

# ADT TITK

1] Procedure makeTitik (output T: Titik)

{ I.S. : - }

{ F.S. : Terdefinisi }

{ proses : mengisi nilai komponen absis dan ordinat T dengan 0 }  
kamus lokal

Algoritma

T.absis  $\leftarrow$  0

T.ordinat  $\leftarrow$  0

2] function getAbsis (T: Titik)  $\Rightarrow$  integer

{ mengembalikan nilai komponen absis T }

kamus lokal

Algoritma

$\rightarrow$  T.absis

3] function getOrdinat (T: Titik)  $\rightarrow$  integer

kamus lokal

Algoritma

$\rightarrow$  T.ordinat

4] Procedure setAbsis (input/output T, input xx)

{ I.S : T terdefinisi }

{ F.S : T.absis = xx }

{ proses : mengisi nilai komponen absis T }

kamus lokal

Algoritma

T.absis  $\leftarrow$  xx

5] Procedure setOrdinat (input/output T, input yy)

{ I.S : T terdefinisi }

{ F.S : T.ordinat = yy }

kamus lokal

Algoritma

T.ordinat  $\leftarrow$  yy

2) function isOrigin (T:titik)  $\rightarrow$  boolean  
{ Mengembalikan true jika titik T ada di perilengsan sumbu x dan y }  
kamus lokal

Algoritma

```
if (T.absis == 0) AND (T.ordinat == 0) then
    → true
else
    → false
```

3) Procedure geserXY (input/output T:titik, input dx; input dy)  
{ I.S : terdefinisi }  
{ F.S : bergeser sejauh (dx,dy) }  
{ proses : mengubah nilai komponen absis dan ordinat T }  
kamus lokal

algoritma

```
T.absis  $\leftarrow$  T.absis + dx
T.ordinat  $\leftarrow$  T.ordinat + dy
```

4) Procedure refleksiSumbuX (input/output T:titik)  
{ I.S : T terdefinisi }  
{ F.S : T dicerminkan ke sumbu X }  
{ proses : mengubah nilai komponen ordinat T }  
kamus lokal

algoritma

```
T.absis  $\leftarrow$  - T.ordinat
```

5) Procedure refleksiSumbuY (input/output T:titik)  
{ I.S : T terdefinisi }  
{ F.S : dicerminkan ke sumbu Y }  
{ proses : mengubah nilai komponen absis T }  
kamus lokal

algoritma

```
T.ordinat  $\leftarrow$  - T.absis
```

|10| Function kuadran ( $T$ : titik)  $\rightarrow$  integer

{ mengembalikan posisi titik  $T$  pada kuadran 1,2,3 atau 4 }

{ asumsi : bila titik memotong sumbu, maka nilai 0 dianggap positif }

kamus lokal

Algoritma

If ( $T.\text{absis} > 0$  AND  $T.\text{ordinat} > 0$ ) then

$\rightarrow 1$

else If ( $T.\text{absis} < 0$  AND  $T.\text{ordinat} > 0$ ) then

$\rightarrow 2$

else If ( $T.\text{absis} < 0$  AND  $T.\text{ordinat} < 0$ ) then

$\rightarrow 3$

else If ( $T.\text{absis} > 0$  AND  $T.\text{ordinat} < 0$ ) then

$\rightarrow 4$

|11| function Jarak ( $T1$ : titik,  $T2$ : titik)  $\rightarrow$  real

{ mengembalikan jarak euclidian  $T1$  dan  $T2$  }

{ asumsi : aplikasikan rumus pitagoras dan fungsi akar sqrt }

kamus lokal

float hasil

Int  $X2$

Int  $Y2$

Int total

Algoritma

$X2 \leftarrow \text{pow}(T.\text{absis}, 2)$

$Y2 \leftarrow \text{pow}(T.\text{ordinat}, 2)$

$\text{total} \leftarrow X2 + Y2$

$\text{hasil} \leftarrow \text{sqrt}(\text{total})$

$\rightarrow \text{hasil}$

# ADT TABEL

function getSize (T: Tabel) → integer

{ mengembalikan banyak elemen pengir T }

kamus lokal

algoritma

→ T.size

Procedure createTable (output T: Tabel)

