자료구조 실습 보고서

[제 07 주] 스택 - 수식 계산

제출일	2017/04/30
학 번	201000287
소 속	일어일문학과
이 름	유다훈

201000287 일어일문학과 유다훈

1 프로그램 설명서

1 주요 알고리즘 및 자료구조

- 알고리즘
  - 입출력
    - ◆ 키보드로부터 infix 수식(흔히 사용하는 수식표기)를 입력받는다.
    - ◆ 수식 표현에 오류는 없다고 가정한다.
      - i 연산자:+, -, *, /, %, ^ (정수연산)
      - ii 0~9 의 한 자리 숫자문자
    - ◆ 매번 하나의 수식을 입력 받을 때 마다 다음의 일을 한다.
    - ◆ - infix 수식을 Postfix 수식으로 변환
      - i 맨 처음 빈 스택은 출력하지 않는다.
      - ii 연산자를 만날 때 마다, 처리 후 스택 내용을 출력한다
      - iii 최종 변환된 수식을 출력한다.
    - ◆ - Postfix 의 수식의 계산
      - i 맨 처음 빈 스택은 출력하지 않는다.
      - ii 스택이 변할 때 마다 그 내용을 출력한다.
      - iii 연산값을 만나서 스택에 넣은 다음, 연산자를 만나 계산이 이루어진 직후 출력
      - iv 최종 계산 값을 출력한다.
    - ◆ !를 입력하면 프로그램을 종료한다.

- 자료구조
  - 입력 받는 infix 형태의 수식을 저장할 char 형 배열
  - infix 형태의 수식을 Postfix 형태로 변환하여 저장할 char 형 배열
  - infix 수식을 Postfix 형태로 변환하는 과정에서 사용할 Character 형 Stack
  - Postfix 형태에서 계산해야할 값을 저장하고, 결과 값을 저장하는 Double 형 Stack
  - Stack 형태의 자료구조를 구현하기 위한 Array 형태의 List
  - 알림 및 오류 등의 메세지를 처리하는 enum

1 함수 설명서

Class	AppController			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	AppController()	없음	없음	_appView 변수와 _calculate 변수, AppController 변수들을 초기화하는 생성자 메소드
	run()	없음	없음	학생들의 성적순 정렬 프로그램을 실행시키는 메소드
	void showMessage(MessageID aMessageID)	에러 메세지 혹은 알림	없음	에러 메세지 혹은 알림에 따라 메세지를 출력하는 메소드
	void evalExpression()	없음	없음	infix 수식을 Postfix 수식으로 변환 후 수식의 계산값을 출력지시하는 메소드

201000287 일어일문학과 유다훈

Class	AppView			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	AppView()	없음	없음	생성자 메소드 값을 입력받는 스캐너 생성
	String inputString()	없음	문자열값	입력 받은 문자열을 리턴하는 메소드
	void outputMessage(String aMessage)	문자열	없음	전달 받은 문자열을 출력하는 메소드
	String inputExpression()	없음	문자열	수식을 입력받아 리턴하는 메소드
	void outputResult(double aValue)	결과값	없음	전달받은 최종값을 출력하는 메소드
	void outputPostfix(String aPostfix)	Postfix 값	없음	infix 수식을 postfix 값으로 변환한 값을 전달받아 출력하는 메소드

Class	interface Stack<T>			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	boolean push(T anElement)	원소	boolean	스택에 원소를 삽입하고자 할 때 사용할 메소드
	T pop()	없음	없음	스택의 맨 위의 원소를 빼내어 삭제할 때 사용할 메소드
	T peek()	없음	없음	스택의 맨 위의 원소를 출력하고자 할 때 사용할 메소드

Class	ArrayList<T> implements Stack<T>			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	ArrayList()	없음	없음	디폴트값으로 스택의 최대 용량과 스택 역할을 할 배열의 최대 값을 초기화하는 생성자 메소드
	ArrayList(int givenCapacity)	스택의 최대 용량 값	없음	입력 받은 값으로 스택의 최대 용량과 스택 역할을 할 배열의 최대값을 초기화 하는 생성자 메소드
	boolean isEmpty()	없음	boolean	스택이 비어있으면 true, 비어있지 않으면 false 를 리턴하는 메소드
	boolean isFull()	없음	boolean	스택이 꽉 차 있다면 true, 그렇지 않다면 false
	int size()	없음	스택 내 원소의 갯수	스택 내부의 원소의 총 갯수 출력
	boolean push(T anElement)	원소	스택에 삽입하면 true	스택이 꽉 차있는지 검사하고, 그렇지않으면 스택에 원소를 추가
	T pop()	없음	원소	스택 내 가장 위의 원소 순번을 줄이고 해당 원소를 반환

	T peek()	없음	원소	스택 내 가장 위의 원소를 리턴
	T elementAT(int aPosition)	원소가 존재하는 위치	원소	입력받은 위치에 존재하는 스택의 원소를 반환

