Push_swap（プッシュ・スワップ

Swap_pushが自然でないため

概要
このプロジェクトでは、限られた命令セットで、できるだけ少ないアクション数で、スタック上のデータをソートさせます。成功するためには、様々な種類のアルゴリズムを操作し、最適化されたデータソートのために（多くの中から）最も適切な解決策を選択する必要があります。

バージョン：6

内容

第一章 序文

- C

```
#include <stdio.h> （英語版のみ

int main(void)
{
    printf("hello, worldn");
    return 0;
}
```

- 空対地ミサイル

```
CSEGセグメント
cs:cseg, ds:cseg org 100h
とする。
メインプ
ロックジ
ャンプデ
ビュー
mess db 'Hello world!$'
debut:
mov dx, offset mess
mov ah, 9
int 21h
ret
メインエン
ドP セグ
エンドエ
ンドプ
ン
```

- LOLCODE

```
ハイ
CAN HAS STDIO?
ビジブル "Hello
world!"KTHXBYE
```

- PHP

```
<?php
 echo "ハローワールド！";
?>
```

- ブレインファック

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

- C#

```
を使用しています。

 public class HelloWorld { public
     static void Main () {
         Console.WriteLine("Hello world!");
     }
 }
```

- HTML5

```
<!DOCTYPE html>
<html>
     <head>
          <meta charset="utf-8">になります。
          <title>ハローワールド!</title
     </head>になります。
     <body>
          <p>Hello World !</p> <p>Hello World !
     </body>。
</html>。
```

- ヤスル

```
"ハローワール
ド！" print
```

- オーカムル

```
メインとする() =
     print_endline "Hello world !"

_ = main ()
```

第二章 はじめに

Push swapプロジェクトは、データをソートする必要があるという、非常にシンプルでわかりやすいアルゴリズムのプロジェクトです。

自由に使えるのは、整数値一式、2つのスタック、そして両方のスタックを操作する命令一式です。

あなたの目標は？引数として受け取った整数をソートする、*Push swap 言語*命令からなる最小のプログラムを計算し、標準出力に表示する push_swap というプログラムを C 言語で作成すること。

簡単？今

にわかる

よ...

第III章 目的

ソートアルゴリズムを書くことは、開発者にとって常に非常に重要なステップである。複雑さの概念との最初の出会いとなることが多いのです。

ソートアルゴリズムとその複雑さは、就職の面接で取り上げられる定番の質問の一つです。いずれは直面することなので、これらの概念に目を向けるのは良い機会でしょう。

このプロジェクトの学習目標は、厳密さ、C言語の使用、基本的なアルゴリズムの使用です。
特に、その複雑さに着目しています。

値の並べ替えは簡単です。しかし、最速でソートすることはそれほど簡単ではありません。特に、整数の構成によって、最も効率的なソート方法は異なることがあるからです。

第四章

共通事項

- プロジェクトはC言語で書かれている必要があります。
- あなたのプロジェクトはNormに従って書かれている必要があります。ボーナスファイル/関数がある場合、それらはノームチェックに含まれ、内部にノームエラーがある場合は0が返されます。
- 未定義の動作とは別に、関数が予期せず終了しないこと（セグメンテーションエラー、バスエラー、ダブルフリーなど）。このような場合、プロジェクトは非機能とみなされ、評価時に0点が与えられます。
- ヒープで確保されたメモリ空間は、必要なときに適切に解放されなければなりません。リークは許されません。
- 対象がそれを要求している場合、ソースファイルを要求された出力にコンパイルする Makefile を -Wall、-Wextra、-Werror フラグで提出し、cc を使用し、その Makefile は再リンクしてはいけません。
- Makefileには、少なくとも$(NAME)、all、clean、fclean、および
再
- プロジェクトにボーナスを回すには、Makefile にルールボーナスを含める必要があります。このルールボーナスは、プロジェクトのメイン部分で禁止されている様々なヘッダー、 librairies または関数をすべて追加します。ボーナスは、主体が何も指定しない場合は、別のファイル _bonus.{c/h} にする必要があります。必須部分とボーナス部分の評価は別々に行われます。
- プロジェクトで libft を使用できるようにする場合、そのソースと関連する Makefile を libft フォルダにコピーしておく必要があります。あなたのプロジェクトの Makefile は、その Makefile を使ってライブラリをコンパイルし、その後プロジェクトをコンパイルしなければなりません。
- この作品は**提出する必要はなく、採点もされませんが**、プロジェクトのためにテストプログラムを作成することをお勧めします。テストプログラムは、自分の作品や仲間の作品を簡単にテストする機会を与えてくれます。このようなテストは、特にデフェンスの際に役立つと思います。実際、審査では、自分のテストや審査する仲間のテストを自由に使用することができます。
- 指定された git リポジトリに作品を提出する。git リポジトリにある作品だけが採点対象となります。Deepthought があなたの作品の採点を担当する場合、採点は以下のように行われます。

