다항식 프로그램 완성과 라인 편집기 구현

자료구조 01분반
20243108 소프트웨어학부 장민주

1. 다항식 프로그램 완성

(1) Polynomial.py 코드

```python
class Poly: # 다항식 만들기  3 usages  MinJuTur

    def __init__(self, capacity = 10):  MinJuTur
        self.capacity = capacity
        self.degree = 0 # 다항식의 차수
        self.coefArr = [None] * capacity # 다항식의 계수 저장 배열

    def readPoly(self):  2 usages  MinJuTur
        self.degree = int(input('다항식의 차수를 입력: '))
        for i in range(self.degree, -1, -1): # 다항식의 차수 ~ 0까지 감소
            coef = int(input(' %d차 항의 계수 : ' % i))
            self.coefArr[i] = coef # i차 항의 계수는 배열의 i번째 인덱스에 저장

    def printPoly(self):  3 usages  MinJuTur
        for i in range(self.degree, 0, -1): # n차~1차 출력
            if (self.coefArr[i] == 0): # 계수가 0이면 출력하지 않음
                continue
            print('%dx^%d + ' % (self.coefArr[i], i), end='')
        print(self.coefArr[0]) # 상수항 출력
```

그림 1 Poly 클래스 코드-1

```
def add(self, other):   # 다항식 더하기  1 usage  MinJuTur *
    if (self.degree >= other.degree):
        larger = self
        smaller = other
    else:
        larger = other
        smaller = self

    sumPoly = Poly(larger.degree+1)
    sumPoly.degree = larger.degree

    for i in range(sumPoly.degree+1):
        if (i > smaller.degree):
            smallerCoef = 0
        else:
            smallerCoef = smaller.coefArr[i]
        largerCoef = larger.coefArr[i]

        sumPoly.coefArr[i] = smallerCoef + largerCoef

    return sumPoly


def evaluate(self, x):  2 usages  MinJuTur
    sum = 0
    for i in range(self.degree, -1, -1):
        sum += self.coefArr[i] * (x ** i)
    return sum
```

그림 2 Poly 클래스 코드-2

(1)-1 add() 함수 설명

두 다항식의 차수를 비교하여, 차수가 더 큰 다항식을 larger 변수에, 차수가 더 작은 다항식을 smaller 변수에 할당합니다. 이는 더 큰 차수를 가진 다항식의 계수를 기준으로 덧셈을 수행하기 위함입니다.

그 다음, 두 다항식의 합을 저장할 새로운 다항식 객체 sumPoly를 생성합니다. 이 객체의 capacity는 larger 다항식의 차수+1로 설정했는데, 계수 배열에 모든 항을 저장할 수 있게 하기 위함입니다. sumPoly.degree는 larger 다항식의 차수와 동일하게 설정합니다.

덧셈은 for 루프를 통해 이루어집니다. 루프는 sumPoly.degree + 1번 반복되며, 각 반복에서 두 다항식의 계수를 더합니다. 만약 작은 차수를 가진 다항식에서 해당 차수의 항이 존재하지 않으면, 해당 계수는 0으로 간주됩니다. 이 방법을 통해 두 다항식의 차수가 다를 경우에도, 누락된 항을 자동으로 처리할 수 있습니다. 각 항을 더한 결과는 sumPoly 객체의 계수 배열에 저장되며, 최종적으로 sumPoly를 반환합니다.

이 함수에서는, 차수가 다른 두 다항식을 더했을 때에도, 하나의 다항식에만 계수가 존재하고 다른 하나에는 계수가 없는 경우를 고려하여 기본적으로 0을 채워주는 방식으로 덧셈을 처리하고 있다는 점이 핵심이었습니다.

(1)-2 evalutate() 함수 설명

이 함수는 매개변수 x를 입력받아 해당 x값에서의 다항식 결과를 계산하는 역할을 합니다.

