {
            return false;
        }
        s.pop();
    }

    return s.empty();
}

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

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

    return 0;
}
```

2. Program ini menggunakan struktur data stack berbasis array untuk membalikkan urutan kata dalam sebuah kalimat. Stack dirancang dengan kapasitas maksimal 100 kata dan dilengkapi variabel `top` untuk menandai posisi teratas.
   Program menggunakan operasi dasar stack, seperti:
   - isEmpty() untuk mengecek apakah stack kosong
   - isFull() untuk memastikan apakah stack penuh
   - push() untuk menambah kata ke stack
   - pop() untuk mengambil dan menghapus kata dari stack
   - size() untuk menghitung jumlah kata dalam stack.

   Pada program utama (`main`), kalimat dimasukkan oleh pengguna, diproses oleh fungsi `reverseSentence`, lalu hasilnya ditampilkan. Sebagai contoh, input "Telkom Purwokerto" akan menghasilkan output "otrekowruP mokleT" melalui proses push kata-kata ke stack, lalu pop untuk membentuk kalimat dalam urutan terbalik.
   Contoh output:



Kode program:

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

// Struktur untuk stack
struct Stack {
    string data[100];  // Array untuk menyimpan kata
    int top;           // Index top of stack

    // Constructor
    Stack() {
        top = -1;     // Inisialisasi stack kosong
    }

    // Fungsi untuk mengecek apakah stack kosong
    bool isEmpty() {
        return top == -1;
    }

    // Fungsi untuk mengecek apakah stack penuh
    bool isFull() {
        return top == 99;
    }

    // Fungsi untuk menambah data ke stack (push)
    void push(string str) {
        if (!isFull()) {
            data[++top] = str;
        }
    }

    // Fungsi untuk mengambil data dari stack (pop)
    string pop() {
        if (!isEmpty()) {
            return data[top--];
        }
        return "";
    }

    // Fungsi untuk melihat jumlah data dalam stack
    int size() {
        return top + 1;
    }
};

// Fungsi untuk membalik kalimat
string reverseSentence(string sentence) {
    Stack wordStack;
    stringstream ss(sentence);
    string word, result;

    // Memisahkan kalimat menjadi kata-kata dan push ke stack
    while (ss >> word) {
        wordStack.push(word);
    }


    // Mengambil kata dari stack dan menyusun kalimat terbalik
    while (!wordStack.isEmpty()) {
        result += wordStack.pop();
        if (!wordStack.isEmpty()) {
            result += " ";
        }
    }

    return result;
}

int main() {
    string input;

    cout << "Masukkan Kata : ";
    getline(cin, input);

    cout << "Datastack Array : " << endl;
    cout << "Data : " << reverseSentence(input) << endl;

    return 0;
}
```

5. Kesimpulan

Stack sebagai salah satu struktur data yang mengadopsi mekanisme LIFO (Last In, First Out), di mana elemen yang terakhir dimasukkan akan menjadi yang pertama dikeluarkan. Laporan ini mencakup dasar teori, operasi-operasi utama dalam stack seperti push, pop, peek, isEmpty, isFull, dan destroy. Selain itu, laporan juga menunjukkan implementasi program yang menggunakan kelas Stack dan metode terkait untuk memanipulasi elemen-elemen dalam stack, baik untuk menambah, menghapus, ataupun menampilkan isi stack. Implementasi praktikum menunjukkan penggunaan stack untuk pengecekan palindrom dan pembalikan kata dalam kalimat. Dengan memanfaatkan stack berbasis array dan linked list, program-program tersebut berhasil memecahkan masalah dengan efisien, menunjukkan fleksibilitas dan kekuatan stack dalam pengolahan data.