Push_swapSwap_pushが自然でないため

ピアエバリュエーション後Deepthoughtの採点中に、あなたの作品のいずれかのセクションでエラーが発生した場合、評価は停止されます。

第五章 必須項目

V.1 のルール

- aとbという名前の2つのスタックを持っています。
- 冒頭で
  - スタックaには、複製不可能な負数および正数のランダムな量が含まれています。
  - スタックbは空である。
- そのためには、以下の操作を自由に行うことができます。

sa (swap a):スタック a の先頭の 2 要素を入れ替える。
要素が1つしかない場合、または要素がない場合は何もしない。

sb (swap b)。スタックbの先頭の2要素を入れ替える。
要素が1つしかない場合、または要素がない場合は何もしない。

ss : saとsbを同時に行う。

pa（aを押す）。bの先頭の要素を取り出し、aの先頭に置く。
bが空の場合は何もしない。

pb (push b)です。aの先頭の要素を取り出し、bの先頭に置く。
aが空の場合は何もしない。

ra (rotate a)。スタック a の全要素を 1 ずつシフトアップする。
最初の要素が最後の要素になる。

rb (rotate b)。スタック b の全要素を 1 ずつシフトアップする.
最初の要素が最後の要素になる。

rr : raとrbを同時に使用。

rra (reverse rotate a)。スタック a の全要素を 1 つ下にシフトします。
最後の要素が最初の要素になる。

rrb (reverse rotate b)。スタック b の全要素を 1 つずつシフトダウンする.
最後の要素が最初の要素になる。

rrr : rraとrrbを同時に使用。

V.2 例

これらの命令のいくつかの効果を説明するために、整数のランダムなリストをソートしてみましょう。この例では、両方のスタックが右から成長していくと考えます。

```
aとbの初期
値：2
1
3
6
5
8
__ a
b
```

```
エクセ
サー：1
2
3
6
5
8
__ a
b
```

```
Exec pb pb pb: 6
3
5 2
8 1
__ a
b
```

```
エグゼクティブラバー（
Rrと同等）：5 2
8 1
6 3
__ a
b
```

```
実行 rra rrb（rrrと同等）：
6 3
5 2
8 1
__ a
b
```

```
エクセ
サ：5 3
6 2
8 1
__ a
b
```

```
エグゼパパパ：
1
2
3
5
6
8
__ a
b
```

aからの整数が12命令でソートされる。あなたはもっとうまくできますか？

V.3 push_swap」プログラム

プログラム名	プッシュスワップ
ファイルを提出する	Makefile, *.h, *.c
メイクファイル	NAME、All、Clean、Fclean、Re
論証	stack a: 整数のリスト
外部機能。	• リード、ライト、マロック、フリー、エグジット • ft_printf と同等の YOU コード。
リブート認可	はい
商品説明	書庫の整理