이 함수에서는 먼저 sum 변수를 0으로 초기화한 후, 다항식의 최고차항부터 상수항까지 반복문을 돌며 값을 계산합니다. 계수들이 저장된 배열 coefArr의 인덱스는 해당 계수가 있었던 항의 차수를 의미하므로, 이를 활용하여 각 항의 값을 계산할 수 있습니다. 즉, 배열의 인덱스 I에 저장된 계수에 x**i 를 곱한 후 이를 sum에 누적하는 방식입니다.

이 과정을 통해 다항식의 모든 항을 계산한 후 최종 값을 반환합니다.

(2) 테스트 코드 실행하기

```
if __name__ == '__main__':
    a = Poly()
    a.readPoly()
    print('a =', end="")
    a.printPoly()

    b = Poly()
    b.readPoly()
    print('b =', end="")
    b.printPoly()

    print('a + b =', end="")
    c = a.add(b)
    c.printPoly()

    print('a(-1) =', a.evaluate(-1))
    print('b(2) =', b.evaluate(2))
```

그림 3 테스트 코드

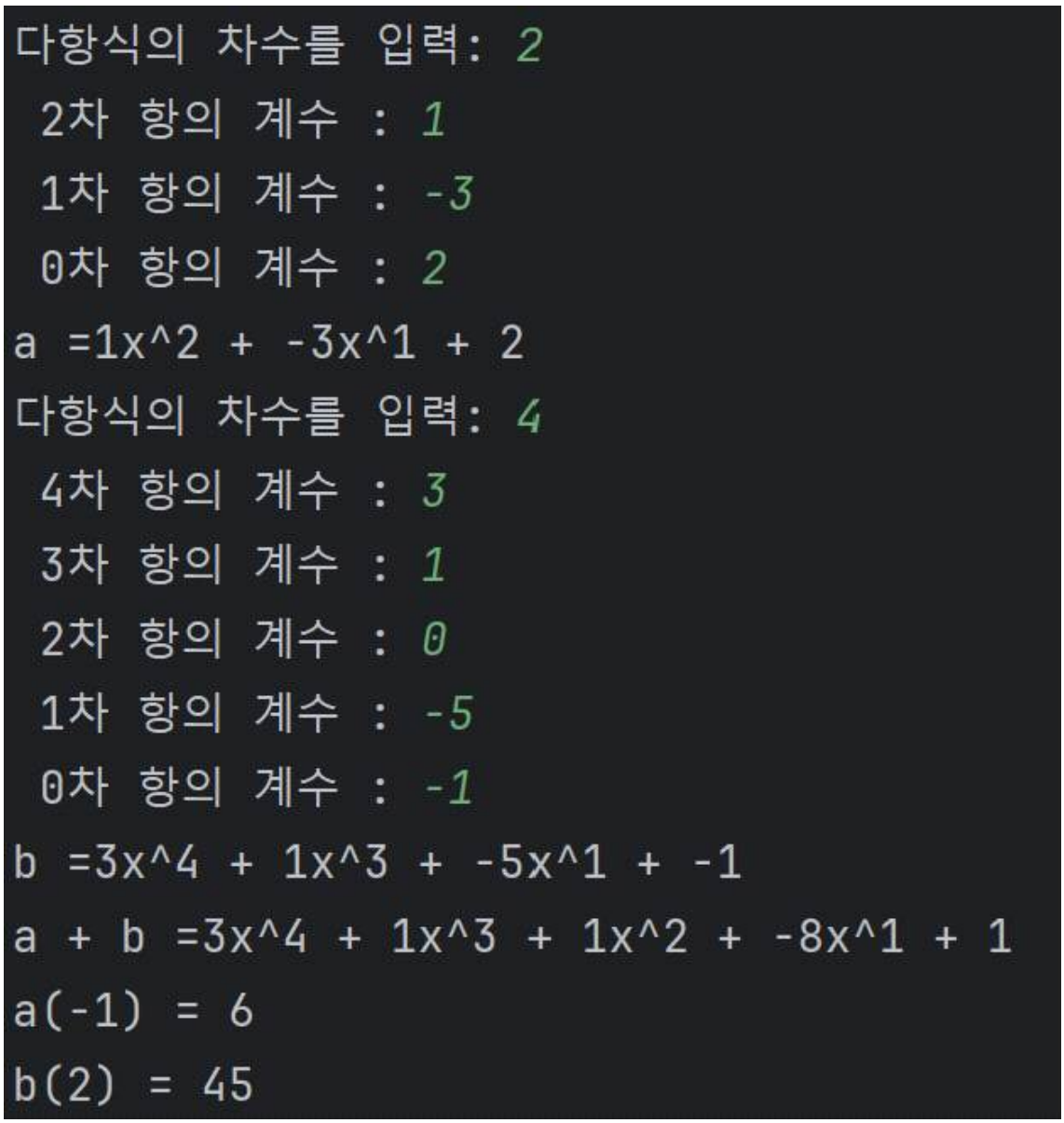

그림 4 테스트 실행 결과

(2)-1 다항식의 덧셈 기능

a = x^2 -3x + 2, b = 3x^4 +x^3 -5x - 1 이라고 입력한 상태입니다.
두 다항식을 더한 다항식을 새로운 c라는 Poly에 저장하여 c를 출력했습니다.
c = 3x^4 + x^3 + 2x^2 -8x + 1 로 잘 출력되는 것을 볼 수 있습니다.

(2)-2 미지수 값이 주어졌을 때 다항식의 계산 값을 구하는 기능

a(-1) 값이 6, b(2) 값이 45로 잘 출력되는 것을 볼 수 있습니다.

2. 라인 편집기 구현

(1) ListByClass.py 코드(ArrayList 클래스)

```
class ArrayList:  3 usages  MinJuTur

    def __init__(self, capacity=100):  MinJuTur
        self.capacity = capacity
        self.size = 0
        self.array = [None] * capacity

    def isEmpty(self):  1 usage  MinJuTur
        return self.size == 0

    def isFull(self):  1 usage  MinJuTur
        return self.size == self.capacity

    def insert(self, pos, e):  8 usages  MinJuTur
        if not self.isFull() and 0 <= pos <= self.size: # pos는 size까지 포함돼야 함
            for i in range(self.size, pos, -1):
                self.array[i] = self.array[i-1]
            self.array[pos] = e
            self.size += 1
        else:
            print("Overflow or Invalid Position.")
```

그림 5 ArrayList 클래스 코드-1

```
def delete(self, pos):  3 usages  MinJuTur
    global size
    if not self.isEmpty() and 0 <= pos < self.size:
