

[22.11.05] 인터프리터IDE2

# TS인터프리터IDE2

김 태 현



# 문 제

줄 단위로 명령을 수행하는 인터프리터 언어 IDE를 구현한다.

- 명령어는 “변수1,변수2,...,변수N=식1,식2,...,식N” 의 형태로 존재하며 **우변의 모든 수식의 계산이 완료된 후 대입한다.**
- 변수는 A,B,...,Z 총 26개이며 식은 0을 포함한 한자리 수, 사칙연산(+-\*/\*), 변수, 괄호로 이루어져 있다.  
연산자 우선순위는 괄호가 가장 높고, **사칙연산의 우선순위는 같다.**
- 중간 단계를 포함한 모든 연산 결과
  - 1) 정수형으로 계산(소수점 버림)
  - 2) 0으로 나누는 경우는 0으로 처리
  - 3) 0보다 작은 경우는 0으로 9,999보다 큰 경우는 10,000으로 나눈 나머지로 계산

## void init()

IDE창이 비워지고 커서는 1번 줄에 위치  
1번 줄을 비어 있으면 모든 변수 값은 0

## int addCommand(char mCommand[200])

커서는 그대로 두고, 커서가 위치한 줄부터 모든 명령을 한 줄씩 내린 후  
커서 위치에 명령 입력 후 커서 줄 번호 반환  
좌변에 동일한 변수가 중복되거나 문법적 오류는 없음이 보장  
mCommand 길이 : 3 ~ 199

\*모든 함수는 각각 5,000번 이내로 호출된다.

## int moveCursor(int mPos)

명령을 수행하거나 되돌리면서 커서를 mPos 만큼 이동  
첫 줄 또는 마지막 빈 줄에 도달하면 멈춤

## void eraseCommand()

커서가 있는 줄의 명령을 지우고 밑의 명령들을 위로 올림  
커서가 빈 줄에 있다면 무시

## int getValue(char mVariable)

mVariable 값을 반환  
mVariable : 'A' ~ 'Z'

# 필요한 로직

## 1. 라벨 별 명령 관리

- 어떤 구조로 관리할지
- 어떤 정보를 저장할지

## 2. 명령 수행

# 라인 별 명령 관리

- 명령 삽입/삭제 : 각 5,000회
- 최대 명령 개수 5,000개

## 1. 라인 관리

- *list*  
명령 삽입/삭제 비용 :  $O(1)$   
커서 관리 : iterator  
커서의 줄 번호 : 따로 전역변수로 관리
- *array, vector*  
명령 삽입/삭제 비용 :  $O(n)$   
커서 관리 : index  
커서의 줄 번호 : index+1

어떤 자료구조여도 상관 없어 보인다

## 2. 라인 별 필요한 정보

- 명령 문자열
- 이 명령을 수행했을 때의 변수 값  
or  
이전 명령을 수행했을 때의 변수 값

# 라인 별 명령 관리 addCommand

- 라인 관리 : list
- 라인별 사용하는 정보  
: 명령 문자열, 이 명령을 수행했을 때의 변수 값

```
struct Data {  
    string cmd;  
    int val[26];  
};
```

before addCommand("B=(2\*A+5)/B+C")

line	command	A	B	C
0	-	0	0	0
1	A,C=5,7+4*7	5	0	77
2	A,B,C=1,3,5			
3	li.end()			

\* line0 을 추가해 놓으면  
예외처리 없이 line1을 계산할 수 있다.



# 라인 별 명령 관리

addCommand

- 라인 관리 : list
- 라인별 사용하는 정보  
: 명령 문자열, 이 명령을 수행했을 때의 변수 값

```
struct Data {  
    string cmd;  
    int val[26];  
};
```

after addCommand("B=(2\*A+5)/B+C")

line	command	A	B	C
0	-	0	0	0
1	A,C=5,7+4*7	5	0	77
2	<b>B=(2*A+5)/B+C</b>			
3	A,B,C=1,3,5			
4	li.end()			



# 라인 별 명령 관리 eraseCommand

- 라인 관리 : list
- 라인별 사용하는 정보  
: 명령 문자열, 이 명령을 수행했을 때의 변수 값

```
struct Data {  
    string cmd;  
    int val[26];  
};
```

before eraseCommand()

line	command	A	B	C
0	-	0	0	0
1	A,C=5,7+4*7	5	0	77
2	A,B,C=1,3,5			
3	li.end()			