Class	Calculate			
Method	메소드	파라미터 설명	리턴값	메소드 설명
	Calculate()	없음	없음	Calculate 클래스의 private 변수들을 초기화시키는 생성자 메소드
	void setInfix(String newInfix)	입력받은 보통의 수식(infix 값)	없음	char 형 배열에 입력받은 수식을 넣는 메소드
	String infix()	없음	문자열	char 형 배열을 String 형으로 변환하여 반환
	String postfix()	없음	문자열	char 형 배열을 String 형으로 변환하여 반환
	boolean infixToPostfix()	없음	변환이 잘 되면 true	infix 형태의 수식을 postfix 형태로 바꾸는 메소드
	double evalPostfix()	없음	계산 결과값	postfix 형태의 수식을 계산하여 결과값을 출력하는 메소드
	boolean isDigit(char aToken)	character 형 한 글자	true or false	전달받은 한 글자가 숫자이면 true, 아니면 false 리턴
	int inComingPrecedence(char aToken)	character 형 한 글자	우선순위	전달받은 연산자의 우선순위를 리턴
	int inStackPrecedence(char aToken)	character 형 한 글자	우선순위	스택 내부에서의 연산자의 우선순위를 리턴
	void showOStackkAll()	없음	없음	infix 를 postfix 로 고친 후 스택의 저장된 값을 출력
	void showVStackAll()	없음	없음	postfix 로 고친 수식의 계산값이 저장된 스택의 내용을 출력

2 종합 설명서

- 평범한 수식을 입력한다.
  - 3+5 를 입력했을 때
- InfixToPostfix()메소드를 이용하여 연산자가 따로 저장되는 OStack 의 내용을 출력하며 Postfix 로 변한 수식을 출력한다.
  - 입력받은 수식이 char 형 배열이 되어 저장되어있는 Caculate 의 _infix 로부터 배열 원소 1 개씩 받아와서 비교를 해본다.
  - 비교할 값이 isDigit()을 이용하여 참일 경우, 그것은 숫자이므로 Postfix 수식을 저장하는 char 형 배열 _postfix 에 저장한다.

- 거짓일 경우 그것은 연산자 이므로 다른 연산을 수행한다.
  - ◆ 만약 닫는 괄호 ‘)’ 라면
    - i 연산자를 저장하는 OStack 이 비어있지 않다면 스택으로부터 값을 한 개 뽑아낸다.
    - ii 스택에서 뽑아낸 연산자가 여는 괄호 ‘(‘ 일때까지 뽑아낸 연산자를 _postifx 에 저장하고 뽑아내고를 반복한다.
  - ◆ 닫는 괄호 이외의 다른 연산자들이라면
    - i 연산자의 스택으로 들어가는 순위를 확인한다
    - ii 만약 스택이 비어있지 않다면
    - iii 스택의 가장 위 연산자를 확인해본다.
    - iv 들어갈 스택보다 스택 내부 가장 위 연산자의 우선순위가 크거나 같다면, 스택으로부터 연산자를 하나 뽑아 _postfix 배열에 넣는다.
    - v 다시 맨 위의 스택의 우선순위를 반복한다.
    - vi 들어갈 우선순위가 더 높거나 스택이 비어있으면 스택에 바로 집어넣는다.
  - ◆ 이후 infix 배열 만큼 while 문이 돌았다면, infix 수식은 다 변환된 것이므로 스택에 남은 나머지 연산자들을 모두 _postfix 에 넣는다.
    - i 위 절차에 따라 3+5 는 35+가 됨.

- 이후 다시 변환된 Postfix 수식을 evalPostifx()를 이용하여 수식을 계산한다.
  - While 문을 postfix 수식이 들어있는 _postfix 배열의 길이만큼 실행한다.
  - 만약 _postfix 에서 뽑아낸 값이 숫자라면
    - ◆ String 형태로 고치고 이것을 다시 Double 형태로 바꾸어 연산값을 넣을 _vStack 에 저장한다.
  - 그렇지 않다면 _vStack 으로 부터 값을 두 개를 뽑아낸다.
    - ◆ 뽑아낸 연산자에 따라 적절한 계산을 하고 그 값을 다시 _vStack 에 저장한다.
  - 만약 스택 내부에 값이 1 개만 남아있으면 그것은 수식 계산의 결과 값이므로 while 문을 탈출한다.
  - While 문이 도는 동안 계속해서 _vStack 의 내용값을 출력한다.
  - While 문이 끝나면 스택에 마지막 남은 결과값을 출력한다.
    - ◆ 35+는 8.0 이다. (Double 형)
- ‘!’(느낌표)가 입력되면 프로그램을 종료한다.

2 프로그램 장단점 분석

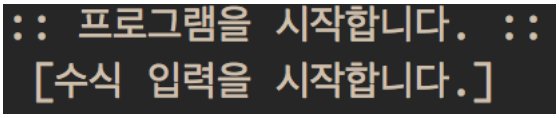
- 장점
  - List 형태의 Stack 자료구조를 구현할 수 있다.
  - 제네릭타입을 이용하여 똑같은 Stack 구조이지만, 여러 자료형태를 사용할 수 있다.
  - Infix 수식이 Postfix 로 변환되는 과정을 볼 수 있다.
  - Postfix 가 계산되는 과정을 볼 수 있다.
  - 수식의 우선순위(괄호)를 따져가며 곱셉과 나눗셈 혹은 괄호 내부의 우선적으로 계산되어야 할 값을 처리할 수 있다.

