**LAPORAN PRAKTIKUM STRUKTUR DATA**

**STACK & QUEUE**

****

**DISUSUN OLEH :**

**MUHAMMAD NURIL ANWARI**

**362458302106**

**TEKNOLOGI REKAYASA PERANGKAT LUNAK**

**POLITEKNIK NEGERI BANYUWANGI**

**TAHUN 2025**

**STACK (TUMPUKAN)**

**DASAR TEORI**

**Stack** adalah salah satu struktur data linear yang prinsip kerjanya menggunakan metode **LIFO (Last In, First Out)**, yang berarti elemen terakhir yang dimasukkan adalah elemen pertama yang dikeluarkan. Stack dapat dianalogikan seperti tumpukan piring: piring yang terakhir diletakkan di atas adalah yang pertama kali diambil.

**TUGAS PENDAHULUAN**

1. **Jelaskan Pengertian Stack**

**Stack** adalah struktur data abstrak yang digunakan untuk menyimpan dan mengelola data dengan aturan khusus, yaitu **LIFO (Last In, First Out)**. Artinya, data yang terakhir kali dimasukkan ke dalam stack akan menjadi data pertama yang dikeluarkan. Stack bekerja layaknya tumpukan benda: hanya bagian atas yang bisa diakses untuk menambah (push) atau mengambil (pop) data.

Ciri-Ciri Utama Stack :

* Data hanya bisa ditambahkan dan diambil dari satu sisi, yaitu **top (atas)**.
* Elemen yang lebih dulu masuk akan berada di bawah elemen yang masuk belakangan.
* Stack bisa diimplementasikan menggunakan **array** atau **linked list**.

1. **Jelaskan operasi pada stack** 
   1. empty**()**

Fungsinya untuk Mengecek apakah stack dalam keadaan kosong. Hasilnya **True (benar)** jika stack tidak memiliki elemen dan **False (salah)** jika terdapat satu atau lebih elemen dalam stack.

* 1. top()

Fungsinya untuk Mengembalikan elemen teratas dari stack tanpa menghapusnya.

Dengan cataatn jika stack kosong, operasi ini bisa menghasilkan error atau exception tergantung implementasi.

* 1. pop()

Menghapus (mengeluarkan) elemen teratas dari stack dan biasanya digunakan untuk pengecekan empty() untuk menghindari error.

* 1. push('T element')

Fungsinya untuk Menambahkan elemen 'T element' ke bagian atas stack.

1. **Ubahlah ekspresi infix dibawah ini menjadi ekspresi postfix**.
2. (A \* B) + (C - D)

Langkah-langkah :

(A \* B) menjadi A B \*

(C - D) menjadi C D -

Gabungkan: A B \* C D - +

Posfix : A B \* C D - +

1. (A - B) - (C \* D) + E

Langkah Langkah :

(A - B) menjadi A B –

(C \* D) menjadi C D \*

A B - C D \* - (pengurangan kedua)

Tambah E: A B - C D \* - E +

**Postfix :** A B - C D \* - E +

1. A \* (B - C) - (D \* E)

Langkah langkah :

(B - C) menjadi B C –

A \* (B - C) menjadi A B C - \*

(D \* E) menjadi D E \*

Gabungkan: A B C - \* D E \* -

Posfix : A B C - \* D E \* -

**PERCOBAAN**

**Percobaan 1**

class Stack {

 int last\_elemen = 0;

 int maxStack = 0;

 List<int> elemen = [];

 void StackOperation(int maxElements) {

 last\_elemen = -1;

 maxStack = maxElements - 1;

 elemen = List<int>.filled(maxElements, 0);

 }

 bool isEmpty() {

 return last\_elemen == -1;

 }

 bool isFull() {

 return last\_elemen == maxStack;

 }

 void push(int data) {

 if (isFull()) {

  print("stack overflow");

 } else {

 last\_elemen = last\_elemen + 1;

 elemen[last\_elemen] = data;

 }

 }

 int pop() {

 int data = 0;

 if (isEmpty()) {

 print("stack Underflow");

 } else {

 data = elemen[last\_elemen];

 elemen[last\_elemen] = 0;

 last\_elemen = last\_elemen - 1;

 }

 return data;

 }

 int top() {

 if (isEmpty()) {

 print("Stack Underflow");

 return 0;