{ I.s : - }

{ F.s : size = 0, setiap elemen wadah = -999 }

kamus lokal

int i

algoritma

T.size ← 0

i traversal [i... T.size]

T.wadah [i] ← -999

Procedure printTable (Input T: Tabel)

{ I.s : T terdefinisi }

{ F.s : - }

{ Proses : menampilkan semua elemen T ke layar }

kamus lokal

int i

algoritma

i traversal [1... T.size]

T.wadah [i]

Procedure viewTable (Input T: Tabel)

{ I.s : T terdefinisi }

{ F.s : - }

{ Proses : menampilkan elemen T yang benar ke layar }

kamus lokal

int i

algoritma

if (T.size > 0) then

i traversal [1... T.size]

T.wadah [i]

else

"tabel kosong"

1.5 Procedure populate1 (input /output T:Tabel, input N : integer)  
{I.S : T terdefinisi, N terdefinisi dalam rentang 1... 10}  
{F.S : T.wadah berisi sebanyak N elemen}

{Proses : Mengisi elemen T.wadah sebanyak N kali dari keyboard}  
{Kondisi : angka - angka merupakan keyboard > 0}

Kamus lokal

int i

int x

algoritma

i traversal [1... N]

If ( $x > 0$ ) then

T.wadah[i]  $\leftarrow x$

else

i --

6) Procedure searchx1 (input T:Tabel, input x : Integer, output pos : integer)

{I.S : T terdefinisi, x terdefinisi}

{F.S : Pos berisi posisi ketemu di T.wadah, atau -999 jika tidak ketemu}

{Proses : mencari elemen bernilai x dalam T.wadah}

Kamus lokal

int i

algoritma

Pos  $\leftarrow -999$

i traversal [T.size... 1]

If ( $x = T.wadah[i]$ ) then

Pos  $\leftarrow i$

7) Procedure countx (input T:Tabel, input x : Integer, output Byk : integer)

{I.S. : T terdefinisi, x terdefinisi}

{F.S. : Byk berisi banyaknya x di T.wadah, atau 0 jika tidak ketemu}

{Proses : menghitung elemen bernilai x dalam T.wadah}

Kamus lokal

int i

algoritma

Byk  $\leftarrow 0$

i traversal [1... T.size]

If ( $x = T.wadah[i]$ ) then

Byk  $\leftarrow$  Byk + 1



[8] function SumEl (T:Tabel)  $\rightarrow$  integer  
{ mengembalikan jumlah semua elemen pengisi T }  
kamus lokal  
 $i$ : integer { iterasi }  
sum: integer {  
algoritma  
 $sum \leftarrow 0$   
 $i$  traversat [1...T.size]  
    sum = sum + T.wadah[i]  
     $\rightarrow$  sum

[9] function AverageEl (T:Tabel)  $\rightarrow$  integer  
{ mengembalikan nilai rata-rata elemen pengisi T }  
kamus lokal

algoritma  
 $\rightarrow$  sumEl(T) / T.size

[10] function getMaxEl (T:Tabel)  $\rightarrow$  integer  
{ mengembalikan nilai elemen terbesar }

kamus lokal

$i$ : integer { iterasi }

max: integer = -999 {

algoritma

$max \leftarrow T.wadah[1]$

$i$  traversal [1..T.size]

    if ( $T.wadah[i] > max$ ) then

$max \leftarrow T.wadah[i]$

$\rightarrow max$

[11] function getMinEl (T:Tabel)  $\rightarrow$  integer  
{ mengembalikan nilai elemen terkecil }

kamus lokal

$i$ : integer { iterasi }

min: integer

algoritma

input J

$\min \leftarrow T.wadah[1]$

$i \text{ traversal } [1..T.size]$

if (( $T.wadah[i] < \min$ ) AND ( $T.wadah[i] \neq \min$ )) then

$\min \leftarrow T.wadah[i]$

$\rightarrow \min$

[12] Procedure populate2 (input/output T: Tabel)

{ I.S. : T terdefinisi }

{ F.S. : T.wadah tersi beberapa elemen positif }

{ Proses : mengisi elemen T.wadah berulang, bila angka dari keyboard  $\leq 0$  maka berhenti, tidak diproses }

