Lab2

Data Structures and AIgorithms

Maira usman

21B-011-SE

In [4]:
```
import ctypes
import random
class Array:
    def __init__(self,size):
        assert size>0 ,"Array size mustbe > 0"
        self._size=size
        PyArrayType=ctypes.py_object*size
        self._elements=PyArrayType()
        self.clear(None)
    def __len__(self):
        return self._size
    def __getitem__(self,index):
        assert index>=0 and index<len(self),"Array index out of range"
        return self._elements[index]
    def __setitem__(self,index,value):
        assert index>=0 and index<len(self),"Array index out of range"
        self._elements[index] =value
    def clear(self,value):
        for i in range(len(self)):
            self._elements[i]=value
    def __iter__(self):
        return _ArrayIterator(self._elements)
class _ArrayIterator:
    def __init__(self,Array):
        self._arrayRef=Array
        self.index=0
    def __iter__(self):
        return self
    def __next__(self):
        if self._index < len(self._arrayRef):
            entry=self.arrayRef[self.index]
            self.index+=1
            return entry
        else:
            raise StopIteration
x=Array(5)
print("lengthof x:",len(x))
items=["apple","pencil","pens","mouse","mobile","books","mango","toys","notes","water","watch"]
for i in range(len(x)):
    item=random.choice(items)
    print("{}--> {}".format(i,item))
    x[i]=item
print("my arrayupto {} values".format(len(x)))
aindex = int(input("enter index value "))
print("array index {}contains item {}".format(aindex,x[aindex]))
x.clear(0)

print("array index {}contains item after clear {}".format(aindex,x[aindex]))
```

```
lengthof x: 5
0--> notes
1--> mobile
2--> mouse
3--> water
4--> toys
my arrayupto 5 values
enter index value 4
array index 4contains item toys
array index 4contains item after clear 0
```

In [1]:

```
import ctypes
import random
class Array:
    def __init__(self,size):
        assert size>0 ,"Array size mustbe > 0"
        self._size=size
        PyArrayType=ctypes.py_object*size
        self._elements=PyArrayType()
        self.clear(None)
    def __len__(self):
        return self._size
    def __getitem__(self,index):
        assert index>=0 and index<len(self),"Array index out of range"
        return self._elements[index]
    def __setitem__(self,index,value):
        assert index>=0 and index<len(self),"Array index out of range"
        self._elements[index] =value
    def clear(self,value):
        for i in range(len(self)):
            self._elements[i]=value
    def __iter__(self):
        return _ArrayIterator(self._elements)
class _ArrayIterator:
    def __init__(self,Array):
        self._arrayRef=Array
        self.index=0
    def __iter__(self):
        return self
    def __next__(self):
        if self._index < len(self._arrayRef):
            entry=self.arrayRef[self.index]
            self.index+=1
            return entry
        else:
            raise StopIteration
class Array2D:
    def __init__(self, rows,cols):
        self._rows=Array(rows)
        self._cols=cols
        for i in range(rows):
            self._rows[i]=Array(cols)
    def numRows(self):
        return len(self._rows)
    def numCols(self):
        return len(self._rows[0])
    def Getitem(self,row,col):
        assert row >=0 and row < self.numRows() and col >= 0 and col < self.numCols(),\
        "Array subscript out of range."
        array=self._rows[row]
        return array[col]
    def SetValue(self, row, col ,value):
        assert row >=0 and row < self.numRows()  and col >= 0 and col < self.numCols(),\
        "Array subscript out of range."
        array=self._rows[row]
        array[col]=value
    def clear(self, value):
        for row in range(self.numRows()):
            self._rows[row].clear(value)
    def PrintValues(self):
        for i in range(self.numRows()):
            for j in range(self.numCols()):
                print(self.Getitem(i,j),end=" ")
            print()
    def SubValues(self,array1, array2):
        array3=Array2D(len(self._rows),self._cols)
        for i in range(len(self._rows)):
            print()
            for j in range(self._cols):
                value=array1.Getitem(i,j)-array2.Getitem(i,j)
                array3.SetValue(i,j,value)
                print(array3.Getitem(i,j),end=" ")
    def MultValues(self,array1, array2):
        value=0
        array3=Array2D(len(self._rows),self._cols)
        for i in range(len(self._rows)):
            print()
            for j in range(self._cols):
                for k in range(self._cols):
                    value += array1.Getitem(i,k)*array2.Getitem(k,j)
                array3.SetValue(i,j,value)
                print(array3.Getitem(i,j),end=" ")


    def Transpose(self):
        array3=Array2D(self._cols,len(self._rows))
        for i in range(len(self._rows)):
            print()
            for j in range(self._cols):
                value=self.Getitem(j,i)
                array3.SetValue(i,j,value)
                print(array3.Getitem(i,j),end=" ")



A = Array2D(3,3) # for a 3 x 3 array
```

```
A.clear(0)
A.SetValue(0,0,1)
A.SetValue(0,1,2)
A.SetValue(0,2,3)
A.SetValue(1,0,4)
A.SetValue(1,1,5)
A.SetValue(1,2,6)
A.SetValue(2,0,7)
A.SetValue(2,1,8)
A.SetValue(2,2,9)

B = Array2D(3,3) # for a 3 x 3 array
B.clear(0)
B.SetValue(0,0,1)
B.SetValue(0,1,2)
B.SetValue(0,2,2)
B.SetValue(1,0,4)
B.SetValue(1,1,5)
B.SetValue(1,2,7)
B.SetValue(2,0,9)
B.SetValue(2,1,1)
B.SetValue(2,2,5)
A.PrintValues()
print()
B.PrintValues()
A.SubValues(A,B)
print()
A.MultValues(A,B)
print()
A.Transpose()
print()
B.Transpose()
```

```
1 2 3
4 5 6
7 8 9

1 2 2
4 5 7
9 1 5

0 0 1
0 0 -1
-2 7 4

36 51 82
160 199 272
392 455 570

1 4 7
2 5 8
3 6 9

1 4 9
2 5 1
2 7 5
```

In [1]:
```
import numpy as np

array1 = np.array([[1,2,3,4],[5,6,7,8]], dtype=np.int64)
print(array1)
x = np.ones((3,4),dtype=np.int64)
print(x)
y = np.zeros((2,3,4),dtype=np.int16)
# print(y)
array2 = np.random.random((2,2))
print(array2)
array3 = np.full((3,3),7)
print(array3)
array4 = np.identity(3,dtype=np.int64)
print(array4)
add = np.add(x,y)
print(add)
diff = np.subtract(x,y)
print(diff)
mult = np.multiply(x,y)
print(mult)
div = np.divide(y,x)
print(div)
rem = np.remainder(y,x)
print(rem)
result = np.array_equal(x,y)
print(result)
```

```
[[1 2 3 4]
 [5 6 7 8]]
[[1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]]
[[[0 0 0 0]
  [0 0 0 0]
  [0 0 0 0]]

 [[0 0 0 0]
  [0 0 0 0]
  [0 0 0 0]]]
[[0.24399135 0.76900499]
 [0.35043376 0.97520498]]
[[7 7 7]
 [7 7 7]
 [7 7 7]]
[[1 0 0]
 [0 1 0]
 [0 0 1]]
[[[1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]]

 [[1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]]]
[[[1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]]

 [[1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]]]
[[[0 0 0 0]
  [0 0 0 0]
  [0 0 0 0]]

 [[0 0 0 0]
  [0 0 0 0]
  [0 0 0 0]]]
[[[0. 0. 0. 0.]
  [0. 0. 0. 0.]
  [0. 0. 0. 0.]]

 [[0. 0. 0. 0.]
  [0. 0. 0. 0.]
  [0. 0. 0. 0.]]]
[[[0 0 0 0]
  [0 0 0 0]
  [0 0 0 0]]

 [[0 0 0 0]
  [0 0 0 0]
  [0 0 0 0]]]
False
```

In [ ]: