Axelliano Rafael Situmeang
2406012113 0089

Fungsi bantuan.

1) Definisi dan spesifikasi :
max2 : 2 integer → integer
{max2(a,b) menghasilkan nilai maksimum dari 2 buah integer}
Realisasi :

```
def max2(a,b):
    if a>=b:
        return a
    else:
        return b
```

Definisi dan spesifikasi :
min2 : 2 integer → integer
{min2(a,b) menghasilkan nilai minimum dari 2 buah integer}
Realisasi :

```
def min2(a,b):
    if a<=b:
        return a
    else:
        return b
```

Definisi dan spesifikasi :
is_one_elmt : list → integer
{ is_one_elmt (x) menghasilkan nilai
True jika List hanya terdiri dari
1 elemen y
Realisasi :

```
def is_one_elmt(x):
    if not (empty_list(x) and
        empty_list(tail(x)) :
            return True
    else :
            return False.
```

1 a) Definisi dan spesifikasi :
max_list : list → integer
{max_list (L) menghasilkan nilai maksimum dari sebuah list y
Realisasi :

```
def max_list(L):
    if is_one_elmt (L) :
        return L
    else :
        return max2 (first_element(L),
                     max_list(tail(L)))
```

b) Definisi dan spesifikasi :
min_list : list → integer
{ min_list(L) menghasilkan nilai minimum dari sebuah list y
Realisasi :

```
def min_list(L):
    if is_one_elmt (L) :
        return L
    else :
        return min2(first_element (L),
                    min_list (tail (L)))
```

2) a) Definisi dan Spesifikasi

total_elemen_daun : Tree → integer

{total_elemen_daun (T) menghasilkan jumlah dari elemen daun dalam pohon T :

Basis 1 : is_one_element (T) : akar (T)

Rekurens : is_biner (T) :
total_elemen_daun (left(T)) +
total_elemen_daun (right (T))
is_uner_right (T) :
total_elemen_daun (right (T))
is_uner_left (T) :
total_elemen_daun (left (T))

Realisasi :

```
def total_elemen_daun (T) :
  if is_one_element (T) :
     return akar (T)
  else :
     if is_biner (T) :
        return total_elemen_daun (left (T)) +
               total_elemen_daun (right(T))
     elif is_uner_left (T) :
        return total_elemen_daun (left (T))
     else :
        return total_elemen_daun (right(T))
```

2)b) Definisi dan spesifikasi :

max_elemen_daun (P) : Pohon → integer

{max_elemen_daun(P) menghasilkan nilai daun terbesar dari sebuah Pohon}

Basis 1 : is_one_element (P) : akar (P)

Rekurens : is_biner (P) :

max2 (max_elemen_daun(left(P)),
max_elemen_daun (right (P)))

is_uner_right (P) :
max_elemen_daun (right (P))

is_uner_left (t) :
max_elemen_daun (left (P))

Realisasi :

```
def max_elemen_daun(P) :
  if is_one_element (P) :
    return akar (P)
  else :
    if is_biner (P) :
      return max2( max_elemen_daun (left (P)),
                   max_elemen_daun (right(P)))
    elif is_uner_left (P) :
      return max_elemen_daun (left (P))
    else :
      return max_elemen_daun (right(P))
```

2) c) Definisi dan spesifikasi

total_elemen_node : Pohon → integer

{total_elemen_node(P) menghasilkan jumlah dari semua elemen node pohon P }

Basis 1 : is_one_element (P) ; akar (P)

Rekurens : is_biner (P) :

akar(P) + total_elemen_node(left(P)) + total_elemen_node(right (P))

is_uner_right (P) :

akar(P) + total_elemen_node(right(P))

is_uner_left(P) :

akar(P) + total_elemen_node (left (P))

Realisasi :

```
def total_elemen_node(P):
    if is_one_element (P):
        return akar (P)
    else:
        if is_biner (P):
            return akar(P) + total_elemen_node(left(P)) +
                   total_elemen_node(right (P))
        elif is_uner_right (P):
            return akar(P) + total_elemen_node(right(P))
        else:
            return akar(P) + total_elemen_node(left(P))
```