- 단점
  - 프로그램을 이해하는데 시간이 걸린다.

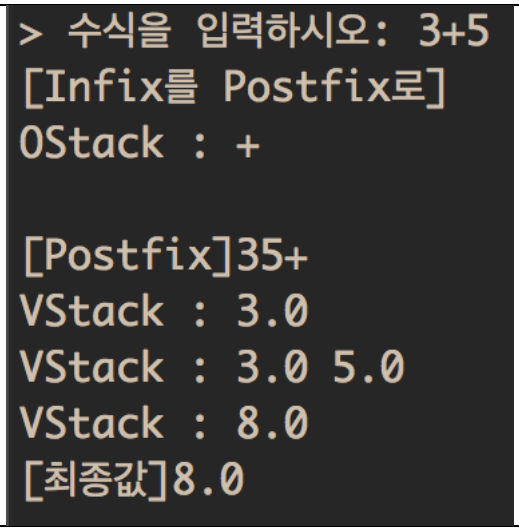
3 실행 결과 분석

1 입력과 출력

프로그램 실행



입출력 1 - 3+5 / 변환 후 수식 계산



입출력 2 - (9/(6-3)) / 변환 후 수식 계산

```
> 수식을 입력하시오: (9/(6-3))
[Infix를 Postfix로]
OStack : (
OStack : ( /
OStack : ( / (
OStack : ( / ( -
OStack : ( /
OStack :

[Postfix]963-/
VStack : 9.0
VStack : 9.0 6.0
VStack : 9.0 6.0 3.0
VStack : 9.0 3.0
VStack : 3.0
[최종값]3.0
```

입출력 3 – 9/6-3 / 수식 변환 후 출력

```
> 수식을 입력하시오: 9/6-3
[Infix를 Postfix로]
OStack : /
OStack : -

[Postfix]96/3-
VStack : 9.0
VStack : 9.0 6.0
VStack : 1.5
VStack : 1.5 3.0
VStack : -1.5
[최종값]-1.5
```

입출력 4 - (8-6)*(2+5*(7-4)) / 수식 변환 후 계산

```
> 수식을 입력하시오: (8-6)*(2+5*(7-4))
[Infix를 Postfix로]
OStack : (
OStack : ( -
OStack :
OStack : *
OStack : * (
OStack : * ( +
OStack : * ( + *
OStack : * ( + * (
OStack : * ( + * ( -
OStack : * ( + *
OStack : *

[Postfix]86-2574-*+*
VStack : 8.0
VStack : 8.0 6.0
VStack : 2.0
VStack : 2.0 2.0
VStack : 2.0 2.0 5.0
VStack : 2.0 2.0 5.0 7.0
VStack : 2.0 2.0 5.0 7.0 4.0
VStack : 2.0 2.0 5.0 3.0
VStack : 2.0 2.0 15.0
VStack : 2.0 17.0
VStack : 34.0
[최종값]34.0
```

프로그램 종료

```
> 수식을 입력하시오: !
[수식 입력을 종료합니다.]
:: 프로그램을 종료합니다. ::
```

2 결과 분석

- Last-In-First-out 개념의 Stack 자료구조를 구현하여 원소를 추가하면 차곡차곡 쌓이고, 원소를 꺼내려면 제일 위의 원소를 꺼내는 형태의 프로그램을 구현하였다.
- 이 특징을 이용하여 우리가 평소 익숙한 infix 형태의 수식을 컴퓨터에게 적합한 postfix 형태의 수식으로 바꾸어볼 수 있다.
- 한 가지의 스택을 구현해놓고, 제네릭 타입으로 선언하여 똑같은 구조의 여러개의 다른 자료형의 스택을 구현할 수 있다.
- Postfix 수식에서 수식을 저장한 문자배열에서 연산자를 얻으면 이것을 메소드를 통해서 애 Double 형으로 바꾸어서 스택에 넣는 방법을 알 수 있다.

- 생각해 볼 점
  - 보다 일반적인 계산기를 작성하려면?
    - i 처음에 문자열을 입력받는 것이 아니라, 정수 혹은 실수형태의 값을 한 번씩 입력받아서 처리해야하지 않을까?
    - ii 현재 상태는 문자열을 입력받아 문자 하나하나 char 형태로 끊는 형태로, 만일 실수를 입력한다면 . 이 들어가게되는 상태가 발생한다.
    - iii 부호가 붙은 수의 입력은 괄호를 이용하여 입력한다면 입력할 수 있지 않을까?

- 결론
  - Stack 구조를 구현해 볼 수 있다.
  - Infix 수식을 postfix 형태의 수식으로 변환할 수 있다.
  - 숫자를 계산할 수 있다.
  - 지금 형태의 자료구조로는 실제 계산기를 제작하기에는 무리가 있다.