# 라인 별 명령 관리 eraseCommand

- 라인 관리 : list
- 라인별 사용하는 정보  
: 명령 문자열, 이 명령을 수행했을 때의 변수 값

```
struct Data {  
    string cmd;  
    int val[26];  
};
```

after eraseCommand()

line	command	A	B	C
0	-	0	0	0
1	A,C=5,7+4*7	5	0	77
2	li.end()			

 cur



# 라인 별 명령 관리

moveCommand

- 라인 관리 : list
- 라인별 사용하는 정보  
: 명령 문자열, 이 명령을 수행했을 때의 변수 값

```
struct Data {  
    string cmd;  
    int val[26];  
};
```

before moveCommand(-1)

line	command	A	B	C
0	-	0	0	0
1	A,C=5,7+4*7	5	0	77
2	B=(2*A+5)/B+C			
3	A,B,C=1,3,5			
4	li.end()			



# 라인 별 명령 관리

moveCommand

- 라인 관리 : list
- 라인별 사용하는 정보  
: 명령 문자열, 이 명령을 수행했을 때의 변수 값

```
struct Data {  
    string cmd;  
    int val[26];  
};
```

after moveCommand(-1)

line	command	A	B	C
0	-	0	0	0
1	A,C=5,7+4*7	5	0	77
2	B=(2*A+5)/B+C			
3	A,B,C=1,3,5			
4	li.end()			



기존 line1 에 있던 값이 의미가 없어진다.

# 라인 별 명령 관리

moveCommand

- 라인 관리 : list
- 라인별 사용하는 정보  
: 명령 문자열, 이 명령을 수행했을 때의 변수 값

```
struct Data {  
    string cmd;  
    int val[26];  
};
```

before moveCommand(2)

line	command	A	B	C
0	-	0	0	0
1	A,C=5,7+4*7			
2	B=(2*A+5)/B+C			
3	A,B,C=1,3,5			
4	li.end()			



# 라인 별 명령 관리

moveCommand

- 라인 관리 : list
- 라인별 사용하는 정보  
: 명령 문자열, 이 명령을 수행했을 때의 변수 값

```
struct Data {  
    string cmd;  
    int val[26];  
};
```

on moveCommand(2)

line	command	A	B	C
0	-	0	0	0
1	A,C=5,7+4*7	5	0	77
2	B=(2*A+5)/B+C			
3	A,B,C=1,3,5			
4	li.end()			



# 라인 별 명령 관리

moveCommand

- 라인 관리 : list
- 라인별 사용하는 정보  
: 명령 문자열, 이 명령을 수행했을 때의 변수 값

```
struct Data {  
    string cmd;  
    int val[26];  
};
```

after moveCommand(2)

line	command	A	B	C
0	-	0	0	0
1	A,C=5,7+4*7	5	0	77
2	B=(2*A+5)/B+C	5	77	77
3	A,B,C=1,3,5			
4	li.end()			



# 라인 별 명령 관리

getValue

- 라인 관리 : list
- 라인별 사용하는 정보  
: 명령 문자열, 이 명령을 수행했을 때의 변수 값

```
struct Data {  
    string cmd;  
    int val[26];  
};
```

getValue('C') = 77 : cur 이전 line의 value 값을 출력해주면 된다.

line	command	A	B	C
0	-	0	0	0
1	A,C=5,7+4*7	5	0	77
2	B=(2*A+5)/B+C	5	77	77
3	A,B,C=1,3,5			
4	li.end()			



# 명령 수행

- 위에서 보듯이 moveCursor(양수) 인 경우만 한 줄씩 명령을 수행하며 내려가면 된다.

## 1) 명령에 여러 개인 식을 파싱

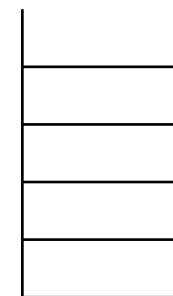
- 변수는 index0부터 2씩 증가하며 확인
- 식은 '=' 이후부터 ',' 단위로 확인

## 2) 하나의 식을 계산하여 대입

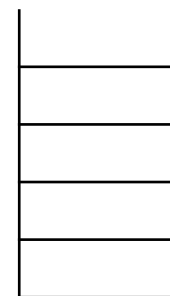
- 괄호만 우선순위가 높고, 사칙연산은 모두 동일하다.
- 연산자 스택, 정수 스택 두개를 활용한다.
  - 정수** : 정수 스택에 push
  - 변수** : prev에서 변수 값을 정수 스택에 push
  - '('** : 연산자 스택에 push
  - )'** : '(' 나올 때까지 연산 진행 후 '('도 pop
  - 사칙연산** : '(' 나올 때까지 연산 진행 후 연산자 스택에 push

prev →  
cur →

line	command	A	B	C
0	-	0	0	0
1	A,C=5,7+4*7	5	0	77
2	B=3+(2*A+5)/B+C			
4	li.end()			



연산자스택



정수스택

감사합니다

