Bab 2. Array

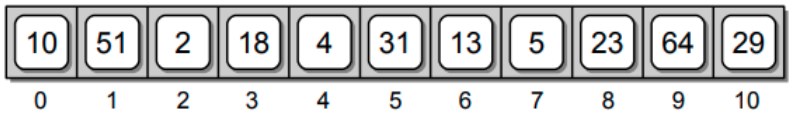
OBJEKTIF:

1. Mahasiswa mampu memahami struktur pada Array.
2. Mahasiswa mampu memahami dan mengimplementasikan ADT pada Array menggunakan *module* Ctypes.
3. Mahasiswa mampu memahami dan mengimplementasikan ADT pada Array Dua Dimensi menggunakan bahasa pemrograman Python.
4. Mahasiswa mampu memahami dan mengimplementasikan ADT Matriks menggunakan operasi-operasi matriks (penjumlahan, pengurangan, perkalian skalar, perkalian matriks, dan matriks transpose).

2.1 Struktur Array

Struktur data paling dasar untuk menyimpan dan mengakses sebuah koleksi dari data adalah array. Array dapat digunakan untuk menyelesaikan berbagai persoalan-persoalan dalam ilmu komputer. Kebanyakan bahasa pemrograman menyediakan array sebagai tipe data primitif dan memungkinkan pembuatan array multi dimensi. Dalam bab ini, kita mengimplementasikan sebuah struktur array untuk array satu dimensi dan lalu menggunakannya untuk mengimplementasi array dua dimensi dan strukur matriks.

Pada tingkat *hardware*, sebagian besar arsitektur komputer menyediakan sebuah mekanisme untuk membuat dan menggunakan array. Array dibentuk oleh rangkaian elemen-elemen yang disimpan dalam lokasi-lokasi memori yang berurutan. Gambar berikut mengilustrasikan contoh sebuah array dengan 11 elemen:



Elemen-elemen individu dalam array dapat diakses dengan indeksnya. Indeks array adalah nomor urut dari elemen-elemen array yang dimulai dari 0.

Mengapa Mempelajari Array?

Anda mungkin menyadari bahwa struktur array sama seperti struktur list Python. Namun keduanya memiliki beberapa perbedaan. Array dan list sama-sama dibentuk dari rangkaian elemen-elemen dan setiap elemennya dapat diakses dengan posisinya (indeksnya). Tetapi terdapat dua perbedaan antara array dan list. Pertama, array mempunyai operasi-operasi terbatas, yang umumnya termasuk pembuatan array, membaca nilai dari elemen spesifik, dan menuliskan nilai untuk elemen tertentu. list, sebaliknya, menyediakan jauh lebih banyak operasi-operasi untuk bekerja dengan isi dari list. Perbedaan kedua, list dapat membesar dan mengecil saat eksekusi penambahan elemen atau pengurangan elemen, sementara pada array, ukuran dari array tidak dapat diubah setelah array tersebut dibuat.

Array dan list mempunyai kegunaannya masing-masing. Terdapat banyak persoalan yang hanya membutuhkan pengunaan array dasar yang dimana banyaknya elemen-elemen telah diketahui sebelumnya dan operasi-operasi fleksibel yang terdapat dalam list tidak dibutuhkan. Array cocok digunakan untuk persoalan-persoalan yang banyak maksimum elemenya diketahui di awal, dimana list adalah pilihan yang lebih baik jika ukuran dari barisan harus dapat berubah

setelah dibuat. List mereservasi ruang penyimpanan pada memori yang lebih besar daripada yang dibutuhkan untuk menyimpan elemen-elemennya. Ruang memori ekstra ini dapat berjumlah sampai dua kali lipat dari ruang yang diperlukan. Ruang memori ekstra ini diperlukan untuk kemudahan ekspansi saat elemen baru ditambahkan ke list. Tetapi ruang memori ekstra ini akan sia-sia jika list digunakan untuk menyimpan elemen-elemen yang banyaknya tetap. Sebagai contoh, misalkan list dengan 25 elemen dapat mereservasi ruang memori sampai dengan ruang memori yang seharusnya cukup untuk 50 elemen.

2.2 ADT Array

Struktur array biasanya terdapat di sebagian besar bahasa pemrograman sebagai tipe primitif, namun Python hanya menyediakan struktur list untuk membuat barisan data. Kita dapat mendefinisikan ADT Array untuk merepresentasikan array satu dimensi yang bekerja sama seperti array pada bahasa pemrograman lain.

Definisi ADT Array

ADT Array adalah koleksi dari elemen-elemen bersebelahan yang setiap elemen individunya diidentifikasi dengan indeks integer unik dimulai dari nol. Setelah array dibuat, ukurannya tidak dapat diubah. Array mendukung operasi-operasi berikut:

- `Array(size)` : Membuat array satu dimensi berisi elemen sebanyak `size` dengan setiap elemennya diinisialisasi dengan `None`. `size` harus lebih besar dari nol.
- `length()` : Mengembalikan panjang array atau banyaknya elemen-elemen dalam array. Diakses melalui fungsi `len()`.
- `getitem(indeks)` : Mengembalikan nilai yang disimpan dalam array pada posisi elemen `indeks`. Diakses menggunakan notasi *subscript*: `arr[indeks]`
- `setitem(indeks, nilai)` : Memodifikasi nilai elemen array pada posisi `indeks` menjadi `nilai`. `indeks` harus dalam rentang yang valid. Diakses menggunakan notasi *subscript*: `arr[indeks] = nilai`.
- `clear(nilai)` : Membersihkan array dengan menetapkan setiap elemen ke `nilai`.
- `iterator()` : Membuat dan mengembalikan sebuah iterator yang dapat digunakan untuk men-traverse elemen-elemen dalam array. Diakses menggunakan loop `for`.

Program berikut memberikan contoh penggunaan ADT Array:

```
# Memanggil constructor Array(size) untuk membuat ADT array
daftarNilai = Array(5)

# Operasi setitem(index) untuk menetapkan nilai-nilai elemen dengan notasi
subscript.
daftarNilai[0] = 90
daftarNilai[1] = 78
daftarNilai[2] = 89
daftarNilai[3] = 67
daftarNilai[4] = 73

# Operasi length() untuk mendapatkan panjang array.
# Diakses menggunakan fungsi len().
print('Panjang array: ', end='')
print(len(daftarNilai))    # Mencetak Panjang array: 5

# Operasi getitem(index) untuk mengakses nilai dari elemen pada index.
# Diakses menggunakan notasi subscript.
print(daftarNilai[1])    # Mencetak 78 yang merupakan nilai elemen indeks ke-1
```

```
# Operasi iterator() untuk meng-traverse array dan mencetak nilai yang ada di
array.'
# Diakses dengan loop for.
# Mencetak:
# 90 78 89 67 73
print('Nilai array: ')
for nilai in daftarNilai:
    print(nilai, end=' ')
print()

# Operasi clear(nilai) untuk membersihkan elemen-elemen dengan sebuah nilai.
daftarNilai.clear(10)

# Cetak nilai masing-masing elemen.
# Mencetak:
# 10 10 10 10 10
print('Nilai array setelah clear dengan nilai 5: ')
for nilai in daftarNilai:
    print(nilai, end=' ')
print()
```

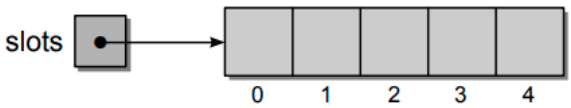
Module `ctypes`

Python adalah bahasa pemrograman yang dibuat dengan bahasa C. Banyak tipe data dan *class* yang tersedia dalam Python sebenarnya diimplementasikan menggunakan tipe data terkait dari bahasa C. Python tidak menyediakan struktur array sebagai bagian dari bahasa Python sendiri, namun Python menyertakan *module* `ctypes` sebagai bagian dari *Python Standard Library*. *Module* ini menyediakan akses ke tipe-tipe data yang tersedia dalam bahasa C, termasuk struktur array dan berbagai fungsionalitas yang ada di *library* C. Kita akan menggunakan *module* `ctypes` untuk mengimplementasikan *abstract data type* (ADT) Array.

