

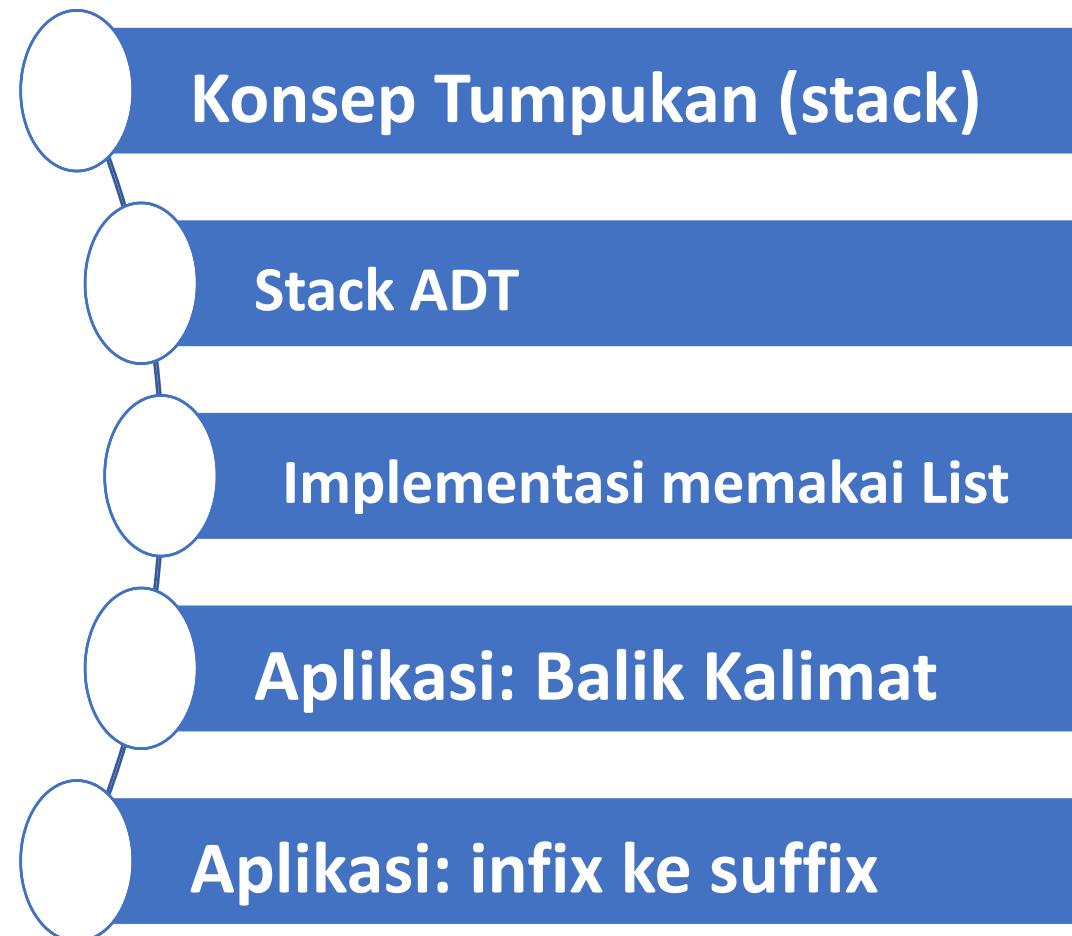


STRUKTUR DATA (PYTHON)

“Struktur Data Tumpukan (Stack)”

[@SUARGA | [Pertemuan 06]

OutLine



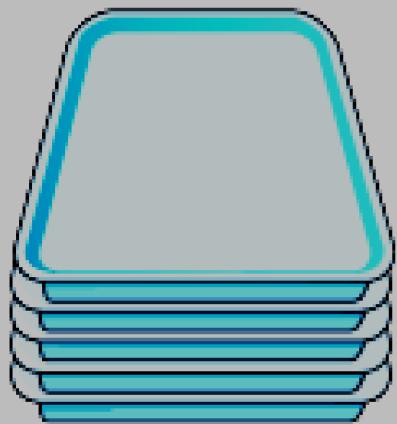


Konsep Tumpukan (STACK)