{ Syarat : angka-angka masukan keyboard  $> 0$  }

kamus lokal

i : integer { iterator }

j : integer { iterator }

algoritma

$i \leftarrow T.size + 1$

while ( $i \leq 10$  AND  $j > 0$ ) do

$T.wadah[i] \leftarrow j$

input j

$i \leftarrow i + 1$

$T.size \leftarrow i - 1$

[13] procedure addXTable (input/output T: Tabel, input x : integer)

{ I.S. : T terdefinisi, x terdefinisi }

{ F.S. : isi T.wadah bertambah 1 elemen jika belum penuh }

{ Proses : mengisi elemen T.wadah dengan nilai x }

kamus lokal

i : integer { iterator }

algoritma

if ( $T.size = 10$ ) then

output "Tabel Penuh"

else

$T.size \leftarrow T.size + 1$

$T.wadah[T.size] \leftarrow x$

[14] procedure delXTable (input/output T: Tabel, input X: integer)  
{I.S.: T terdefinisi}  
{F.S.: isi T.wadah berkurang 1 elemen jika belum kosong}  
{Proses: menghapus 1 elemen bernilai X, geser semua elemen sisanya}  
Kamus lokal

i : integer {iterator}

position : integer { }  
temp : integer { }

algoritma

searchX1 (T, X, position)

If (position != -999) then

T.wadah [position] = -999

T.size < T.size - 1

i traversal [position ... getSize (T)]

temp <- T.wadah [i]

T.wadah [i] <- T.wadah [i+1] <- T.wadah [i+2]

T.wadah [i+1] <- temp

[15] procedure delAllXTable (input/output T: Tabel, input X: integer)

{I.S.: T terdefinisi}

{F.S.: isi T.wadah berkurang semua elemen bernilai X jika belum kosong}

{Proses: menghapus semua elemen bernilai X, geser elemen sisanya}

Kamus lokal

position : integer { }

algoritma

do

delXTable (T, X)

searchX1 (T, X, Position)

while (position != -999)

1b)

function Modus ( $T$ : Tabel)  $\rightarrow$  Integer  
{ mengembalikan elemen pengisi  $T$  yang paling banyak muncul }  
{ asumsi: bila terdapat banyak yang sama maka yang diambil yang pertama }  
| kamus lokal |

$i$  : Integer { iterator }  
 $j$  : Integer { iterator }  
modus : Integer { }  
jum : Integer { }  
jumax : Integer { }

algoritma

modus  $\leftarrow 0$

jumax  $\leftarrow 0$

$i$  traversal [1..T.size]

jum  $\leftarrow 0$

$j$  traversal [1... T.size]

If ( $T.wadah[i] = T.wadah[j]$ ) then

jum  $\leftarrow$  jum + 1

If (jum > jumax) then

jumax  $\leftarrow$  jum

modus  $\leftarrow i$

$\rightarrow T.wadah[modus]$

# ADT STACK

1) procedure create stack (output T : Tstack)  
{ I.S. : - }

{ F.S. : T terdefinisi, semua nilai elemen T.wadah = '#' }  
{ proses : menginisialisasi T }  
kamus lokal

i : Integer { iterator }

algoritma

T.top ← 0

i traversal [1...10]

T.wadah [i] ← '#'

2) function top (T: Tstack) → Integer

{ mengembalikan posisi puncak stack}

kamus lokal

algoritma

→ T.top

3) function infotop (T: Tstack) → character

{ mengembalikan nilai elemen top stack}

kamus lokal

algoritma

If (T.top != 0) then  
    → T.wadah [T.top]

else

    → '-'

4) function isEmptyStack (T: Tstack) → boolean

{ mengembalikan True jika T kosong }

kamus lokal

algoritma

If (T.top = 0) then  
    → true

else

    → false

[5] function isfullStack ( $T$ : Tstack)  $\rightarrow$  boolean  
{ mengembalikan True jika  $T$  penuh }  
kamus lokal

algoritma

If ( $T.top = 10$ ) then  
    → true  
else  
    → false

[6] procedure push (input/output  $T$ : Tstack, input  $E$  : character)  
{ I.S. :  $T, E$  terdefinisi }  
{ F.S. : info top tetap, atau berisi nilai  $E$  }  
{ Proses : mengisi elemen top baru, bila belum penuh }  
kamus lokal

algoritma

If ( $!isfullStack(T)$ ) then  
     $T.top \leftarrow T.top + 1$   
     $T.wadah[T.top] \leftarrow E$

