Arrays-draft

September 9, 2021

3. Arrays

0.0.1 3.1 The Array structure

A **one-dimensional array** is composed of multiple sequential elements stored in contiguous bytes of memory and allows for random access to the individual elements.

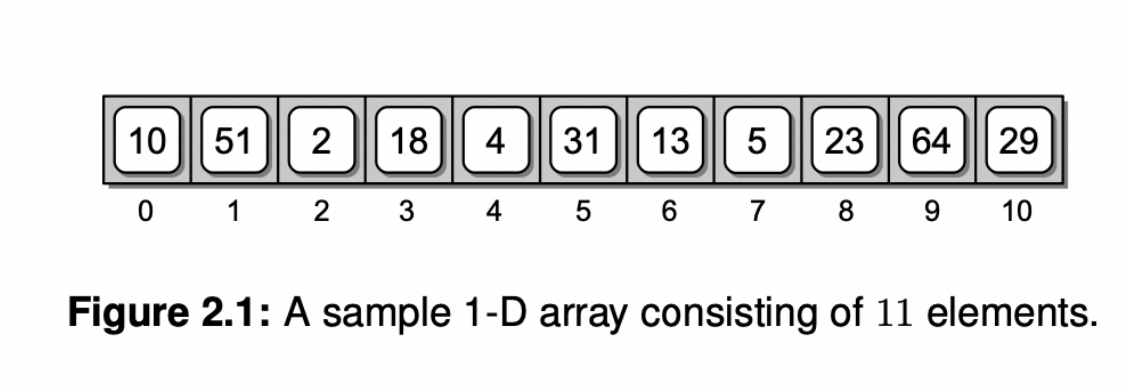

Figure 2.1: A sample 1-D array consisting of 11 elements.

0.0.2 3.1.1 Why Study Arrays?

The array structure looks very similar to Python's list structure. That's because the two structures are both sequences that are composed of multiple sequential elements that can be accessed by position. But there are two major differences between the array and the list. - **First**, an array has a limited number of operations, which commonly include those for array creation, reading a value from a specific element, and writing a value to a specific element. The list, on the other hand, provides a large number of operations for working with the contents of the list. - **Second**, the list can grow and shrink during execution as elements are added or removed while the size of an array cannot be changed after it has been created.

0.0.3 3.1.2 The Array Abstract Data Type

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 or number of elements in the array.

- **getitem (index)**: Returns the value stored in the array at element position index. The index argument must be within 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.

```
# Fill a 1-D array with random values, then print them, one per line.
from array import Array
import random

# The constructor is called to create the array.
valueList = Array( 100 )

# Fill the array with random floating-point values. for i in range( len( valueList ) ) :
valueList[ i ] = random.random()

# Print the values, one per line.
for value in valueList :
    print( value )
```

[15]:
```
import ctypes

ArrayType = ctypes.py_object

slots = ArrayType()

p[0] = 1
p[4] = 1000

print(p[0])
print(p[4])
```

```
1
1000
```

[117]:
```
# Implements the Array ADT using array capabilities of the ctypes module.
import ctypes

class Array :
    def __init__( self, size ):
        '''Creates an array with size elements.'''
        assert size > 0, "Array size must be > 0"
        self.size = size
        # Create the array structure using the ctypes module.
        ArrayType = ctypes.py_object * size
```

```
        self.items = ArrayType()

    def expand(self):
        '''Double the cpacity of the array'''
        ArrayType = ctypes.py_object * (2 * self.size)
        newArray = ArrayType()
        self.copy(newArray)
        self.items = newArray;
        self.size = self.size * 2

    def copy(self, other):
        i =0;
        for item in self.items:
            other[i] = item
            i +=1

    def __len__( self ):
        '''Returns the size of the array.'''
        return self.size


    def __getitem__( self, index ):
        '''Gets the contents of the index element'''
        assert index >= 0 and index < len(self), "Array subscript out of range"
        return self.items[index]


    def __setitem__( self, index, value ):
        '''Puts the value in the array element at index position'''
        assert index >= 0 and index < len(self), "Array subscript out of range"
        self.items[index] = value


    def clear( self, value ):
        '''Clears the array by setting each element to the given value'''
        for i in range(len(self)):
            self.items[i] = value


    def __iter__(self):
        '''Returns the array's iterator for traversing the elements'''
        return MyArrayIterator(self.items)

class MyArrayIterator():
    def __init__(self, arrayItems):
        self._arrayRef = arrayItems;
        self.curInd = 0;
```

```
    def __iter__(self):
        return self

    def __next__(self):
        if self.curInd < len(self._arrayRef):
            item = self._arrayRef[self.curInd]
            self.curInd +=1
            return item
        else:
            raise StopIteration
```

[123]:
```
p = Array(5)

p.clear('*')

s = str(len(p))
print(s)

p[0] = "cat"
p[2] = "tiger"
print(p[0])
print(p[2])

q = Array(5)

p.copy(q)
print(q[0])
print(q[2])


help(p.clear)

print(p)
p.expand()

p.clear('_')
s = str(len(p))
print(s)

p[1] = "dog"
print(p[2])
print(p[1])
print(p[0])
```

```
for a in p:
    print(a)
```

```
5
cat
tiger
cat
tiger
Help on method clear in module __main__:

clear(value) method of __main__.Array instance
    Clears the array by setting each element to the given value

<__main__.Array object at 0x106357a00>
10
-
dog
-
-
dog
-
-
-
-
-
-
-
-
```

0.0.4 Reference:

[1] **Chapter 2**: Necaise, Rance D - Data structures and algorithms using Python-John Wiley & Sons (2010_2011)

[ ]: