푸시 스왑

Swap_push가 자연스럽지 않기 때문에

요약: 이 프로
젝트는 가능한 가장 적은 수의 작업을 사용하여 제한된 지침 세트로 스택의 데이터를 정렬합니다. 성공하려면 다양한 유형의 알
고리즘을 조작하고 최적화된 데이터 정렬에 가장 적합한 솔루션(많은 솔루션 중에서)을 선택해야 합니다.

버전: 6

내용물

1장

머리말

- 씨

```
#include <stdio.h>

정수 메인(무효) {

        printf("안녕하세요, 세계\n"); 반환 0;

}
```

- ASM

```
cseg 세그먼트는
cs:cseg, ds:cseg org 100h main proc
jmp 데뷔 엉망 db 'Hello world!$' 데뷔를 가
정합니다.



mov dx, 오프셋 엉망 mov ah, 9 int
21h

오른쪽
메인 앤드 cseg 끝

엔드 메인
```

- 롤코드

```
HAI
에 STDIO가 있습니까?
보이는 "Hello World!"
KTHXBYE
```

- PHP

```
<?php
  echo "안녕하세요!"; ?>
```

- 브레인퍽

```
++++++++++[>++++++++>++++++++++>++++>+<<<<<-] >++.>+.+ ++++++..+++
+.>++.
<<+++++++++++++++.>.+++.------.--------.>+.>.
```

• 씨#

```
시스템 사용;

공개 클래스 HelloWorld {
    public static void Main() { Console.WriteLine("안
        녕하세요!");
    }
}
```

• HTML5

```
<!DOCTYPE HTML>
<html>
    <머리>
        <meta charset="utf-8"> <title>Hello
        World !</title> </head> <body> <p>Hello World !
    </p> </body> </html>
```

• 야슬

```
"안녕 세계!"
인쇄
```

• 오캠

```
let main() =
    print_endline "안녕하세요!"

허락하다 _ = 메인()
```

제2장

소개

푸시 스왑 프로젝트는 데이터를 정렬해야 하는 매우 간단하고 매우 간단한 알고리즘 프로젝트입니다 .

정수 값 세트, 스택 2개 및 명령어 세트를 마음대로 사용할 수 있습니다.
두 스택을 조작합니다.

네 목표? 인수로 받은 정수를 정렬하는 푸시 스왑 언어 명령어 로 구성된 가장 작은 프로그램을 계산하고 표준 출력에 표시하는 push_swap이라는 프로그램을 C로 작성하십시오 .

쉬운?

우리는 볼 것입니다...

제3장

목표

정렬 알고리즘을 작성하는 것은 항상 개발자의 여정에서 매우 중요한 단계입니다. 그것은 종종 복잡성의 개념을 처음 접합니다.

정렬 알고리즘과 그 복잡성은 면접에서 논의되는 고전적인 질문의 일부입니다. 언젠가는 직면해야 하므로 이러한 개념을 살펴보는 것이 좋습니다.

이 프로젝트의 학습 목표는 엄격함, C 사용 및 기본 알고리즘 사용입니다.
특히 복잡성에 중점을 둡니다.

값 정렬은 간단합니다. 그것들을 가장 빠른 방법으로 정렬하는 것은 덜 간단합니다. 특히 하나의 정수 구성에서 다른 정수 구성으로 가장 효율적인 정렬 솔루션이 다를 수 있기 때문입니다.

제4장

공통 지침

- 프로젝트는 C로 작성해야 합니다.
- 프로젝트는 규범에 따라 작성되어야 합니다. 보너스 파일/기능이 있는 경우 표준 검사에 포함되며 내부에 표준 오류가 있으면 0이 표시됩니다.
- 기능은 정의되지 않은 동작을 제외하고 예기치 않게 종료되어서는 안 됩니다(세그먼트 오류, 버스 오류, 이중 해제 등). 이 경우 프로젝트는 작동하지 않는 것으로 간주되고 평가 중에 0을 받습니다.
- 필요한 경우 모든 힙 할당 메모리 공간을 적절하게 해제해야 합니다. 누출 없음
  용납됩니다.
- 주제가 요구하는 경우 -Wall, -Wextra 및 -Werror 플래그를 사용하여 소스 파일을 필수 출력으로 컴파일하고 cc를 사용하는 Makefile을 제출해야 하며 Makefile은 다시 연결되어서는 안 됩니다.
- Makefile은 최소한 $(NAME), all, clean, fclean 및 규칙을 포함해야 합니다.
  답장.
- 프로젝트에 보너스를 제공하려면 Makefile에 규칙 보너스를 포함해야 합니다. 이 규칙은 프로젝트의 주요 부분에서 금지된 다양한 헤더, 라이브러리 또는 기능을 모두 추가합니다. 제목이 다른 것을 지정하지 않으면 보너스는 다른 파일 _bonus.{c/h}에 있어야 합니다. 필수 및 보너스 부분 평가는 별도로 수행됩니다.
- 프로젝트에서 libft를 사용할 수 있는 경우 해당 소스 및 관련 Makefile을 관련 Makefile과 함께 libft 폴더에 복사해야 합니다. 프로젝트의 Makefile은 해당 Makefile을 사용하여 라이브러리를 컴파일한 다음 프로젝트를 컴파일해야 합니다.
- 이 작업 이 제출되거나 채점되지 않더라도 프로젝트에 대한 테스트 프로그램을 만들 것을 권장합니다 . 귀하의 작업과 동료의 작업을 쉽게 테스트할 수 있는 기회를 제공합니다. 이러한 테스트는 방어 중에 특히 유용하다는 것을 알게 될 것입니다. 실제로, 방어하는 동안 자신의 테스트 및/또는 평가 중인 동료의 테스트를 자유롭게 사용할 수 있습니다.
- 작업을 할당된 git 저장소에 제출합니다. git 저장소에 있는 작업만 채점됩니다. 작업을 평가하기 위해 Deepthought가 할당되면 완료됩니다.