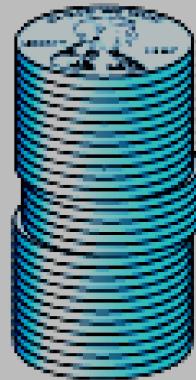
- Struktur Tumpukan (Stack) adalah struktur data yang meniru bagaimana proses menyimpan dan mengambil satu benda pada suatu tumpukan benda yang ada dilantai.
- Apabila diperhatikan dengan saksama maka proses menyimpan buku (disebut *push*) dan proses mengambil buku (disebut *pop*) dari suatu tumpukan selalu dilakukan pada bagian atas tumpukan (*top of the stack*).
- Sehingga terjadi urutan yang disebut LIFO (Last In First Out), artinya benda yang terakhir disimpan pada tumpukan adalah benda yang pertama yang bisa diambil karena benda inilah yang berada pada urutan teratas dari tumpukan.

Contoh Tumpukan

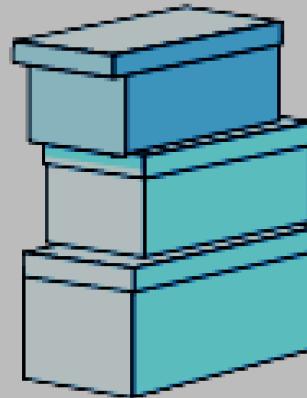
A stack of
cafeteria trays



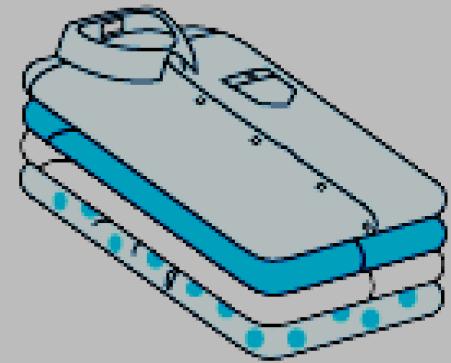
A stack
of pennies



A stack of
shoe boxes



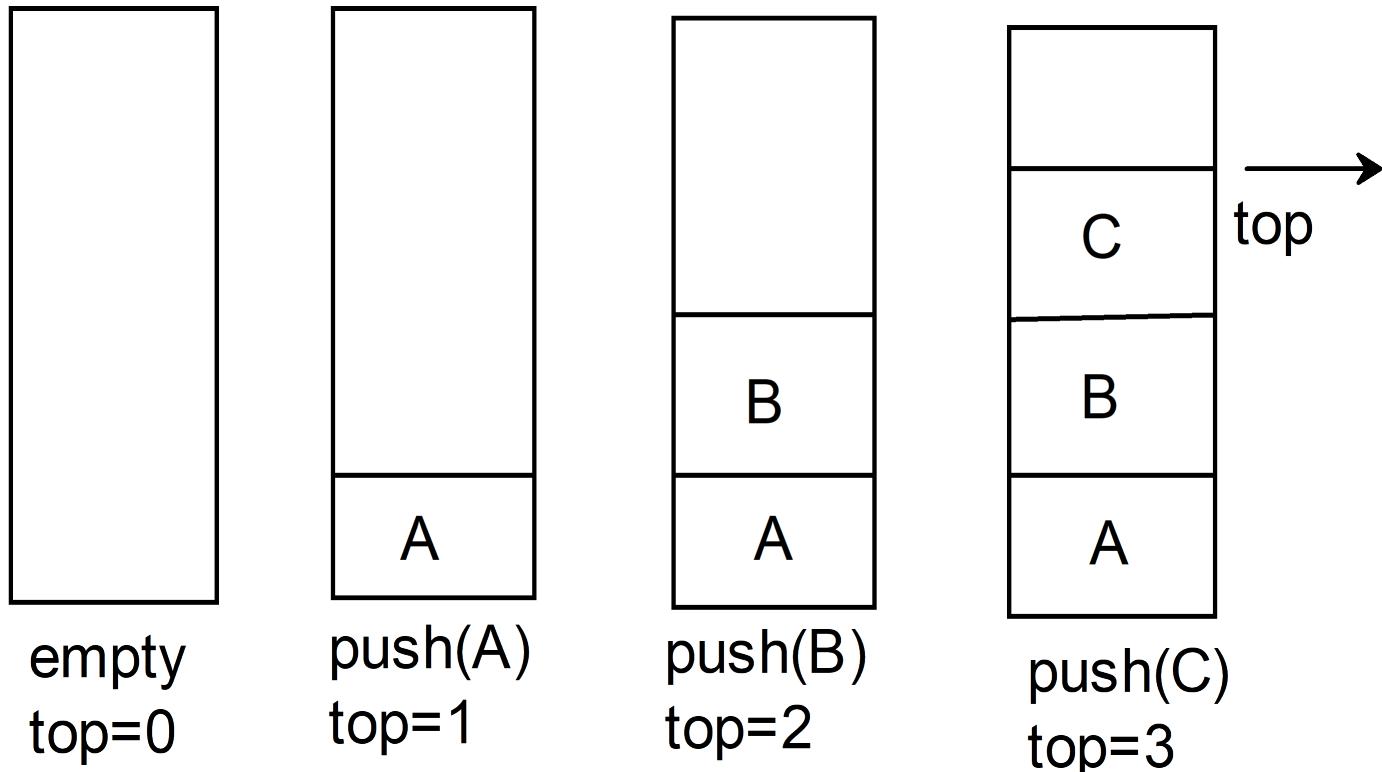
A stack of
neatly folded shirts



Stack ADT

- Tumpukan (Stack) digunakan apabila data akan di-akses dengan urutan “Last In First Out” (LIFO), data yang pertama bisa diakses adalah data yang terakhir dimasukkan. Beberapa fungsi pada ADT tumpukan yang perlu dibentuk untuk keperluan akses data adalah sebagai berikut:
- **push(x)** : menambahkan elemen x ke bagian top
- **pop()** : mengambil data dari posisi top
- **peek()** : melihat isi yang teratas/pertama
- **top()** : elemen teratas, sama dengan peek()
- **isEmpty()** : memeriksa apakah tumpukan kosong
- **len()** : menghitung jumlah elemen dalam tumpukan

Contoh : push(x)



Algoritma Push(x)

```
Prosedur push (input x : item; in-out S : Stack)
{ prosedur menempatkan item x pada posisi top dari stack }
```

Definisi Variabel

```
int atas;
```

Rincian Langkah

```
If S.top = maks
  then write ("sudah penuh");
else
  S.top ← S.top + 1;
  atas ← S.top;
  S.isi[atas] ← x;
endif.
```

Python: Fungsi push(x)

```
def push(self, x) :  
    if self.top == maks:  
        print ('Sudah penuh')  
    else:  
        self.top += 1  
        atas = self.top  
        self.isi[atas] = x
```

- menggunakan python list sebagai struktur data dasar akan memudah implementasi ADT dari stack, perhatikan fungsi push berikut:

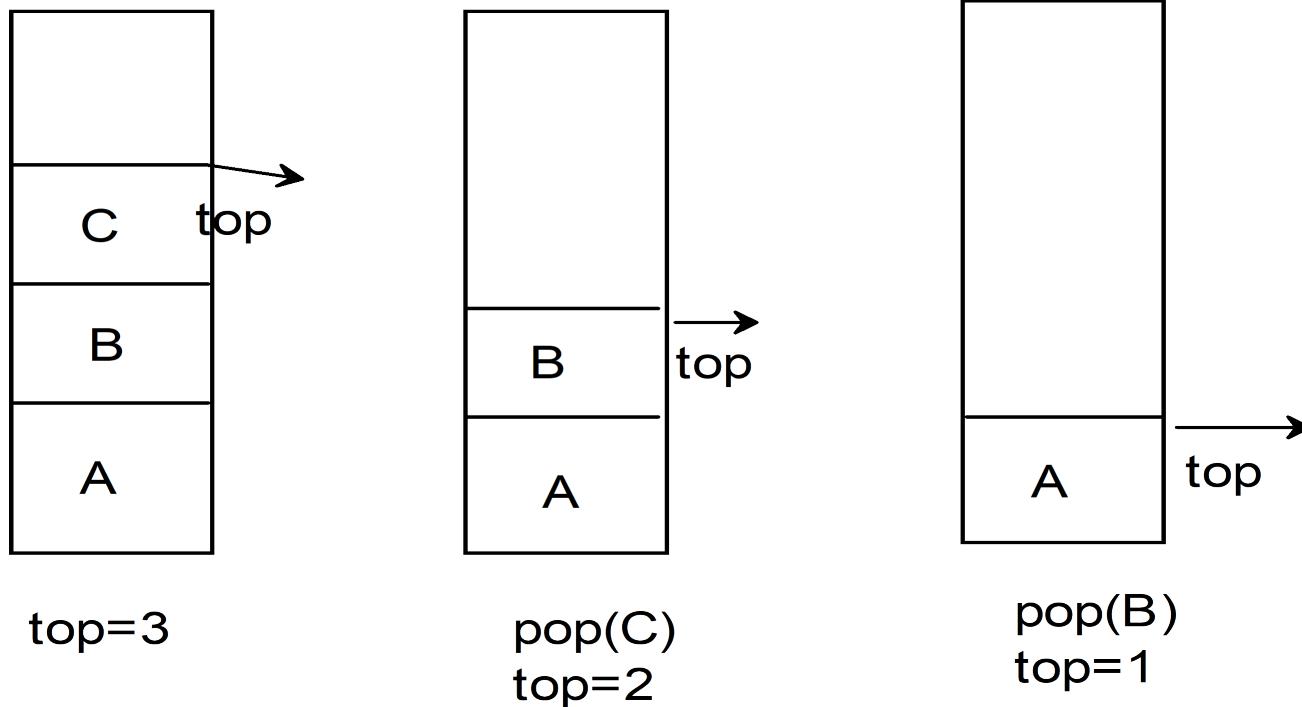
```
def push(self, e) :  
    # memasukkan elemen e di posisi top  
    self._data.append(e)
```

- fungsi append() pada list otomatis menempatkan elemen di posisi atas
- memakai list maka stack tidak akan penuh (list adalah dynamic array)

Class ArrayStack

```
class ArrayStack:  
    #LIFO Stack implementation using a Python list  
    #as underlying storage.  
  
    def __init__(self):  
        #Create an empty stack.  
        self._data = [] # nonpublic list instance  
  
    def __len__(self):  
        #Return the number of elements in the stack.  
        return len(self._data)
```

Contoh fungsi pop()



Algoritma fungsi Pop()

Prosedur pop (output x : item; in-out S:Stack)
{ prosedur mengambil data dari tumpukan pada posisi top
dari stack }

Definisi Variabel

int atas;

Rincian Langkah:

```
If S.top = 0
  then write ("tumpukan kosong");
else
  atas ← S.top;
  x ← S.isi[atas];
  S.top ← S.top - 1;
endif
```

Python: fungsi pop()

- berbasis operasi python list, prosedur pop()

```
def pop(self):  
    # mengambil elemen di posisi top  
    #Raise Empty exception if the stack is empty.  
  
    if self.is_empty():  
        raise Empty('Stack is empty')  
    return self._data.pop()
```

Python: fungsi top()

- ke pointer ter-atas

```
def top(self):  
    #bila stack kosong error  
    if self.is_empty():  
        raise Empty('stack is empty')  
    #bila stack ber-isi, ambil yang ter-atas  
    return self._data[-1]
```

