PEMBAHASAN UAS DASAR PEMROGRAMAN 2020/2021

1. a. max2()

```
# max2 : 2 integer → integer
# max2 (a,b) menentukan bilangan maksimum dari 2 bilangan integer
# Realisasi
def max2 (a,b):
    if (a>b):
        return a
    else:
        return b
```

b. min2()

```
# min2 : 2 integer → integer
# min2 (a,b) menentukan bilangan maksimum dari 2 bilangan integer
# Realisasi
def min2 (a,b):
    if (a<b):
        return a
    else:
        return b
```

c. max_list()

```
# max_list : list → integer
# max_list (L) menentukan bilangan maksimum dari sebuah list
# Realisasi
def max_list (L):
    if empty_list (L):
        return 0
    elif is_one_element (L):
        return first_element (L)
    else:
        return max2(first_element (L), max_list (tail(L)))
```

untuk no. 1c dan 1d

```
def is_one_element(L):
    if ((not empty_list (L)) and
        empty_list(tail(L))):
        return True
    else:
        return False
```

d. Min_list()

```
# min_list : list → integer
# min_list (L) menentukan bilangan minimum dari sebuah list
# Realisasi
def min_list (L):
    if empty_list (L):
        return 0
    elif is_one_element (L):
        return first_element (L)
    else:
        return min2 (first_element (L), min_list
                     (tail(L)))
```

2 a. total_elemen_daun ()

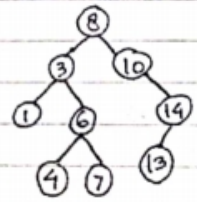
```
# total_elemen_daun : PohonBiner → integer
# total_elemen_daun (P) menghitung jumlah daun pada pohon biner
# Realisasi
def total_elemen_daun (P) :
    If is_one_element (P) :
        return akar (P)
    else:
        If is_biner (P) :
            return total_elemen_daun (left (P)) + total_elemen_daun (right (P))
        elif is_uner_left (P) :
            return total_elemen_daun (left (P)
        elif is_uner_right (P) :
            return total_elemen_daun (right (P))
```

b. total_elemen_node ()

```
# total_elemen_node : PohonBiner → integer
# total_elemen_... (P) menghitung jumlah semua elemen pada pohon biner
# Realisasi
def total_elemen_node (P)
    If is_one_element (P) :
        return akar (P)
    else:
        If is_biner (P) :
            return total_elemen_node (left (P)) + akar (P) + total_elemen_node (right (P))
        elif is_uner_left (P) :
            return total_elemen_node (left (P)) + akar (P)
        elif is_uner_right (P) :
            return akar (P) + total_elemen_node (right (P))
```

C. BST ()

Jika P = /L A R\



1. Cek apakah akar 8 is_biner, ternyata, cek Apakah L < R
2. Cek apakah akar 3 is_biner, ternyata, cek apakah L < R.
3. Cek apakah akar 1 is_biner, ternyata akar 1 tidak ada cabang (daun)
4. Cek ke bagian kanan akar 3, yaitu 6. Apakah akar 6 is_biner. ∴ ternyata iya dan cek apakah L < R
5. L dan R dari akar 6 ada 4 dan 7 yang tidak memiliki cabang lagi (daun)

∴ P = [8, 3, 1, 6, 4, 7, 10, 14, 13]

6. setelah dari kiri akar 8, lalu cek bagian kanannya. yaitu 10.
7. Akar 10 hanya memiliki cabang kanan (uner_right) yaitu 14.
8. Akar 14 hanya memiliki cabang kiri (uner_left) yaitu 13
9. akar 13 tidak memiliki cabang (daun)

3.a. Fungsi pada python

```
# kelipatan 10 : integer → boolean
# kelipatan10 (x) bernilai true bila bilangan merupakan kelipatan 10
# Realisasi
def kelipatan10 (x) :
    if x % 10 == 0
        return True
    else :
        return False

# bukan_kelipatan10 : integer → boolean
# bukan_kelipatan10 (x) bernilai true bila bilangan ~~ber~~ bukan merupakan kelipatan 10
# Realisasi
def bukan_kelipatan10 (x) :
    if x % 10 != 0
        return True
    else :
        return False

# Filter_list : list → list
# Filter_list (L) mem-filter elemen list yang bernilai kelipatan 10 atau bukan
# Realisasi
def Filter_list (L,f) :
    if empty_list (L) :
        return []
    if not (f (first_elmt (L))) :
        return Filter_list (tail (L), f)
    else :
        return konso (first_elmt (L), Filter_list (tail), f))
```

b. menulis ekspresi lambda untuk L2 dan L3

```
L2 = Filter_list (L1, lambda x : x mod 10 = 0)
L3 = Filter_list (L1, lambda x : x mod 10 ≠ 0)
```

4. Himpunan

```
# minus : 2 list → list
# minus (H1, H2) menampilkan himpunan hasil selisih antara himpunan H1 dan H2
#realisasi
def minus (H1, H2) :
    if is_sub_set (H1, H2):
        return []
    elif is_member ( First_elmt (H1), H2):
        return minus (tail (H1), H2)
    else :
        return konso ( First_elmt (H1), minus (tail (H1), H2))
```