CPE111: Programming with Data Structures

Week2: Array

Practice I) 1-Dimensional array

From the definition of **Array ADT** (1-D array), please code "array.py" following in **List1** and then try to create an Array object and test some operations.

Array ADT

A one-dimensional array is a collection of contiguous elements in which individual elements are identified by a unique integer subscript starting with zero. Once an array is created, its size cannot be changed.

- *Array(size)* : Creates a one-dimensional array consisting of **size** elements with each element initially set to **None**. Size must be greater than zero.
- *length( )*: Returns the length of number of elements in the array.
- *getitem(index)*: Returns the value stored in the array at element position index. The index argument must be with in the valid range. Accessed using the subscript operator.
- setitem*(index, value)*: Modifies the contents of the array element at position index to contain value. The index must be within the valid range. Accessed using the subscript operator.
- *clearing(value)*: Clears the array by setting every element to value.
- *iterator( )*: Creates and returns an iterator that can be used to traverse the elements of the array.

List1: array.py

```
1 # Implements the Array ADT using array capabilities of the ctypes module.
2 import ctypes
3
4 class Array :
5       # Creates an array with size elements.
6       def __init__( self, size ):
7             assert size > 0, "Array size must be > 0"
8             self._size = size
9             # Create the array structure using the ctypes module.
10            PyArrayType = ctypes.py_object * size
11            self._elements = PyArrayType()
12            # Initialize each element.
13            self.clear( None )
14
15      # Returns the size of the array.
16      def __len__( self ):
```

```
17              return self._size
18
19      # Gets the contents of the index element.
20      def __getitem__( self, index ):
21              assert index >= 0 and index < len(self), "Array subscript out of range"
22              return self._elements[ index ]
23
24      # Puts the value in the array element at index position.
25      def __setitem__( self, index, value ):
26              assert index >= 0 and index < len(self), "Array subscript out of range"
27              self._elements[ index ] = value
28
29      # Clears the array by setting each element to the given value.
30      def clear( self, value ):
31              for i in range( len(self) ) :
32                      self._elements[i] = value
33
34      # Returns the array's iterator for traversing the elements.
35      def __iter__( self ):
36              return _ArrayIterator( self._elements )
37
38 # An iterator for the Array ADT.
39 class _ArrayIterator :
40      def __init__( self, theArray ):
41              self._arrayRef = theArray
42              self._curNdx = 0
43
44      def __iter__( self ):
45              return self
46
47      def __next__( self ):
48              if self._curNdx < len( self._arrayRef ) :
49                      entry = self._arrayRef[ self._curNdx ]
50                      self._curNdx += 1
51                      return entry
52              else :
53                      raise StopIteration
```

Practice II) 2- dimensional array

From the definition of Array2D ADT (2-D array) add code in list2 to "array.py" and then try to create an Array2D object and test its operations.

Array2D ADT

A two-dimensional array consists of a collection of elements organized into rows and columns. Individual elements are referenced by specifying the specific row and column indices (r, c), both of which start at 0.

- *Array2D(nrows, ncols):* Creates a two-dimensional array organized into rows and columns. The **nrows** and **ncols** arguments indicate the size of the table. The individual elements of the table are initialized to None.
- *numRows():* Returns the number of rows in the 2-D array.
- *numCols():* Returns the number of columns in the 2-D array.
- *clear(value):* Clears the array by setting each element to the given value.
- *getitem(i1, i2):* Returns the value stored in the 2-D array element at the position indicated by the 2-tuple (i1; i2), both of which must be within the valid range. Accessed using the subscript operator: y = x[1,2].
- *setitem(i1, i2, value):* Modifies the contents of the 2-D array element indicated by the 2-tuple (i1; i2) with the new value. Both indices must be within the valid range. Accessed using the subscript operator: x[0,3] = y.

List2: adding to array.py

```
# Implementation of the Array2D ADT using an array of arrays.

class Array2D :
    # Creates a 2-D array of size numRows x numCols.
    def __init__( self, numRows, numCols ):
        # Create a 1-D array to store an array reference for each row.
        self._theRows = Array( numRows )

        # Create the 1-D arrays for each row of the 2-D array.
        for i in range( numRows ) :
            self._theRows[i] = Array( numCols )

    # Returns the number of rows in the 2-D array.
    def numRows( self ):
        return len( self._theRows )

    # Returns the number of columns in the 2-D array.
    def numCols( self ):
```

```
19              return len( self._theRows[0] )
20
21      # Clears the array by setting every element to the given value.
22      def clear( self, value ):
23              for row in range( self.numRows() ):
24                      self._theRows[row].clear(value)
25
26      # Gets the contents of the element at position [i, j]
27      def __getitem__( self, ndxTuple ):
28              assert len(ndxTuple) == 2, "Invalid number of array subscripts."
29              row = ndxTuple[0]
30              col = ndxTuple[1]
31              assert row >= 0 and row < self.numRows() \
32                      and col >= 0 and col < self.numCols(), \
33                              "Array subscript out of range."
34              the1dArray = self._theRows[row]
35              return the1dArray[col]
36
37      # Sets the contents of the element at position [i,j] to value.
38      def __setitem__( self, ndxTuple, value ):
39              assert len(ndxTuple) == 2, "Invalid number of array subscripts."
40              row = ndxTuple[0]
41              col = ndxTuple[1]
42              assert row >= 0 and row < self.numRows() \
43                      and col >= 0 and col < self.numCols(), \
44                              "Array subscript out of range."
45              the1dArray = self._theRows[row]
46              the1dArray[col] = value
```