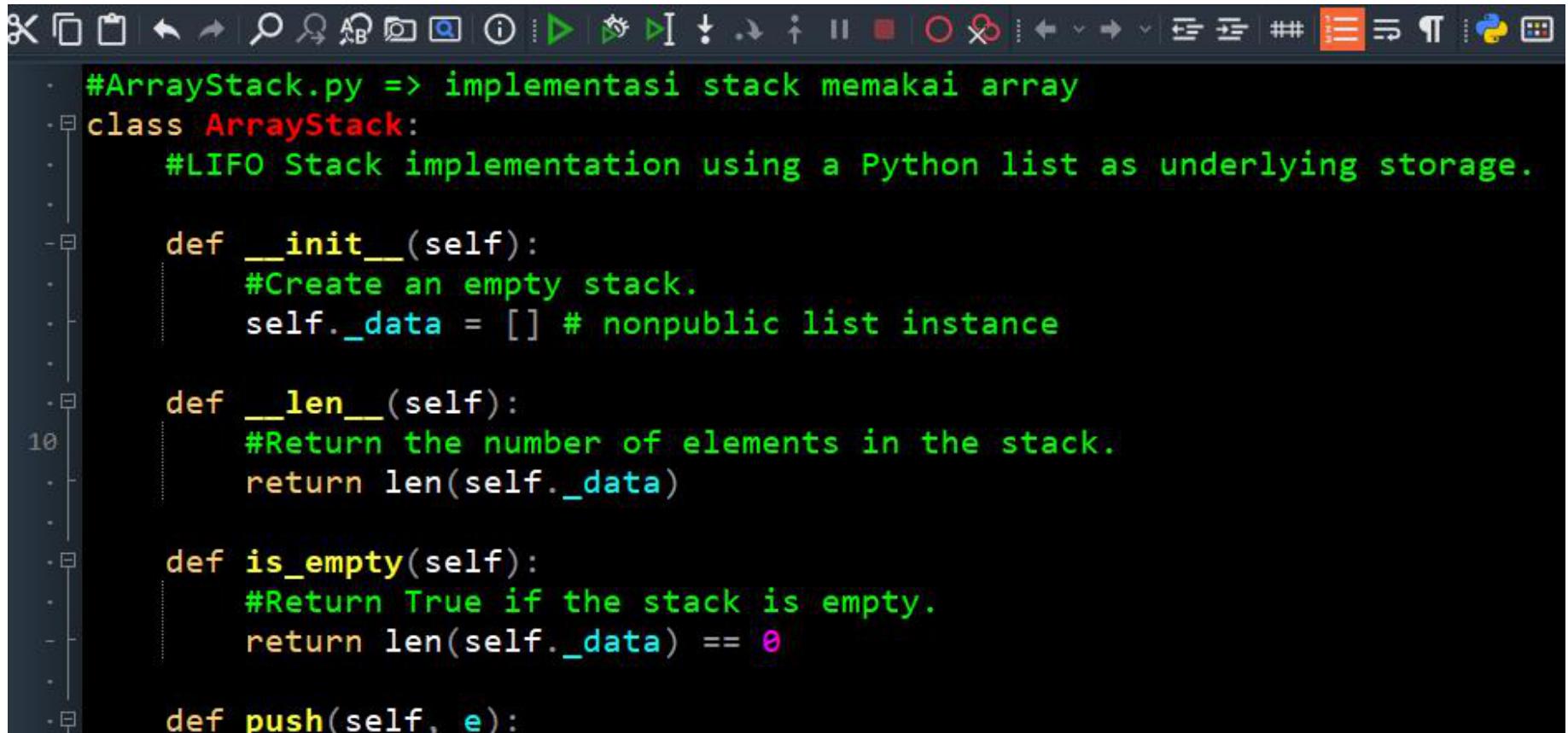
Fungsi peek() : melihat data ter-atas, sama dengan fungsi top()

Fungsi: lihat_isi()

- untuk melihat susunan data dalam stack
- dari yang ter-atas hingga ter-bawah

```
def lihat_isi(self):  
    n = len(self._data)  
    for i in range(n):  
        print(self._data[-i-1])
```

Implementasi Stack



```
#ArrayStack.py => implementasi stack memakai array
class ArrayStack:
    #LIFO Stack implementation using a Python list as underlying storage.

    def __init__(self):
        #Create an empty stack.
        self._data = [] # nonpublic list instance

    def __len__(self):
        #Return the number of elements in the stack.
        return len(self._data)

    def is_empty(self):
        #Return True if the stack is empty.
        return len(self._data) == 0

    def push(self, e):
```

```
def push(self, e):
    #Add element e to the top of the stack.
    self._data.append(e) # new item stored at end of list
20
def top(self):
    #Return (but do not remove) the element at the top
    #of the stack.
    #Raise Empty exception if the stack is empty.

    if self.is_empty():
        raise Empty('Stack is empty')
    return self._data[-1] # the last item in the list

30
def pop(self):
```

```
30 def pop(self):
    """
    Remove and return the element from the top of
    the stack (i.e., LIFO).
    Raise Empty exception if the stack is empty.

35     if self.is_empty():
        raise Empty('Stack is empty')
    return self._data.pop()

40 def peek(self):
    """
    peek the top element
    if self.is_empty():
        raise Empty('Stack is empty')
    return self._data[-1] #peek() sama dengan top()
```