동료 평가 후. Deepthink의 채점 중 작업의 어떤 부분에서 오류가 발생하면 평가가 중지됩니다.

제5장

필수 부분

V.1 규칙

- 스택 이 2개 있습니다. 이름과 b.
- 처음에:
  - 스택 a에는 임의의 양의 음수 및/또는 양수가 포함됩니다.
    복제할 수 없는 것. ◦ 스택 b가 비어 있
    습니다.
- 목표는 오름차순 번호로 스택으로 정렬하는 것입니다. 이를 위해 다음 작업을 마음대로 사용할 수 있습니다.

sa (swap a): 스택 a의 맨 위에 있는 처음 2개 요소를 교환합니다.
요소가 하나만 있거나 없는 경우 아무 작업도 수행하지 않습니다.

sb (swap b): 스택 b의 맨 위에 있는 처음 2개의 요소를 교체합니다.
요소가 하나만 있거나 없는 경우 아무 작업도 수행하지 않습니다.

ss : sa와 sb를 동시에 사용합니다.

pa (푸시 a): b의 맨 위에 있는 첫 번째 요소를 가져와서 맨 위에 놓습니다.
b가 비어 있으면 아무 것도 하지 않습니다.

pb (푸시 b): 의 맨 위에 있는 첫 번째 요소를 가져 와서 b의 맨 위에 놓습니다.
비어 있으면 아무 것도 하지 마십시오.

ra (a 회전): 스택 a의 모든 요소를 1만큼 위로 이동합니다.
첫 번째 요소가 마지막 요소가 됩니다.

rb (회전 b): 스택 b의 모든 요소를 1만큼 위로 이동합니다.
첫 번째 요소가 마지막 요소가 됩니다.

rr : ra와 rb를 동시에

rra (역회전 a): 스택 a의 모든 요소를 1만큼 아래로 이동합니다.
마지막 요소가 첫 번째 요소가 됩니다.

rrb (역회전 b): 스택 b의 모든 요소를 1만큼 아래로 이동합니다.
마지막 요소가 첫 번째 요소가 됩니다.

rrr : rra와 rrb를 동시에 사용합니다.

V.2 예

이러한 명령 중 일부의 효과를 설명하기 위해 임의의 정수 목록을 정렬해 보겠습니다.
이 예에서는 두 스택이 모두 오른쪽에서 증가하는 것으로 간주합니다.

```
--------------------------------

초기화 및 b:
2
1
3
6
5
8

- -
ab
--------------------------------

실행: 1

2
3
6
5
8

- -
ab
--------------------------------

임원 pb pb pb: 6 3 5 2


8 1

- -
ab
--------------------------------

Exec ra rb(rr에 해당): 5 2

8 1 6
3

- -
ab
--------------------------------

Exec rra rrb(rrr와 동일):
6 3 5
2 8 1


- -
ab
--------------------------------

실행:
5 3 6
2
8 1

- -
ab
--------------------------------

단계별 실행: 1 2


3
5
6
8

- -
ab
--------------------------------
```

get의 정수는 12개 명령어로 정렬됩니다. 당신은 더 잘할 수 있습니까?

V.3 "push_swap" 프로그램

프로그램 이름	push_swap
파일 제출	Makefile, *.h, *.c NAME, all,
Makefile 인	clean, fclean, re stack a: 정수 목록
수 외부 함수.	
	• 읽기, 쓰기, malloc, 무료, 출구 • ft_printf 및 이와 동등한 것 당신은 코딩
Libft 승인	예
설명	스택 정렬

프로젝트는 다음 규칙을 준수해야 합니다.