 } else {

 return elemen[last\_elemen];

 }

 }

 void printStack() {

 if (!isEmpty()) {

 print("Menampilkan urutan dari posisi tertinggi");

 for (int i = last\_elemen; i > -1; i--) {

 print("Elemen ke-" + i.toString() + " = " + elemen[i].toString());

 }

 } else {

 print("Stack masih kosong");

 }

 }

}

void main() {

 Stack s = Stack();

 s.StackOperation(100);

 s.push(10);

 s.push(20);

 s.push(30);

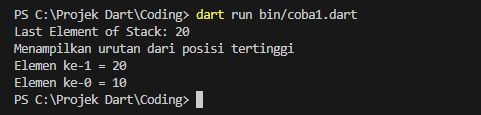
 s.pop();

 print("Last Element of Stack: " + s.top().toString());

 s.printStack();

}

**Output**

****

**Percobaan 2**

class Stack<T> {

 final List<T> \_stack = [];

 void push(T element) {

 \_stack.add(element);

 }

 T pop() {

 if (\_stack.isEmpty) {

 throw StateError("No elements in the Stack");

 } else {

  T lastElement = \_stack.last;

 \_stack.removeLast();

 return lastElement;

 }

 }

 T top() {

 if (\_stack.isEmpty) {

 throw StateError("No elements in the Stack");

 } else {

 return \_stack.last;

 }

 }

 bool isEmpty() {

 return \_stack.isEmpty;

 }

 @override

 String toString() => \_stack.toString();

}

void main() {

 var stack = Stack<int>();

 var test = stack.isEmpty();

 print(test); // Output: true

 stack.push(1);

 stack.push(2);

 stack.push(3);

 stack.push(4);

 stack.push(5);

 stack.push(7);

 print(stack); // Output: [1, 2, 3, 4, 5, 7]

 var myNumber = stack.pop();

 print('My 1st number is $myNumber');

 var mySecNumber = stack.pop();

 print('My 2nd number is $mySecNumber');

 print(stack.pop()); // Output: 4

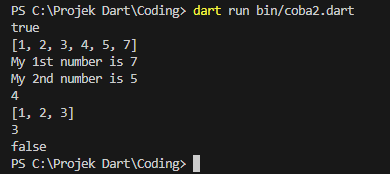
 print(stack); // Output: [1, 2, 3]

 print(stack.top()); // Output: 3

 print(stack.isEmpty()); // Output: false

}

**Output**

****

**Percobaan 3**

**Percobaan 4**

import 'dart:io';

class Stack {

 List<String> \_items = [];

 void push(String item) {

 \_items.add(item);

 }

 String pop() {

 if (isEmpty()) {

 throw Exception('Stack is empty');

 }

 return \_items.removeLast();

 }

 String top() {

 if (isEmpty()) {

 throw Exception('Stack is empty');

 }

 return \_items.last;

 }

 bool isEmpty() {

 return \_items.isEmpty;

 }

}

class PostfixToInfix {

 String postfixExp = "";

 String toInfix(String postfixExp) {

 var stack = Stack();

 for (var ch in postfixExp.split('')) {

 if (\_isOperator(ch)) {

 var operand2 = stack.pop();

 var operand1 = stack.pop();

 var result = '($operand1$ch$operand2)';

 stack.push(result);

 } else {

 stack.push(ch);

 }

 }

 return stack.pop();

 }

  bool \_isOperator(String ch) {

 return ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch ==

'^';

 }

}

void main() {

 var pti = PostfixToInfix();

 String postfixExp;

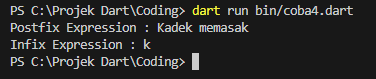
 stdout.write("Postfix Expression : ");

 postfixExp = stdin.readLineSync()!;

 print("Infix Expression : ${pti.toInfix(postfixExp)}");

}

**Output**

****

**LATIHAN**

**Latihan 1 a.**

import 'dart:io';

class KonversiBilangan {

  int desimal;

  KonversiBilangan(this.desimal);