Contoh pemakaian:

```
>>> S=ArrayStack()          #menciptakan stack S
>>> S.push(5)              #memasukkan data 5,4,3,2
>>> S.push(4)
>>> S.push(3)
>>> S.push(2)
>>> print(len(S))         #tinggitumpukan
4
>>> print(S.pop())         #ambil elemen teratas => 2
2
>>> print(S.top())          #elemen teratas berikutnya
3
>>> print(S.peek())         #melihat elemen ter-atas
3
>>> print(S.is_empty())     #apakah stack s kosong?
False
>>>
```

Aplikasi: Balik Kalimat

- Salah satu contoh aplikasi dari struktur data tumpukan ini adalah membalik suatu kalimat yang dimasukkan lewat keyboard. Misalkan input-nya adalah “program tumpukan” maka output-nya adalah “nakupmut margorp”.
- Prosesnya dapat dijelaskan secara sederhana, setiap huruf, mulai dari kiri, di masukkan (push) ke dalam tumpukan (stack) S, hingga seluruhnya ada di dalam S. Kemudian satu persatu huruf ini diambil (pop) dari S, dengan demikian kalimat akan terbaca terbalik.

```
#balik.py => membalikkan kalimat
from ArrayStack import *      #memanggil implementasi Stack
S=ArrayStack()                  #menciptakan tumpukan S

kalimat = input('Masukkan satu kalimat : ')
panjang=len(kalimat)

#masukkan tiap huruf ke stack
for c in kalimat:
    S.push(c)

#baca isi stack
x=''
for i in range(panjang):
    x=x+S.pop()

print('Baca terbalik : ')
print(x)
```

Python Interpreter

```
*** Remote Interpreter Reinitialized ***
Masukkan satu kalimat : Selamat Pagi
Baca terbalik :
igaP tamaleS
>>>
*** Remote Interpreter Reinitialized ***
Masukkan satu kalimat : Ini program tumpukan
Baca terbalik :
nakupmut margorp inI
>>> |
```

Aplikasi membalik isi file

```
#reverse_file.py => membalikkan isi file teks
from ArrayStack import *

def reverse_file(filename):
    S=ArrayStack()
    original = open(filename)
    print("Ini isi file aslinya:")
    for line in original:
        print(line.rstrip('\n'))
        S.push(line.rstrip('\n'))
    original.close()

    outfile = "D:\\USER\\Python\\reverse.txt"
    output = open(outfile, 'w')
    while not S.is_empty():
        output.write(S.pop() + '\n')

    output.close()
```

```
def main():
    filename = input("Masukkan nama file teks: ")
    reverse_file(filename) # contoh:'D:\\USER\\Python\\emma2.txt')
    print("\n\nIni hasil proses reverse:")
    outfile="D:\\USER\\Python\\reverse.txt"
    reverse = open(outfile)
    for line in reverse:
        print(line.rstrip('\\n'))

main()
```

Hasil proses

The image shows two Microsoft Notepad windows side-by-side. The top window, titled '*emma2.txt - Notepad', contains the first half of a Jane Austen passage. The bottom window, titled 'reverse.txt - Notepad', contains the second half of the same passage, with the text reversed. A cursor is visible in the 'reverse.txt' window at the start of the first sentence.

*emma2.txt - Notepad

File Edit Format View Help

Emma Woodhouse, handsome, clever, and rich, with a comfortable home and happy disposition, seemed to unite some of the best blessings of existence; and had lived nearly twenty-one years in the world with very little to distress or vex her.

She was the youngest of the two daughters of a most affectionate, indulgent father; and had, in consequence of her sister's marriage, been mistress of his house from a very early period. Her mother had died too long ago for her to have more than an indistinct remembrance of her caresses; and her place had been supplied by an excellent woman as governess, who had fallen little short of a mother in affection.|

Ln 12, Col 26

reverse.txt - Notepad

File Edit Format View Help

|bf a mother in affection.
by an excellent woman as governess, who had fallen little short
remembrance of her caresses; and her place had been supplied
had died too long ago for her to have more than an indistinct
been mistress of his house from a very early period. Her mother
indulgent father; and had, in consequence of her sister's marriage,
She was the youngest of the two daughters of a most affectionate,

with very little to distress or vex her.
of existence; and had lived nearly twenty-one years in the world
and happy disposition, seemed to unite some of the best blessings
Emma Woodhouse, handsome, clever, and rich, with a comfortable home

