Laporan Praktikum Algoritma dan Pemrograman

Semester Genap 2023/2024

NIM	**71230986**
Nama Lengkap	**TOMAS BECKET**
Minggu ke / Materi	**13 / Rekursif**

SAYA MENYATAKAN BAHWA LAPORAN PRAKTIKUM INI SAYA BUAT DENGAN USAHA SENDIRI TANPA MENGGUNAKAN BANTUAN ORANG LAIN. SEMUA MATERI YANG SAYA AMBIL DARI SUMBER LAIN SUDAH SAYA CANTUMKAN SUMBERNYA DAN TELAH SAYA TULIS ULANG DENGAN BAHASA SAYA SENDIRI.

SAYA SANGGUP MENERIMA SANKSI JIKA MELAKUKAN KEGIATAN PLAGIASI, TERMASUK SANKSI TIDAK LULUS MATA KULIAH INI.

PROGRAM STUDI INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
UNIVERSITAS KRISTEN DUTA WACANA
YOGYAKARTA
2024

BAGIAN 1: MATERI MINGGU INI (40%)

MATERI 1

PENGERTIAN REKURSIF

Rekursif adalah suatu teknik pemrograman di mana sebuah fungsi memanggil dirinya sendiri sebagai bagian dari eksekusi. Ini biasanya digunakan untuk menyelesaikan masalah yang dapat dipecah menjadi submasalah yang lebih kecil dengan struktur yang serupa dengan masalah aslinya. Rekursif sering digunakan dalam berbagai algoritma, seperti pencarian, pengurutan, dan operasi pada struktur data seperti pohon dan graf.

Ada dua bagian utama dalam rekursif:

1. **Basis Kasus (Base Case):**
    - Basis kasus adalah kondisi yang menghentikan rekursi. Ini adalah kondisi yang paling sederhana di mana masalah dapat diselesaikan tanpa memerlukan pemanggilan rekursif lebih lanjut. Tanpa basis kasus, fungsi rekursif akan memanggil dirinya sendiri tanpa henti, yang akan menyebabkan tumpukan panggilan (call stack) meluap dan program mengalami crash.
    - Contoh sederhana: Pada fungsi faktorial, basis kasusnya adalah **faktorial(0) = 1**.
2. **Panggilan Rekursif (Recursive Call):**
    - Panggilan rekursif adalah bagian dari fungsi di mana fungsi tersebut memanggil dirinya sendiri untuk memecahkan submasalah yang lebih kecil. Setiap panggilan rekursif harus membawa masalah lebih dekat ke basis kasus.
    - Contoh sederhana: Pada fungsi faktorial, panggilan rekursifnya adalah **rekrusif(x) = rekrusif(x-1)**.

```
# Bentuk Umum dan studi Kasus
def rekrusif(x):
    print(x)
    if x == 1:
        return 0
    else:
        return rekrusif(x-1)

print(rekrusif(5))
```

sehingga Pada Source code diatas akan menunjukan angka 5 sampai dengan 1 dengan memanggil fungsi rekursif(x – 1) untuk melakukan perulangan.

MATERI 2

KELEBIHAN DAN KEKURANGAN

Beberapa keunggulan fungsi rekursif adalah salah satunya :

- Kode program lebih singkat dan elegan, Rekursif sering menghasilkan kode yang lebih sederhana dan lebih mudah dibaca, terutama untuk masalah yang secara alami bersifat rekursif (seperti pohon biner, graf, dan algoritma pembagian dan penaklukan).
- Masalah Kompleks dapat di breakdown menjadi sub masalah yang lebih kecil dalam rekursif, Rekursif dapat mengurangi jumlah kode yang perlu ditulis, terutama jika dibandingkan dengan solusi iteratif yang mungkin memerlukan lebih banyak kode boilerplate atau pengaturan struktur data tambahan.