  String keBiner() {

    return desimal.toRadixString(2);  // 2 berarti basis biner

  }

  String keOktal() {

    return desimal.toRadixString(8);  // 8 berarti basis oktal

  }

  String keHeksadesimal() {

    return desimal.toRadixString(16).toUpperCase();  // 16 = basis hex

  }

}

void main() {

  stdout.write("Masukkan nilai desimal = ");

  int? input = int.tryParse(stdin.readLineSync()!);

  if (input != null) {

    var konversi = KonversiBilangan(input);

    print("Hasil nilai biner       = ${konversi.keBiner()}");

    print("Hasil nilai oktal       = ${konversi.keOktal()}");

    print("Hasil nilai heksadesimal = ${konversi.keHeksadesimal()}");

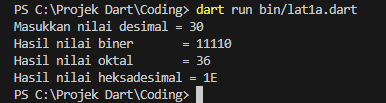
  } else {

    print("Input tidak valid.");

  }

}

**Output**

****

**Penjelasan**

1. **1mport dart:io**  
   Digunakan untuk mengambil input dari pengguna di terminal.
2. **Class KonversiBilangan**
   * Menerima nilai desimal pada konstruktor.
   * Tiga metode:
     + toRadixString(2) untuk biner.
     + toRadixString(8) untuk oktal.
     + toRadixString(16) untuk heksadesimal.
3. **Fungsi main()**
   * Mengambil input desimal dari user.
   * Membuat objek dari class dan menampilkan hasil konversi.

**Latihan 1 b.**

import 'dart:io';

class Kalimat {

  String teks;

  Kalimat(this.teks);

  String balikKalimat() {

    return teks.split('').reversed.join('');

  }

  bool isPalindrom() {

    String asli = teks.replaceAll(' ', '').toLowerCase();

    String dibalik = balikKalimat().replaceAll(' ', '').toLowerCase();

    return asli == dibalik;

  }

}

void main() {

  stdout.write("Masukkan kalimat : ");

  String input = stdin.readLineSync()!;

  var kalimat = Kalimat(input);

  String hasilBalik = kalimat.balikKalimat();

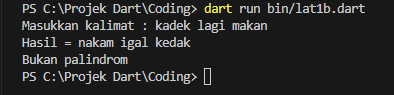
  bool palindrom = kalimat.isPalindrom();

  print("Hasil = $hasilBalik");

  print(palindrom ? "Palindrom" : "Bukan palindrom");

}

**Output**

****

**Penjelasan**

(import 'dart:io';) : Digunakan agar kita bisa membaca input dari keyboard (terminal).

### **(Cla**ss**** Kalimat)

Class ini menyimpan kalimat dan memiliki dua metode utama:

a. Konstruktor

dart

CopyEdit

Kalimat(this.teks);

* Menyimpan input kalimat dari pengguna ke variabel teks.

b. Metode balikKalimat()

dart

CopyEdit

String balikKalimat() {

return teks.split('').reversed.join('');

}

* split(''): Memecah string menjadi list karakter.
* reversed: Membalik urutan karakter.
* join(''): Menggabungkan kembali karakter menjadi string.

Kemudian ada Metode isPalindrom()

dart

CopyEdit

bool isPalindrom() {

String asli = teks.replaceAll(' ', '').toLowerCase();

String dibalik = balikKalimat().replaceAll(' ', '').toLowerCase();

return asli == dibalik;

}

* replaceAll(' ', ''): Menghapus spasi.
* toLowerCase(): Mengabaikan kapitalisasi.
* Bandingkan string asli dan yang dibalik. Jika sama, maka itu **palindrom**.

**QUEUE (ANTRIAN)**

**DASAR TEORI**

**Queue** atau dalam bahasa Indonesia yang berarti [antrean](https://kbbi.kemdikbud.go.id/entri/antrean" \t "_blank) adalah struktur data yang menyusun elemen-elemen data dalam urutan linier. Prinsip dasar dari struktur data ini adalah “First In, First Out” (FIFO) yang berarti elemen data yang pertama dimasukkan ke dalam antrean akan menjadi yang pertama pula untuk dikeluarkan.

Caranya bekerja adalah seperti jejeran orang yang sedang menunggu antrean di supermarket di mana orang pertama yang datang adalah yang pertama dilayani (**First In, First Out**). Pada struktur data ini, urutan pertama (data yang akan dikeluarkan) disebut **Front** atau **Head**. Sebaliknya, data pada urutan terakhir (data yang baru saja ditambahkan) disebut **Back**, **Rear**, atau **Tail**. Proses untuk menambahkan data pada antrean disebut dengan **Enqueue**, sedangkan proses untuk menghapus data dari antrean disebut dengan **Dequeue**.

