LAPORAN PRAKTIKUM
Modul 7
Stack

Disusun Oleh:
Zhafir Zaidan Avail
S1-SE-07-2

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

PROGRAM STUDI S1 SOFTWARE ENGINEERING
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
PURWOKERTO

2024

1. Tujuan

1. Memahami konsep stack.
2. Mengimplementasikan stack dengan menggunakan representasi pointer dan tabel.
3. Memahami konsep queue.
4. Mengimplementasikan queue dengan menggunakan pointer dan tabel.

2. Landasan Teori

- Pengertian Stack
  Stack merupakan salah satu bentuk struktur data dimana prinsip operasi yang digunakan seperti tumpukan. Seperti halnya tumpukan, elemen yang bisa diambil terlebih dahulu adalah elemen yang paling atas, atau elemen yang pertama kali masuk, prinsip ini biasa disebut LIFO (Last In First Out).
- Komponen-komponen dalam Stack
  Komponen – komponen dalam stack pada dasarnya sama dengan komponen pada single linked list. Hanya saja akses pada stack hanya bisa dilakukan pada awal stack saja.
  Komponen utama dalam stack yang berfungsi untuk mengakses data dalam stack adalah elemen paling awal saja yang disebut "Top". Pendeklarasian tipedata stack:

```
#ifndef stack_H
#define stack_H
#define Nil NULL
#define info(P) (P)->info
#define next(P) (P)->next
#define Top(S) ((S).Top)
typedef int infotype; /* tipe data dalam stack */
typedef struct elmStack *address;
/* tipe data pointe untuk elemen stack */
struct elmStack {
infotype info;
address next;
}; /*tipe data elemen stack */
/* pendeklarasian tipe data stack */
struct stack {
address Top;
};
#endif
```

  Keterangan:
  1. Dalam stack hanya terdapat TOP.
  2. Tipe address adalah tipe elemen stack yang sama dengan elemen dalam list lainnya.
- Operasi dalam sebuah Stack
  Dalam stack ada dua operasi utama, yaitu operasi penyisipan (Push) dan operasi pengambilan (Pop).
  - Push
    Adalah operasi menyisipkan elemen pada tumpukan data. Fungsi ini sama dengan fungsi insert first pada list biasa.

```
/* buat dahulu elemen yang akan disisipkan */
address createElm(int x){
address p = alokasi(x);
next(p) = null;
return p;
}
/* contoh sintak push */
void push(address p){
next(p) = top(s);
top(s) = p;
}
```

- Pop
  Adalah operasi pengambilan data dalam list. Operasi ini mirip dengan operasi delete first dalam list linear, karena elemen yang paling pertama kali diakses adalah elemen paling atas atau elemen paling awal saja. Langkah-langkah dalam proses Pop:

```
/* contoh sintak pop */
address pop(address p){
p = top(s);
top(s) = next(top(s));
return p;
}
```

- Primitif-Primitif dalam Stack
  Primitif-primitif dalam stack pada dasarnya sama dengan primitif-primitif pada list lainnya. Malahan
  primitif dalam stack lebih sedikit, karena dalam stack hanya melakukan operasi-operasi terhadap
  elemen paling atas.
  Primitif -primitif dalam stack :
  1. createStack().
  2. isEmpty().
  3. alokasi().
  4. dealokasi().
  5. Fungsi – fungsi pencarian.
  6. Dan fungsi – fungsi primitif lainnya.
  Seperti halnya pada model list yang lain, primitif-primitifnya tersimpan pada file *.c dan file *.h.
- Stack (Representasi Tabel)
  Pada prinsipnya representasi menggunakan tabel sama halnya dengan menggunakan pointer. Perbedaannya terletak pada pendeklarasian struktur datanya, menggunakan array berindeks dan jumlah tumpukan yang terbatas.
  - Operasi-operasi Dalam Stack
    Operasi-operasi dalam stack representasi tabel pada dasarnya sama dengan representasi pointer, yaitu PUSH dan POP.
    - Push
      Push merupakan operasi penyisipan data ke dalam stack, penyisipan dilakukan dengan menggeser indeks dari TOP ke indeks berikutnya.
    - Pop
      Pop merupakan operasi pengambilan data di posisi indeks TOP berada dalam sebuah stack. Setelah data diambil, indeks TOP akan bergeser ke indeks sebelum TOP tanpa menghilangkan info dari indeks TOP sebelumnya.
  - Primitif-primitif Dalam Stack
    Primitif-primitif pada stack representasi tabel pada dasarnya sama dengan representasi pointer.
    Perbedaanya hanya pada manajemen memori, pada representasi tabel tidak diperlukan manajemen
    memori. antara lain sebagai berikut,
    1. createStack().
    2. isEmpty().
    3. Fungsi – fungsi pencarian.

4. Dan fungsi – fungsi primitif lainnya.
Seperti halnya pada model list yang lain, primitif-primitifnya tersimpan pada file *.c dan file *.h.
Untuk lebih memahami struktur data dari stack representasi tabel, berikut ini contoh ADT stack
representasi tabel dalam file *.h:

```
/* file : stack.h
   contoh ADT stack dengan representasi tabel */
#ifndef STACK_H_INCLUDED
#define STACK_H_INCLUDED

#include <stdio.h>
#include <conio.h>
struct infotype {
 char nim[20];
 char nama[20];
};
struct Stack {
 infotype info[10];
 int Top;
};

/* prototype */
/******** pengecekan apakah Stack kosong *********/
int isEmpty(Stack S);
/* mengembalikan nilai 0 jika stack kosong */
/******** pembuatan Stack ********/
void createStack(Stack &S);
/* I.S. sembarang
   F.S. terbentuk stack dengan Top=0 */
/******** penambahan elemen pada Stack ********/
void push(Stack &S, infotype X);
/* I.S. Stack mungkin kosong
   F.S. menambahkan elemen pada stack dengan nilai X, TOP=TOP+1 */
/******** penghapusan elemen pada list ********/
void pop(Stack &S, infotype &X);
/* I.S. stack tidak kosong
   F.S. nilai info pada indeks TOP disimpan pada X, kemudian TOP=TOP-1
*/
/******** proses semua elemen list *********/
void viewStack(Stack S);
/* I.S. stack mungkin kosong
   F.S. jika stack tidak kosong menampilkan semua info yang ada pada
stack
*/
#endif // STACK_H_INCLUDED
```

