Pohon Keluarga Pak Adam

Kelompok:

01 Abyan Ardiatama
24060120140161

02 Ilham Rismawan Faadhi
24060122140182

✓ Studi Kasus

Pohon Keluarga Pak Adam

Pak adam merupakan kakek tua berusia lanjut, di umurnya yang sudah lansia ia ingin menyusun sebuah pohon keluarganya sendiri dengan detail sebagai berikut.

- Hadi dan Rudi merupakan anak dari pak Adam
- Gani dan Ian merupakan anak dari pak Hadi
- Opick dan Sarah merupakan anak dari Rudi
- Dara dan Hilmi merupakan anak dari Gani
- Husna dan jamal merupakan anak dari Ian
- Nonik dan Gofur merupakan anak dari Opic
- Raden dan Tarno merupakan anak dari Sarah

✓ Permasalahan

Permasalahan 1

Buatlah sebuah fungsi yang mengembalikan anak dari pak Adam, cucu dari pak Adam, dan cicit dari pak Adam.

Permasalahan 2

Selain itu, karena ia sudah tua, Ia berencana untuk membagikan harta warisannya juga pada anak cucunya. Ia penasaran apakah ada dari anak cucunya yang memiliki nama samaran dengan anak cucunya yang lain, jika ada ia memberikan bonus warisan padanya.

Buatlah fungsi yang akan mengembalikan nama anak cucu pak adam yang akan mendapatkan bonus

✓ Gambaran solusi dari permasalahan

Solusi 1

Dengan berinput list of list, lalu diubah menjadi tree yang nantinya akan dijalankan menjadi BST dengan selektor lambda

Solusi 2

Dengan berinput list lalu program mengecek apakah ada nama yang sama dengan menggunakan set Intersect()

solusi realisasi fungsional

DEFINISI DAN SPESIFIKASI TYPE

DEFINISI DAN SPESIFIKASI KONSTRUKTOR

```
family_tree(t):
   if t == [] then []:
   else:
      akar = t[0]
      left = t[1]
      right = t[2]
      then [root, left, right]
```

{dengan family_tree(t) untuk mengubah list menjadi tree}

Konso: character,text → text

{konso(X,L)menghasilkan sebuah list dari X dan L dengan X sebagai elemen pertama}

DEFINISI DAN SPESIFIKASI SELEKTOR

```
# family tree
root = lambda t: t[0]
```

root(t) untuk mengidentifikasi akar pusatnya

```
left_child = lambda t: t[1]
```

left_child(t) untuk menavigasikan jalur bagian kiri

```
right_child = lambda t: t[2]
```

right_child(t) untuk menavigasikan jalur bagian kanan

Head(L): element, List (Tidak kosong) → list

{Head(L) menghasilkan list tanpa elemet terakhir

Tail(L): element, List (Tidak kosong) → list

{Tail(L) menghasilkan list tanpa elemet pertama

DEFINISI DAN SPESIFIKASI PREDIKAT

Is_empty(t) : element, List (Tidak kosong) → Boolean

{is_empty(t) benar jika kosong}

root_child(t): element, List (Tidak kosong) → list

{root_child(t) memberikan baris ke dua pada tree}

grand_child(t): element, List (Tidak kosong) →list

{grand_child(t) mengembalikan baris ke tiga pada tree}

Greatgrandchild(t): element, List (Tidak kosong) →list

{greatgrandchild(t) mengembalikan baris ke tiga pada tree}

Last(L): element, List (Tidak kosong) →list

{last(l) mengembalikan bilangan terakhir}

First(L): element, List (Tidak kosong) →list

{First(l) mengembalikan bilangan pertama}

Is_member(X,L): element, List (Tidak kosong) → Boolean

{ Is_member(X,L) benar jika x element pertama L}

REALISASI

child = root_child(t)

cucu= grandchild(t)

cicit = greatgrandchild(t)

make_intersect(H1,H2):

depend on H1,H2

is_empty(H1) and is_empty(H2)	:[]
not is_empty(H1) and is_empty(H2)	:[]
is_empty(H1) and not is_empty(H2)	:{rekurensi}

if is_member(First(H1), H2):

then konso(First(H1), make_intersect(Tail(H1),H2)

else make_intersect(tail(H1)H2)