Namun adapun beberapa kelemahannya dari fungsi rekursif ini sebagau contoh:

- Memakan memori yang lebih besar karena setiap kali bagian dirinya dipanggil maka dibu tuhkan sejumlah ruang memori tambahan,
- Mengorbankan efisiensi dan kecepatan, Beberapa algoritma rekursif dapat menjadi sangat tidak efisien jika tidak dioptimalkan dengan teknik seperti memoization atau dynamic programming. Misalnya, algoritma rekursif untuk menghitung deret Fibonacci tanpa optimasi memiliki kompleksitas eksponensial.
- Fungsi rekursif sulit dilakukan debugging dan kadang sulit dimenegerti

MATERI 3

BENTUK UMUM DAN STUDI KASUS

MATERI 4

Secara teori, semua fungsi rekursif memang memiliki solusi iteratif. Ini karena rekursi dan iterasi adalah dua teknik yang dapat digunakan untuk mencapai hasil yang sama, meskipun dengan cara yang berbeda. Namun, mengubah fungsi rekursif menjadi iteratif atau sebaliknya bisa bervariasi dalam tingkat kesulitan tergantung pada masalah yang dihadapi. Misal contoh sederhana yaitu faktorial

```
def faktorial(n):
    if n == 0:
        return 1
    else:
        return n * faktorial(n - 1)
```

dan fungsi iteratifnya yaitu

```
def faktorial_iteratif(n):
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
```

sehingga kita tahu Meskipun semua fungsi rekursif bisa diubah menjadi solusi iteratif, dan sebaliknya, penting untuk memilih pendekatan yang paling sesuai dengan masalah yang dihadapi. Rekursi menawarkan solusi yang lebih sederhana dan alami untuk beberapa jenis masalah, sementara iterasi bisa lebih efisien dalam hal penggunaan memori dan kontrol alur program. Keputusan antara menggunakan rekursi atau iterasi seringkali bergantung pada preferensi pribadi, kinerja yang diinginkan, dan karakteristik spesifik dari masalah yang diselesaikan.

MATERI 5

CONTOH PROBLEM SOLVING 1

Membuat sebuah program yang dapat melakukan perkalian antara 2 buah bilangan dengan menggunakan fungsi rekursif. Misalkan kita ingin mengalikan angka 2 dengan 4. Dengan metode penjumlahan diperoleh 2*4 = 2 + 2 + 2 + 2 = 8.

```
# KEGIATAN PRAKTIKUM
def kali(angka1, angka2):
    if angka2 == 1:
        print(f"{angka1} = ", end="")
        return angka1
    else:
        print(f"{angka1} + ", end="")
        return angka1 + kali(angka1, angka2-1)

print(kali(2, 4))
```

contoh ini Base case terdapat pada if angka2 == 1 maka return angka 1 dimana perulangan akan berhenti dan memnuhi kondisi tersebut. Dan fungsi rekursif terdapat pada else: return angka1 + kali(angka1, angka2-1) untuk memanggil fungsi kali dan melakukan perulangan hingga angka2 = 1. Sehingga…

Hasil yang muncul yaitu : 2 + 2 + 2 + 2 = 8

```
ekursif\tempCodeRunnerFile.py"
2 + 2 + 2 + 2 = 8
```

MATERI 6

CONTOH PROBLEM SOLVING 3 FIBONAC

Bilangan fibonacci adalah bilangan yang berasal dari penjumlahan 2 bilangan sebelumnya. Sehingga jika dibuat fungsi rekursif nya maka :

```
def fibonac(n):
    if n == 1:
        return [1]
    elif n == 2:
        return [1, 1]
    else:
        call = fibonac(n-1)
        call.append(call[-1] + call[-2])
        return call

print(fibonac(7))
```

kita perlu membuat base case dari fungsi fibonac sendiri dimana base case nya yaitu n == 1 atau n == 2 yang akan memenuhi kondisi perulangannya nanti, dan rekursif case nya yaitu terdapat pada call = fibonac(n-1) yang akan memanggil fungsi fibonac dengan mengurangi variable n dengan 1 untuk menjalankan perulangan. Sehingga jika dilihat prosesnya maka akan seperti ini :

ketika kita memanggil **fibonac(7)**, ada beberapa alur eksekusinya:

1. **fibonac(7)** memanggil **fibonac(6)**
2. **fibonac(6)** memanggil **fibonac(5)**
3. **fibonac(5)** memanggil **fibonac(4)**
4. **fibonac(4)** memanggil **fibonac(3)**
5. **fibonac(3)** memanggil **fibonac(2)** (base case, mengembalikan **[1, 1]**)
6. **fibonac(3)** kemudian menambahkan **1 + 1** ke daftar, menghasilkan **[1, 1, 2]**
7. **fibonac(4)** menambahkan **2 + 1**, menghasilkan **[1, 1, 2, 3]**
8. **fibonac(5)** menambahkan **3 + 2**, menghasilkan **[1, 1, 2, 3, 5]**
9. **fibonac(6)** menambahkan **5 + 3**, menghasilkan **[1, 1, 2, 3, 5, 8]**
10. **fibonac(7)** menambahkan **8 + 5**, menghasilkan **[1, 1, 2, 3, 5, 8, 13]**

Hasil akhir dari **fibonac(7)** adalah **[1, 1, 2, 3, 5, 8, 13]**.

