System Programming

Assignment : Infix Notation Calculator

Note

> **중위 표기법**(中位表記法, 영어: infix notation)은 산술 및 논리학 공식과 문장에서 일반적으로 사용되는 표기법이다. 연산자를 피연산자 사이에 배치하는 것이 특징이며, 이 위치에 있는 연산자를 "중위 연산자"라고 한다. 예를 들어 2 **+** 2에서의 더하기 기호가 중위 연산자에 해당한다.
>
> **역폴란드 표기법**(RPN, reverse Polish notation) 또는 **후위 표기법(후치 표기법)**(後位 -, postfix notation)은 연산자를 연산 대상의 뒤에 쓰는 연산 표기법이다.
>
> 예를 들어, (2 + 3) * 4를 역폴란드 표기법으로 쓰면 다음과 같다.
>
> 2 3 + 4 *
>
> 이와 같은 방식은 수식을 계산할 때 특별한 변환이 필요없이, 수식을 앞에서부터 읽어 나가면서 스택에 저장하면 된다는 장점이 있다. 또한, 중위 표기법에서는 연산자의 우선순위가 모호해서 괄호가 필요한 경우가 있지만, 역폴란드 표기법에서는 그러한 문제점이 발생하지 않는다. 그러나 인간의 눈으로 쉽게 이해하거나 계산하기 힘들다는 문제점이 있어 눈에 보이는 표기보다는 주로 프로그램 내부의 표기법으로 사용한다.

Goal

본 과제의 목표는 **콘솔을 통해 중위표기법 수식을 입력받아, 이를 계산한 결과를 출력하는 프로그램을 작성하는 것**입니다.

Example

중위표기식을 입력받아, 후위표기식으로 변환하고, 이를 계산한 결과를 출력하는 예시

```
2+3*4  # infix notation
234*+  # postfix notation
14     # result
```

```
8-3+5*4  # infix notation
83-54*+  # postfix notation
25       # result
```

Background

중위표기법에서 후위표기법으로의 변환

1. 먼저, 하나의 스택(Stack)을 준비한다.
2. 중위표기식을 왼쪽부터 순차적으로 읽는다.
3. 읽은 문자가 ***피연산자(operand, 숫자)***일 경우, 그대로 출력한다.
4. 읽은 문자가 ***연산자(operator)***일 경우, 다음 기준에 따라 처리한다:
    a. 스택이 비어있다면, 연산자를 스택에 push한다.
    b. 스택의 top에 있는 연산자의 우선순위가 현재 연산자보다 낮다면, 연산자를 스택에 push한다.
    c. 그렇지 않은경우, 위 조건(a, b)이 만족될 때까지 스택에서 연산자를 pop하여 출력한 뒤, 현재 연산자를 push 한다.
5. 모든 입력을 처리한 후, 스택에 남아 있는 연산자를 모두 pop하여 출력한다.
6. 연산자의 우선순위는 다음과 같다

- +, - (덧셈, 뺄셈) < *, /(곱셈, 나눗셈)

후위표기식의 계산

1. 먼저, 하나의 스택을 준비한다.
2. 후위표기식을 왼쪽부터 순차적으로 읽는다.
3. 읽은 문자가 ***피연산자(operand, 숫자)***일 경우, 스택에 push한다.
4. 읽은 문자가 ***연산자(operator)***일 경우, 스택에서 피연산자 두 개를 pop하여 연산을 수행한다.

- 이때, 먼저 pop한 값을 두 번째 피연산자로, 나중에 pop한 값을 첫번째 피연산자로 사용해야 한다.

5. 계산 결과는 다시 스택에 push한다.
6. 모든 입력이 처리되면, 스택에는 최종 결과값이 남아있게 된다.

음수 고려 안해도됨

Requirement / Submission

- 프로그램은 **SicTools에서 시뮬레이션 가능한 asm 파일**로 작성하여 제출합니다.
- 코드에는 필히 주석이 포함되어야합니다.
- 과제 제출 시 다음 사항을 준수하세요.
    - 제출 기한 : 4월 9일 (수) 11:59 PM
    - asm 파일을 자신의 학번을 이름으로하여 제출한다.

제출파일 예시

- 202224210.asm

Grading Guidline

- **출력되는 후위표기식과 계산 결과는 개행 문자로 구분되어야 합니다.**
  - 자세한 형식은 문서 상단의 예시(* 주석 제외)를 참고하세요.
- **모든 출력이 끝난 후에는 HALT 구조 등을 사용하여 프로그램의 오동작을 방지하는 것을 권장합니다.**
- 본 과제의 평가는 **프로그램이 주어진 테스트케이스를 정확히 통과하는지 여부**를 기준으로 이루어집니다. 아래의 테스트케이스 예시를 참고하세요.