Practice III) Matrix

From the definition of Matrix ADT, create “matrix.py” by code in **List3** and try to complete **transpose( )** , **sub( )** and **mul( )** method.

Matrix ADT

A matrix is a collection of scalar values arranged in rows and columns as a rectangular grid of a fixed size. The elements of the matrix can be accessed by specifying a given row and column index with indices starting at 0.

- *Matrix(rows, ncols):* Creates a new matrix containing **nrows** and **ncols** with each element initialized to 0.
- *numRows():* Returns the number of rows in the matrix.
- *numCols():* Returns the number of columns in the matrix.
- *getitem (row, col):* Returns the value stored in the given matrix element. Both row and col must be within the valid range.
- *setitem (row, col, scalar):* Sets the matrix element at the given row and col to scalar. The element indices must be within the valid range.
- *scaleBy(scalar):* Multiplies each element of the matrix by the given scalar value. The matrix is modified by this operation.
- *transpose():* Returns a new matrix that is the transpose of this matrix.
- *add (rhsMatrix):* Creates and returns a new matrix that is the result of adding this matrix to the given **rhsMatrix**. The size of the two matrices must be the same.
- *subtract (rhsMatrix):* The same as the add() operation but subtracts the two matrices.
- *multiply (rhsMatrix):* Creates and returns a new matrix that is the result of multiplying this matrix to the given **rhsMatrix**. The two matrices must be of appropriate sizes as defined for matrix multiplication.

List3: matrix.py

```
# Implementation of the Matrix ADT using a 2-D array.
from array import Array2D

class Matrix :
    # Creates a matrix of size numRows x numCols initialized to 0.
    def __init__( self, numRows, numCols ):
        self._theGrid = Array2D( numRows, numCols )
        self._theGrid.clear( 0 )

    # Returns the number of rows in the matrix.
    def numRows( self ):
        return self._theGrid.numRows()

    # Returns the number of columns in the matrix.
    def numCols( self ):
        return self._theGrid.numCols()

    # Returns the value of element (i, j): x[i,j]
    def __getitem__( self, ndxTuple ):
        return self._theGrid[ ndxTuple[0], ndxTuple[1] )

```

```
22      # Sets the value of element (i,j) to the value s: x[i,j] = s
23      def __setitem__( self, ndxTuple, scalar ):
24              self._theGrid[ ndxTuple[0], ndxTuple[1] ] = scalar
25
26      # Scales the matrix by the given scalar.
27      def scaleBy( self, scalar ):
28              for r in range( self.numRows() ) :
29                      for c in range( self.numCols() ) :
30                              self[ r, c ] *= scalar
31
32      # Creates and returns a new matrix that is the transpose of this matrix.
33      def tranpose( self ):
34              ……
35
36       # Creates and returns a new matrix that results from matrix addition.
37      def __add__( self, rhsMatrix ):
38              assert rhsMatrix.numRows() == self.numRows() and \
39                      rhsMatrix.numCols() == self.numCols(), \
40                      "Matrix sizes not compatible for the add operation."
41              # Create the new matrix.
42              newMatrix = Matrix( self.numRows(), self.numCols() )
43              # Add the corresponding elements in the two matrices.
44              for r in range( self.numRows() ) :
45                      for c in range( self.numCols() ) :
46                              newMatrix[ r, c ] = self[ r, c ] + rhsMatrix[ r, c ]
47              return newMatrix
48
49       # Creates and returns a new matrix that results from matrix subtraction.
50      def __sub__( self, rhsMatrix ):
51              ……
52
53      # Creates and returns a new matrix resulting from matrix multiplication.
54      def __mul__( self, rhsMatrix ):
55              ……
```

Reference

Rance D. Necaise. *Data Structures and algorithms using python*. *Chapter2*. John Wiley&Sons,Inc., 2011

Michael T.Goodrich, Roberto Tamassia, Michael H. Goodwasser. *Data Structures and Algorithms in python*. *Chapter5*. John Wiley&Sons,Inc. 2013