        e = self.array[pos]
        for i in range(pos,self.size-1):
            self.array[i] = self.array[i+1]
        self.size -= 1
        return e
    else:
        print("Underflow or Invalid Position.")

def getEntry(self, pos):  3 usages  MinJuTur
    if (0 <= pos < self.size):
        return self.array[pos]
    else:
        return None

def __str__(self):  MinJuTur
    return str(self.array[0:self.size])
```

그림 6 ArrayList 클래스 코드-2

(2) LineEditor.py 코드

```
from practice.Day0324.ListByClass import ArrayList

list = ArrayList()
while True:
    commmand = input("[메뉴선택] i-입력, d-삭제, r-변경, p-출력, l-파일읽기, s-저장, q-종료=> ")

    if commmand == 'i': # 입력
        pos = int(input(" 입력행 번호: "))
        str = input(" 입력행 내용: ")
        list.insert(pos, str)
    elif commmand == 'd': # 삭제
        pos = int(input(" 삭제행 번호: "))
        list.delete(pos)
    elif commmand == 'r': # 변경
        pos = int(input(" 변경행 번호: "))
        str = input(" 변경행 내용: ")
        list.delete(pos)
        list.insert(pos, str)
    elif commmand == 'p': # 출력
        print("Line Editor")
        for i in range(list.size):
            print("[ %d] " % i, list.getEntry(i))
```

그림 7 LineEditor 코드-1

```
elif commmand == 'l': # 파일읽기
    with open("text.txt", "r", encoding="utf-8") as file:
        list.__init__() # 리스트 초기화

        i = 0
        for line in file:
            list.insert(i, line.rstrip()) # 줄바꿈 제거 후 리스트에 추가
            i += 1

        print("Line Editor")
        for i in range(list.size):
            print("[ %d] " % i, list.getEntry(i))
elif commmand == 's': # 저장
    with open("text.txt", "w", encoding="utf-8") as file:
        for i in range(list.size):
            file.write(list.getEntry(i) + "\n")
elif commmand == 'q': # 종료
    break
```

