**LAPORAN PRAKTIKUM**

**STRUKTUR DATA**

**(Rekursif)**

**Dosen Pengampu : Lutfi Hakim, S.Pd., M.T.**



**Disusun Oleh :**

Vina Faizatus Sofita

362458302095

**Program Studi Teknologi Rekayasa Perangkat Lunak**

**Jurusan Bisnis Dan Informatika**

**Politeknik Negeri Banyuwangi**

**2025**

**REKURSIF**

**<https://github.com/viinafai/STRUKTUR-DATA-REKRUSIF>**

1. **Tujuan**
2. Memahami mengenai konsep rekursif
3. Mampu memecahkan permasalahan dengan konsep rekursif
4. **Dasar Teori**

Rekursif adalah suatu proses atau prosedur dari fungsi yang memanggil dirinya sendiri secara berulangulang. Karena proses dalam Rekursif ini terjadi secara berulang-ulang maka harus ada kondisi yang membatasi pengulangan persebut, jika tidak maka proses tidak akan pernah berhenti sampai memori yang digunakan untuk menampung proses tersebut tidak dapat menampung lagi/penuh.

Kelebihan Fungsi Rekursif adalah program menjadi lebih singkat. Pada beberapa kasus, lebih mudah menggunakan fungsi rekursif, contohnya: pangkat, factorial, dan fibonacci, dan beberapa proses deret lainnya. Fungsi rekursif lebih efisien dan cepat dibandingkan proses secara iteratif. Kekurangan Fungsi Rekursif adalah memakan memori lebih besar, karena setiap bagian dari dirinya dipanggil, akan membutuhkan sejumlah ruang memori untuk penyimpanan. Rekursif sering kali tidak bisa berhenti sehingga memori akan terpakai habis dan program bisa hang.

1. **Tugas Pendahuluan**
2. **Apa yang dimaksud dengan rekursif?**

Rekursif adalah teknik dalam pemrograman di mana sebuah fungsi memanggil dirinya sendiri untuk menyelesaikan suatu permasalahan yang lebih kecil hingga mencapai kondisi dasar (base case). Rekursi sering digunakan dalam permasalahan seperti faktorial, Fibonacci dan pangkat.

1. **Tuliskan fungsi untuk menghitung nilai faktorial**

int faktorial(int n) {

  if (n == 0 || n == 1) {

    return 1; // Base case

  }

  return n \* faktorial(n - 1); // Recursive case

}

void main() {

  int angka = 5; // Ubah angka sesuai kebutuhan

  print('Faktorial dari \$angka adalah: ${faktorial(angka)}');

}

Fungsi di atas menggunakan rekursi untuk menghitung faktorial. Jika Anda ingin mencoba dengan angka lain, cukup ubah nilai variabel angka dalam main()

1. **Tuliskan fungsi untuk menampilkan nilai fibonacci dari dere fibonacci**

import 'dart:io';

int fibonacci(int n) {

  if (n == 0) {

    return 0; // Base case pertama

  } else if (n == 1) {

    return 1; // Base case kedua

  }

  return fibonacci(n - 1) + fibonacci(n - 2); // Recursive case

}

void main() {

  int jumlah = 10; // Ubah jumlah sesuai kebutuhan

  print('Deret Fibonacci:');

  for (int i = 0; i < jumlah; i++) {

    stdout.write('${fibonacci(i)} ');

  }

  print(''); // Pindah ke baris baru setelah mencetak semua angka

}

Fungsi di atas menggunakan rekursi untuk menghitung nilai Fibonacci. Anda dapat mengubah nilai jumlah dalam main() untuk menampilkan lebih banyak angka dalam deret Fibonacci.

1. **Apa yang dimaksud dengan rekursif tail?**

**Rekursif tail** (tail recursion) adalah bentuk rekursi di mana pemanggilan rekursif dilakukan sebagai **operasi terakhir** dalam fungsi. Artinya, setelah pemanggilan rekursif dilakukan, tidak ada lagi perhitungan tambahan yang harus dilakukan oleh fungsi.