중위표기	후위표기	계산결과
2+3*5	235*+	17
8*2-4	82*4-	12
9+7-4*3	97+43*-	4
5*5+3-2	55*3+2-	26
6+2*3-1	623*+1-	11
7-2+3*4	72-34*+	17
1+5+2*4	15+24*+	14
8*3-5+6	83*5-6+	25

입출력만 제대로되면 됨,
while로 구현까지 안해도됨
중위표기 표시 x, 후위랑 결과만 표기
공백 고려 x
백그라운드 따라할 필요 x
코드효율성 체크 x-> 구현 못했으면 보고서에서 체크
sicTools/tests/linker/facotiral/print 를 보면 (이외에도 참고할거많음)16진수에서 10진수 아스키코드로 변환해서 출력하는거 참고

⚠️ **주의: 실제 채점에 사용될 테스트케이스는 공개하지 않습니다.**

또한, 과제 수행에 집중할 수 있도록 아래와 같은 편의 조건들이 적용됩니다. 음수, 괄호 X.

- 입력 버퍼의 크기는 자유롭게 설정할 수 있습니다. 다만, 테스트케이스는 **최대 20바이트 이내의 입력**으로 준비될 예정입니다.
- **모든 피연산자는 한 자릿수의 양의 정수**입니다.
- **괄호는 수식에 포함하지 않습니다.**
- 연산자의 종류는 덧셈(+), 뺄셈(-), 곱셈(*)으로 한정합니다.
- **계산결과는 세자리 수를 초과하지 않습니다.**
- 모든 테스트케이스는 **정상적인 입력과 결과를 보장하므로**, 별도의 예외 처리는 필요하지 않습니다.

⚠️ **다음 사항에 해당할 경우, 프로그램의 실행 여부와 관계없이 감점됩니다.**

- **주석 없이 코드만 제출한 경우**: 예외없이 **0점 처리**
- **지각 제출 시: 1시간당 0.5점 감점**
- **제출 형식 미준수 시: 0.5점 감점**
- **부정행위 적발 시 : 0점**

Appendix (Hint)

1. **스택(Stack) 구현** extdef:외부변수 참고 처럼 필요 없는 부분은 조금 생략하고 구현
   - 스택 구현은 기존에 제공된 PPT 파일을 참고하세요.
   - 또한, SicTools 폴더의 tests/linker/factorial/stack.asm 예시도 참고하면 도움이 됩니다.
2. **A 레지스터 오염 방지**
   - RD, LDCH처럼, 레지스터 **A의 오른쪽 1바이트만을 로드**하는 명령을 사용할 때는, 기존 A 레지스터 값에 의해 오염이 발생할 수 있습니다.
   - 해당 명령을 사용하기 전에, 'CLEAR A'와 같은 명령을 통해 **A 레지스터를 초기화**하는 방식을 활용할 수 있습니다.
3. **C 언어 활용**
   - **SIC/XE 어셈블리**로 직접 구현하기가 어렵다면, 먼저 **C 언어로 프로그램을 작성**해보세요.
   - 이후, **C 코드**를 참고하면서 **SIC/XE**로 옮겨 쓰면 보다 쉽게 구현할 수 있습니다.
4. **추가 힌트** 한줄씩실행되는기능.
   - **단계별 디버깅**을 권장합니다. SicTools에서 **step 실행**을 활용하여 명령어 실행 후 레지스터 상태를 확인해 보세요.
   - 충분히 작은 단위(예: 입력, 스택 저장, 연산 처리 등)으로 기능을 나눈 뒤, **각 단계별로 동작을 확인**하면서 전체 프로그램을 완성하는 것을 권장합니다.
5. **HALT**

```
...
      J      HALT    . 프로그램의 로직이 끝나면 HALT로 JUMP

...
HALT    J      HALT    . SicTool이 해당 라인에서 멈추게됨
```

6. 참고할만한 사이트
   - 중위표기식 → 후위표기식 변환 사이트: https://www.calcont.in/Conversion/infix_to_postfix
   - 중위 표기법과 후위표기법에 대한 설명: https://todaycode.tistory.com/73

tab을 그냥 스페이스바로 진행하면 sictool이 정상적으로 작동안함