APLIKASI

t = ["adam",["hadi", ["gani", ["dara", [],[]],["hilmi", [],[]],["ian",["husna", [],[]],["jamal", [],[]]], ["rudy", ["opick", ["husna",[],[]], ["gofur",[],[]], ["sarah", ["raden",[],[]], ["tarno",[],[]]]]

solusi realisasi dalam bahas pemrograman

```
###Binary tree Menggunakan implementasi list
def family_tree(t):
    if t == []:
        return []
    else:
        akar = t[0]
        left = t[1]
        right = t[2]
        return [root, left, right]
```

```
#selektor menggunakan fungsi lambda
root = lambda t: t[0]
left_child = lambda t: t[1]
right_child = lambda t: t[2]

def is_empty(t):
    return t == []
def root_child(t):
    child = [root(left_child(t))] + [root(right_child(t))]
    return child
def grandchild(t):
    grandchild = [root_child(left_child(t))] + [root_child(right_child(t))
]
    return grandchild
def greatgrandchild(t):
    greatgrandchild = (grandchild(left_child(t))) + (grandchild(right_chil
d(t)))
    return greatgrandchild

t =   ["adam",["hadi", ["gani",  ["dara",  [],[]],["hilmi",  [],[]
],["ian",["husna",  [],[]],["jamal",  [],[] ]], ["rudy", ["opick",
```

```
["husna",[],[]],  ["gofur",[],[]] ], ["sarah", ["raden",[],[]],
["tarno",[],[]] ]]]

child = root_child(t)
cucu = grandchild(t)
cicit = greatgrandchild(t)
print("Anak dari pak Adam   = ", child)
print("Cucu dari pak adam   = ", cucu)
print("Cicit dari pak Adam  = ", cicit)
```

Output:

Anak dari pak Adam = ['hadi', 'rudy']

Cucu dari pak adam = [['gani', 'ian'], ['opick', 'sarah']]

Cicit dari pak Adam = [['dara', 'hilmi'], ['husna', 'jamal'], ['husna', 'gofur'], ['raden', 'tarno']]

```
#Mencari pewaris yang akan mendapatkan bonus menggunakan make_intersect
def konso(x,L):
    return [x]+L
def Last(L):
    return L[-1]
def First(L):
    return L[0]
def Head(L):
    return L[:-1]
def Tail(L):
    return L[1:]
def is_empty(L):
    if L==[]:
        return True
def is_member(x,L):
    if is_empty(L):
        return False
    else:
        if Last(L)==x:
            return True
        elif Last(L)!=x:
            return is_member(x,Head(L))
def make_intersect(H1,H2):
    if is_empty(H1) and is_empty(H2):
        return []
    elif not is_empty(H1) and is_empty(H2):
```

```
        return []
    elif is_empty(H1) and not is_empty(H2):
        return []
    else:
        if is_member(First(H1),H2):
            return konso(First(H1),make_intersect(Tail(H1),H2))
        else:
            return make_intersect(Tail(H1),H2)
print("list cucu dari anak pak Adam")
hadi = cucu[0]
rudy = cucu[1]
print("hadi   = ",hadi)
print("rudy   = ",rudy)
print("\n")

print("list cicit dari cucu pak Adam")
gani  = cicit[0]
ian   = cicit[1]
opick = cicit[2]
sarah = cicit[3]
print("gani   = ",gani)
print("ian    = ",ian)
print("opick  = ",opick)
print("sarah  = ", sarah)


print("pewaris yang mendapat bonus =", make_intersect(ian,opick))
```

Output:

list cucu dari anak pak Adam

hadi = ['gani', 'ian']

rudy = ['opick', 'sarah']

list cicit dari cucu pak Adam

gani = ['dara', 'hilmi']

ian = ['husna', 'jamal']

opick = ['husna', 'gofur']

sarah = ['raden', 'tarno']

pewaris yang mendapat bonus = ['husna']

solusi realisasi fungsional

DEFINISI DAN SPESIFIKASI TYPE

DEFINISI DAN SPESIFIKASI KONSTRUKTOR

```
family_tree(t):
   if t == [] then []:
   else:
      akar = t[0]
      left = t[1]
      right = t[2]
      then [root, left, right]
```

{dengan family_tree(t) untuk mengubah list menjadi tree}

Konso: character,text → text

{konso(X,L)menghasilkan sebuah list dari X dan L dengan X sebagai elemen pertama}

DEFINISI DAN SPESIFIKASI SELEKTOR

```
# family tree
root = lambda t: t[0]
```

root(t) untuk mengidentifikasi akar pusatnya

```
left_child = lambda t: t[1]
```

left_child(t) untuk menavigasikan jalur bagian kiri

```
right_child = lambda t: t[2]
```

right_child(t) untuk menavigasikan jalur bagian kanan

Head(L): element, List (Tidak kosong) → list

{Head(L) menghasilkan list tanpa elemet terakhir

Tail(L): element, List (Tidak kosong) → list

{Tail(L) menghasilkan list tanpa elemet pertama

DEFINISI DAN SPESIFIKASI PREDIKAT

Is_empty(t) : element, List (Tidak kosong) → Boolean

{is_empty(t) benar jika kosong}

root_child(t): element, List (Tidak kosong) → list

{root_child(t) memberikan baris ke dua pada tree}

grand_child(t): element, List (Tidak kosong) →list

{grand_child(t) mengembalikan baris ke tiga pada tree}

Greatgrandchild(t): element, List (Tidak kosong) →list

{greatgrandchild(t) mengembalikan baris ke tiga pada tree}

Last(L): element, List (Tidak kosong) →list

{last(l) mengembalikan bilangan terakhir}

First(L): element, List (Tidak kosong) →list

{First(l) mengembalikan bilangan pertama}

Is_member(X,L): element, List (Tidak kosong) → Boolean

{ Is_member(X,L) benar jika x element pertama L}

REALISASI

child = root_child(t)