**TUGAS PENDAHULUAN**

**Queue** atau **antrean** adalah salah satu struktur data linear yang bekerja berdasarkan prinsip **FIFO (First In, First Out)**, artinya data yang pertama kali masuk akan menjadi data pertama yang keluar. Dalam kehidupan sehari-hari, konsep ini bisa diibaratkan seperti antrean di loket: orang yang datang lebih dulu akan dilayani lebih dulu. Queue memiliki dua titik utama, yaitu **front** (tempat elemen dikeluarkan) dan **rear** (tempat elemen ditambahkan). Operasi dasar pada queue meliputi enqueue() untuk menambahkan elemen ke belakang antrean, dequeue() untuk menghapus elemen dari depan, front() untuk melihat elemen paling depan tanpa menghapusnya, serta isEmpty() dan isFull() untuk mengecek kondisi antrean. Ada beberapa jenis queue seperti **linear queue**, **circular queue** (yang memungkinkan penggunaan ruang memori lebih efisien dengan membuat rear kembali ke awal saat mencapai batas), **priority queue** (yang memproses elemen berdasarkan prioritas, bukan urutan masuk), dan **deque** (double-ended queue, di mana penambahan dan penghapusan bisa dilakukan dari kedua ujung). Dalam bidang komputasi, queue sangat berguna dan sering digunakan dalam manajemen tugas sistem operasi, penjadwalan proses, antrian pada printer, dan banyak aplikasi lainnya yang membutuhkan pemrosesan data secara berurutan. Dengan kemampuannya mengatur aliran data, queue merupakan salah satu struktur data penting dalam pemrograman dan sistem informasi.

**PERCOBAAN**

**Percobaan 1**

class Queue {

 List<int> \_elements = [];

 int \_front = 0;

 int \_rear = 0;

 int \_maxQueue = 0;

 void QueueOperation(int max) {

 \_front = 0;

 \_rear = -1;

 \_elements = List<int>.filled(max, 0);

 \_maxQueue = max - 1;

 }

 bool isEmpty() {

 return \_rear == -1 && \_front == 0;

 }

 bool isFull() {

 return \_rear == \_maxQueue - 1;

 }

 void enqueue(int data) {

 if (isFull()) {

 print("Queue Overflow, tidak dapat mengisi data lagi");

 } else {

 \_rear += 1;

 \_elements[\_rear] = data;

 }

 }

 int dequeue() {

 int data = 0;

 if (isEmpty()) {

 print("Queue Underflow");

 } else {

 data = \_elements[\_front];

 \_elements[\_front] = 0;

 \_front += 1;

 }

 return data;

 }

 void printQueue() {

 if (!isEmpty()) {

 print("Menampilkan urutan dari posisi tertinggi");

 for (int i = \_rear; i > -1; i--) {

 print("Elemen ke-$i = ${\_elements[i]}");

 }

 } else {

 print("Queue masih kosong");

 }

 }

}

void main() {

 Queue q = Queue();

 q.QueueOperation(100);

 q.enqueue(10);

 q.enqueue(20);

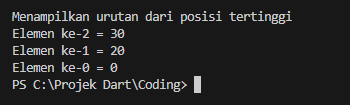
 q.enqueue(30);

 q.dequeue();

 q.printQueue();

}

**Output**

****

**Percobaan 2**

class Queue {

 int front = 0, rear = 0, size = 0;

 int capacity = 0;

 List<int> array = [];

 void QueueOperation(int capacity) {

 this.capacity = capacity;

 front = size = 0;

 rear = capacity - 1;

 array = List<int>.filled(capacity, 0);

 }

 bool isFull() {

 return (size == capacity);

 }

 bool isEmpty() {

 return (size == 0);

 }

 void enqueue(int item) {

 if (isFull()) return;

 rear = (rear + 1) % capacity;

 array[rear] = item;

 size++;

 print('$item enqueued to queue');

 }

 int dequeue() {

 if (isEmpty()) return -1;

 int item = array[front];

 front = (front + 1) % capacity;

 size--;

 return item;

 }

 int frontElement() {

 if (isEmpty()) return -1;

 return array[front];

 }

 int rearElement() {

 if (isEmpty()) return -1;

 return array[rear];