3. Guided

Guided1:

```
#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" << endl;
            return;
        }
        arr[++top] = x;
    }

    int pop()
    {
        if (isEmpty())
        {
            cout << "Stack Underflow" << endl;
            return -1;
        }
        return arr[top--];
    }

    int peek()
    {
        if (isEmpty())
        {
            cout << "Stack Underflow" << endl;
            return -1;
        }
        return arr[top];
    }

    void display()
    {
        if (isEmpty())
        {
            cout << "Stack Underflow" << endl;
            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);
    s.display();
    cout << "Stack elements: ";
```

```
    s.display();

    cout << "Top element: " << s.peek() << "\n";
    return 0;
}
```

Output:

```
30 20 10
Stack elements: 30 20 10
Top element: 30
```

Gudied2

```
#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())
```

```
        {
            cout << "Stack is empty\n";
            return -1;
        }
        return top->data;
    }

    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();
    cout << "Stack after pop: ";
    s.display();

    return 0;
}
```

Output:

```
Stack elements: 30 20 10
Top element: 30
Stack after pop: 20 10
```

4. Unguided

Unguided1

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

// Fungsi untuk memeriksa apakah kalimat adalah palindrom
bool isPalindrome(const string& input)
{
    string cleanedInput;
    stack<char> charStack;

    // Normalisasi input: hapus spasi dan ubah ke huruf kecil
    for (char c : input)
    {
```

```
        if (!isspace(c)) // Abaikan karakter spasi
        {
            char lowerChar = tolower(c);
            cleanedInput += lowerChar;
            charStack.push(lowerChar);
        }
    }

    // Periksa apakah string asli sama dengan versi terbaliknya
    for (char c : cleanedInput)
    {
        if (c != charStack.top())
        {
            return false;
        }
        charStack.pop();
    }

    return true;
}

int main()
{
    string userInput;

    // Membaca input dari pengguna
    cout << "Masukkan sebuah kalimat: ";
    getline(cin, userInput); // Mendukung input dengan spasi

    // Output hasil
    if (isPalindrome(userInput))
    {
        cout << "Kalimat tersebut adalah palindrom." << endl;
    }
    else
    {
        cout << "Kalimat tersebut bukan palindrom." << endl;
    }

    return 0;
}
```

Output:

```
Masukkan sebuah kalimat: Hai
Kalimat tersebut bukan palindrom.
Masukkan sebuah kalimat: ini
Kalimat tersebut palindrom.
```

Unguided2

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

// Fungsi untuk membalikkan seluruh kalimat
string reverseSentence(const string& input)
{
    stack<char> charStack;
    string result;

    // Masukkan setiap karakter ke dalam stack
    for (char ch : input)
    {
        charStack.push(ch);
    }

    // Bangun string terbalik dari stack
    while (!charStack.empty())
```

```
    {
        result += charStack.top();
        charStack.pop();
    }

    return result;
}

int main()
{
    string userInput;

    // Meminta input dari pengguna
    cout << "Masukkan kalimat (minimal 3 kata): ";
    getline(cin, userInput);

    // Membalikkan kalimat dan menampilkan hasil
    string reversedResult = reverseSentence(userInput);
    cout << "Data stack array:" << endl;
    cout << "Data: " << reversedResult << endl;

    return 0;
}
```

Output

```
Masukkan kalimat (minimal 3 kata): Tian Raja Jomok
Data stack array:
Data: komoJ ajaR naiT
```

5. Kesimpulan

Stack adalah struktur data yang bekerja berdasarkan prinsip LIFO (Last In, First Out), mirip seperti tumpukan buku. Elemen yang terakhir dimasukkan akan menjadi yang pertama dikeluarkan. Stack terdiri dari komponen utama yaitu **top** yang menunjuk ke elemen paling atas. Operasi dasar pada stack adalah **push** (menambahkan elemen) dan **pop** (menghapus elemen). Stack dapat diimplementasikan menggunakan **pointer** atau **tabel**. Pada representasi pointer, elemen-elemen stack saling terhubung melalui pointer. Sedangkan pada representasi tabel, elemen-elemen stack disimpan dalam sebuah array. **Primitif-primitif** pada stack meliputi operasi-operasi dasar seperti membuat stack kosong, memeriksa apakah stack kosong, menambahkan elemen, menghapus elemen, dan menampilkan seluruh elemen.

Representasi tabel pada stack menggunakan array untuk menyimpan elemen-elemen. Operasi push dan pop dilakukan dengan menggeser indeks top. Keuntungan representasi tabel adalah implementasinya yang lebih sederhana dan tidak memerlukan manajemen memori secara eksplisit. Namun, kapasitas stack terbatas oleh ukuran array yang telah ditentukan. **Primitif-primitif** pada stack representasi tabel serupa dengan representasi pointer, namun manajemen memorinya lebih sederhana.