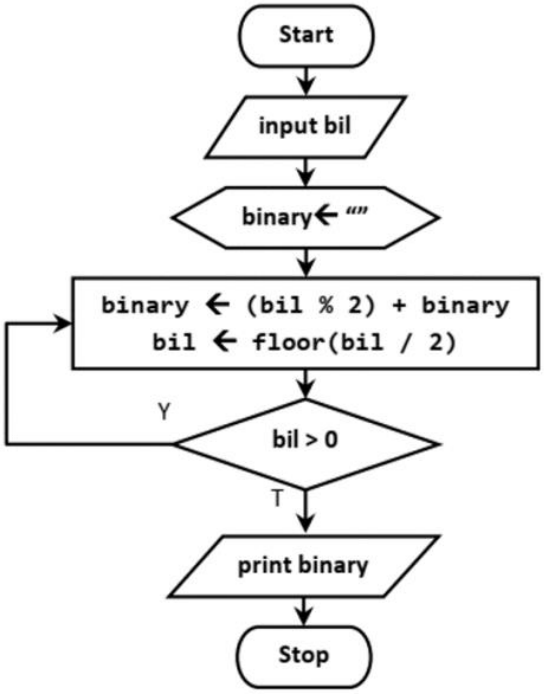
MATERI 7

CONTOH PROBLEM SOLVING 4 KONVERSI BASIS

Akan dibuat suatu program yang dapat mengkonversi suatu bilangan dari basis 10 ke basis lainnya. Input berupa bilangan dalam bais 10 dan basis bilangan (selain basis 10) dan program akan dibuat secara fungsi rekursif sehingga…

```
def konversi(angka, basis):
    x = '01'
    if angka < basis:
        return x[angka]
    else:
        return konversi(angka // basis, basis) + x[angka % basis]

print(konversi(8, 2))
```



- Jika kita mengetahui bentuk dari rekursifnya maka akan mudah menentukan base case dan rekursif case nya, sehingga jika kita ingin membuat rekursif casenya kita harus cari tahu terlebih dahulu kondisi untuk memenuhi syarat mengubah angka menjadi basis 10 bagaimana. Seperti yang ditunjukan pada Flowchart disamping. Sehingga
- **Base Case:**

  if angka < basis: return x[angka]
- **Recursive Case:**

  return konversi(angka // basis, basis) + x[angka % basis]

dan fungsi ini menggunakan rekursi untuk membagi angka menjadi bagian-bagian yang lebih kecil dan membangun string representasi basis dari hasil-hasil rekursif tersebut.

BAGIAN 2: LATIHAN MANDIRI (60%)

SOAL 1

1.1 SOURCE CODE

```
def ini_prima(n, pembagi=None):

    if pembagi is None:
        pembagi = n - 1

    if n < 2:
        return False

    if pembagi == 1:
        return True

    if n % pembagi == 0:
        return False

    return ini_prima(n, pembagi - 1)


bilangan = 29
if ini_prima(bilangan):
    print(f"{bilangan} bilangan prima")
else:
    print(f"{bilangan} bukan bilangan prima")
```

1.2 PENJELASAN

fungsi rekursif ini akan mengecek apakah suatu bilangan n adalah bilangan prima atau bukan. Mari kita jelaskan baris per baris:

1. **def ini_prima(n, pembagi=None):**: Mendefinisikan fungsi **ini_prima** dengan dua parameter, yaitu **n** (bilangan yang akan diperiksa) dan **pembagi** (pembagi saat ini dalam proses pencarian faktor).

   ```
   def ini_prima(n, pembagi=None):
   ```

2. **if pembagi is None:**: Ini adalah kondisi yang mengecek apakah **pembagi** belum ditentukan saat ini. Jika ya, maka **pembagi** akan diinisialisasi dengan nilai **n - 1**.

