



ALGORITMA DAN STRUKTUR DATA

INF1083

LAPORAN PRAKTIKUM 6 : Struktur Data Dasar (1)

Oleh :

Akhmad Qasim

2211102441237

Teknik Informatika
Fakultas Sains & Teknologi
Universitas Muhammadiyah Kalimantan Timur

Samarinda, 2023

Laporan Praktikum 6:

Struktur Data Dasar (1)

Pokok Bahasan:

- ❖ Stack.
- ❖ Infix, Prefix, dan Postfix.

Tujuan Pembelajaran:

- ✓ Memahami implementasi *Stack* pada struktur data *Python*.
- ✓ Memahami penulisan ekspresi aritmatika di *Python*.

Percobaan & Latihan:

1. Stack

```
1 from pythonds.basic.stack import Stack
2
3 s = Stack()
4
5
6 print(s.isEmpty())
7 s.push(4)
8 s.push('dog')
9 print(s.peek())
10 s.push(True)
11 print(s.size())
12 print(s.isEmpty())
13 s.push(8.4)
14 print(s.pop())
15 print(s.pop())
16 print(s.size())
```

a) Berikan tampilan output dari perintah diatas!

Tampilan Output:

```
True
dog
3
False
8.4
True
2
```

b) Analisa hasil dari perintah diatas!

Analisa syntax:

```

1 # from pythonds.basic.stack import Stack
2
3 class Stack: # Membuat kelas Stack
4     new *
5     def __init__(self): # Membuat konstruktor untuk kelas Stack
6         self.items = [] # Membuat list kosong
7
8     new *
9     def isEmpty(self): # Membuat method untuk mengecek apakah stack kosong
10        return self.items == [] # Mengembalikan nilai True jika stack kosong
11
12    new *
13    def push(self, item): # Membuat method untuk menambahkan item ke stack
14        self.items.append(item) # Menambahkan item ke stack
15
16    new *
17    def pop(self): # Membuat method untuk menghapus item dari stack
18        return self.items.pop() # Menghapus item dari stack dan mengembalikan nilai item
19
20    new *
21    def peek(self): # Membuat method untuk melihat item teratas dari stack
22        return self.items[len(self.items) - 1] # Mengembalikan nilai item teratas dari stack
23
24    new *
25    def size(self): # Membuat method untuk mengetahui ukuran stack
26        return len(self.items) # Mengembalikan nilai panjang stack
27
28 s = Stack() # Membuat objek s dari kelas Stack
29
30 print(s.isEmpty()) # Menampilkan hasil dari method isEmpty
31 s.push(4) # Menambahkan item ke stack
32 s.push('dog') # Menambahkan item ke stack
33 print(s.peek()) # Menampilkan hasil dari method peek
34 s.push(True) # Menambahkan item ke stack
35 print(s.size()) # Menampilkan hasil dari method size
36 print(s.isEmpty()) # Menampilkan hasil dari method isEmpty
37 s.push(8.4) # Menambahkan item ke stack
38 print(s.pop()) # Menampilkan hasil dari method pop
39 print(s.pop()) # Menampilkan hasil dari method pop
40 print(s.size()) # Menampilkan hasil dari method size

```

2. Stack

a) Jalankan perintah berikut dan analisa hasil output.

```

24 print(parChecker('(((((')))
25 print(parChecker('(()'))