 }

}

void main() {

 Queue q = Queue();

 q.QueueOperation(1000);

 q.enqueue(10);

 q.enqueue(20);

 q.enqueue(30);

 q.enqueue(40);

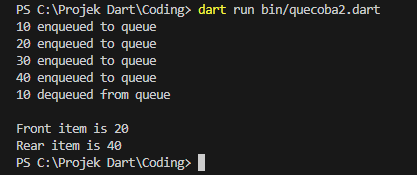
 print('${q.dequeue()} dequeued from queue\n');

 print('Front item is ${q.frontElement()}');

 print('Rear item is ${q.rearElement()}');

}

**Output**

****

**LATIHAN**

**Latihan 1**

import 'dart:io';

class CircularQueue {

  List<int> \_queue = [];

  int \_front = -1;

  int \_rear = -1;

  int \_size = 0;

  CircularQueue(int capacity) {

    \_queue = List<int>.filled(capacity, 0);

    \_size = capacity;

  }

  bool isEmpty() {

    return \_front == -1;

  }

  bool isFull() {

    return (\_rear + 1) % \_size == \_front;

  }

  void enqueue(int data) {

    if (isFull()) {

      print("Queue Overflow (Penuh), tidak bisa menambahkan data $data");

    } else {

      if (\_front == -1) \_front = 0;

      \_rear = (\_rear + 1) % \_size;

      \_queue[\_rear] = data;

      print("Data $data dimasukkan ke queue.");

    }

  }

  int dequeue() {

    if (isEmpty()) {

      print("Queue Underflow (Kosong)");

      return -1;

    }

    int data = \_queue[\_front];

    if (\_front == \_rear) {

      // Queue hanya punya 1 elemen

      \_front = -1;

      \_rear = -1;

    } else {

      \_front = (\_front + 1) % \_size;

    }

    print("Data $data dihapus dari queue.");

    return data;

  }

  void printQueue() {

    if (isEmpty()) {

      print("Queue masih kosong.");

      return;

    }

    print("Isi queue:");

    int i = \_front;

    while (true) {

      print("-> ${\_queue[i]}");

      if (i == \_rear) break;

      i = (i + 1) % \_size;

    }

  }

}

void main() {

  CircularQueue q = CircularQueue(5);

  q.enqueue(10);

  q.enqueue(20);

  q.enqueue(30);

  q.enqueue(40);

  q.enqueue(50); // Seharusnya penuh

  q.dequeue();

  q.dequeue();

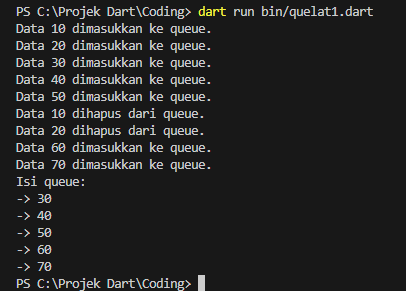
  q.enqueue(60);

  q.enqueue(70); // Sekarang bisa masuk karena circular

  q.printQueue();

}

**Output**

****

**Penjelasan**

| **Komponen** | **Fungsi** |
| --- | --- |

|  |  |
| --- | --- |
| \_front | Menunjukkan elemen pertama di queue |

|  |  |
| --- | --- |
| \_rear | Menunjukkan elemen terakhir di queue |

|  |  |
| --- | --- |
| % \_size | Digunakan untuk membungkus indeks agar kembali ke 0 setelah mencapai batas array (circular) |

**KESIMPULAN**

Dengan adanya praktkum Stack dan Queue, dapat dipahami bahwa Pemilihan struktur data yang tepat sangat bergantung pada kebutuhan dan karakteristik permasalahan. **Stack** cocok digunakan pada aplikasi seperti fungsi rekursif, undo-redo, dan pemeriksaan ekspresi matematika sedangkan **Queue** cocok untuk aplikasi seperti penjadwalan tugas, antrian cetak, atau sistem pemros esan data secara berurutan.

**REFERENSI**

**Modul Stack & Queue :** <https://lms.poliwangi.ac.id/pluginfile.php/26879/mod_resource/content/1/Modul%20Struktur%20Data%2006%20-%20Queue.pdf>

**Chat Gpt :** <https://openai.com/index/chatgpt/>

**W3School :** <https://www.w3schools.com/>

**LINK GitHub**