1 Konsep

Tulis ringkasan atau penjelasan hal-hal berikut, dengan kata-kata kalian sendiri mengenai, Konsep struktur data Queue dan Deque, operasi-operasi yang terdapat pada Queue dan Deque, dan implementasi operasioperasi Queue dan Deque tersebut dengan bahasa Pemrograman Python.

Jawaban

Queue

1. Konsep Struktur Data Queue

Queue merupakan sebuah stuktur data yang penambahan dan penghapusan datanya berada pada ujung yang berbeda atau bisa disebut dengan antrian pada queue penambahan data hanya dapat dilakukan pada ujung Rear/belakang dan penghapusan data hanya dapat dilakukan di Front/Depan atau kebalikannya jadi data yang pertama kali di tambahkan akan menjadi pertama kali yang keluar (First In First Out)

2. Operasi operasi yang terdapat pada Queue

```
- queue,     Digunakan untuk inisialisasi awal untuk tempat menampung data queue
- enqueue,   Digunakan untuk penambahan data baru pada queue
- dequeue,   Digunakan untuk penghapusan data pada queue
- isEmpty,   Digunakan untuk pengecekan apakah queue dalam keadaan kosong atau tidak
- size,      Digunakan untuk mengetahui jumlah data yang terdapat pada queue
```

3. Implementasi operasi-operasi Queue

```
def createQueue():
    q=[]
    return (q)
def enqueue(q,data):
    q.insert(0,data)
    return(q)
def dequeue(q):
    data=q.pop()
    return(data)
def isEmpty(q):
    return (q==[])
def size(q):
    return (len(q))
```

2. Deque

1. Konsep Struktur Data Queue

Deque merupakan sebuah struktur data yang penambahan datanya dapat dilakukan di kedua ujung baik Front maupun Rear begitu pula untuk penghapusan juga dapat di lakukan di kedua ujung baik pada Front maupun Rear

2. Operasi operasi yang terdapat pada Queue

```
- deque,       Digunakan untuk inisialisasi awal untuk tempat menampung data deque
- addFront,    Digunakan untuk penambahan data baru di ujung front pada deque
- addRear,     Digunakan untuk penambahan data baru di ujung rear pada deque
- removeRear,  Digunakan untuk untuk penghapusan data pada ujung rear
- removeFront, Digunakan untuk penghapusan data pada ujung front
- isEmpty,     Dugunakan untuk pengecekan apakah deque dalam keadaan kosong
- size,        Digunakan untuk mengatahui jumlah data yang terdapat pada deque
```

3. Implementasi operasi-operasi Queue

```
def createDeque():
    d=[]
    return (d)
def addFront(d,data):
    d.insert(0,data)
    return(d)
def addRear(d,data):
```

```
    d.append(data)
    return(d)
def removeRear(d):
    data=d.pop()
    return(data)
def removeFront(d):
    data=d.pop(0)
    return(data)
def isEmpty(d):
    return (d==[])
def size(d):
    return (len(d))
```