[7] procedure pop (input/output  $T$ : Tstack, output  $x$  : character)  
{ I.S. :  $T$  terdefinisi }  
{ F.S. :  $x$  = info top stack lama, atau '#' }  
{ Proses : mengambil elemen top, bila belum kosong }  
kamus lokal

algoritma

If ( $T.top \neq 0$ ) then  
     $x \leftarrow T.wadah[T.top]$   
     $T.wadah[T.top] \leftarrow '#'$   
     $T.top \leftarrow T.top - 1$   
else  
     $x \leftarrow '#'$

[8] procedure printStack (input T: Tstack)  
{I.S.: T terdefinisi}  
{F.S.: - }

{Proses: menampilkan kondisi wadah T ke layar}  
{setiap elemen dipisah tanda titik koma}  
kamus lokal  
 $i$  : Integer {Iterator}

algoritma

T traversal [1...10]

output T.wadah [ $i$ ]

[9] procedure viewStack (input T: Tstack)

{I.S.: T terdefinisi}

{F.S.: - }

{Proses: menampilkan elemen tak kosong T ke layar}

{setiap elemen dipisah tanda titik koma}

kamus lokal

$i$  : Integer {Iterator}

T traversal [1...10]

If (T.wadah [ $i$ ] = '#') then

output T.wadah [ $i$ ]

[10] procedure pushN (input/output T: Tstack, input N: Integer)

{I.S.: T, N terdefinisi}

{F.S.: infotop tetap, atau top = N}

{Proses: mengisi elemen top baru N kali, bila belum penuh}

kamus lokal

$i$  : Integer {Iterator}

algoritma

If (T.top < 10) then

T.top  $\leftarrow$  T.top + 1

T.wadah [T.top]  $\leftarrow$  N

[P1] pushBabel1 (input/output T: Tstack, input E: character)

{ I.S. : T terdefinisi }

{ F.S. : T bertambah 1 elemen (E) atau menjadi kosong bila penuh }

{ Proses: menumpuk top atau menghapus semua elemen }

Kamus lokal

E : character { }

i : integer { iterator }

algoritma

if (T.top >= 9) then

    createStack (T)

else

    T.top ← T.top + 1

    T.wadah [i.top] ← E

versi resmi

[P2] procedure pushN (input/output T: Tstack, input N: integer)

{ I.S.: T, N terdefinisi }

{ F.S.: intotop tetap, atau top = N }

{ Proses: mengisi elemen top baru N kali, bila belum penuh }

Kamus lokal

i : integer { iterator }

algoritma

i ← 1

if (!isFullStack (T)) then

    i traversal [1... N]

        input T.wadah [0]

        push (T, T.wadah [0])

do

    input T.wadah [0]

    push (T, T.wadah [0])

    i ← i + 1

while (i ≤ N)

function isEmpty (s: Stackhur)  $\rightarrow$  boolean  
{mengembalikan true jika wadah kosong}

function isFull (s: stackhur)  $\rightarrow$  boolean  
{mengembalikan true jika top mencapai kapasitas}

procedure push (input/output T: Tstack, input E : character)  
{I.S. : T, E terdefinisi}.

{F.S : Infotop tetap, atau berisi nilai E}

{Proses : mengisi elemen top baru, bila belum penuh}

procedure pop (input/output T: Tstack, output X : character)  
{ I.S. : T terdefinisi }

{F.S. : X = infotop stack lama, atau '#'}

{Proses : mengambil elemen top, bila belum kosong}

procedure PushBall (input/output S: stackhur, input H : character)

{I.S. : S terdefinisi mungkin kosong, H terdefinisi}

{F.S. : bertambah atau berkurang 1 elemen}

{Proses : menambah elemen stack atau menghapus elemen puncak}

Isamus lokal

algoritma

if (S.top = E) then

if (infoTop(S) = E) then

InfoTop(S)  $\leftarrow$  '-'

top  $\leftarrow$  top - 1

else

push (S, E)

else

if (infoTop(S) then

InfoTop(S)  $\leftarrow$  '-'

top  $\leftarrow$  top - 1

else

output ("Stack Penuh")