Ln 1, Col 1 | 100% Windows (CRL| UTF-8

Aplikasi: infix ke suffix

- Aplikasi yang lain dari tumpukan (stack) adalah penulisan notasi “polish” dari ekspresi matematis. Ketika compiler akan melaksanakan ekspresi matematis maka notasi yang dikenal (notasi infix) diubah ke notasi polish (atau notasi suffix), misalnya sebagai berikut:

infix : $a - b * c$

suffix (polish): $a b c * -$

$(a + b) * (c + d)$

$a b + c d + *$

$a - (b + c) / d + e$

$a b c + d / - e +$

Agar proses penterjemahan notasi infix ke notasi polish dapat dilakukan maka diperlukan tabel atau vektor dari urutan (precedence) simbol operator serta rank-nya, seperti pada tabel berikut ini.

Simbol	Precedence (f)	Rank (r)
+	1	-1
*	2	-1
a, b, c, ...	3	1
#	0	

- Input (masukan) algoritma adalah string infix yang diakhiri oleh tanda '#', output (keluaran) algoritma adalah notasi polish. Secara umum algoritma-nya sebagai berikut:

1. Inisialisasi isi tumpukan S dengan simbol #
2. Baca notasi infix, kemudian ambil satu simbol mulai dari sisi paling kiri
3. Lakukan perulangan hingga langkah ke-6 selama simbol input bukan #
4. Ambil dan keluarkan (pop) simbol pada tumpukan apabila nilai precedence-nya \geq nilai precedence dari input saat ini.
5. Masukkan (push) simbol input ke dalam tumpukan
6. Baca kembali simbol berikutnya dari string infix.

Listing Program: `polish.py`

The screenshot shows a Java IDE interface with a dark theme. The title bar reads "D:\USER\Python\polish.py". The menu bar includes "File", "View", "Project", "Run", "Tools", and "Help". Below the menu is a toolbar with various icons. The main area displays Python code for converting infix expressions to postfix. The code defines a function `f` that maps operators to precedence levels (1 for +/-, 2 for */, and 3 for #). The code is as follows:

```
#polish.py => ubah infix menjadi postfix
from ArrayStack import *

#memeriksa precedence
def f(c):
    if c in ('+', '-'):
        return 1
    else:
        if c in ('*', '/'):
            return 2
        else:
            if c == '#':
                return 0
            else:
                return 3
```

```
·
·     #memeriksa ranking
·     def r(c):
·         if c in ('+', '-'):
·             return -1
·         else:
·             if c in ('*', '/'):
·                 return -1
·             else:
·                 if c == '#':
·                     return 0
·                 else:
·                     return 1
·
·
30     def main():
```

```
30 def main():
    S=ArrayStack()
    S.push('#')
    infix = input('Masukkan notasi infix diakhir # : ')
    current = infix[0]
    polish=''
    idx=0
    rank=0
    while (current != '#'):
        if S.is_empty():
            print('Invalid, stack kosong')
            break

        while (f(current) <= f(S.top())):
            temp = S.pop()
            polish = polish + temp
            rank = rank + r(temp)
            if (rank < 1):
                print('invalid infix !')
                break

        S.push(current)
        idx=idx+1
        current=infix[idx]

    while (S.top() != '#'):
```

```
54 |
|     while (S.top() != '#'):
|         temp = S.pop()
|         polish = polish + temp
|         rank = rank + r(temp)
|         if rank < 1:
|             print('Invalid !!!')
|             break
|
|         if (rank==1):
|             print('Valid-polish : ')
|             print(polish)
|         else:
|             print('Invalid result')
|
|
70 main()
```

Contoh RUN

```
|           while (S.top() != '#'):  
Python Interpreter  
*** Remote Interpreter Reinitialized ***  
Masukkan notasi infix diakhir # : a*b-c/d/e+f#  
Valid-polish :  
ab*c d/e/-f+  
>>>  
*** Remote Interpreter Reinitialized ***  
Masukkan notasi infix diakhir # : 3*x+5*x/y-2*y/x#  
Valid-polish :  
3x*5x*y/+2y*x/-  
>>>
```