2 Implementasi

2.1 Konversi Infix ke Postfix

Buatlah Fungsi dan code yang diperlukan untuk mengkonversi ekspressi aritmatika infix menjadi ekspresi aritmatika postfix, dengan menggunakan struktur data stack, dengan output seperti yang ditunjukkan pada Gambar 4 (terdapat visualisasi token yang dibaca dan isi dari stack).

```
print('((a+b)*(c-d))/f : ',inf2Post('((a+b)*(c-d))/f'))

read token-1 :(
stack :  ['(']

read token-2 :(
stack :  ['(', '(']

read token-3 :a
stack :  ['(', '(']

read token-4 :+
stack :  ['(', '(', '+']

read token-5 :b
stack :  ['(', '(', '+']

read token-6 :)
stack :  ['(']

read token-7 :*
stack :  ['(', '*']

read token-8 :(
stack :  ['(', '*', '(']

read token-9 :c
stack :  ['(', '*', '(']

read token-10 :-
stack :  ['(', '*', '(', '-']

read token-11 :d
stack :  ['(', '*', '(', '-']

read token-12 :)
stack :  ['(', '*']

read token-13 :)
stack :  []

read token-14 :/
stack :  ['/']

read token-15 :f
stack :  ['/']

((a+b)*(c-d))/f :  ab+cd-*f/
```

```
print('a+b*c/d-f+g*h : ',inf2Post('a+b*c/d-f+g*h'))

read token-1 :a
stack :  []

read token-2 :+
stack :  ['+']

read token-3 :b
stack :  ['+']

read token-4 :*
stack :  ['+', '*']

read token-5 :c
stack :  ['+', '*']

read token-6 :/
stack :  ['+', '/']

read token-7 :d
stack :  ['+', '/']

read token-8 :-
stack :  ['-']

read token-9 :f
stack :  ['-']

read token-10 :+
stack :  ['+']

read token-11 :g
stack :  ['+']

read token-12 :*
stack :  ['+', '*']

read token-13 :h
stack :  ['+', '*']

a+b*c/d-f+g*h :  abc*d/+f-gh*+
```

```
print('(A*(B+C))-D : ',inf2Post('(A*(B+C))-D'))
```

```
read token-1 :(
stack :  ['(']

read token-2 :A
stack :  ['(']

read token-3 :*
stack :  ['(', '*']

read token-4 :(
stack :  ['(', '*', '(']

read token-5 :B
stack :  ['(', '*', '(']

read token-6 :+
stack :  ['(', '*', '(', '+']

read token-7 :C
stack :  ['(', '*', '(', '+']

read token-8 :)
stack :  ['(', '*']

read token-9 :)
stack :  []

read token-10 :-
stack :  ['-']

read token-11 :D
stack :  ['-']

(A*(B+C))-D :  ABC+*D-
```

Gambar 1: Konversi Infix ke Postfix dengan Stack

```
# Import modul

from google.colab import drive
drive.mount('/content/drive')

import sys
sys.path.append('/content/drive/MyDrive/Colab_Notebooks/')

from modules import *
```

```
# Jawaban
def inf2Post(strMath):

    hasil = ""
    oprStc = stack()
    opr = {'*':2,'/':2,'+':1,'-':1}
    count=0

    for i in strMath :
        count+=1
        print('read token-',count," : ",i)

        if i == '(':
            push(oprStc,i)

        elif i == ')':
            while peek(oprStc) != '(':
                hasil+=pop(oprStc)
            pop(oprStc)

        elif i in opr.keys():
            while not(isEmpty(oprStc)) and peek(oprStc) != '(' and opr[i] <= opr[peek(oprStc)]:
                hasil+=pop(oprStc)
            push(oprStc,i)

        else:
            hasil+=i

        print('stack : ',oprStc,"\n")

    while not isEmpty(oprStc):
        hasil+= pop(oprStc)

    return (strMath+" : "+hasil)

print(inf2Post('((A+B)*(C-D))/F'))
```

```
read token- 1  :  (
stack :  ['(']

read token- 2  :  (
stack :  ['(', '(']

read token- 3  :  A
stack :  ['(', '(']

read token- 4  :  +
stack :  ['(', '(', '+']
```

```
read token- 5  :  B
stack :  ['(', '(', '+']

read token- 6  :  )
stack :  ['(']

read token- 7  :  *
stack :  ['(', '*']

read token- 8  :  (
stack :  ['(', '*', '(']

read token- 9  :  C
stack :  ['(', '*', '(']

read token- 10  :  -
stack :  ['(', '*', '(', '-']

read token- 11  :  D
stack :  ['(', '*', '(', '-']

read token- 12  :  )
stack :  ['(', '*']

read token- 13  :  )
stack :  []

read token- 14  :  /
stack :  ['/']

read token- 15  :  F
stack :  ['/']

((A+B)*(C-D))/F : AB+CD-*F/
```

2.2 Evaluasi Postfix

Buatlah Fungsi dan code yang diperlukan untuk evaluasi ekspresi aritmatika postfix, dengan menggunakan struktur data stack, dengan output seperti yang ditunjukkan pada Gambar 2 (terdapat visualisasi operasi matematika yang dikerjakan tiap tahap).

```
print(evaluatePostFix("2 6 +"))
```

```
[1]: 2 + 6 = 8.0
8.0
```

```
print(evaluatePostFix("10 2 3 + * 4 -"))
```

```
[1]: 2 + 3 = 5.0
[2]: 10 * 5.0 = 50.0
[3]: 50.0 - 4 = 46.0
46.0
```

```
print(evaluatePostFix("2 3 4 * 10 / + 11 - 9 2 * +"))
```

```
[1]: 3 * 4 = 12.0
[2]: 12.0 / 10 = 1.2
[3]: 2 + 1.2 = 3.2
[4]: 3.2 - 11 = -7.8
[5]: 9 * 2 = 18.0
[6]: -7.8 + 18.0 = 10.2
10.2
```