# ADT STACK

[1] procedure createStack (output T : Tstack)

{ I.S. : - }

{ F.S. : T terdefinisi, semua nilai elemen T.wadah = '#' }

{ Proses : menginisialisasi T }

kamus lokal

i : integer { iterator }

algoritma

T.top ← 0

i traversal [1... 10]

T.wadah[i] ← '#'

[2] function top (T: Tstack) → integer

{ mengembalikan posisi puncak stack }

kamus lokal

algoritma

→ T.top

atau #define top(T) (T).top

[3] function infotop (T: Tstack) → character

{ mengembalikan nilai elemen top stack }

kamus lokal

algoritma

if (T.top != 0) {

    → T.wadah[T.top]

atau #define infotop(T) (T).wadah[(T).top]

else

    → '-'

[4] function isEmptyStack (T: Tstack) → boolean

{ mengembalikan True jika T Kosong }

kamus lokal

algoritma

if (T.top = 0)

    → true

else

    → false

[5] function isfullstack (T: Tstack) → boolean  
{ mengembalikan True jika T penuh }  
kamus lokal

algoritma

if ( $T.\text{top} = 10$ )  
→ true

else

→ false

[6] procedure push (Input/output T: Tstack, input E: character)  
{I.S.: T, E terdefinisi}  
{F.S.: infotop tetap, atau bensí nilai E}  
kamus lokal

algoritma

if ( $! \text{isfullstack}(T)$  then  
 $T.\text{top} \leftarrow T.\text{top} + 1$

# QUEUE1

1) Procedure createQueue (output  $Q : t\text{Queue}$ )  
{I.S. : - }  
{F.S. : terdefinisi, kosong }  
{Proses : mensin komponen dan elemen dengan '#' }  
Kamus lokal  
 $i : \text{integer} \quad \{ \text{indexator} \}$

algoritma  
 $Q.\text{head} \leftarrow 0$   
 $Q.\text{tail} \leftarrow 0$   
 $\underset{i \text{ traversal } [0 \dots 5]}{\cdots}$   
 $Q.\text{wadah}[i] \leftarrow \#$

2) function Head ( $Q : t\text{Queue}$ )  $\rightarrow \text{integer}$   
{mengembalikan elemen terdepan antian  $Q$ }

3) function Tail ( $Q : t\text{Queue}$ )  $\rightarrow \text{integer}$   
{mengembalikan elemen terakhir antian  $Q$ }

4) function infoHead ( $Q : t\text{Queue}$ )  $\rightarrow \text{character}$   
{mengembalikan nilai elemen terdepan antian  $Q$ }  
Kamus lokal

algoritma  
if (NOT isEmpty ( $Q$ )) then  
     $Q.\text{wadah}[Q.\text{head}]$   
else  
     $\rightarrow \#$

5) function infoTail ( $Q : t\text{queue}$ )  $\rightarrow \text{character}$   
{mengembalikan nilai elemen terakhir antian  $Q$ }  
Kamus lokal

algoritma  
if (NOT isEmpty ( $Q$ )) then  
     $Q.\text{wadah}[Q.\text{tail}]$   
else  
     $\rightarrow \#$

19) function BEmptyQueue ( $Q: tqueue$ )  $\rightarrow$  boolean  
{ mengembalikan true jika  $Q$  kosong }  
|kamus lokal|

algoritma

If  $((head(Q) = 0) \text{ AND } (tail(Q) = 0))$  then  
 $\rightarrow$  true  
else  
 $\rightarrow$  false

10) function IsFullQueue ( $Q: tqueue$ )  $\rightarrow$  boolean  
{ mengembalikan nilai true jika  $Q$  penuh }  
|kamus lokal|

algoritma

If  $((head(Q) = 1) \text{ AND } (tail(Q) = 5))$  then  
 $\rightarrow$  true  
else  
 $\rightarrow$  false