1. **Tuliskan fungsi untuk menghitung deret fibonacci menggunakan tail rekursif!**

****

1. Fungsi fibonacciTail (n, a, b) :

* a adalah nilai fibonacci ke-( n – 2)
* b adalah nilai fibonacci ke-( n – 1)
* fungsi teu memanggil dirinya sendiri sampai n == 0 atau n == 1.

1. Fungsi fibonacci (n) :

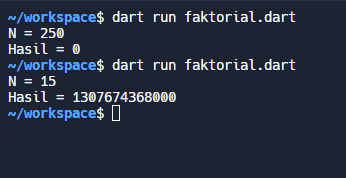
* Memulai rekusi dengan a = 0 dan b = 1, yaitu dua angka pertama dalam deret fibonacci.

**Percobaan**

1. **Percobaan 1 : Fungsi rekursif untuk menghitung nilai faktorial**



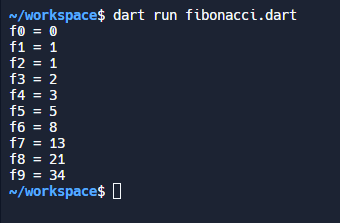
**Output :**

****

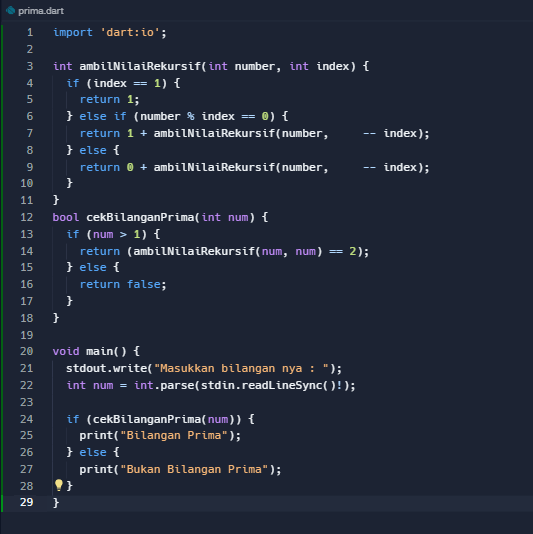
1. **Percobaan 2 : Fungsi rekursif untuk menampilkan deret fibonacci**

****

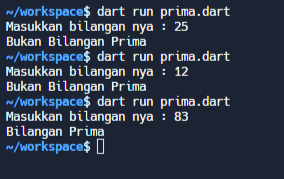
**Output :**

****

1. **Percobaan 3 : Fungsi rekursif untuk menentukan bilangan prima atau bukan prima**

****

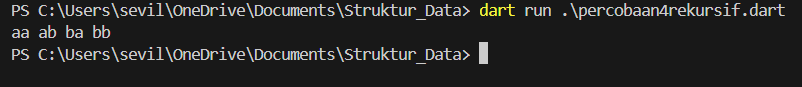
**Output :**

****

1. **Percobaan 4 : Fungsi rekursi untuk menampilkan kombinasi 2 karakter**

****

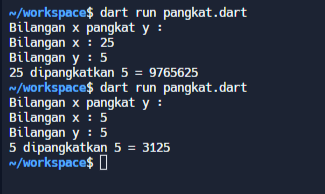
**Output :**

****

1. **Percobaan 5 : Fungsi rekursi untuk menghitung pangkat**

****

**Output :**

****

1. **Percobaan 6 : Fungsi tail rekursif untuk menampilkan i**

****

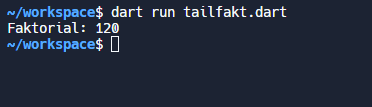
**Output :**

****

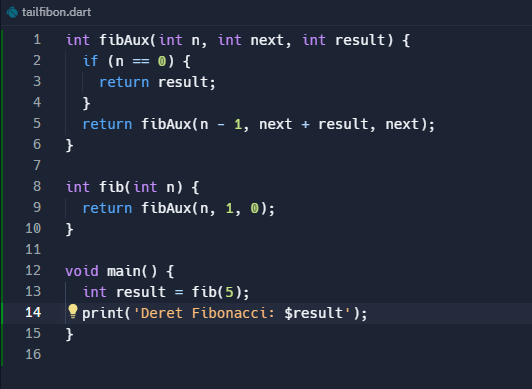
1. **Percobaan 7 : Fungsi tail rekursif untuk menghitung faktorial**