```

Tampilan Syntax:

```

1 from pythonds.basic.stack import Stack # Mengimpor kelas Stack dari modul pythonds.basic.stack
2
3 # parChecker merupakan fungsi yang menerima parameter symbolString
4 # Fungsi ini akan mengembalikan nilai True jika symbolString memiliki jumlah kurung yang sama
  akhmadqasim*
5 def parChecker(symbolString): # Membuat fungsi parChecker
6     s = Stack() # Membuat objek s dari kelas Stack
7     balanced = True # Membuat variabel balanced dengan nilai True
8     index = 0 # Membuat variabel index dengan nilai 0
9     # Melakukan perulangan selama index kurang dari panjang symbolString dan balanced bernilai True
10    while index < len(symbolString) and balanced:
11        symbol = symbolString[index] # Mengambil nilai symbol dari symbolString
12        if symbol == "(": # Jika symbol bernilai (
13            s.push(symbol) # Menambahkan symbol ke stack
14        else: # Jika symbol tidak bernilai (
15            if s.isEmpty(): # Jika stack kosong
16                balanced = False # Mengubah nilai balanced menjadi False karena jumlah kurung tidak sama
17            else: # Jika stack tidak kosong
18                s.pop() # Menghapus item dari stack
19
20        index = index + 1 # Menambahkan nilai index dengan 1
21
22    if balanced and s.isEmpty(): # Jika balanced bernilai True dan stack kosong
23        return True # Mengembalikan nilai True
24    else: # Jika balanced bernilai False atau stack tidak kosong
25        return False # Mengembalikan nilai False
26
27
28 print(parChecker('((()))')) # Menampilkan hasil dari fungsi parChecker
29 print(parChecker('(()')) # Menampilkan hasil dari fungsi parChecker

```

Tampilan Output:

```

True
False

```

Nilai yang pertama bernilai True karena tanda kurung yang muncul seimbang, sedangkan nilai yang kedua kekurangan 1 tanda kurung ')' yang berarti tidak seimbang.

- b) Jalankan perintah berikut dan berikan alasan mengapa menghasilkan output False!

```

27 print(parChecker('{{}}'))
28 print(parChecker('{{}}'))

```

Tampilan Syntax:

```

1 from pythonds.basic.stack import Stack # Mengimpor kelas Stack dari modul pythonds.basic.stack
2
3 # parChecker merupakan fungsi yang menerima parameter symbolString
4 # Fungsi ini akan mengembalikan nilai True jika symbolString memiliki jumlah kurung yang sama
  akhmadqasim*
5 def parChecker(symbolString): # Membuat fungsi parChecker
6     s = Stack() # Membuat objek s dari kelas Stack
7     balanced = True # Membuat variabel balanced dengan nilai True
8     index = 0 # Membuat variabel index dengan nilai 0
9     # Melakukan perulangan selama index kurang dari panjang symbolString dan balanced bernilai True
10    while index < len(symbolString) and balanced:
11        symbol = symbolString[index] # Mengambil nilai symbol dari symbolString
12        if symbol == "(": # Jika symbol bernilai (
13            s.push(symbol) # Menambahkan symbol ke stack
14        else: # Jika symbol tidak bernilai (
15            if s.isEmpty(): # Jika stack kosong
16                balanced = False # Mengubah nilai balanced menjadi False karena jumlah kurung tidak sama
17            else: # Jika stack tidak kosong
18                s.pop() # Menghapus item dari stack
19
20        index = index + 1 # Menambahkan nilai index dengan 1
21
22    if balanced and s.isEmpty(): # Jika balanced bernilai True dan stack kosong
23        return True # Mengembalikan nilai True
24    else: # Jika balanced bernilai False atau stack tidak kosong
25        return False # Mengembalikan nilai False
26
27
28 print(parChecker('{{}}'))
29 print(parChecker('{{}}'))

```

Tampilan Output:

```

False
False

```

Pada output baris pertama dan baris kedua mengembalikan nilai False dikarenakan pada struktur kontrol hanya mengecek tanda kurung saja.

3. Stack

a) Berikan tampilan output dari perintah diatas!

Tampilan Output:

```

True
False

```

b) Jelaskan perbedaan fungsi diatas dengan fungsi yang terdapat di Percobaan & Latihan 6.2!