11) function isOneElement ( $Q: tqueue$ )  $\rightarrow$  boolean  
{ mengembalikan true jika hanya ada 1 elemen }  
|kamus lokal|

algoritma

$\rightarrow (Q.\text{head} = Q.\text{tail}) \text{ AND } (\text{NOT } \text{IsEmptyQueue})$

12) procedure enqueue (input/output  $Q: tqueue$ , input  $E: \text{charakter}$ )  
{ I.S. : terdefinisi }  
{ F.S. : elemen wadah  $Q$  bertambah 1, bila belum penuh }  
{ proses : menambah elemen wadah  $Q$  }  
|kamus lokal|

algoritma

If  $(\text{NOT } \text{IsFullQueue})$  then  
If  $(\text{IsEmptyQueue}(Q))$  then  
 $tail(Q) \leftarrow tail(Q) + 1$   
 $Q.\text{wadah}[tail(Q)] \leftarrow E$

13) procedure dequeue (input/output Q: tqueue, output t: character)

{ I.S.: Q terdefinisi }

{ F.S.: E = infohead(Q) atau E = '#' bila Q kosong, elemen wadah Q berkurang 1 }

{ proses : mengurangi elemen wadah Q, semua elemen di belakang head dgeser maju }

{ bila awalnya 1 elemen, maka Head dan Tail menjadi 0 }

kamus lokal

i : integer (iteratur)

algoritma

if isEmptyQueue (Q) then

E ← '#'

else

E ← infohead (Q)

i traversal [1... (Tail(Q) - 1)]

Q.wadah[i] ← Q.wadah[i+1]

Q.wadah[Tail(Q)] ← '#'

Tail(Q) ← Tail(Q) - 1

if Tail(Q) = 0 then

Head(Q) ← 0

14) function longer(Q<sub>1</sub>: tqueue, Q<sub>2</sub>: tqueue) → integer

{ mengembalikan ukuran yang terpanjang dari 2 antrean }

kamus lokal

algoritma

if (tail(Q<sub>1</sub>) > tail(Q<sub>2</sub>)) then

→ tail(Q<sub>1</sub>)

else if (tail(Q) < tail(Q<sub>2</sub>)) then

→ tail(Q<sub>2</sub>)

else

→ 0

15) procedure enqueueShort (Input/Output  $Q_1$ : tqueue, Input/Output  $Q_2$  : +Queue, Input  $E$ : character)  
{I.S. = terdefinisi}

{F.S. = elemen wadah  $Q_1$  atau  $Q_2$  bertambah 1, bila belum penuh}  
{proses : menambah elemen wadah pada antrian terpendek (salah satu,  $Q_1$  atau  $Q_2$ )  
kamus lokal}

algoritma

If ( $\text{sizeQueue}(Q_1) < \text{sizeQueue}(Q_2)$ ) then  
enqueue ( $Q_1, E$ )  
else if ( $\text{sizeQueue}(Q_1) > \text{sizeQueue}(Q_2)$ ) then  
enqueue ( $Q_2, E$ )  
else

('Jumlah kedua elemen sama atau penuh')

16) procedure dequeueLong (Input/Output  $Q_1$ : tqueue, Input/Output  $Q_2$  : tqueue, Input  $E$ : character)

{I.S. :-}

{F.S. :  $E = \text{InfoHead}$   $Q_1$  atau  $Q_2$ , atau  $E = \#$  bila  $Q_1$  dan  $Q_2$  kosong, elemen wadah  $Q_1$  atau  $Q_2$  berkurang 1}  
{proses : mengurangi elemen wadah antrian terpanjang  $Q_1$  atau  $Q_2$ , semua elemen di belakang head digeser maju}

{ bila awalnya 1 elemen, maka head and tail antrian menjadi 0 }

kamus lokal

algoritma

If ( $\text{sizeQueue}(Q_1) > \text{sizeQueue}(Q_2)$ ) then  
dequeue ( $Q_1, E$ )  
else if ( $\text{sizeQueue}(Q_1) < \text{sizeQueue}(Q_2)$ ) then  
dequeue ( $Q_2, E$ )  
else  
( ' Jumlah elemen kedua Queue sama' )

