CH3. LIST_SET

List

List ADT

데이터: 같은 유형의 요소들의 순서 있는 모임

연산

- List(): 비어 있는 새로운 리스트를 만든다.
- insert(pos, e): pos 위치에 새로운 요소 e를 삽입한다.
- delete(pos): pos 위치에 있는 요소를 꺼내고(삭제) 반환한다.
- isEmpty(): 리스트가 비어있는지를 검사한다.
- getEntry(pos): pos 위치에 있는 요소를 반환한다.
- size(): 리스트안의 요소의 개수를 반환한다.
- clear(): 리스트를 초기화한다.
- find(item): 리스트에서 item이 있는지 찾아 인덱스를 반환한다.
- replace(pos, item): pos에 있는 항목을 item으로 바꾼다.
- sort(): 리스트의 항목들을 어떤 기준으로 정렬한다.
- merge(lst): 다른 리스트 lst를 리스트에 추가한다.
- display(): 리스트를 화면에 출력한다.
- append(e): 리스트의 맨 뒤에 새로운 항목을 추가한다.

1. 함수로 구현

1) insert(pos, elem)

```
def insert(pos, elem):
  items.insert(pos, elem)
```

2) delete(pos)

```
def delete(pos):
  return items.pop(pos)
```

3) getEntry(pos)

```
def getEntry(pos): return items[pos]
```

4) isEmpty()

```
#1.

def isEmpty():
  if len(items) == 0:
    return True
  else:
    return False


#2.

def isEmpty(): return len(items) == 0
```

5) 기타 함수들

```
def size(): return len(items)

def clear():
  global items
```

```
  items = []

def find(item): return items.index(item)

def replace(pos, elem): items[pos] = elem

def sort(): items.sort()

def merge(lst): items.extend(lst)
```

6) display()

```
def display(msg='ArrayList'):
  print(msg, size(), items)
```

2. 클래스로 구현

```
class ArrayList:
  def __init__(self):
    self.items = []

  def insert(self, pos, elem):  #O(n)->중간에 삽입할 경우 다 뒤로 밀림
    items.insert(pos, elem)

  def delete(self, pos): #O(n)->첫 항목을 삭제할 경우 다 앞으로 당겨야함
    return items.pop(pos)

  def getEntry(self, pos): return items[pos]

  def isEmpty(self): return len(items) == 0

  def size(self): return len(items)

  def clear(self):
    items = []

  def find(self,item): return items.index(item)

  def replace(self, pos, elem): items[pos] = elem

  def sort(self): items.sort()

  def merge(self, lst): items.extend(lst)
```

```
def display(self, msg='ArrayList'):
  print(msg, size(), items)
```

Set

Set ADT

데이터: 같은 유형의 유일한 요소들의 모임. 원소들은 순서는 없지만 서로 비교할 수는 있어야 함.

연산

- Set(): 비어 있는 새로운 집합을 만든다.
- size(): 집합의 원소의 개수를 반환한다.
- contains(e): 집합이 원소 e를 포함하는지를 검사하고 반환함.
- insert(e): 새로운 원소 e를 삽입함. 이미 e가 있다면 삽입하지 않음.

- delete(e): 원소 e를 집합에서 꺼내고(삭제) 반환한다.
- equals(setB): setB와 같은 집합인지를 검사.
- union(setB): setB와의 합집합을 만들어 반환한다.
- intersect(setB): setB와의 교집합을 만들어 반환한다.
- difference(setB): setB와의 차집합을 만들어 반환한다.
- display(): 집합을 화면에 출력한다.

1. 클래스로 구현

```
class Set:
    def __init__(self):
        self.items = []

    def size(self):
        return len(self.items)

    def display(self, msg):
        print(msg, self.items)

    def contains(self, item):
        return item in self.items

    def insert(self, elem): #O(n2)
        if elem not in self.items:
            self.items.append(elem)

    def delete(self, elem): #O(n2)
        if elem in self.items:
            self.items.remove(elem)

    def union(self, setB): #O(n^2)
        setC = Set()
        setC.items = list(self, items)

        for elem in setB.items:
            if elem not in self.items: #n번
                setC.items.append(elem)
        return setC

    def intersect(self, setB): #O(n^2)
        setC = Set()
        for elem in setB.items:
            if elem in self.items: #n번
                setC.items.append(elem)
        return setC

    def difference(self, setB): #O(n^2)
        setC = Set()
        for elem in setB.items:
            if elem not in setB.items: #n번
                setC.items.append(elem)
        return setC


# 3.9
    def properSubset(self, setB):
        setC = Set()
        setD = Set()

        for elem in self.items:
            if elem in setB.items:
```

```
                setC.items.append(elem)
            else: setD.items.append(elem)

        if setC.items == setA.items and setD.size() == 0 and self.items != setB.items:
            return True
        else: return False
```