Module `ctypes` menyediakan suatu teknik untuk membuat array *low-level* (tingkat *hardware*) yang dapat menyimpan referensi ke *object* Python. Potongan kode berikut:

```
import ctypes

CArray = ctypes.py_object * 5
slots = CArray()
```

membuat sebuah *array low-level* bernama `slots` yang berisi lima elemen. Gambar berikut mengilustrasikan *array low-level* `slots` yang dibuat setelah kode di atas dieksekusi:



Array low-level yang dibuat dengan *module* `ctypes` harus diinisialisasi sebelum dapat digunakan. Kita menginisialisasinya dengan memberikan sebuah nilai ke masing-masing elemennya. Kita dapat memberikan nilai ke masing-masing elemen dengan menggunakan notasi *subscript* (notasi pengindeksan). Nilai apa saja dapat digunakan untuk menginisialisasi elemen, tapi pilihan yang logis adalah dengan menugaskan `None` ke setiap elemen. *Keyword* `None` pada Python berarti *null*. Jika variabel ditugaskan dengan `None` maka variabel adalah variabel kosong.

Kode berikut adalah kode yang digunakan untuk menginisialisasi setiap elemen pada array *low-level* dengan `None`:

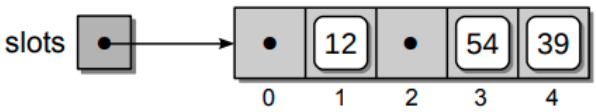
```
for i in range(5):
    slots[i] = None
```

Pada kode di atas kita menggunakan angka 5 dengan fungsi `range()` untuk menandakan banyaknya elemen yang diinisialisasi. Ini diperlukan karena array *low-level* tidak memberikan cara untuk mendapatkan ukuran array.

Kita dapat menugaskan elemen-elemen dalam array *low-level* dengan nilai-nilai dari tipe data apapun. Kita menugaskan elemen dari array *low-level* menggunakan notasi pengindeksan seperti yang kita gunakan dalam `list`. Sebagai contoh, kode berikut menugaskan `integer` ke tiga elemen dari `array`:

```
slots[1] = 12
slots[3] = 54
slots[4] = 39
```

Gambar berikut mengilustrasikan array *low level* setelah kode di atas dieksekusi:



Operasi pengindeksan juga dapat digunakan untuk mengakses nilai dari elemen array *low-level*. Sebagai contoh, kode berikut:

```
print(slots[1])
```

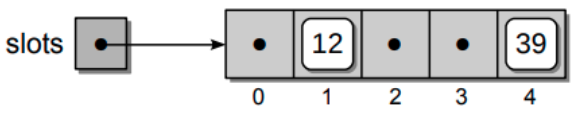
akan memberikan *output*:

```
12
```

Operasi-operasi yang terdapat dalam array *low-level* hanya memungkinkan untuk menetapkan dan mengakses elemen. Untuk menghapus elemen dalam array, kita menugaskan elemen tersebut ke `None`. Sebagai contoh, misalkan kita menghapus nilai 54 dari array:

```
slots[3] = None
```

Kode di atas merubah array `slots` menjadi seperti berikut:



Ukuran dari array *low-level* tidak dapat diubah, menghapus sebuah elemen dari array tidak mempunyai efek ke ukuran dari array. Array *low-level* tidak mempunyai operasi-operasi seperti `append()` atau `pop()` yang terdapat dalam list.

Mengimplementasikan ADT Array

Untuk mengimplementasikan ADT Array kita menggunakan array *low-level* dari *module* `ctypes` dan mengimplementasikannya ke *class* bernama `Array`:

```
import ctypes

class Array:
    # ...
```

`# ...` pada kode di atas berarti terdapat kode di bawahnya.

Berikutnya kita perlu menuliskan kode implementasi untuk *method-method* dari operasi-operasi yang dimiliki ADT Array.

Constructor `Array(size)`

Constructor menangani pembuatan dan inisialisasi array dengan menggunakan teknik yang kita bahas sebelumnya. *Constructor* ADT Array menerima sebuah argumen berupa ukuran dari array yang ingin dibuat. Misalkan, *statement* berikut:

```
arr = Array(10)
```

membuat sebuah array dengan ukuran 10 elemen. Kita harus memastikan nilai argumen ukuran array yang diberikan lebih besar daripada 0. Oleh karena ini, kita menuliskan kondisional pada *constructor* seperti berikut:

```
def __init__(self, size):
    if (size <= 0):
        raise ValueError('Array harus mempunyai ukuran > 0')
    else:
        # Inisialisasi field-field
        # ...
```

Statement yang kita tuliskan dalam *statement* `if` pada baris 3 adalah *statement* yang meng-*raise* sebuah eksepsi `ValueError` jika pengguna memberikan nilai argumen bernilai <= 0. Misalkan, kode berikut:

```
arr = Array(-1)
```

akan menghasilkan sebuah eksepsi `ValueError`.

Pada klausa `else` dari kondisional di atas, yang berarti pengguna memasukkan argumen `size` yang benar, kita menuliskan kode-kode untuk mendefinisikan *field-field*. Kita memerlukan dua *field* data untuk implementasi ADT Array: *field* pertama untuk menyimpan banyaknya elemen yang dialokasikan untuk array tersebut dan *field* kedua untuk menyimpan sebuah referensi ke struktur array yang dibuat menggunakan *module* `ctypes`.

Kita menamakan *field* pertama kita dengan `_size` dan kita menginisialisasi *field* ini dengan menugaskannya dengan nilai argumen yang diberikan, seperti berikut:

```
self._size = size
```

Lalu untuk *field* kedua yang menyimpan sebuah referensi ke struktur `array` yang dibuat menggunakan *module* `ctypes` kita namakan dengan `_refArray`. Kita menginisialisasi *field* ini dengan membuat struktur `array` dengan elemen sebanyak nilai `size` yang diberikan menggunakan *module* `ctypes`, seperti terlihat pada kode berikut:

```
CArray = ctypes.py_object * size
self._refArray = CArray()
```

Namun, seperti yang telah kita bahas sebelumnya, kita harus menginisialisasi elemen-elemen dalam struktur `array` dengan `None`. Untuk melakukan ini, kita memanggil *method* `clear`, yang akan kita implementasikan berikutnya, dengan memberikan argumen `None`:

```
self.clear(None)
```

Sehingga kita akan mempunyai kode *constructor* ADT Array lengkap seperti berikut:

```
def __init__(self, size):
    if (size <= 0):
        raise ValueError("Array harus mempunyai ukuran > 0")
    else:
        # Inisialiasi field-field.
        self._size = size
        # Buat array low-level menggunakan module ctype.
        CArray = ctypes.py_object * size
        self._isi = CArray()
        # Inisialisasi setiap elemen.
        self.clear(None)
```

Method `length()`

Kita ingin membuat ADT Array dapat bekerja dengan fungsi *built-in* `len()` untuk mendapatkan panjang dari array (banyaknya elemen dalam array). Misalkan, kode berikut:

```
arrContoh = Array(10)
print(len(arrContoh))
```

akan memberikan *output* :

```
10
```

Pemanggilan fungsi `len(arrContoh)` mengembalikan panjang array `arrContoh`. Untuk membuat ADT Array dapat bekerja dengan fungsi `len` kita harus mengimplentasikan *method* spesial bernama `__len__`. *Method* `__len__` harus mengembalikan panjang dari array. Untuk membuat *method* `__len__` mengembalikan panjang dari array kita hanya perlu mengembalikan nilai *field* `_size`. Sehingga implementasi *method* `__len__` dapat dituliskan seperti berikut:

```
def __len__(self):
    return self._size
```

Method `getitem(indeks)`

Elemen individu pada array umumnya diakses menggunakan notasi pengindeksan atau sering disebut juga dengan notasi *subscript*. Misalkan, untuk mendapatkan nilai elemen dengan indeks 3 dari array `arrContoh`, kita menuliskan notasi *subscript* berikut: `arrContoh[3]`.

Pada Python, untuk membuat ADT Array dapat bekerja dengan notasi *subscript* kita harus mengimplementasikan *method* spesial bernama `__getitem__`. *Method* `__getitem__` menerima sebuah argument berupa indeks dari elemen yang ingin diakses dan mengembalikan nilai elemen dengan indeks tersebut.