あなたのプロジェクトは、以下のルールに従わなければなりません。

- ソースファイルをコンパイルするMakefileを提出する必要があります。再リンクはしてはいけません。
- グローバル変数の使用は禁止されています。
- 引数として、整数のリストとしてフォーマットされたスタックを受け取る push_swap という名前のプログラムを書かなければならない。最初の引数はスタックの一番上にあるべきである (順序に注意)。
- プログラムは、スタックをソートするために可能な限り小さな命令のリストを表示する必要があります
  aで、小さい数字が一番上になります。
- 指示は「'」で区切り、それ以外のものは使用しないでください。
- 目標は、可能な限り少ない操作回数でスタックをソートすることです。評価プロセスでは、あなたのプログラムによって見つかった命令の数が、制限、つまり許容される最大操作回数と比較されます。あなたのプログラムがより長いリストを表示するか、数字が適切にソートされていない場合、あなたの成績は0となります。
- パラメータが指定されない場合、プログラムは何も表示せず、プロンプトを返さなければならない。
- エラーが発生した場合は、標準エラーに「Error」の後に「♪」を表示する必要があります。エラーとは、例えば、引数が整数でない、引数が整数より大きい、重複している、などです。

```
$>./push_swap 2 1 3 6 5 8 sa
pb
pb
pb
sa
pa
pa
pa
pa
$>./push_swap 0 one 2 3
Error
$>
```

評価プロセスでは、お客様のプログラムを適切にチェックするために、バイナリが提供されます。

以下のように動作します。

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

もしプログラムチェッカー_OSが「KO」と表示したら、あなたのpush_swapが数字をソートしない命令リストを出してきたということです。

checker_OSプログラムは、イントラネットのプロジェクトのリソースで公開されています。
その仕組みは、本書のボーナスパートでご紹介しています。

第VI章 ボーナスパート

このプロジェクトは、そのシンプルさゆえに、余分な機能を追加する余地はほとんどありません。しかし、自分だけのチェッカーを作るのはいかがでしょうか？

このチェッカープログラムにより、以下のことが確認できます。
は、push_swap プログラムが生成する命令のリストが、実際にスタックを正しくソートすることを意味します。

VI.1 チェッカー」 プログラム

プログラム名	チェッカー
ファイルを提出する	***.h, *.c**
メイクファイル	ボーナス
論証	stack a: 整数のリスト
外部機能。	• リード、ライト、マロック、フリー、エグジット • ft_printf と同等の YOU コード。
リブート認可	はい
商品説明	仕分け指示の実行

- 引数として、整数のリストとしてフォーマットされたスタックを受け取るcheckerという名前のプログラムを書きなさい。最初の引数はスタックの一番上にあるべきである(順序に注意)。引数が与えられない場合、プログラムは停止し、何も表示しない。
- その後、待機して標準入力の命令を読み込みます。各命令の後には'␊'が続きます。すべての命令が読み込まれたら、プログラムは引数として受け取ったスタック上で命令を実行しなければなりません。

- これらの命令を実行した後、実際にスタックaがソートされ、スタックbが空になった場合、プログラムは標準出力に「OK」の後に「 \n 」と表示しなければなりません。
- それ以外の場合は、標準出力に "KO "の後に"„¯¯"と表示すること。
- エラーの場合は、"Error "の後に"„¯¯"**を表示する必要があります**。エラーとは、例えば、引数が整数でない、引数が整数より大きい、重複している、命令が存在しない、書式が正しくない、などです。

```
$>./checker 3 2 1 0
rra
pb
sa
rra
pa
OK
$>./checker 3 2 1 0
sa
RR
A
PB
KO
$>./checker 3 2 one 0
Error
$>./checker "" 1
エラー
$>
```

提供されたバイナリと全く同じ挙動を再現する必要はありません。エラーを管理することは必須ですが、引数をどのように解析するかはあなた次第です。

ボーナスパーツは、必須パーツがPERFECTである場合にのみ査定されます。パーフェクトとは、必須パートが統合的に行われ、誤動作することなく動作することを意味します。必須条件をすべてクリアしていない場合、ボーナスパーツの評価は一切行われません。

第七章

提出と相互評価

通常通り、Git リポジトリに課題を提出する。リポジトリ内の作品だけが、ディフェンスで評価されます。ファイル名が正しいかどうか、遠慮なく再確認してください。

　これらの課題は、プログラムによる検証は行われませんので、必須ファイルを提出し、要件を満たしていれば、自由にファイルを整理してください。