   ```
   if pembagi is None:
           pembagi = n - 1
   ```

3. **if n < 2:**: Ini adalah basis kasus pertama. Jika **n** kurang dari 2, maka **n** bukanlah bilangan prima dan fungsi akan mengembalikan **False**.

```
if n < 2:
        return False
```

4. **if pembagi == 1:**: Ini adalah basis kasus kedua. Jika **pembagi** turun hingga 1 tanpa menemukan faktor yang membagi **n**, maka **n** adalah bilangan prima dan fungsi akan mengembalikan **True**.

```
if pembagi == 1:
        return True
```

5. **if n % pembagi == 0:**: Ini adalah kondisi yang mengecek apakah **n** habis dibagi oleh **pembagi**. Jika ya, maka **n** bukanlah bilangan prima dan fungsi akan mengembalikan **False**.

```
if n % pembagi == 0:
        return False
```

6. **return ini_prima(n, pembagi - 1)**: Ini adalah langkah rekursif. Fungsi akan memanggil dirinya sendiri dengan **n** dan **pembagi - 1** sebagai argumen. Ini akan mengulangi proses pencarian faktor dari **n** hingga **pembagi** turun hingga 1.

```
return ini_prima(n, pembagi - 1)
```

1.3 HASIL OUTPUT

```
29 bilangan prima
```

SOAL 2

2.1 SOURCE CODE

```python
def palindrom(kalimat):

    if len(kalimat) <= 1:
        return True

    if kalimat[0] != kalimat[-1]:
        return False

    return palindrom(kalimat[1:-1])


kalimat1 = "kasur rusak"
kalimat2 = "saya makan"

print(f"{kalimat1} = {palindrom(kalimat1)}")
print(f"{kalimat2} = {palindrom(kalimat2)}")
```

2.2 PENJELASAN

Fungsi plaindrom yang dibuat pada Source code diatas menunjukan :

1. **def palindrom(kalimat):**: Mendefinisikan fungsi **palindrom** yang mengambil satu parameter, yaitu **kalimat** (kalimat yang akan diperiksa).
2. **if len(kalimat) <= 1:**: Ini adalah basis kasus pertama. Jika panjang **kalimat** kurang dari atau sama dengan 1, maka **kalimat** merupakan palindrom (karena kalimat dengan satu karakter atau kosong dianggap sebagai palindrom).
3. **if kalimat[0] != kalimat[-1]:**: Ini adalah kondisi yang mengecek apakah karakter pertama tidak sama dengan karakter terakhir dari **kalimat**. Jika ya, maka **kalimat** bukanlah palindrom dan fungsi akan mengembalikan **False**.
4. **return palindrom(kalimat[1:-1])**: Ini adalah langkah rekursif. Fungsi akan memanggil dirinya sendiri dengan **kalimat[1:-1]** sebagai argumen. Ini akan memeriksa sisa kalimat tanpa karakter pertama dan terakhirnya.

2.3 HASIL OUTPUT

```
kasur rusak = True
saya makan = False
```

SOAL 3

3.1 SOURCE CODE

```
def jumlah_deret_ganjil(n):
    if n == 1:
        return 1
    else:
        if n % 2 == 0:
            return 0 + jumlah_deret_ganjil(n - 1)
        else:
            return n + jumlah_deret_ganjil(n - 2)

n = 11
print("Jumlah deret ganjil untuk n =", n, "adalah", jumlah_deret_ganjil(n))
```

3.2 PENJELASAN

Source code diatas dibuat untuk melakukan fungsi rekursif dalam menghitung jumlah deret ganjil hingga suku ke-n. sehingga penjelasannya :

1. **def jumlah_deret_ganjil(n):**: Mendefinisikan fungsi **jumlah_deret_ganjil** yang mengambil satu parameter, yaitu **n** (jumlah suku dalam deret).
2. **if n == 1:**: Ini adalah basis kasus. Jika **n** sama dengan 1, maka fungsi akan mengembalikan 1, karena suku pertama dari deret ganjil adalah 1.
3. **if n % 2 == 0:**: Ini adalah kondisi yang mengecek apakah **n** adalah bilangan genap. Jika ya, itu berarti **n** tidak termasuk dalam deret ganjil. Oleh karena itu, jumlah deret ganjil dari suku ke-n adalah jumlah deret ganjil dari suku ke-(n-1).
4. **return 0 + jumlah_deret_ganjil(n - 1)**: Jika **n** adalah bilangan genap, fungsi akan mengembalikan hasil rekursi dengan **n-1**, karena **n** tidak termasuk dalam deret ganjil.
5. **return n + jumlah_deret_ganjil(n - 2)**: Jika **n** adalah bilangan ganjil, itu berarti **n** termasuk dalam deret ganjil. Oleh karena itu, jumlah deret ganjil dari suku ke-n adalah **n** ditambah dengan jumlah deret ganjil dari suku ke-(n-2).