Tampilan syntax:

```

1  from pythonds.basic.stack import Stack # Mengimpor modul Stack
2
3
4  * akhmadqasim *
5  def parChecker(symbolString): # Membuat fungsi parChecker
6      s = Stack() # Membuat objek Stack
7      balanced = True # Membuat variabel balanced dengan nilai True
8      index = 0 # Membuat variabel index dengan nilai 0
9      while index < len(symbolString) and balanced: # Melakukan perulangan while
10         symbol = symbolString[index] # Membuat variabel symbol dengan nilai symbolString[index]
11         if symbol in "([{": # Jika salah satu symbol berada di dalam "([{"
12             s.push(symbol) # Maka masukkan nilai symbol ke dalam stack
13         else: # Maka jika tidak
14             if s.isEmpty(): # Jika stack kosong
15                 balanced = False # Maka variabel balanced akan bernilai False
16             else: # Maka jika tidak
17                 top = s.pop() # Maka variabel top akan bernilai nilai yang di pop dari stack
18                 if not matches(top, symbol): # Jika nilai top dan symbol tidak sama
19                     balanced = False # Maka variabel balanced akan bernilai False
20
21             index = index + 1 # Menambahkan nilai index dengan 1
22
23         if balanced and s.isEmpty(): # Jika variabel balanced bernilai True dan stack kosong
24             return True # Maka kembalikan nilai True
25         else: # Maka jika tidak
26             return False # Maka kembalikan nilai False
27
28  new *
29  def matches(open, close): # Membuat fungsi matches berfungsi untuk mengecek apakah open dan close sama
30      opens = "([{" # Membuat variabel opens dengan nilai "([{"
31      closers = ")]}" # Membuat variabel closers dengan nilai ")]}"
32      return opens.index(open) == closers.index(close) # Mengembalikan nilai True jika open dan close sama
33
34  print(parChecker('{([[]])}()')) # Menampilkan hasil dari fungsi parChecker
35  print(parChecker('{([()])}')) # Menampilkan hasil dari fungsi parChecker

```

Terdapat perbedaan antara fungsi pada percobaan & latihan 6.2 dengan percobaan & latihan 6.3 dimana terdapat tambahan pada struktur kontrol dan fungsi matcher yang dapat mengenali tanda kurung, kurung siku, dan kurung kurawal.

4. Stack

a) Jalankan fungsi diatas menggunakan perintah berikut!

```

17 print(divideBy2(42))
18 print(divideBy2(55))

```

Tampilan Syntax:

```
1 from pythonds.basic.stack import Stack # Mengimpor modul Stack
2
3
4 def divideBy2(decNumber): # Fungsi divideBy2 untuk mengubah bilangan desimal menjadi biner
5     remstack = Stack() # Membuat objek Stack
6
7     while decNumber > 0: # Melakukan perulangan while
8         rem = decNumber % 2 # Membuat variabel rem dengan nilai sisa pembagian decNumber dengan 2
9         remstack.push(rem) # Memasukkan nilai rem ke dalam stack
10        decNumber = decNumber // 2 # Membuat variabel decNumber dengan nilai pembagian decNumber dengan 2
11
12        binString = "" # Membuat variabel binString dengan nilai string kosong
13        while not remstack.isEmpty(): # Melakukan perulangan while
14            binString = binString + str(remstack.pop()) # Menambahkan nilai binString dengan nilai yang di pop dari stack
15
16        return binString # Mengembalikan nilai binString
17
18
19 print(divideBy2(42)) # Menampilkan hasil dari fungsi divideBy2
20 print(divideBy2(55)) # Menampilkan hasil dari fungsi divideBy2
```

b) Analisa hasil ouput diatas!

Tampilan Output:

```
101010
110111
```

Hasil output merupakan hasil operasi fungsi divideBy2 yang merubah bilangan desimal menjadi bilangan biner.

5. Stack

a) Jelaskan perbedaan ouput dari kedua perintah diatas (ouput baris 19 dan 20)!

Tampilan Syntax:

```

1 from pythonds.basic.stack import Stack # Mengimpor modul Stack
2
3
4 def baseConverter(decNumber, base): # Fungsi baseConverter untuk mengubah bilangan desimal menjadi bilangan lain
5     # Variabel digits untuk menyimpan nilai dari 0-9 dan A-F
6     # yang akan digunakan untuk mengubah bilangan desimal menjadi bilangan lain
7     digits = "0123456789ABCDEF"
8
9     remstack = Stack() # Membuat objek Stack
10
11     while decNumber > 0: # Melakukan perulangan while
12         rem = decNumber % base # Membuat variabel rem dengan nilai sisa pembagian decNumber dengan base
13         remstack.push(rem) # Memasukkan nilai rem ke dalam stack
14         decNumber = decNumber // base # Membuat variabel decNumber dengan nilai pembagian decNumber dengan base
15
16     newString = "" # Membuat variabel newString dengan nilai string kosong
17     while not remstack.isEmpty(): # Melakukan perulangan while
18         # Menambahkan nilai newString dengan nilai yang di pop dari stack
19         newString = newString + digits[remstack.pop()]
20
21     return newString # Mengembalikan nilai newString
22
23
24 print(baseConverter(25, 2)) # Merubah bilangan desimal menjadi bilangan biner
25 print(baseConverter(25, 16)) # Merubah bilangan desimal menjadi bilangan heksadesimal