Header dari *method* `__getitem__` dapat kita tuliskan seperti berikut:

```
def __getitem__(self, indeks):
```

Sekarang yang perlu kita pikirkan adalah bagaimana cara mengembalikan nilai elemen pada indeks dalam struktur array. Seperti yang telah kita lihat sebelumnya, pada array *low-level* kita juga dapat melakukan operasi pengindeksan dengan notasi *subscript*, sehingga kita dapat menuliskan:

```
def __getitem__(self, indeks):
    return self._isi[indeks]
```

Namun kita perlu memastikan pengguna memberikan indeks yang valid, yaitu di rentang 0 sampai dengan satu indeks sebelum panjang array. Sebagai contoh, misalkan panjang array adalah 5, maka indeks yang valid berada dalam rentang 0 sampai dengan 4.

Untuk memastikan pengguna memberikan indeks yang valid kita menuliskan sebuah kondisi yang meng-*raise* sebuah eksepsi `IndexError` ketika indeks yang diberikan tidak berada dalam rentang yang valid. Rentang indeks yang tidak valid adalah ketika indeks < 0 atau indeks >= panjang array. Panjang array kita dapatkan dengan memanggil fungsi `len()` dengan argumen *object* `array` `self`.

Sehingga, kode lengkap implementasi *method* `__getitem__` dapat dituliskan seperti berikut:

```
def __getitem__(self, indeks):
    if (indeks < 0 or indeks >= len(self)):
        raise IndexError()
    else:
        return self._isi[indeks]
```

Pada kode di atas, kita memanggil fungsi `len()` dengan argumen *object* array `self` untuk mendapatkan panjang dari *object* array.

Method `setitem(indeks, nilai)`

Kita ingin membuat nilai-nilai elemen dalam ADT Array kita dapat ditetapkan dengan menggunakan notasi *subscript*, seperti contoh berikut:

```
arrContoh[0] = 12
arrContoh[2] = 10
```

Statement baris pertama, menetapkan nilai elemen dengan indeks 0 dengan 12 dan *statement* baris kedua, menetapkan nilai elemen dengan indeks 2 dengan 10.

Untuk membuat ADT Array kita dapat ditetapkan nilai-nilai elemennya dengan notasi *subscript* kita harus mengimplementasikan *method* spesial bernama `__setitem__`. *Method* ini menerima dua argumen, argumen pertama berupa indeks dari elemen yang ingin ditetapkan nilainya dan argumen kedua adalah nilai yang ingin ditetapkan ke elemen tersebut. Sehingga *header method* `__setitem__` dapat dituliskan seperti berikut:

```
def __setitem__(self, indeks, nilai):
```

Pada *body* dari *method* `__setitem__`, kita harus menuliskan kode-kode yang menetapkan `nilai` pada elemen dengan indeks sesuai dengan `indeks`. Seperti yang telah kita lihat sebelumnya pada pembahasan *module* `ctypes`, kita juga dapat menggunakan notasi *subscript* pada array *low-level*. Di dalam *body method* `__set_item__` kita dapat menggunakan *statement* berikut untuk menetapkan elemen pada indeks yang diberikan:

```
self._isi[indeks] = nilai
```

Namun, seperti pada *method* `__getitem__`, kita juga harus memastikan indeks yang diberikan berada dalam rentang yang valid, yaitu 0 sampai dengan panjang array. Sehinga, kode lengkap untuk *method* `__setitem__` dapat dituliskan seperti berikut:

```
def __setitem__(self, index, nilai):
    if (index <= 0 or index > len(self)):
        raise IndexError()
    else:
        self._isi[index] = nilai
```

Method `clear(nilai)`

Method `clear(nilai)` digunakan untuk membersihkan nilai-nilai dari semua elemen pada `array` dengan menetapkan sebuah nilai ke semua elemen pada `array`. *Method* ini menerima sebuah argumen berupa sebuah nilai yang ingin ditetapkan ke semua elemen pada array. *Header* dari *method* `clear(nilai)` dapat kita tuliskan seperti berikut:

```
def clear(self, nilai):
```

Pada *body method*, kita dapat menggunakan sebuah *loop* yang mengiterasi setiap elemen pada array dan menetapkan nilai setiap elemen dengan nilai. Sehingga kode lengkap *method* `clear` dapat dituliskan seperti berikut:

```
def clear(self, nilai):
    for i in range(len(self)):
        self._isi[i] = nilai
```

Method `iterator()`

Untuk membuat `array` kita dapat di-*traverse* atau diiterasi menggunakan *loop*, kita harus mengimplementasikan sebuah iterator. Iterator diimplementasi dengan menambahkan *method* `__iter__` pada *class* yang ingin diiterasi dan menuliskan sebuah *class* iterator. *Class* iterator untuk ADT Array mirip dengan *class* iterator yang kita tuliskan untuk ADT *Bag* pada bagian sebelumnya. Kita menamakan *class* iterator ini dengan `_ArrayIterator` dengan definisi seperti berikut:

```
class _ArrayIterator:
    def __init__(self, strkArray):
        self._refArr = strkArray
        self._curIndeks = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self._curIndeks < len(self._refArr):
            entry = self._refArray[self._curIndeks]
            self._curIndeks += 1
            return entry
        else:
            raise StopIteration
```

Class `_ArrayIterator` memiliki dua *field*: *field* pertama, `_refArr`, adalah referensi ke array *low level* dan *field* kedua, `_curIndeks`, adalah variabel indeks *loop* yang digunakan saat iterasi terhadap array *low level*. *Field* `_curIndeks` diinisialisasi ke 0 untuk membuat iterasi pertama dari *loop* dimulai dari indeks ke-0 dari `array`.

Method `__iter__` pada *class* `_ArrayIterator` hanya perlu mengembalikan objek dari *class* ini. Lalu pada *method* `__next__` pada *class* `_ArrayIterator` kita mengembalikan nilai elemen pada indeks `_curIndeks` menggunakan notasi *subscript* terhadap *field* pertama kemudian menginkrementasikan `_curIndeks`. Dan jika `curIndeks` sudah melebihi panjang dari array, kita meng-*raise* eksepsi `StopIteration`.

Pada *class* Array, *method* `__iter__` dituliskan seperti berikut:

```
def __iter__(self):
    return _ArrayIterator(self._isi)
```

Kode Lengkap Impelementasi ADT Array

Berikut adalah kode lengkap implementasi ADT Array:

Module *`array1d.py`*

```
import ctypes

# Implementasi ADT Array menggunakan module ctypes
class Array:
    # Buat array dengan ukuran size.
    def __init__(self, size):
        if (size <= 0):
            raise ValueError("Array harus mempunyai ukuran > 0")
        else:
            # Inisialiasi field-field
            self._size = size
            # Buat struktur array menggunakan module ctype.
            CArray = ctypes.py_object * size
            self._isi = CArray()
            # Inisialisasi setiap elemen.
            self.clear(None)
```

```
    # Mengembalikan ukuran dari array.
    # Diakses menggunakan fungsi len().
    def __len__(self):
        return self._size

    # Method getitem mendapatkan nilai dari elemen dengan indeks tertentu
    # menggunakan notasi subscript.
    # Contoh: arr[5] mengembalikan nilai elemen pada indeks ke-5.
    # Method ini menerima satu argument:
    # indeks dari elemen yang ingin diakses nilainya.
    # Method ini mengembalikan nilai elemen dari indeks.
    def __getitem__(self, index):
        if (index < 0 or index >= len(self)):
            raise IndexError('Indeks harus dalam rentang yang valid.')
        else:
            return self._isi[index]

    # Method setitem(nilai) menetapkan nilai elemen
    # menggunakan notasi subscript.
    # Contoh: arr[4] = 11.
    # Method ini menerima dua argument:
    # (1) indeks - indeks dari elemen yang ingin ditetapkan nilainya; dan
    # (2) nilai - nilai yang ingin ditugaskan ke elemen tersebut
    # Method ini tidak mengembalikan nilai.
    def __setitem__(self, index, nilai):
        if (index < 0 or index >= len(self)):
            raise IndexError('Indeks harus dalam rentang yang valid.')
        else:
            self._isi[index] = nilai

    # Method clear(nilai) membersihkan array dengan
    # mengganti nilai setiap elemen array dengan nilai yang diberikan.
    # Method ini menerima satu argument:
    # nilai yang ingin ditetepkan ke semua elemen.
    # Method ini tidak mengembalikan nilai.
    def clear(self, nilai):
        for i in range(len(self)):
            self._isi[i] = nilai

    # Method iterator() meng-traverse nilai-nilai elemen pada array.
    # Diakses menggunakan loop for.
    def __iter__(self):
        return _ArrayIterator(self._isi)

# Class iterator untuk ADT Array
class _ArrayIterator:
    def __init__(self, iniArray):
        self._refArray = iniArray
        self._curIndex = 0

    def __iter__(self):
        return self

    def __next__(self):
        if self._curIndex < len(self._refArray):
            entry = self._refArray[self._curIndex]
            self._curIndex += 1
            return entry
```

```
        else:
            raise StopIteration
```