3.3 HASIL OUTPUT

```
Jumlah deret ganjil untuk n = 11 adalah 36
```

SOAL 4

4.1 SOURCE CODE

```
def jumlah_digit(bilangan):
    if len(bilangan) == 1:
        return int(bilangan), bilangan
    else:
        jumlah_sisa = jumlah_digit(bilangan[1:])
        return int(bilangan[0]) + jumlah_sisa[0], bilangan[0] + "+" +
jumlah_sisa[1]

bilangan = "234"
jumlah, deret = jumlah_digit(bilangan)
print(f"{bilangan} maka jumlah digitnya adalah {deret} = {jumlah}")
```

4.2 PENJELASAN

Untuk problem solve ini akan menentukan atau menghitung jumlah digit yang ada pada sebuah string seperti contoh “123” maka akan dihitung jumlahnya berapa dengan menuliskan 1+2+3 = 6. Sehingga penjelasan pada source code diatas :

1. **def jumlah_digit(bilangan):**: Mendefinisikan fungsi **jumlah_digit** yang mengambil satu parameter, yaitu **bilangan** (bilangan yang akan dihitung jumlah digitnya).
2. **if len(bilangan) == 1:**: Ini adalah basis kasus. Jika panjang **bilangan** hanya satu digit, maka fungsi akan mengembalikan tuple yang berisi bilangan tersebut sebagai integer dan string.
3. **jumlah_sisa = jumlah_digit(bilangan[1:])**: Ini adalah langkah rekursif. Fungsi akan memanggil dirinya sendiri dengan parameter **bilangan[1:]**, yaitu **bilangan** tanpa digit pertamanya. Ini akan menghitung jumlah digit dan deret digit dari sisa bilangan.
4. **return int(bilangan[0]) + jumlah_sisa[0], bilangan[0] + "+" + jumlah_sisa[1]**: Ini adalah langkah untuk menghitung jumlah digit dan menyusun deret digit. Kami menambahkan digit pertama dari **bilangan** dengan jumlah digit dari sisa bilangan. Kami juga menyusun deret digit dengan menggabungkan digit pertama dengan tanda tambah (**+**) dan deret digit dari sisa bilangan.

4.3 HASIL OUTPUT

```
234 maka jumlah digitnya adalah 2+3+4 = 9
```

SOAL 5

5.1 SOURCE CODE

```python
def kombinasi(n, k):
    if k == 0 or k == n:
        return 1
    else:
        return kombinasi(n - 1, k - 1) + kombinasi(n - 1, k)

n = 5
k = 2
print(f"C({n}, {k}) = {kombinasi(n, k)}")
```

5.2 PENJELASAN

1. **def kombinasi(n, k):**: Mendefinisikan fungsi **kombinasi** yang mengambil dua parameter, yaitu **n** dan **k** (dari $C(n,k)C(n,k)$).
2. **if k == 0 or k == n:**: Ini adalah basis kasus. Jika kk sama dengan 0 atau kk sama dengan nn, maka hasilnya adalah 1 karena setiap himpunan memiliki kombinasi yang unik yaitu himpunan itu sendiri dan himpunan kosong.
3. **return kombinasi(n - 1, k - 1) + kombinasi(n - 1, k)**: Ini adalah langkah rekursif. Fungsi akan memanggil dirinya sendiri dua kali, yaitu untuk mencari kombinasi dengan memilih elemen terakhir dan untuk mencari kombinasi tanpa memilih elemen terakhir. secara garis besar bentuk dari kombinasi seperti ini $C(n,k)=C(n-1,k-1)+C(n-1,k)C(n,k)=C(n-1,k-1)+C(n-1,k)$.
4.

5.3 HASIL OUTPUT

```
C(5, 2) = 10
```