```

Tampilan Output:

```

11001
19

```

Perbedaan antara baris 24 dan 25 adalah kita menggunakan fungsi baseConverter untuk mengubah bilangan desimal menjadi bilangan lain. Pada baris 24 kita menggunakan parameter base 2 yang berarti merubah bilangan desimal menjadi bilangan biner, sedangkan pada baris 25 kita menggunakan fungsi baseConverter dengan parameter base 16 yang berarti kita akan mengubah bilangan desimal menjadi bilangan heksadesimal.

6. Infix, Prefix, dan Postfix

a) Berikan hasil output dari perintah diatas!

Tampilan Output:

```

A B * C D * +
A B + C * D E - F G + *
A B + C D + *
A B + C *
A B C * +

```

b) Berikan penjelasan tiap baris pada coding diatas!

Tampilan Syntax:


```

1 from pythonds.basic.stack import Stack # Mengimpor modul Stack
2
3
4 def infixToPostfix(infixexpr): # Fungsi infixToPostfix untuk mengubah ekspresi infix menjadi postfix
5     prec = {} # Membuat variabel prec dengan nilai dictionary kosong
6     prec["*"] = 3 # Menambahkan nilai ke dalam variabel prec
7     prec["/"] = 3 # Menambahkan nilai ke dalam variabel prec
8     prec["+"] = 2 # Menambahkan nilai ke dalam variabel prec
9     prec["-"] = 2 # Menambahkan nilai ke dalam variabel prec
10    prec["("] = 1 # Menambahkan nilai ke dalam variabel prec
11    opStack = Stack() # Membuat objek Stack
12    postfixList = [] # Membuat variabel postfixList dengan nilai list kosong
13    tokenList = infixexpr.split() # Membuat variabel tokenList dengan nilai list yang di split dari variabel infixexpr
14
15    for token in tokenList: # Melakukan perulangan for
16        # Melakukan pengecekan apakah token berada di dalam variabel prec
17        if token in "ABCDEFGHIJKLMNOPQRSTUVWXYZ" or token in "0123456789":
18            postfixList.append(token) # Menambahkan nilai token ke dalam variabel postfixList
19        elif token == '(': # Jika token bernilai (
20            opStack.push(token) # Memasukkan nilai token ke dalam stack
21        elif token == ')': # Jika token bernilai )
22            topToken = opStack.pop() # Membuat variabel topToken dengan nilai yang di pop dari stack
23            while topToken != '(': # Melakukan perulangan while
24                postfixList.append(topToken) # Menambahkan nilai topToken ke dalam variabel postfixList
25                topToken = opStack.pop() # Membuat variabel topToken dengan nilai yang di pop dari stack
26        else: # Jika token tidak ada di dalam variabel prec
27            while (not opStack.isEmpty()) and \
28                (prec[opStack.peek()] >= prec[token]): # Melakukan perulangan while
29                # Menambahkan nilai yang di pop dari stack ke dalam variabel postfixList
30                postfixList.append(opStack.pop())
31            opStack.push(token) # Memasukkan nilai token ke dalam stack
32
33    while not opStack.isEmpty(): # Melakukan perulangan while
34        postfixList.append(opStack.pop()) # Menambahkan nilai yang di pop dari stack ke dalam variabel postfixList
35    return " ".join(postfixList) # Mengembalikan nilai variabel postfixList yang di join dengan spasi
36
37
38 print(infixToPostfix("A * B + C * D")) # Menampilkan hasil dari fungsi infixToPostfix
39 print(infixToPostfix("( A + B ) * C - ( D - E ) * ( F + G )")) # Menampilkan hasil dari fungsi infixToPostfix
40 print(infixToPostfix("( A + B ) * ( C + D )")) # Menampilkan hasil dari fungsi infixToPostfix
41 print(infixToPostfix("( A + B ) * C")) # Menampilkan hasil dari fungsi infixToPostfix
42 print(infixToPostfix("A + B * C")) # Menampilkan hasil dari fungsi infixToPostfix