- 소스 파일을 컴파일할 Makefile을 제출해야 합니다. 안된다
  다시 연결합니다.
- 전역 변수는 금지되어 있습니다.
- 정수 목록 형식의 스택을 인수로 사용하는 push_swap이라는 프로그램을 작성해야 합니다. 첫 번째 인수는 스택의 맨 위에 있어야 합니다(순서에 주의하십시오).
- 프로그램은 스택을 정렬할 수 있는 가장 작은 명령어 목록을 표시해야 하며 가장 작은 번호가 맨 위에 표시됩니다.
- 명령어는 '\n'으로 구분해야 하며 다른 것은 사용하지 않아야 합니다.
- 목표는 가능한 가장 적은 수의 작업으로 스택을 정렬하는 것입니다. 평가 프로세스 동안 프로그램에서 찾은 명령어의 수는 허용되는 최대 작업 수인 한도와 비교됩니다. 프로그램에 더 긴 목록이 표시되거나 숫자가 제대로 정렬되지 않은 경우 성적은 0이 됩니다.
- 매개변수가 지정되지 않은 경우 프로그램은 아무 것도 표시하지 않고 다음을 제공해야 합니다.
  다시 프롬프트합니다.
- 오류의 경우 표준 오류에 "오류" 다음에 '\n'을 표시해야 합니다.
  오류는 예를 들면 다음과 같습니다. 일부 인수가 정수가 아니거나 일부 인수가 정수보다 크거나 중복이 있습니다.

```
$>./push_swap 2 1 3 6 5 8
~에
삐
삐
삐삐
~에
pa
pa
pa
$>./push_swap 0 하나 2 3 오류 $>
```

평가 과정에서 제대로 확인하기 위해 바이너리가 제공됩니다.
당신의 프로그램.

다음과 같이 작동합니다.

```
$>ARG="4 67 3 87 23"; ./push_swap $ARG | 화장실 -l
       6
$>ARG="4 67 3 87 23"; ./push_swap $ARG | ./checker_OS $ARG 확인 $>
```

프로그램 checker_OS가 "KO"를 표시하면 push_swap이 발생했음을 의미합니다.
숫자를 정렬하지 않는 지침 목록과 함께.

checker_OS 프로그램은 인트라넷의 프로젝트 리소스에서 사용할 수 있습니다.

이 보너스 파트에서 작동 방식에 대한 설명을 찾을 수 있습니다.
문서.

제6장

보너스 파트

이 프로젝트는 단순성으로 인해 추가 기능을 추가할 여지가 거의 없습니다. 그러나 자신의 체커를 만드는 것은 어떻습니까?

체커 프로그램 덕분에
push_swap 프로그램에 의해 생성된 명령어 목록은 실제로 스택을 적절하게 정렬합니다.

VI.1 "체커" 프로그램

프로그램 이름 파일 제출	체커
Makefile 인수 외부 기능.	*.h, *.c 보너
	스
	스택 a: 정수 목록
	• 읽기, 쓰기, malloc, 무료, 출구 • ft_printf 및 이와 동등한 것 당신은 코딩
Libft 승인	예
설명	정렬 명령 실행

- 정수 목록으로 형식화된 스택을 인수로 사용하는 checker라는 프로그램을 작성하십시오. 첫 번째 인수는 스택의 맨 위에 있어야 합니다(순서에 주의하십시오). 인수가 제공되지 않으면 중지되고 아무 것도 표시하지 않습니다.

- 그런 다음 표준 입력에 대한 명령을 기다리고 읽습니다. 각 명령 뒤에 '\n'이 옵니다. 모든 명령어를 읽고 나면 프로그램은 인수로 받은 스택에서 명령어를 실행해야 합니다.

- 해당 명령어를 실행한 후 스택 a가 실제로 정렬되고 스택 b가 비어 있으면 프로그램은 표준 출력에서 "OK" 다음에 '\n'을 표시해야 합니다.

- 다른 모든 경우에는 표준 출력에 "KO" 다음에 '\n'이 표시되어야 합니다.
- 오류의 경우 표준 오류에 "오류" 다음에 '\n'을 표시해야 합니다 . 오류는 예를 들면 다음과 같습니다. 일부 인수는 정수가 아니거나 일부 인수가 정수보다 크거나 중복이 있거나 명령이 존재하지 않거나 형식이 잘못되었습니다.

```
$>./체커 3 2 1 0
라라
PB
~에
라라
pa
OK
$>./checker 3 2 1 0
~에
라라
pb
KO
$>./checker 3 2 하나 0 오류 $>./
checker ""
                    1
오류 $>
```

제공된 바이너리와 똑같은 동작을 재현할 필요는 없습니다. 오류를 관리하는 것은 필수이지만 인수를 구문 분석하는 방법을 결정하는 것은 사용자에게 달려 있습니다.

필수 부분이 PERFECT인 경우에만 보너스 부분이 평가됩니다. Perfect는 필수 부분이 완벽하게 수행되어 오작동 없이 작동함을 의미합니다. ALL을 통과하지 못한 경우

필수 요구 사항, 보너스 부분은 전혀 평가되지 않습니다.

제7장

제출 및 동료 평가

평소와 같이 Git 리포지토리에서 할당을 제출합니다. 방어 중에는 리포지토리 내부의 작업만 평가됩니다. 파일 이름이 올바른지 다시 한 번 확인하는 것을 주저하지 마십시오.

이러한 할당은 프로그램에서 확인되지 않으므로 필수 파일을 제출하고 요구 사항을 준수하는 한 원하는 대로 파일을 구성할 수 있습니다.