2d) Fungsi bantuan

Definisi dan Spesifikasi

NbElmtPohon : Pohon → integer

{NbElmtPohon(P) menghasilkan banyaknya node dalam suatu Pohon P

Basis 1 : is_one_element(P) : 1

Rekurens : is_biner(P) :
1 + NbElmtPohon(left(P)) +
NbElmtPohon(right(P))

is_uner_right(P) :
1 + NbElmtPohon(right(P))

is_uner_left(P) :
1 + NbElmtPohon(left(P))

Realisasi :

```
def NbElmtPohon(P):
 if is_one_element(P):
     return 1
 else:
   if is_biner(P):
     return   1 + NbElmtPohon(left(P)) +
              NbElmtPohon(right(P))
   elif is_uner_right(P):
     return   1 + NbElmtPohon(right(P))
   else:
     return   1 + NbElmtPohon(left(P))
```

2 d) Definisi dan Spesifikasi

rata2_elemen_node : Pohon → real

{rata2_elemen_node(P) menghasilkan nilai rata-rata dari semua elemen node pohon P }

Realisasi :

```
def rata2_elemen_node(P):
    return total_elemen_node(P) /
           NbElmtPohon(P)
```

2)e) Definisi dan Spesifikasi

BST → elemen, Pohon → boolean

{BST(x,P) menghasilkan True jika elemen x ada di dalam pohon P}

Realisasi :

```
def BST(x,P):
    if x = akar(P):
        return True
    elif is_one_element(P):
        return False
    else:
        if x > akar(P):
            return BST(x, right(P))
        else:
            return BST(x, left(P))
```

→Langkah-langkah :

1) Cek apakah 19 = 50, ternyata tidak.
Kemudian cek apakah pohon hanya 1 elemen, ternyata tidak.
Kemudian Cek apakah 19>50, ternyata tidak. karena itu ambil pohon sebelah kiri.

2) Cek apakah 19=17, ternyata tidak.
Kemudian cek apakah pohon hanya 1 elemen, ternyata tidak.
Kemudian cek apakah 19>17 ternyata iya.
Karena itu ambil pohon sebelah kanan

3) Cek apakah 19=23, ternyata tidak.
Kemudian cek apakah pohon hanya 1 elemen, ternyata tidak.
Kemudian cek apakah 19>23, ternyata tidak. Karena itu ambil pohon sebelah kiri

4) Cek apakah 19 = 19, ternyata iya.
maka keluarkan hasil True

3) a)

Definisi dan spesifikasi :
kelipatan 5 : integer → boolean
{kelipatan 5(x) bernilai True bila bilangan merupakan kelipatan 5 }
Realisasi
```
def kelipatan5(x):
    return x%5==0
```

Definisi dan spesifikasi :
bukan_kelipatan5 : integer → boolean
{bukan_kelipatan5(x) bernilai True bila bilangan bukan merupakan kelipatan 5 }
Realisasi
```
def bukan_kelipatan5(x):
    return x%5!=0
```

Definisi dan spesifikasi
Filter_list : list → list
{Filter_list(L,f) memfilter list sesuai dengan fungsi f }
Realisasi :
```
def Filter_list(L,f):
    if empty_list(L):
        return []
    elif f(first_elmt(L)):
        return Konso(first_elmt(L),
                     Filter_List(tail(L),f))
    else:
        return Filter_list(tail(L),f)
```

3b
```
L1 = [60, 18, 7, 20, 19, 23, 50]
L2 = Filter_list(L1, lambda x: x % 5 == 0)
L3 = Filter_list(L1, lambda x: x % 5 != 0)
```

4 Definisi dan Spesifikasi

minus : 2 list → list

{minus (a, b) menghasilkan list dimana isi himpunan a tanpa himpunan list b}

Realisasi

```
def minus(a,b):
    if is_sub_set(a,b):
        return []
    elif is_member(first_element(a),b):
        return minus(tail(a),b)
    else:
        return Konso(first_element(a),
                     minus(tail(as,b))
```