Menguji ADT Array

Script berikut dapat digunakan untuk menguji implementasi array satu dimensi:

```
from array1d import Array

# Memanggil constructor Array(size) untuk membuat ADT array
daftarNilai = Array(5)

# Operasi setitem(index) untuk menetapkan nilai-nilai elemen dengan notasi
subscript.
daftarNilai[0] = 90
daftarNilai[1] = 78
daftarNilai[2] = 89
daftarNilai[3] = 67
daftarNilai[4] = 73

# Operasi length() untuk mendapatkan panjang array.
# Diakses menggunakan fungsi len().
print('Panjang array: ', end='')
print(len(daftarNilai))    # Mencetak Panjang array: 5

# Operasi getitem(index) untuk mengakses nilai dari elemen pada index.
# Diakses menggunakan notasi subscript.
print(daftarNilai[1])    # Mencetak 78 yang merupakan nilai elemen indeks ke-1

# Operasi iterator() untuk meng-traverse array dan mencetak nilai yang ada di
array.'
# Diakses dengan loop for.
# Mencetak:
# 90 78 89 67 73
print('Nilai array: ')
for nilai in daftarNilai:
    print(nilai, end=' ')
print()

# Operasi clear(nilai) untuk membersihkan elemen-elemen dengan sebuah nilai.
daftarNilai.clear(10)

# Cetak nilai masing-masing elemen.
# Mencetak:
# 10 10 10 10 10
print('Nilai array setelah clear dengan nilai 5: ')
for nilai in daftarNilai:
    print(nilai, end=' ')
print()

# Mencoba mengakses elemen dengan indeks tidak valid
try:
    print(daftarNilai[-1])
except IndexError:
    print('Indeks -1 bukan indeks yang valid.')

# Mencoba menetapkan elemen dengan indeks yang tidak valid
```

```
try:
    daftarNilai[5]
except IndexError:
    print('Array tidak mempunyai indeks 5.')
```

Output dari *script* di atas:

```
Panjang array: 5
78
Nilai array:
90 78 89 67 73
Nilai array setelah clear dengan nilai 5:
10 10 10 10 10
Indeks -1 bukan indeks yang valid.
Array tidak mempunyai indeks 5.
```

2.3 ADT Array Dua Dimensi

Array dua dimensi (array 2-D) sangat umum di pemrograman komputer. Array dua dimensi biasanya digunakan untuk menyelesaikan persoalan-persoalan yang membutuhkan data diorganisasi ke dalam baris dan kolom. Pada bagian ini kita akan mengimplementasikan ADT Array2D.

Definisi ADT Array2D

Array2D atau array dua dimensi terdiri dari sebuah koleksi elemen-elemen yang diorganisasi dalam baris dan kolom. Elemen individu direferensikan dengan indeks baris dan kolom `(b, k)`, keduanya dimulai dari 0. Operasi-operasi yang dapat dilakukan terhadap ADT Array2D:

- `Array2D(bykBaris, bykKolom)`: Membuat array dua dimensi dengan jumlah baris sebanyak `bykBaris` dan jumlah kolom sebanyak `bykKolom`. Semua elemen diinisialisasi ke `None`.
- `bykBaris()`: Mengembalikan banyak baris dalam array dua dimensi.
- `bykKolom()`: Mengembalikan banyak kolom dalam array dua dimensi.
- `clear(nilai)`: Membersihkan array dua dimensi dengan menetapkan setiap elemen dengan `nilai`.
- `getitem(brs, klm)`: Mengembalikan nilai dalam elemen dengan posisi ditandakan oleh pasangan indeks (`brs`, `klm`). `brs` dan `klm` harus berada dalam rentang yang valid. Diakses menggunakan notasi *subscript*: `y = x[1, 2]`
- `setitem(brs, klm, nilai)`: Memodifikasi isi dari elemen dengan pasangan indeks `(brs, klm)` dengan `nilai`. Kedua indeks harus berada dalam rentang yang valid. Diakses menggunakan notasi *subscript*: `x[0, 3] = nilai`.

Untuk mengilustrasikan penggunaan array 2-D, misalkan kita mempunyai sebuah koleksi dari nilai-nilai *exam* dalam sebuah *file* teks yang dikelompokkan berdasarkan kelompok mahasiswa yang perlu diproses.

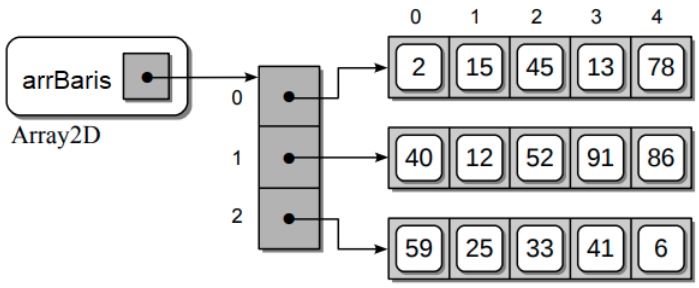
Program berikut mencontohkan penggunaan array 2-D:

```
from array2d import Array2D

import random

# Membuat array 2-d dengan banyak baris 5 dan banyak kolom 5
```

```
daftarNilai = Array2D(5, 5)

# Operasi setitem(i1, i2, nilai)
# Menetapkan nilai pada elemen-elemen dalam array 2-d dengan
# integer random antara 0 s.d 100.
for i in range(daftarNilai.bykBaris()):
    for j in range(daftarNilai.bykKolom()):
        daftarNilai[i, j] = random.randint(0, 100)

# Operasi getitem(i1, i2)
# Mengakses dan mencetak elemen-elemen dalam array 2-d
for i in range(daftarNilai.bykBaris()):
    for j in range(daftarNilai.bykKolom()):
        print(f'{daftarNilai[i, j]:3d}', end = ' ')
    print('\n')

# Method bykBaris()
print('Banyak baris: ', end='')
print(daftarNilai.bykBaris())

# Method bykKolom()
print('Banyak kolom: ', end='')
print(daftarNilai.bykKolom())

# Operasi clear(nilai)
# Membersihkan elemen-elemen dengan menetapkan semua elemen
# dengan nilai 10.
daftarNilai.clear(10)

# Cetak elemen-elemen setelah operasi clear
print()
print('Isi array setelah clear dengan 10:')
for i in range(daftarNilai.bykBaris()):
    for j in range(daftarNilai.bykKolom()):
        print(f'{daftarNilai[i, j]:3d}', end = ' ')
    print('\n')
```

Mengimplementasikan ADT Array 2D

Untuk mengimplementasikan array dua dimensi, kita dapat menggunakan pendekatan array dari array. Kita menyimpan setiap baris dari array 2-D dalam satu array 1-D. Lalu, array 1-D lain digunakan untuk menyimpan referensi-referensi ke setiap array yang digunakan untuk menyimpan elemen-elemen baris. Gambar berikut mengilustrasikan array 2-D yang akan kita implementasikan:



Constructor `Array2D(bykBaris, bykKolom)`

Constructor dari array 2-D dapat dituliskan seperti berikut:

```
def __init__(self, bykBaris, bykKolom):
    # Buat array 1-D untuk menyimpan array referensi untuk setiap baris.
    self._arrBaris = Array(bykBaris)

    # Buat array 1-D untuk setiap baris dari array 2-D.
    for i in range(bykBaris):
        self._arrBaris[i] = Array(bykKolom)
```

Constructor Array2D menerima dua argumen: argumen pertama yaitu `bykBaris` untuk menyatakan banyak baris dari array 2-D yang ingin dibuat dan argumen kedua yaitu `bykKolom` untuk menyatakan banyak kolom dari array 2-D yang ingin dibuat. Contoh *statement* yang membuat array 2-D ini:

```
daftarNilai = Array2D(3, 5)
```

Field `_arrRef` adalah *field* yang kita gunakan untuk menyimpan array 1-D yang berisi referensi ke array yang menyimpan elemen-elemen setiap baris. Kita menginisialisasi `_arrRef` dengan Array 1-D yang kita buat sebelumnya. Lalu kita membuat array baris dengan array 1-D sebanyak `banyakBaris` dan menugaskan referensi ke masing-masing array baris tersebut ke elemen-elemen pada `_arrRef` sesuai dengan indeks barisnya.