```

7. Infix, Prefix, dan Postfix

a) Berikan hasil output dari perintah diatas!

Tampilan Output:

3.0

b) Berikan penjelasan tiap baris pada coding diatas!

Tampilan Syntax:

```

1 from pythonds.basic.stack import Stack # Mengimpor modul Stack
2
3
4 def postfixEval(postfixExpr): # Fungsi postfixEval untuk mengubah ekspresi postfix menjadi nilai
5     operandStack = Stack() # Membuat objek Stack
6     # Membuat variabel tokenList dengan nilai list yang di split dari variabel postfixExpr
7     tokenList = postfixExpr.split()
8
9     for token in tokenList: # Melakukan perulangan for untuk setiap token di dalam variabel tokenList
10         if token in "0123456789": # Melakukan pengecekan apakah token berada di dalam variabel prec
11             operandStack.push(int(token)) # Memasukkan nilai token ke dalam stack
12         else: # Jika token tidak ada di dalam variabel prec
13             operand2 = operandStack.pop() # Membuat variabel operand2 dengan nilai yang di pop dari stack
14             operand1 = operandStack.pop() # Membuat variabel operand1 dengan nilai yang di pop dari stack
15             result = doMath(token, operand1, operand2) # Membuat variabel result dengan nilai dari fungsi doMath
16             operandStack.push(result) # Memasukkan nilai result ke dalam stack
17     return operandStack.pop() # Mengembalikan nilai yang di pop dari stack
18
19
20 new *
21 def doMath(op, op1, op2): # Fungsi doMath untuk melakukan operasi matematika
22     if op == "*": # Melakukan pengecekan apakah op bernilai *
23         return op1 * op2 # Mengembalikan nilai op1 * op2
24     elif op == "/": # Melakukan pengecekan apakah op bernilai /
25         return op1 / op2 # Mengembalikan nilai op1 / op2
26     elif op == "+": # Melakukan pengecekan apakah op bernilai +
27         return op1 + op2 # Mengembalikan nilai op1 + op2
28     else: # Jika op bernilai -
29         return op1 - op2 # Mengembalikan nilai op1 - op2
30
31 print(postfixEval('7 8 + 3 2 + /')) # Menampilkan hasil dari fungsi postfixEval

```

Kesimpulan:

Di dalam pemrograman, "stack" (tumpukan) adalah struktur data yang digunakan untuk menyimpan dan mengakses elemen dengan aturan Last-In-First-Out (LIFO). Artinya, elemen yang terakhir dimasukkan ke dalam stack adalah elemen yang pertama kali diambil.

Dalam matematika, "infix", "prefix" dan "postfix" adalah notasi yang digunakan untuk menulis ekspresi aritmatika. Contoh ekspresi aritmatika adalah "3 + 4 * 2". Notasi infix adalah notasi yang paling umum digunakan, yaitu operator ditempatkan antara dua operand. Notasi prefix dan postfix adalah notasi yang operatornya ditempatkan sebelum atau sesudah operand.

Contoh notasi infix: 3 + 4 * 2

Contoh notasi prefix: + 3 * 4 2

Contoh notasi postfix: 3 4 2 * +

Untuk mengubah notasi infix menjadi prefix atau postfix, kita perlu menggunakan struktur data stack. Dalam konversi infix ke postfix, kita menggunakan stack untuk menyimpan

operator dan operand sementara sebelum menulis ulang ekspresi dalam notasi postfix. Dalam konversi infix ke prefix, kita juga menggunakan stack dengan aturan yang sama, hanya saja penulisan ulang dilakukan dalam notasi prefix.