Gambar 2: Evaluasi Ekspressi Aritmatika Postfix

```
#Jawaban
def evaluatePostfix(strPost):
    strPost = strPost.split()
    oprndStc = stack()
    opr = '*/+-'
    count=1

    for i in strPost:
        if i not in opr:
            push(oprndStc,i)

        else:
            oprnd1 = pop(oprndStc)
            oprnd2 = pop(oprndStc)

            if i == "*":
                hsl = float(oprnd2) * float(oprnd1)

            elif i == "/":
                hsl = float(oprnd2) / float(oprnd1)

            elif i == "-":
                hsl = float(oprnd2) - float(oprnd1)

            elif i == "+":
                hsl = float(oprnd2) + float(oprnd1)

            print(f"[{count}]: {oprnd2} {i} {oprnd1} = {hsl}")
            count+=1
            push(oprndStc,hsl)

    return pop(oprndStc)

print(evaluatePostfix("2 3 4 * 10 / + 11 - 9 2 * +"))
```

```
[1]: 3 * 4 = 12.0
[2]: 12.0 / 10 = 1.2
[3]: 2 + 1.2 = 3.2
[4]: 3.2 - 11 = -7.8
[5]: 9 * 2 = 18.0
[6]: -7.8 + 18.0 = 10.2
10.2
```

2.3 Palindrome

Buatlah fungsi dan code yang diperlukan untuk pengecekan palindrome suatu kata atau kalimat, dengan menggunakan struktur data Deque, dengan output seperti yang ditunjukkan pada Gambar 3

```
print(cekPalindrom('katak'))
```

```
front and rear --> 'k' == 'k' ? : True
front and rear --> 'a' == 'a' ? : True
True
```

Gambar 3: Palindrome

```
# Jawaban
def cekPalindrom(strCh):
    palindrom = createDeque()
    cek = True

    for i in strCh :
        addRear(palindrom,i)

    while size(palindrom) > 1 :
        rear = removeRear(palindrom)
        front = removeFront(palindrom)

        if rear == front :
            cek = True

        else:
            cek = cek and False

        print(f"front and rear --> '{front}' == '{rear}' ? : {cek}")

    return cek

```

```
23 print(cekPalindrom('katak'))
```

```
front and rear --> 'k' == 'k' ? : True
front and rear --> 'a' == 'a' ? : True
True
```

2.4 Scheduling / Penjadwalan

Terdapat proses penjadwalan CPU (resource sharing). Pada penjadwalan CPU terdapat beberapa proses sebagai berikut :

- Semua task masuk kedalam antrian, setiap task ini terdapat atribut waktu yang dibutuhkan masingmasing task untuk menggunakan CPU. Misalkan task A membutuhkan waktu untuk diproses di CPU selama 7 detik
- Task yang berada di posisi paling depan dari antrian, mendapat kesempatan pertama untuk diproses ke dalam CPU
- CPU memiliki atribut waktu, yaitu semua task hanya boleh menggunakan CPU selama waktu tertentu. Misalkan waktu CPU adalah 3 detik, maka semua task hanya boleh mengakses/menggunakan waktu CPU 3 detik saja
    - Jika suatu task membutuhkan waktu lebih dari waktu CPU, maka task tersebut akan diproses selama waktu CPU saja, kemudian task akan dikeluarkan dan dimasukkan kembali (posisi rear) ke dalam antrian, agar task tersebut dapat diproses selanjutnya
    - Jika task membutuhkan waktu tidak melebih waktu proses CPU, maka task diproses sesuai waktu proses task, kemudian dikeluarkan dari antrian, yang berarti task tersebut telah selesai diproses oleh CPU.