Method `bykBaris()`

Method `bykBaris()` mengembalikan banyak baris dari array 2-D. Contoh penggunaan dari *method* `bykBaris()` yang akan kita implementasikan adalah sebagai berikut:

```
daftarNilai = Array2D(3, 5)
bykBaris = daftarNilai.bykBaris();
```

Setelah *statement-statement* di atas dieksekusi, variabel `bykBaris` akan menyimpan nilai 3. Ini karena banyak baris dari Array 2-D `daftarNilai` adalah 3.

Untuk mengimplementasikan *method* `bykBaris()` kita hanya perlu mengembalikan panjang dari *field* `_arrBaris`. Kita dapat melakukannya dengan menggunakan fungsi *built-in* `len()` pada *field* `_arrBaris`. Sehingga kode untuk *method* `banyakBaris()` dapat dituliskan seperti berikut:

```
def bykBaris(self):
    return len(self._arrBaris)
```

Method `bykKolom()`

Method `bykKolom()` mengembalikan banyak kolom dari array 2-D. Contoh penggunaan dari *method* `bykKolom()` yang akan kita implementasikan adalah sebagai berikut:

```
daftarNilai = Array2D(3, 5)
bykKolom = daftarNilai.bykBaris();
```

Setelah *statement-statement* di atas dieksekusi, variabel `bykKolom` akan menyimpan nilai 5. Ini karena banyak kolom dari Array 2-D `daftarNilai` adalah 5.

Untuk mengimplementasikan *method* `bykKolom()` kita hanya perlu mamanggil fungsi `len()` dari array 1-D yang direferensikan oleh salah satu elemen pada array *field* `_arrBaris`. Kode implementasi *method* `bykKolom()` dapat dituliskan seperti berikut:

```
def bykKolom(self):
    return len(self._arrBaris[0])
```

Pada kode di atas kita mengembalikan panjang dari array 1-D yang direferensikan oleh elemen ke-0 dari array 1-D `_arrBaris`.

Method `clear(nilai)`

Method `clear(nilai)` membersihkan nilai semua elemen pada array 2-D dengan menetapkan semua elemennya dengan nilai. Untuk mengimplementasikan *method* ini, kita cukup mengiterasi setiap array dalam `_arrRef` dan memanggil *method* `clear()` pada array 1-D setiap iterasinya. Kode implementasi *method* `clear(nilai)` dapat dituliskan seperti berikut:

```
def clear(self, nilai):
    for baris in range(self.bykBaris()):
        self._arrBaris[baris].clear(nilai)
```

Method `getitem(i1, i2)`

Kita ingin struktur array dua dimensi kita dapat diakses menggunakan notasi *subscript* dengan indeks baris dan kolom. Misalkan, ekspresi berikut:

```
arr2D[2, 3]
```

Digunakan untuk mendapatkan nilai dari elemen pada baris ke-2 dan kolom ke-3 dari array dua dimensi `arr2D`. Ketika notasi *subscript* menggunakan dua indeks digunakan untuk mendapatkan nilai elemen, argumen yang diberikan ke *method* `__getitem__` adalah berupa tuple. Pada contoh, `arr2D[2, 3]` memanggil *method* `__getitem__` dengan argumen berupa tuple `(2, 3)`. Sehingga kita menuliskan *header* dari *method* `__getitem__` seperti berikut:

```
def __getitem__(self, indeksTuple):
```

Untuk mendapatkan baris dan kolom dari `indeksTuple`, kita dapat menggunakan notasi *subscript* seperti berikut:

```
brs = indeksTuple[0]
klm = indeksTuple[1]
```

Setelah mendapatkan baris dan kolom dari elemen yang diakses, kita dapat menggunakan `brs` untuk mendapatkan array 1-D tempat elemen itu berada dan menggunakan `klm` untuk mendapatkan elemen yang diakses di array 1-D tersebut.

```
arr1D = self._arrBaris[brs]
elm = arr1D[klm]
```

Kita perlu menambahkan kondisional dimana indeks dari notasi *subscript* tidak sama dengan dua dan juga kondisional dimana indeks di luar rentang yang valid. Sehingga kode implementasi *method* `__getitem__` dituliskan seperti berikut:

```
def __getitem__(self, indeksTuple):
    # jika argumen yang diberikan bukan dua indeks (brs, klm)
    # raise ValueError
    if len(indeksTuple) != 2:
        raise IndexError('Indeks tidak valid.')
    else:
        brs = indeksTuple[0]
        klm = indeksTuple[1]

        # Jika brs dan klm tidak valid raise IndexError
        if brs < 0 or brs >= self.bykBaris() \
            or klm < 0 or klm >= self.bykKolom():
            raise IndexError('Indeks tidak valid.')
        else:
            arr1D = self._arrBaris[brs]
            elm = arr1D[klm]
            return elm
```

Method `setitem(i1, i2, nilai)`

Kode untuk *method* `__setitem__` mirip dengan kode untuk `__getitem__` hanya pada *method* ini kita menetapkan nilai untuk elemen dengan indeks (`brs`, `klm`) yang diberikan. Berikut adalah kode untuk *method* `__setitem__`:

```
def __setitem__(self, indexTuple, nilai):
    if len(indexTuple) != 2:
        raise IndexError('Indeks tidak valid.')
    else:
        brs = indexTuple[0]
        klm = indexTuple[1]
        # Jika brs dan klm tidak valid raise IndexError
        if brs < 0 or brs >= self.bykBaris() \
            or klm < 0 or klm >= self.bykKolom():
            raise IndexError('Indeks tidak valid.')
        else:
            arr1D = self._arrBaris[brs]
            arr1D[klm] = nilai
```

Kode Lengkap Implementasi Array 2D

Berikut adalah kode lengkap dari class `Array2D` untuk ADT Array 2D:

Module *`array2d.py`*

```
from array1d import Array

# Implementasi ADT Array 2 Dimensi
class Array2D:
    # Buat array 2-D dengan ukuran numBaris x numKolom
    # Mendefinisikan sebuah field berupa array 1-D yang masing-masing
```

```
    # elemennya mereferensikan array 1-D yang berisi nilai-nilai pada kolom-
kolom
    # dari sebuah baris.
    def __init__(self, bykBaris, bykKolom):
        # Buat array 1-D untuk menyimpan array referensi untuk setiap baris.
        self._arrBaris = Array(bykBaris)

        # Buat array 1-D untuk setiap baris dari array 2-D.
        for i in range(bykBaris):
            self._arrBaris[i] = Array(bykKolom)

    # Mengembalikan banyaknya baris dalam array 2-D.
    def bykBaris(self):
        return len(self._arrBaris)

    # Mengembalikan banyaknya kolom dalam array 2-D.
    def bykKolom(self):
        return len(self._arrBaris[0])

    # Membersihkan array dengan menetapkan setiap element dengan nilai yang
diberikan.
    def clear(self, nilai):
        for baris in range(self.bykBaris()):
            self._arrBaris[baris].clear(nilai)

    # Mendapatkan isi dari elemen pada posisi [i, j].
    # Diakses menggunakan notasi subscript: arr2D[i, j]
    def __getitem__(self, indeksTuple):
        # jika argumen yang diberikan bukan dua indeks (brs, klm)
        # raise ValueError
        if len(indeksTuple) != 2:
            raise IndexError('Indeks tidak valid.')
        else:
            brs = indeksTuple[0]
            klm = indeksTuple[1]

            # Jika brs dan klm tidak valid raise IndexError
            if brs < 0 or brs >= self.bykBaris() \
                or klm < 0 or klm >= self.bykKolom():
                raise IndexError('Indeks tidak valid.')
            else:
                arr1D = self._arrBaris[brs]
                elm = arr1D[klm]
                return elm

    # Menetapkan nilai dari elemen pada posisi [i, j].
    # Diakses menggunakan notasi subscript: arr2D[i, j] = nilai.
    def __setitem__(self, indexTuple, nilai):
        if len(indexTuple) != 2:
            raise IndexError('Indeks tidak valid.')
        else:
            brs = indexTuple[0]
            klm = indexTuple[1]
            # Jika brs dan klm tidak valid raise IndexError
            if brs < 0 or brs >= self.bykBaris() \
                or klm < 0 or klm >= self.bykKolom():
                raise IndexError('Indeks tidak valid.')
            else:
```

```
                arr1D = self._arrBaris[brs]
                arr1D[klm] = nilai
```