cucu= grandchild(t)

cicit = greatgrandchild(t)

make_intersect(H1,H2):

depend on H1,H2

is_empty(H1) and is_empty(H2)	:[]
not is_empty(H1) and is_empty(H2)	:[]
is_empty(H1) and not is_empty(H2)	:{rekurensi}

if is_member(First(H1), H2):

then konso(First(H1), make_intersect(Tail(H1),H2)

else make_intersect(tail(H1)H2)

APLIKASI

t = ["adam",["hadi", ["gani", ["dara", [],[]],["hilmi", [],[]],["ian",["husna", [],[]],["jamal", [],[]]], ["rudy", ["opick", ["husna",[],[]], ["gofur",[],[]], ["sarah", ["raden",[],[]], ["tarno",[],[]]]]

solusi realisasi dalam bahas pemrograman

```
###Binary tree Menggunakan implementasi list
def family_tree(t):
    if t == []:
        return []
    else:
        akar = t[0]
        left = t[1]
        right = t[2]
        return [root, left, right]
```

```
#selektor menggunakan fungsi lambda
root = lambda t: t[0]
left_child = lambda t: t[1]
right_child = lambda t: t[2]

def is_empty(t):
    return t == []
def root_child(t):
    child = [root(left_child(t))] + [root(right_child(t))]
    return child
def grandchild(t):
    grandchild = [root_child(left_child(t))] + [root_child(right_child(t))
]
    return grandchild
def greatgrandchild(t):
    greatgrandchild = (grandchild(left_child(t))) + (grandchild(right_chil
d(t)))
    return greatgrandchild

t =   ["adam",["hadi", ["gani",  ["dara",  [],[]],["hilmi",  [],[]
],["ian",["husna",  [],[]],["jamal",  [],[] ]], ["rudy", ["opick",
```

```
["husna",[],[]],  ["gofur",[],[]] ], ["sarah", ["raden",[],[]],
["tarno",[],[]] ]]]

child = root_child(t)
cucu = grandchild(t)
cicit = greatgrandchild(t)
print("Anak dari pak Adam   = ", child)
print("Cucu dari pak adam   = ", cucu)
print("Cicit dari pak Adam  = ", cicit)
```

Output:

Anak dari pak Adam = ['hadi', 'rudy']

Cucu dari pak adam = [['gani', 'ian'], ['opick', 'sarah']]

Cicit dari pak Adam = [['dara', 'hilmi'], ['husna', 'jamal'], ['husna', 'gofur'], ['raden', 'tarno']]

```
#Mencari pewaris yang akan mendapatkan bonus menggunakan make_intersect
def konso(x,L):
    return [x]+L
def Last(L):
    return L[-1]
def First(L):
    return L[0]
def Head(L):
    return L[:-1]
def Tail(L):
    return L[1:]
def is_empty(L):
    if L==[]:
        return True
def is_member(x,L):
    if is_empty(L):
        return False
    else:
        if Last(L)==x:
            return True
        elif Last(L)!=x:
            return is_member(x,Head(L))
def make_intersect(H1,H2):
    if is_empty(H1) and is_empty(H2):
        return []
    elif not is_empty(H1) and is_empty(H2):
```

```
        return []
    elif is_empty(H1) and not is_empty(H2):
        return []
    else:
        if is_member(First(H1),H2):
            return konso(First(H1),make_intersect(Tail(H1),H2))
        else:
            return make_intersect(Tail(H1),H2)
print("list cucu dari anak pak Adam")
hadi = cucu[0]
rudy = cucu[1]
print("hadi   = ",hadi)
print("rudy   = ",rudy)
print("\n")

print("list cicit dari cucu pak Adam")
gani  = cicit[0]
ian   = cicit[1]
opick = cicit[2]
sarah = cicit[3]
print("gani   = ",gani)
print("ian    = ",ian)
print("opick  = ",opick)
print("sarah  = ", sarah)


print("pewaris yang mendapat bonus =", make_intersect(ian,opick))
```

Output:

list cucu dari anak pak Adam

hadi = ['gani', 'ian']

rudy = ['opick', 'sarah']

list cicit dari cucu pak Adam

gani = ['dara', 'hilmi']

ian = ['husna', 'jamal']

opick = ['husna', 'gofur']

sarah = ['raden', 'tarno']

pewaris yang mendapat bonus = ['husna']