****

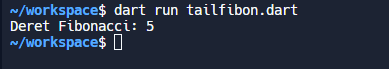
**Output :**

****

1. **Percobaan 8 : Fungsi tail rekursif untuk menghitung fibonacci**

****

**Output :**

****

1. **Latihan**
2. **Buatlah program rekursif untuk menghitung segitiga Pascal !**

import 'dart:io';

// Fungsi rekursif untuk menghitung elemen pada Segitiga Pascal

int pascal(int n, int k) {

  if (k == 0 || k == n) {

    return 1;

  } else {

    return pascal(n - 1, k - 1) + pascal(n - 1, k);

  }

}

// Fungsi untuk mencetak Segitiga Pascal dengan format sama kaki

void printPascalsTriangle(int rows) {

  for (int i = 0; i < rows; i++) {

    // Menambahkan spasi untuk membuat segitiga sama kaki

    for (int j = 0; j < rows - i - 1; j++) {

      stdout.write(' ');

    }

    // Mencetak elemen-elemen segitiga Pascal pada baris ke-i

    for (int j = 0; j <= i; j++) {

      stdout.write('${pascal(i, j)} ');

    }

    // Pindah ke baris berikutnya

    print('');

  }

}

void main() {

  // Meminta input jumlah baris segitiga Pascal

  print('Masukkan jumlah baris segitiga Pascal: ');

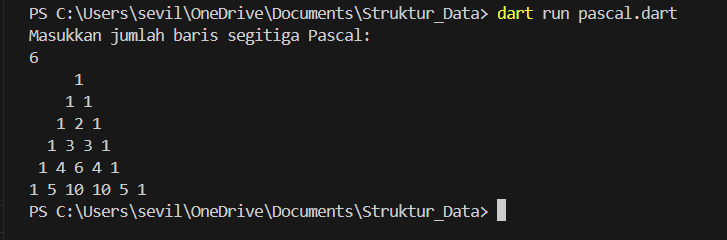
  int rows = int.parse(stdin.readLineSync()!);

  // Mencetak Segitiga Pascal

  printPascalsTriangle(rows);

}

**Output :**

****

**Penjelasan :**

### **Fungsi** pascal(n, k) **– Menghitung Nilai Segitiga Pascal**

int pascal(int n, int k) {

  if (k == 0 || k == n) {

    return 1;

  } else {

    return pascal(n - 1, k - 1) + pascal(n - 1, k);

  }

}

* Jika k == 0 atau k == n, maka hasilnya **1** (karena setiap baris Segitiga Pascal selalu dimulai dan diakhiri dengan angka 1).
* Nilai Segitiga Pascal pada posisi (n, k) dihitung sebagai jumlah dari dua elemen di atasnya, yaitu (n-1, k-1) dan (n-1, k)
* Fungsi printPascalsTriangle(rows) – Mencetak Segitiga Pascal

void printPascalsTriangle(int rows) {

  for (int i = 0; i < rows; i++) {

    // Menambahkan spasi untuk membuat segitiga sama kaki

    for (int j = 0; j < rows - i - 1; j++) {

      stdout.write(' ');

    }

    // Mencetak elemen-elemen segitiga Pascal pada baris ke-i

    for (int j = 0; j <= i; j++) {

      stdout.write('${pascal(i, j)} ');