Menguji Implementasi ADT Array 2D

Kita dapat menggunakan *script* berikut untuk menguji implementasi ADT Array 2D:

```
from array2d import Array2D

import random

# Membuat array 2-d dengan banyak baris 5 dan banyak kolom 5
daftarNilai = Array2D(5, 5)

# Operasi setitem(i1, i2, nilai)
# Menetapkan nilai pada elemen-elemen dalam array 2-d dengan
# integer random antara 0 s.d 100.
for i in range(daftarNilai.bykBaris()):
    for j in range(daftarNilai.bykKolom()):
        daftarNilai[i, j] = random.randint(0, 100)

# Operasi getitem(i1, i2)
# Mengakses dan mencetak elemen-elemen dalam array 2-d
for i in range(daftarNilai.bykBaris()):
    for j in range(daftarNilai.bykKolom()):
        print(f'{daftarNilai[i, j]:3d}', end = ' ')
    print('\n')

# Method bykBaris()
print('Banyak baris: ', end='')
print(daftarNilai.bykBaris())

# Method bykKolom()
print('Banyak kolom: ', end='')
print(daftarNilai.bykKolom())

# Operasi clear(nilai)
# Membersihkan elemen-elemen dengan menetapkan semua elemen
# dengan nilai 10.
daftarNilai.clear(10)

# Cetak elemen-elemen setelah operasi clear
print()
print('Isi array setelah clear dengan 10:')
for i in range(daftarNilai.bykBaris()):
    for j in range(daftarNilai.bykKolom()):
        print(f'{daftarNilai[i, j]:3d}', end = ' ')
    print('\n')

# Uji mengakses elemen menggunakan indeks yang tidak valid pada array 2-D
try:
    print(daftarNilai[4, 7])
except IndexError:
    print('Indeks tidak valid.')

try:
    print(daftarNilai[4, 3, 3])
```

```
except IndexError:
    print('Banyak indeks tidak valid.')
```

Output dari *script* di atas:

```
 95 100   4  60  67

 49  20  74  44  24

 17   1  85  23   9

  4  67  69  71  19

 86   3   0  48  49

Banyak baris: 5
Banyak kolom: 5

Isi array setelah clear dengan 10:
 10  10  10  10  10

 10  10  10  10  10

 10  10  10  10  10

 10  10  10  10  10

 10  10  10  10  10

Indeks tidak valid.
Banyak indeks tidak valid.
```

2.4 ADT Matriks

Dalam matematika, matriks didefiniskan sebagai nilai-nilai numerik yang disusun dalam baris dan kolom. Matriks digunakan dalam banyak aplikasi seperti grafik komputer dan aljabar linear. Pada bagian ini, kita mendefinisikan ADT Matriks dan mengimplementasikannya menggunakan array 2-D yang telah kita buat sebelumnya.

Definisi ADT Matriks

Sebuah matriks adalah koleksi nilai-nilai skalar yang disusun dalam baris dan kolom yang tetap. Elemen-elemen dari matriks dapat diakses dengan menentukan indeks baris dan kolom yang masing-masing dimulai dari 0. Operasi-operasi yang dapat dilakukan terhadap matriks:

- `Matriks(bykBaris, bykKolom)`: Membuat matriks baru dengan banyak baris sama dengan `bykBaris` dan banyak kolom sama dengan `bykKolom`. Setiap elemen diinisialisasi ke 0.
- `bykBaris()`: Mengembalikan banyak baris dalam matriks.
- `bykKolom()`: Mengembalikan banyak kolom dalam matriks.
- `getitem(brs, klm)`: Mengembalikan nilai yang disimpan dalam elemen matriks dengan index (`brs`, `klm`). Indeks (`brs`, `klm`) harus dalam rentang yang valid. Diakses dengan dengan notasi *subscript*: `mtrk[brs, klm]`.
- `setitem(brs, klm, nilai)`: Menetapkan nilai elemen pada indeks (`brs`, `klm`) dengan `nilai`. Diakses dengan notasi *subscript*: `mtrk[brs, klm] = 10`.
- `kaliSkalar(skalar)`: Mengalikan setiap elemen matriks dengan nilai `scalar` yang diberikan. Operasi ini memodifikasi matriks.

- `transpos()` : Mengembalikan sebuah matriks baru yang merupakan matriks transpos dari matriks ini.
- `add(matriksLain)` : Membuat dan mengembalikan matriks baru yang merupakan hasil penjumlahan matriks ini dengan `matriksLain`. Ukuran dari kedua matriks harus sama. Diakses dengan operator `+`. Contoh: `mtrkC = mtrkA + mtrkB`.
- `substract(matriksLain)` : Membuat dan mengembalikan matriks baru yang merupakan hasil pengurangan matriks ini dengan `matriksLain`. Ukuran dari kedua matriks harus sama. Diakses dengan operator `-`. Contoh: `mtrkC = mtrkA - mtrkB`.
- `multiply(matriksLain)` : Membuat dan mengembalikan matriks baru yang merupakan hasil perkalian matriks ini dengan `matriksLain`. Kedua matriks harus mempunyai ukuran yang sesuai dengan yang didefinisikan pada perkalian matriks. Diakses dengan operator `*`. Contoh: `mtrkC = mtrkA * mtrkB`.

Script berikut mencontohkan penggunaan ADT Matriks:

```
# Definisikan matrixA = | 0 1 |
#                       | 2 3 |

# Buat matriks
matrixA = Matriks(2, 2)

# Operasi setitem(brs, klm, nilai) menetapkan elemen matriks
# pada brs dan klm dengan nilai.
# Operasi ini digunakan dengan notasi subscript.
matrixA[0, 0] = 0
matrixA[0, 1] = 1
matrixA[1, 0] = 2
matrixA[1, 1] = 3

# Definisikan matrixB = |  4  5 |
#                       |  6  7 |

# Buat matriks
matrixB = Matriks(2, 2)

# Operasi setitem(brs, klm, nilai) menetapkan elemen matriks
# pada brs dan klm dengan nilai.
# Operasi ini digunakan dengan notasi subscript.
matrixB[0, 0] = 4
matrixB[0, 1] = 5
matrixB[1, 0] = 6
matrixB[1, 1] = 7

# Operasi add(matriksLain) untuk menjumlahkan dua matriks.
# Operasi ini dilakukan dengan operator +.
# matrixC = | 4  6 |
#           | 8 10 |
matrixC = matrixA + matrixB

# Operasi substract(matriksLain) untuk pengurangan dua matriks.
# Operasi ini dilakukan dengan operator -.
# matrixD = | -4 -4 |
#           | -4 -4 |
matrixD = matrixA - matrixB
```

```
# Operasi multiply(matriksLain) untuk mengalikan dua matriks.
# Operasi ini dilakukan dengan operator *.
# matrixE = |  6  7 |
#           | 26 31 |
matrixE = matrixA * matrixB

# Operasi transpos() untuk mendapatkan transpos dari suatu matriks.
# matrixF = | 0 2 |
#           | 1 3 |
matrixF = matrixA.transpos()

# Operasi kaliSkalar(skalar) memodifikasi matriks dengan mengalikan
# setiap elemennya dengan skalar.
# matrix A = |
#            |
matrixA.kaliSkalar(2)
```

Operasi-operasi Matriks

Sebelum kita mengimplementasikan ADT Matriks, kita akan membahas operasi-operasi yang dapat dilakukan terhadap matriks-matriks.