그림 8 LineEditor 코드-2

(2)-1 기능 설명

이 프로그램은 ArrayList 클래스를 이용하여 간단한 라인 편집기 기능을 구현한 코드입니다. 사용자는 프로그램을 실행하면서 텍스트의 행을 입력하거나 수정, 삭제할 수 있으며, 내용을 파일에서 읽어오거나 저장하는 기능도 제공합니다. 프로그램은 while True 반복문을 통해 사용자로부터 명령을 입력받고 이를 처리하는 방식으로 동작합니다.

1) 메뉴 선택: 프로그램은 사용자에게 메뉴를 선택하라는 메시지를 출력하고, 사용자는 다양한 명령을 입력할 수 있습니다. 입력할 수 있는 명령은 i(입력), d(삭제), r(변경), p(출력), l(파일 읽기), s(저장), q(종료)입니다. 사용자에게 메뉴를 계속해서 물어보고, 사용자가 q를 입력할 때까지 프로그램은 반복됩니다.

2) 입력 (i): 사용자가 i를 입력하면, 프로그램은 입력할 행 번호와 내용을 받아 리스트에 추가합니다. 이때, insert 메서드를 사용하여 지정한 위치에 텍스트를 삽입합니다. 사용자가 입력한 위치에 새로운 텍스트가 추가되며, 기존 텍스트는 한 칸씩 밀려납니다.

3) 삭제 (d): 사용자가 d를 입력하면, 삭제할 행 번호를 입력받고 해당 행을 삭제합니다. delete 메서드를 통해 지정된 위치의 요소를 제거하며, 삭제된 후에는 리스트의 크기가 감소하고 뒤의 요소들이 한 칸씩 앞으로 밀립니다.

4) 변경 (r): 사용자가 r을 입력하면, 변경할 행 번호와 새로운 내용을 입력받고 해당 행을 변경합니다. 삭제 후 새로운 텍스트를 삽입하는 방식으로 동작합니다. 기존 내용을 삭제하고, 그 자리에 변경된 내용을 새로 입력받아 삽입합니다.
5) 출력 (p): 사용자가 p를 입력하면, 현재 리스트에 저장된 모든 텍스트를 출력합니다. 각 텍스트 항목은 행 번호와 함께 출력됩니다. getEntry 메서드를 사용하여 각 항목을 가져오고, for 반복문을 통해 하나씩 출력합니다.
6) 파일 읽기 (l): 사용자가 l을 입력하면, 프로그램은 text.txt 파일을 읽고 파일에 저장된 내용을 리스트에 추가합니다. open() 함수로 파일을 열고, 각 줄을 읽어서 insert 메서드를 사용하여 리스트에 삽입합니다. 파일을 읽어올 때 줄바꿈 문자는 제거됩니다. 파일의 모든 내용을 리스트에 추가한 후, 다시 전체 내용을 출력합니다.
7) 저장 (s): 사용자가 s를 입력하면, 현재 리스트에 저장된 모든 내용을 text.txt 파일에 저장합니다. open() 함수로 파일을 열고, 리스트의 각 항목을 줄바꿈과 함께 파일에 기록합니다. 리스트에 있는 텍스트는 getEntry 메서드를 통해 가져오며, 이를 파일에 씁니다.
8) 종료 (q): 사용자가 q를 입력하면 프로그램이 종료됩니다. break 문을 사용하여 무한 루프를 종료하고, 프로그램 실행이 끝납니다.