    }

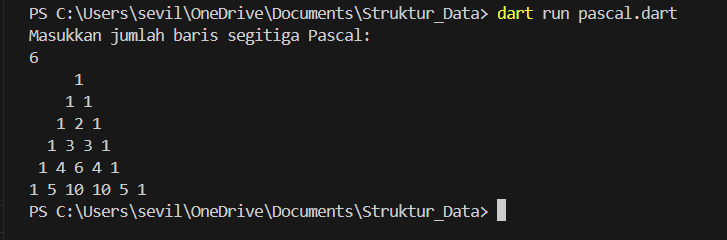
    // Pindah ke baris berikutnya

    print('');

  }

}

* **Loop pertama** (for (int i = 0; i < rows; i++)) mengontrol jumlah baris yang akan dicetak.
* **Loop kedua** (for (int j = 0; j < rows - i - 1; j++)) menambahkan **spasi** agar hasil berbentuk segitiga.
* **Loop ketiga** (for (int j = 0; j <= i; j++)) mencetak nilai Segitiga Pascal dengan memanggil pascal(i, j).
* Menggunakan stdout.write() untuk mencetak angka dalam satu baris, lalu print('') agar berpindah ke baris baru.
* **Output**



1. **Buat program BinarySearch dengan Rekursif ! (data tentukan sendiri): Data : 2,5,8,10,14,32, 35, 41, 67, 88, 90, 101, 109 Data yang dicari : 10 Data 10 berada pada indek ke – 3!**

int binarySearchRecursive(List<int> arr, int left, int right, int target) {

  if (left > right) {

    return -1; // Target tidak ditemukan

  }

  int mid = left + (right - left) ~/ 2;

  if (arr[mid] == target) {

    return mid; // Target ditemukan pada indeks mid

  } else if (arr[mid] > target) {

    return binarySearchRecursive(arr, left, mid - 1, target); // Cari di kiri

  } else {

    return binarySearchRecursive(arr, mid + 1, right, target); // Cari di kanan

  }

}

void main() {

  List<int> data = [2, 5, 8, 10, 14, 32, 35, 41, 67, 88, 90, 101, 109];

  int target = 10;

  int result = binarySearchRecursive(data, 0, data.length - 1, target);

  if (result != -1) {

    print('Data $target berada pada indeks ke-$result');

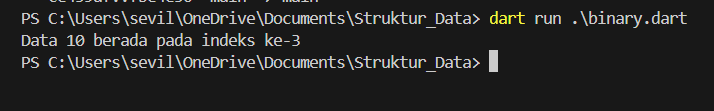
  } else {

    print('Data $target tidak ditemukan');

  }

}

**Output :**

****

**Penjelasan :**

* **Deklarasi list** data

List<int> data = [2, 5, 8, 10, 14, 32, 35, 41, 67, 88, 90, 101, 109];

* Daftar ini sudah **terurut**, yang merupakan syarat utama **Binary Search**.
* Menentukan target yang akan dicari

int target = 10;

* Memanggil fungsi binarySearchRecursive()

int result = binarySearchRecursive(data, 0, data.length - 1, target);

* Pencarian dilakukan mulai dari indeks **0** hingga **data.length - 1**.
* Menampilkan hasil pencarian

if (result != -1) {

    print('Data $target berada pada indeks ke-$result');

  } else {

    print('Data $target tidak ditemukan');

  }

}

· Jika result != -1, berarti elemen ditemukan dan indeksnya ditampilkan.

· Jika result == -1, berarti elemen tidak ada dalam daftar.

1. **Kesimpulan**

Dalam praktikum ini dapat disimpulkan bahwa, Rekursif adalah metode yang efektif untuk menyelesaikan masalah berulang seperti faktorial, deret Fibonacci, Segitiga Pascal, dan Binary Search. Meskipun rekursi membuat kode lebih sederhana, beberapa kasus seperti Fibonacci memerlukan optimasi untuk menghindari perhitungan berulang. Binary Search rekursif lebih efisien dibanding pencarian linier, sedangkan Segitiga Pascal mudah dihitung namun kurang optimal untuk jumlah baris besar. Kesimpulannya, rekursi sangat berguna tetapi harus dioptimalkan agar tidak menyebabkan penggunaan memori berlebihan.

1. **Referensi**

* Modul
* **<https://sko.dev/>**