Penjumlahan. Dua buah matriks dengan ukuran ***m x n*** dapat dijumlahkan dengan menjumlahkan setiap elemen dari kedua matriks yang berindeks sama. Berikut adalah contoh operasi penjumlahan dua matriks:

Pengurangan. Dua buah matriks dengan ukuran ***m×n*** dapat dikurangkan dengan mengurangkan setiap elemen dari matriks ruas kiri dengan elemen berindeks sama dari matriks ruas kanan. Berikut adalah contoh operasi pengurangan dua matriks:

Perkalian Skalar. Sebuah matriks dapat dikalikan dengan sebuah skalar *c* dengan mengalikan skalar *c* dengan setiap elemen dalam matriks. Berikut adalah contoh perkalian skalar:

Perkalian Matriks. Perkalian matriks hanya bisa dilakukan untuk dua matriks dimana banyak kolom dari matriks pada ruas kiri sama dengan banyak baris dari matriks ruas kanan. Hasil perkalian dua buah matriks adalah matriks baru dengan banyak baris sama seperti matriks ruas kiri dan banyak kolom sama seperti matriks ruas kanan. Dengan kata lain, perkalian matriks ukuran ***m×n*** dan matriks ukuran ***n×p*** menghasilkan matriks ukuran ***m×p***.

Setiap elemen dari matriks hasil perkalian adalah jumlah dari perkalian elemen sebuah baris dari matriks ruas kiri dengan sebuah kolom dari matriks ruas kanan. Pada contoh di bawah, baris dan kolom untuk mendapatkan elemen (0, 0) dari matriks hasil perkalian di-*highlight* dengan warna abu-abu.

Melihat perkalian matriks berdasarkan indeks elemen dapat membantu untuk lebih memahami operasi perkalian matriks. Misalkan dua matriks ***A*** dan ***B*** seperti berikut:

Misalkan hasil kali matriks ***A*** dan matriks ***B*** menghasilkan matriks ***C***, maka perhitungan elemen individu dari matriks ***C*** adalah sebagai berikut:

Sehingga,

Transpos. Diberikan matriks ***A*** dengan ukuran ***m×n***, matriks transpose ***AT*** didapatkan dengan menukar baris dan kolom dari matriks A. Contoh operasi transpose dapat dilihat pada contoh berikut:

Mengimplementasikan ADT Matriks

Kita mengimplementasikan ADT Matriks menggunakan array dua dimensi yang sebelumnya telah kita buat.

Constructor `Matriks(bykBaris, bykKolom)`

Class `Matriks` hanya membutuhkan satu *field* data untuk menyimpan array 2-D. Setelah membuat array 2-D, elemen-elemennya harus diinisialisasi ke 0. Sehingga *method constructor* dari `Matriks` dapat dituliskan seperti berikut:

```
def __init__(self, bykBaris, bykKolom):
    self._arr2D = Array2D(bykBaris, bykKolom)
    self._arr2D.clear(0)
```

Method `bykBaris()`

Method `bykBaris()` mengembalikan banyak baris dari matriks. Pada *method* ini kita hanya perlu memanggil *method* `bykBaris()` dari array 2-D dan mengembalikan nilainya:

```
def bykBaris():
    return self._arr2D.bykBaris()
```

Method `bykKolom()`

Method `bykKolom()` mengembalikan banyak kolom dari matriks. Kita hanya perlu memanggil *method* `bykKolom()` dari array 2-D dan mengembalikan nilainya:

```
def bykKolom():
    return self._arr2D.bykKolom()
```

Method `getitem(brs, klm)`

Untuk mengimplementasikan *method* `__getitem__` kita hanya perlu mengembalikan nilai elemen dalam array 2-D menggunakan notasi *subscript*. Sehingga, *method* `__getitem__` untuk struktur matriks dapat dituliskan seperti berikut:

```
def __getitem__(self, indexTuple):
    return self._arr2D[indexTuple[0], indexTuple[1]]
```

Kita tidak perlu menuliskan kondisi untuk `(brs, klm)` yang valid karena kondisi ini sudah ditangani oleh *method* `getitem()` dari array 2-D.

Method `setitem(brs, klm, nilai)`

Sama seperti pada implementasi *method* `__getitem__`, untuk mengimplementasikan *method* `__setitem__` kita hanya perlu menggunakan notasi *subscript* untuk menetapkan nilai pada array 2-D. Sehingga, *method* `__setitem__` untuk struktur matriks dapat dituliskan seperti berikut:

```
def __setitem__(self, indexTuple, nilai):
    self._arr2D[indexTuple[0], indexTuple[1]] = nilai
```

Kita tidak perlu menuliskan kondisi untuk `(brs, klm)` yang valid karena kondisi ini sudah ditangani oleh *method* `setitem()` dari array 2-D.

Method `kaliSkalar(skalar)`

Method `kaliSkalar(skalar)` memodifikasi elemen-elemen pada objek matriks dengan mengalikan setiap elemen dalam matriks dengan `skalar`. Untuk mengalikan setiap elemen dengan `skalar`, kita menuliskan *loop* `for` tersarang yang mengiterasi baris dan kolom dari objek matriks untuk mengakses setiap elemen dari objek matriks, lalu mengalikan elemen tersebut dengan `skalar`:

```
def kaliSkalar(self, skalar):
    for brs in range(self.bykBaris()):
        for klm in range(self.bykKolom):
            self[brs, klm] *= skalar
```

Method `transpos()`

Method `transpos()` mengembalikan matriks transpos dari objek matriks. *Method* ini tidak memodifikasi objek matriks. Untuk mengimplementasikan *method* ini, kita pertama membuat sebuah matriks baru dengan banyak baris sebanyak banyak kolom dari objek matriks dan banyak kolom sebanyak banyak baris dari objek matriks. Lalu, kita mengakses setiap elemen dari objek matriks menggunakan *loop* `for` tersarang. Pada setiap iterasinya, kita mengisi matriks baru dengan menetapkan elemen indeks `(brs, klm)` pada objek matriks menjadi elemen indeks `(klm, brs)` dari matriks baru. Setelah mengisi semua elemen pada matriks baru, kita mengembalikan matriks tersebut.

Berikut adalah kode implementasi dari method `transpos()`:

```
def transpos(self):
    trMatrix = Matriks(self.bykKolom(), self.bykBaris())
    for brs in range(self.bykBaris()):
        for klm in range(self.bykKolom()):
            trMatrix[klm, brs] = self[brs, klm]
    return trMatrix
```

Method `add(matriksLain)`

Method `add(matriksLain)` mengembalikan sebuah matriks baru hasil penjumlahan matriks antara objek matriks dengan `matriksLain`. *Method* ini tidak memodifikasi objek matriks.

Penjumlahan matriks hanya dapat dilakukan antara dua matriks dengan ukuran yang sama (banyak baris sama dan banyak kolom sama). Oleh karena ini, dalam *method* ini kita pertama-pertama menguji apakah banyaknya baris dan banyaknya kolom pada `matriksLain` sama dengan banyaknya baris dan banyaknya kolom pada objek matriks. Jika tidak sama, kita meng-*raise* `ValueEror` dan jika sama, kita melakukan operasi penjumlahan matriks.

Dalam operasi penjumlahan matriks, kita membuat matriks baru sebagai matriks hasil penjumlahan. Lalu, kita menuliskan *loop* `for` tersarang untuk mengiterasi setiap elemen dari kedua matriks. Pada setiap iterasi, kita mengisi elemen pada baris dan kolom pada matriks baru dengan hasil penjumlahan elemen objek matriks dengan elemen `matriksLain` pada baris dan kolom yang sama.

