

Lab 04 (3/24) Postfix Evaluation

- arranged by TA Sangheon Yang

Due : ~2021.03.29.(Tue) 23:59, Late Submission : ~2021.03.30.(wed) 23:59

- **Postfix Evaluation Implementation (using Stack) - Push, Pop, Postfix, DeleteStack.**
- 아래의 사진과 같이 [input].txt 파일을 입력받아 Stack을 생성하고 편집하는 기능을 구현하고, **Postfix Evaluation** 기능 수행후 결과 값을 [output].txt 파일에 출력하여 저장한다.
- **Implement the function of creating and editing a Stack by receiving the [input].txt file as input as below and the output result of postfix evaluation is stored in the [output].txt file.**

```
4736%+*42/-9+23*-#
top numbers : 4 7 3 6 3 10 40 4 2 2 38 9 47 2 3 6 41
evaluation result : 41
```

<input>

- postfix expression (한 줄, 100자 이하의 character, 최대 길이 100자)
- Operand -> single-digit numbers(1,2,3,4,5,6,7,8,9), “0”은 포함되지 않음.
- Operator -> “+”, “-”, “*”, “/”, “%” 5 가지. (int 연산 이므로 “/” 시에는 나머지는 버림.)
- '#' -> End of Postfix Expression. (# 입력시 종료.)
- postfix expression (a single line, less or equal to the length of 100 character)
- Operand -> single-digit numbers(1,2,3,4,5,6,7,8,9), “0” is not included.
- Operator -> “+”, “-”, “*”, “/”, “%”. (Since all the operation is based on integer operation, when operating “/”, the remainder should be truncated.)
- '#' -> End of Postfix Expression. (# denotes the end of expression.)

<output>

- Stack의 상태가 변할 때마다 Top Element를 출력. -> (첫째줄)
- Postfix evaluation 최종결과 출력. -> (둘째줄)
- 각 상황에 맞는 에러메시지 출력. -> (둘째줄)
- Top Element of the stack should be printed out whenever the state of Stack has changed. -> (1st line)
- Final result of postfix evaluation should be printed out. -> (2nd line)
- Appropriate error messages for each situation. -> (2nd line)

<Structure & Function Format>

Structure	Function
<pre>typedef struct Stack{ int* key; int top; int max_stack_size; }Stack;</pre>	<pre>Stack* CreateStack(int max); void Push(Stack* S, int X); int Pop(Stack* S); int Top(Stack* S); void DeleteStack(Stack* S); int IsEmpty(Stack* S); int IsFull(Stack* S);</pre>

- 위 사진과 같은 Struct 구조체를 사용하셔야 합니다.
- 위 사진과 같은 함수들을 형식에 맞게 구현해주시면 됩니다.
- **Struct format above should be used for implementation**
- **Functions should be implemented in appropriate format as above.**

<File Name Format>

- [StudentID].c ex) 20XXXXXXXXXX.c

<Execution>

- gcc 20XXXXXXXXXX.c -o 20XXXXXXXXXX
- ./20XXXXXXXXXX [input_file_name] [output_file_name]
- !!! 꼭 제공되는 testcase로 실행시켜보시기 바랍니다!!!!,
- !!! Run your solution code with the provided test case above and check whether it works properly !!!

<Issue>

- 코드 작성시 주석을 적어주시기 바랍니다. 주석이 없는 경우 Cheating으로 간주될 수 있습니다.
 - 제공된 testcase는 채점 case에 포함됩니다. 모두 알맞게 나오는지 확인해보시기 바랍니다.
 - 파일 입출력은 `argv[]` 를 사용하여 구현해주시기 바랍니다.
 - 제출 마감 시간 이전의 가장 최신 버전의 commit을 기준으로 채점할 예정입니다.
 - 제출 파일과, 폴더 naming 은 꼭 지정된 형식으로 해주셔야 합니다.
 - Please write down the detailed comments when writing the code. If there is no comment, it might be considered cheating.
 - Provided test case is included in the test case for grading. Please check to see if it makes a proper result.
 - Do not use a fixed file name when inputting and outputting files, but implement it using `argv[]` as in skeleton code.
 - Scoring will be based on the latest version of commit before the deadline.
 - The names of the .c file and directory should be named in proper format.
-
- 출력시, 위 사진의 예시와 같은 형식으로 출력해주시면 됩니다. 모든 공백은 띄어쓰기 한칸입니다. 모든 출력 메시지의 알파벳은 소문자만 사용하여 출력 합니다.
->둘째줄의 경우, 줄바꿈이 된 후 EOF("\n", 사용)
 - Operation은 [(2nd popped value) OP (1st popped value)] 의 형식으로 수행해야 합니다. ->[**Stack : 9, 20(top) & Operator : % -> result : 9 % 20 = 9 (not 20 % 9)**]
 - **max_stack_size** 는 20 으로 고정시켜 주셔야 합니다.

- All the messages must be printed out according to the appropriate format as shown in the example above. you have to change the line for each error message. All spaces except for line change are one space. Only lowercase letters should be used for the alphabets in output messages.
 - >in the case of 2nd line, newline("\n") is added before EOF.
 - **Order of operand in operation is as follows :** [(2nd popped value) OP (1st popped value)] -> [Stack : 9, 20(top) & Operator : % -> result : 9 % 20 = 9 (not 20 % 9)]
 - **max_stack_size should be fixed to 20.**

<Error Message Format>

1. $/$, $\%$ 연산시 0으로 나누는 경우
(Divide by Zero in “ $/$ ” or “ $\%$ ”)
-> "error : invalid postfix expression, divide by zero!"
 2. Stack이 꽉 차있는 상태에서 Push 하는 경우
(Pushing element when stack is already full)
-> "error : invalid postfix expression, stack is full!"
 3. Stack이 비어있는 상태에서 Pop 하는 경우
(Popping element when stack is empty)
-> "error : invalid postfix expression, stack is empty!"
 4. Stack안의 원소가 1개보다 많은 상태에서 Expression이 종료되는 경우("#" 입력)
(When the expression ended with "#", while more than one elements left in the stack)
-> "error : invalid postfix expression, [num of El't in S] elements are left!"
----->[num of El't in S]은 종료되는 시점에 스택안에 남아있는 Element의 갯수를 의미.
----->[num of El't in S] is the number of elements left in the stack at the moment when expression ended with "#".
- 위 4종류의 에러 발생시, 에러메시지 출력후 프로그램이 종료되도록 구현하시면 됩니다.
- If the above 4 types of errors occur, the program should be terminated with printing out the appropriate error messages.

<Directory Format>

- 아래와 같이 git 프로젝트 폴더에 "lab04" 폴더 생성후, "lab04" 폴더 안에 "20XXXXXXXXXX.c" 파일을 위치시키시면 됩니다.
 - After creating the "lab04" directory in the git-project-directory as below, place the "20XXXXXXXXXX.c" file in the "lab04" directory.

2022_CSE2010_20XXXXXXX/ (GitLab project directory)

---...
---lab03/

2

1184

---lab04/

2

10