      Buatlah ilustrasi penjadwalan CPU tersebut dengan menggunakan modul queue yang telah dibuat sebelumnya, dengan ketentuan sebagai berikut :

1. Input berupa jumlah *task* atau proses yang akan diproses pada CPU
2. Nama proses, beserta atribut waktu yang dibutuhkan masing-masing proses untuk menggunakan CPU
3. Waktu proses CPU Contoh ilustrasi penjadwalan CPU dapat dilihat pada Gambar 4 HashtagVideo : Modul4 SchedulingQueue dan PraktikumStrukturData KelasX

```
Jumlah Proses yang akan dijadwal di CPU = 3
Nama Proses ke-0 : A
Waktu proses : 5
Nama Proses ke-1 : B
Waktu proses : 9
Nama Proses ke-2 : C
Waktu proses : 2
```

```
Antrian Proses :
[['C', 2], ['B', 9], ['A', 5]]

waktu proses CPU = 3
Antrian Proses beserta Waktunya =  [['C', 2], ['B', 9], ['A', 5]]
Iterasi ke- 1 :
     Proses  A sedang diproses, dan sisa waktu proses  A = 2
     Data proses yang tersisa :  [['A', 2], ['C', 2], ['B', 9]]
Iterasi ke- 2 :
     Proses  B sedang diproses, dan sisa waktu proses  B = 6
     Data proses yang tersisa :  [['B', 6], ['A', 2], ['C', 2]]
Iterasi ke- 3 :
     Proses  C telah selesai diproses
     Data proses yang tersisa :  [['B', 6], ['A', 2]]
Iterasi ke- 4 :
     Proses  A telah selesai diproses
     Data proses yang tersisa :  [['B', 6]]
Iterasi ke- 5 :
     Proses  B sedang diproses, dan sisa waktu proses  B = 3
     Data proses yang tersisa :  [['B', 3]]
Iterasi ke- 6 :
     Proses  B telah selesai diproses
     Data proses yang tersisa :  []
```

Gambar 4: Penjadwalan dengan Struktur Data Queue

```
1 # Jawaban
2 def inputTask():
3     task ={}
4     jmlh = int(input("Jumlah proses yang akan di jadwal di CPU = "))
5     for i in range(jmlh):
6         nama = input(f"Nama proses ke-{i} : ")
7         waktu =  int(input("waktu proses : "))
8         task[nama]=[waktu,0]
9     return task
```

```
    return task

def scheduling(task):
    tskQueue = createQueue()
    for i in task:
        enqueue(tskQueue,[i,task[i][0]])
    print("Antrian Proses : \n",tskQueue,'\n')

    limitTime = int(input("waktu proses CPU = "))
    print("Antrian Proses beserta waktunya =B
     ",tskQueue)
    totalWaktu = 0
    no=0
    while not isEmpty(tskQueue):
        no+=1
        print("Iterasi ke- ",no)
        temp = dequeue(tskQueue)
        waktuTunggu = temp[1] - limitTime

        if waktuTunggu > 0 :
            enqueue(tskQueue,temp)
            prosTime = limitTime
            print(f'\tProses {temp[0]} sedang diproses, dan sisa waktu proses {temp[0]} = {waktuTunggu}')
        else:
            prosTime = temp[1]
            waktuTunggu = 0
            print(f'\tProses {temp[0]} telah selesai diproses')

        totalWaktu += prosTime
        temp[1] = waktuTunggu
        task[temp[0]][0] = totalWaktu
        print("\tData proses tersisa : ",tskQueue)

dataTask = inputTask()
scheduling(dataTask)
```

```
Jumlah proses yang akan di jadwal di CPU = 3
Nama proses ke-0 : A
waktu proses : 5
Nama proses ke-1 : B
waktu proses : 9
Nama proses ke-2 : C
waktu proses : 2
Antrian Proses :
 [['C', 2], ['B', 9], ['A', 5]]

waktu proses CPU = 3
Antrian Proses beserta waktunya =  [['C', 2], ['B', 9], ['A', 5]]
Iterasi ke-  1
        Proses A sedang diproses, dan sisa waktu proses A = 2
        Data proses tersisa :  [['A', 2], ['C', 2], ['B', 9]]
Iterasi ke-  2
        Proses B sedang diproses, dan sisa waktu proses B = 6
        Data proses tersisa :  [['B', 6], ['A', 2], ['C', 2]]
Iterasi ke-  3
        Proses C telah selesai diproses
        Data proses tersisa :  [['B', 6], ['A', 2]]
Iterasi ke-  4
        Proses A telah selesai diproses
        Data proses tersisa :  [['B', 6]]
Iterasi ke-  5
        Proses B sedang diproses, dan sisa waktu proses B = 3
        Data proses tersisa :  [['B', 3]]
Iterasi ke-  6
        Proses B telah selesai diproses
        Data proses tersisa :  []
```