Berikut adalah kode implementasi dari *method* `add(matriksLain)`:

```
def __add__(self, matriksLain):
    if (matriksLain.bykBaris() != self.bykBaris()) or \
        (matriksLain.bykKolom() != self.bykKolom()):
        raise ValueError('Matriks yang dijumlahkan harus berukuran sama.')
    else:
        hasil = Matriks(self.bykBaris(), self.bykKolom())
        for brs in range(self.bykBaris()):
            for klm in range(self.bykKolom()):
                hasil[brs, klm] = self[brs, klm] + matriksLain[brs, klm]
        return hasil
```

> ***Catatan.*** Implementasi *method* `substract(mtrkLain)` dan *method* `multiply(mtrksLain)` dijadikan latihan.

Kode Implementasi ADT Matriks

Berikut adalah kode implementasi ADT Matriks:

Module *`matriks.py`*

```
from array2d import Array2D

# Implementasi ADT Matriks
class Matriks:
    # Constructor: membuat sebuah matrix dengan ukuran bykBaris x bykKolom
    # dengan semua elemen diinisialisasi ke 0
    def __init__(self, bykBaris, bykKolom):
        self._arr2D = Array2D(bykBaris, bykKolom)
        self._arr2D.clear(0)

    # Method bykBaris() mengembalikan banyaknya baris dalam matrix
    def bykBaris(self):
        return self._arr2D.bykBaris()

    # Method bykKolom() mengembalikan banyaknya kolom dalam matrix
    def bykKolom(self):
        return self._arr2D.bykKolom()

    # Operasi getitem(brs, klm) mengembalikan nilai elemen pada brs dan klm.
    # Diakses dengan notasi subscript: x[i, j]
    def __getitem__(self, indexTuple):
        return self._arr2D[indexTuple[0], indexTuple[1]]

    # Operasi setitem(brs, klm, nilai) menetapkan nilai elemen pada brs dan klm
    # dengan nilai.
    # Diakses dengan notasi subscript: x[i, j] = nilai
    def __setitem__(self, indexTuple, nilai):
        self._arr2D[indexTuple[0], indexTuple[1]] = nilai

    # Operasi kaliSkalar(skalar) memodifikasi matriks dengan mengalikan
    # setiap elemen matriks dengan skalar yang diberikan.
    def kaliSkalar(self, skalar):
        for brs in range(self.bykBaris()):
            for klm in range(self.bykKolom):
                self[brs, klm] *= skalar
```

```
    # Operasi transpos() mengembalikan matriks baru yang merupakan
    # transpos dari matriks ini.
    def transpos(self):
        trMatrix = Matriks(self.bykKolom(), self.bykBaris())
        for brs in range(self.bykBaris()):
            for klm in range(self.bykKolom()):
                trMatrix[klm, brs] = self[brs, klm]
        return trMatrix

    # Operasi add(matriksLain) mengembalikan matriks baru
    # hasil penjumlahan matriks ini dengan matriksLain.
    # Diakses menggunakan operator +.
    # Contoh: mtrkC = mtrkA + mtrkB
    def __add__(self, matriksLain):
        # Cek apakah matriksLain mempunyai baris dan kolom yang sama dengan
        # object matriks.
        if (matriksLain.bykBaris() != self.bykBaris()) or \
            (matriksLain.bykKolom() != self.bykKolom()):
            raise ValueError('Matriks yang dijumlahkan harus berukuran sama.')
        else:
            hasil = Matriks(self.bykBaris(), self.bykKolom())
            for brs in range(self.bykBaris()):
                for klm in range(self.bykKolom()):
                    hasil[brs, klm] = self[brs, klm] + matriksLain[brs, klm]
            return hasil

    # Operasi substract(matriksLain) mengembalikan matriks baru
    # hasil pengurangan matriks ini dengan matriksLain.
    # Diakses menggunakan operator -.
    # Contoh: mtrkC = mtrkA - mtrkB
    def __sub__(self, matriksLain):
        # ...

    # Operasi multiply(matriksLain) mengembalikan matriks baru
    # hasil perkalian matriks ini dengan matriksLain.
    # Diakses menggunakan operator *.
    def __mul__(self, matriksLain):
        # ...
```

Menguji Implementasi ADT Matriks

Kita dapat menggunakan program berikut untuk menguji hasil implementasi ADT Matriks:

```
from matriks import Matriks
from random import randint

def cetakMatriks(matriks):
    for brs in range(matriks.bykBaris()):
        for klm in range(matriks.bykKolom()):
            print(f'{matriks[brs, klm]: 4d}', end=' ')
        print()

def main():
    # Buat sebuah matriks
    matrix1 = Matriks(3, 4)
```

```
    # Isi elemen matriks dengan integer acak menggunakan
    # operasi setitem melalui notasi subscript.
    for i in range(matrix1.bykBaris()):
        for j in range(matrix1.bykKolom()):
            matrix1[i, j] = randint(0, 100)

    # Cetak isi matriks menggunakan loop for tersarang
    # dan menggunakan operasi getitem untuk mendapatkan
    # nilai elemen.
    print('Matrix 1:')
    cetakMatriks(matrix1)
    print()

    # Cetak matriks transpos dari matrix1 menggunakan
    # operasi transpos().
    matrixTranspose = matrix1.transpos()
    print('Transpose matrix 1:')
    cetakMatriks(matrixTranspose)
    print()

    # Buat matriks lain dengan ukuran yang sama seperti
    # matrix1
    matrix2 = Matriks(3, 4)

    # Isi elemen matrix2 dengan integer acak
    for i in range(matrix2.bykBaris()):
        for j in range(matrix2.bykKolom()):
            matrix2[i, j] = randint(0, 100)

    # Cetak isi matrix2.
    print('Matrix 2:')
    cetakMatriks(matrix2)
    print()

    # Lakukan penjumlahan matrix1 + matrix2
    # menggunakan operasi add dengan operator +.
    jumlahMatriks = matrix1 + matrix2

    # Cetak jumlah matriks
    print('Matrix 1 + Matrix 2:')
    cetakMatriks(jumlahMatriks)
    print()

    # Lakukan pengurangan matrix1 - matrix2
    # menggunakan operasi substract dengan operator -.
    selisihMatriks = matrix1 - matrix2

    # Cetak jumlah matriks
    print('Matrix 1 - Matrix 2:')
    cetakMatriks(selisihMatriks)
    print()

    # Definisikan matrixA = | 1 2 3 |
    #                       | 4 5 6 |
    matrixA = Matriks(2, 3)
    matrixA[0, 0] = 1
    matrixA[0, 1] = 2
    matrixA[0, 2] = 3
```

```
    matrixA[1, 0] = 4
    matrixA[1, 1] = 5
    matrixA[1, 2] = 6

    # Cetak matrix A
    print('Matrix A: ')
    cetakMatriks(matrixA)
    print()

    # Definisikan matrixB = |  7  8 |
    #                       |  9 10 |
    #                       | 11 12 |
    matrixB = Matriks(3, 2)
    matrixB[0, 0] = 7
    matrixB[0, 1] = 8
    matrixB[1, 0] = 9
    matrixB[1, 1] = 10
    matrixB[2, 0] = 11
    matrixB[2, 1] = 12

    # Cetak matrix B
    print('Matrix B: ')
    cetakMatriks(matrixB)
    print()

    # Hitung hasil kali dan cetak hasilnya
    print('Hasil kali matrix A dan matrix B: ')
    cetakMatriks(matrixA * matrixB)
    print()

main()
```

Output dari program di atas:

```
Matrix 1:
  24   73   22   33
  13   11   29   88
  17   43    7   14

Transpose matrix 1:
  24   13   17
  73   11   43
  22   29    7
  33   88   14

Matrix 2:
  77   21   92   92
  72   80   38    1
  23   63   27   51

Matrix 1 + Matrix 2:
 101   94  114  125
  85   91   67   89
  40  106   34   65

Matrix 1 - Matrix 2:
 -53   52  -70  -59
```

```
 -59  -69   -9   87
  -6  -20  -20  -37

Matrix A:
   1    2    3
   4    5    6

Matrix B:
   7    8
   9   10
  11   12

Hasil kali matrix A dan matrix B:
  58   64
 139  154
```

REFERENSI

[1] Necaise, Rance D. 2011. Data structures and algorithms using Python .