16) function  $\text{sizeQueue}(Q$ : tqueue)  $\rightarrow$  integer

{mengembalikan panjang antrian Q}

# QUEUE2

1) Procedure create Queue2 (output Q: tQueue2)  
{I.S.: - }  
{F.S.: Q terdefinisi, kosong }

{proses: mengisi komponen dengan 0, elemen kosong = '@'}

2) function isEmptyQueue2 (Q: tQueue2)  $\rightarrow$  boolean  
{mengembalikan true jika Q kosong}  
Kamus lokal

Algoritma

IF (Q.head = 0 AND Q.tail = 0) then  
 $\rightarrow$  true

else

$\rightarrow$  false

3) function isFullQueue2 (Q: tQueue2)  $\rightarrow$  boolean

{mengembalikan true jika Q penuh}

Kamus lokal

Algoritma

IF (Q.head = 1 AND Q.tail = 5)  
 $\rightarrow$  true

else

$\rightarrow$  false

4) function isOneElement2 (Q: tQueue2)  $\rightarrow$  boolean

{mengembalikan true jika Q berisi 1 elemen}

Kamus lokal

Algoritma

IF (Not isEmptyQueue2(Q)) then  
 $\rightarrow$  (Q.tail - Q.head + 1) = 1

else

$\rightarrow$  false

[7] Function InfotHead2 (Q : Tqueue2) → character  
{mengembalikan nilai elemen berdepan}  
kamus lokal

Algoritma

→ Q.wadah [Q.head]

[8] Function InfotTail2 (Q : Tqueue2) → character  
{mengembalikan nilai elemen terakhir}  
kamus lokal

Algoritma

→ Q.wadah [Q.tail]

[9] function sizeQueue2 ( Q : tQueue2 ) → integer  
{mengembalikan panjang antrean Q}  
kamus lokal

Algoritma

If (IsEmptyQueue2(Q)) then  
→ 0

else

→ Q.tail - Q.head + 1

[10] function isTailStop2 (Q : TQueue2) → boolean  
{mengembalikan true jika Tail tidak dapat lagi geser}  
{karena sudah di posisi kapasitas}  
kamus lokal

Algoritma

If ( Q.tail = 5 ) then  
→ true

else

→ false

$\boxed{13}$  function resetthead (input/output Q : Tqueue2)  
 { I.S. : Tail = kapasitas, head > 1, F.S : Head = 1 }  
 { Proses : mengembalikan head ke indeks 1 }  
 { Elemen selain head ikut bergeser menyesuaikan }  
 kamus lokal  
 i : integer { iterator }  
 j : integer { iterator }

Algoritma

```

if (NOT isEmpty Queue2(Q)) then
  j ← 0
  i traversal [Q.head ... Q.tail]
  j = j + 1
  Q.wadah [j] ← Q.wadah [i]
  Q.wadah [i] ← '@'
  Q.head ← 1
  Q.tail ← j
  
```

$\boxed{14}$  Procedure enqueue2 (input/output Q : Tqueue2, Input E : character)

{ I.S. : tail definisi }  
 { F.S : elemen wadah Q bertambah 1 bila belum penuh }  
 { Proses : menambah elemen wadah Q, jika tail(Q) = kapasitas, maka semua  
 elemen digeser lebih dulu sehingga head(Q) = 1 }  
 kamus lokal

Algoritma

```

if (NOT isfull Queue2(Q)) then
  if (isTaistop2(Q)) then
    resetthead(Q)
  if (isEmpty Queue2(Q)) then
    Q.head = Q.head + 1
    Q.wadah [Q.tail] = E
  
```

15) procedure dequeue2 ( input/output Q : tqueue2, output E : character )

{ I.S. : }

{ F.S. : elemen wadah Q berkurang 1 (Head), E = infohead(Q) lama, bila kosong,  
E = '@' }

{ proses : mengurangi elemen wadah Q, bila 1 elemen, maka Head dan Tail mengacai  
ke 0 }

kamus lokal

algoritma

If ( NOT isEmpty ( Queue2 ( Q ) ) then

E = Q.wadah [Q.head]

Q.wadah [Q.head] <- @

If ( isOnelement ( Q ) ) then

Q.head <- 0

Q.tail <- 0

else

Q.head = Q.head + 1

else

E <- 0