(4) 실행하기

```
[메뉴선택] i-입력, d-삭제, r-변경, p-출력, l-파일읽기, s-저장, q-종료=> i
 입력행 번호: 0
 입력행 내용: hello
[메뉴선택] i-입력, d-삭제, r-변경, p-출력, l-파일읽기, s-저장, q-종료=> i
 입력행 번호: 1
 입력행 내용: this is content
[메뉴선택] i-입력, d-삭제, r-변경, p-출력, l-파일읽기, s-저장, q-종료=> i
 입력행 번호: 1
 입력행 내용: content added
[메뉴선택] i-입력, d-삭제, r-변경, p-출력, l-파일읽기, s-저장, q-종료=> p
Line Editor
[ 0]  hello
[ 1]  content added
[ 2]  this is content
```

그림 9 입력과 출력

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

    def insert(self, pos, elem) : self.items.insert(pos, elem)
    def delete(self, pos) : self.items.pop(pos)
    def isEmpty(self) : return self.size() == 0
```

그림 10 text.txt 파일 내용

```
[메뉴선택] i-입력, d-삭제, r-변경, p-출력, l-파일읽기, s-저장, q-종료=> l
Line Editor
[ 0]  class ArrayList:
[ 1]      def __init__(self):
[ 2]          self.items = []
[ 3]
[ 4]      def insert(self, pos, elem) : self.items.insert(pos, elem)
[ 5]      def delete(self, pos) : self.items.pop(pos)
[ 6]      def isEmpty(self) : return self.size() == 0
[메뉴선택] i-입력, d-삭제, r-변경, p-출력, l-파일읽기, s-저장, q-종료=> p
Line Editor
[ 0]  class ArrayList:
[ 1]      def __init__(self):
[ 2]          self.items = []
[ 3]
[ 4]      def insert(self, pos, elem) : self.items.insert(pos, elem)
[ 5]      def delete(self, pos) : self.items.pop(pos)
[ 6]      def isEmpty(self) : return self.size() == 0
```

그림 11 text.txt 파일 읽기와 출력

```
[메뉴선택] i-입력, d-삭제, r-변경, p-출력, l-파일읽기, s-저장, q-종료=> r
 변경행 번호: 5
 변경행 내용:     def delete(self, pos) : this.items.pop(pos)
[메뉴선택] i-입력, d-삭제, r-변경, p-출력, l-파일읽기, s-저장, q-종료=> p
Line Editor
[ 0]  class ArrayList:
[ 1]      def __init__(self):
[ 2]          self.items = []
[ 3]
[ 4]      def insert(self, pos, elem) : self.items.insert(pos, elem)
[ 5]      def delete(self, pos) : this.items.pop(pos)
[ 6]      def isEmpty(self) : return self.size() == 0
[메뉴선택] i-입력, d-삭제, r-변경, p-출력, l-파일읽기, s-저장, q-종료=> s
[메뉴선택] i-입력, d-삭제, r-변경, p-출력, l-파일읽기, s-저장, q-종료=> l
Line Editor
[ 0]  class ArrayList:
[ 1]      def __init__(self):
[ 2]          self.items = []
[ 3]
[ 4]      def insert(self, pos, elem) : self.items.insert(pos, elem)
[ 5]      def delete(self, pos) : this.items.pop(pos)
[ 6]      def isEmpty(self) : return self.size() == 0
```

그림 15 내용변경 후 출력, 파일 저장 후 파일읽기

self를 this(빨간색 동그라미)로 변경 후, 출력했을 때와 파일 저장 후 파일 읽기를 했을 때 모두 변경 사항이 적용된 것을 볼 수 있습니다.

```
[메뉴선택] i-입력, d-삭제, r-변경, p-출력, l-파일읽기, s-저장, q-종료=> d
 삭제행 번호: 0
[메뉴선택] i-입력, d-삭제, r-변경, p-출력, l-파일읽기, s-저장, q-종료=> p
Line Editor
[ 0]      def __init__(self):
[ 1]          self.items = []
[ 2]
[ 3]      def insert(self, pos, elem) : self.items.insert(pos, elem)
[ 4]      def delete(self, pos) : this.items.pop(pos)
[ 5]      def isEmpty(self) : return self.size() == 0
[메뉴선택] i-입력, d-삭제, r-변경, p-출력, l-파일읽기, s-저장, q-종료=> q

Process finished with exit code 0
```

그림 16 삭제 후 출력, 종료