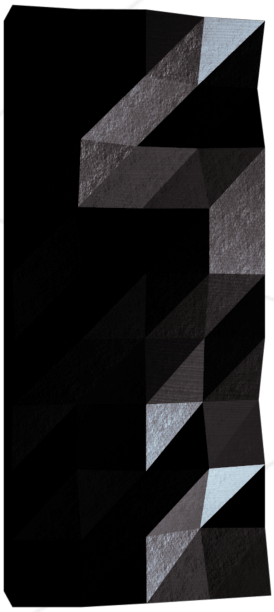

Projet plutôt d'algo

push_swap

42 staff staff@42.fr

Résumé: Coucou, tu veux voir ma pile ?

Table des matières

Chapitre I

Préambule

Hello world !

- C

```
#include <stdio.h>

int main(void)
{
    printf("hello, world\n");
    return 0;
}
```

- ASM

```
cseg segment
assume cs:cseg, ds:cseg
org 100h
main proc
jmp debut
mess db 'Hello world!$'
debut:
mov dx, offset mess
mov ah, 9
int 21h
ret
main endp
cseg ends
end main
```

- LOLCODE

```
HAI
CAN HAS STDIO?
VISIBLE "HELLO WORLD!"
```

```
  KTHXBYE
```

- PHP

```
<?php
 echo "Hello world!";
 ?>
```

- BrainFuck

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

- C#

```
using System;

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

- HTML5

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

Chapitre II

Description du jeu

- Le jeu est constitué de 2 piles nommées `a` et `b` (si vous savez pas ce qu'est une pile, Google est votre ami).
- Au départ `b` est vide et `a` contient un certain nombre de nombres positifs ou négatifs (sans doublons).
- Le but du jeu est de faire en sorte que `a` contienne les mêmes nombres mais dans l'ordre croissant.
- Pour ce faire, on ne dispose que des opérations suivantes :
  - `sa` : swap a - intervertit les 2 premiers éléments au sommet de la pile a.
    (ne fait rien s'il n'y en a qu'un ou aucun).
  - `sb` : swap b - intervertit les 2 premiers éléments au sommet de la pile b.
    (ne fait rien s'il n'y en a qu'un ou aucun).
  - `ss` : sa et sb en même temps.
  - `pa` : push a - prend le premier élément au sommet de b et le met sur a.
    (ne fait rien si b est vide).
  - `pb` : push b - prend le premier élément au sommet de a et le met sur b.
    (ne fait rien si a est vide).
  - `ra` : rotate a - décale d'une position tous les élements de la pile a.
    (vers le haut, le premier élément devient le dernier).
  - `rb` : rotate b - décale d'une position tous les élements de la pile b.
    (vers le haut, le premier élément devient le dernier).
  - `rr` : ra et rb en meme temps.
  - `rra` : reverse rotate a
    (vers le bas, le dernier élément devient le premier).
  - `rrb` : reverse rotate b
    (vers le bas, le dernier élément devient le premier).
  - `rrr` : rra et rrb en même temps.

Chapitre III

Exemples

- Les piles a et b seront definies ainsi :
  `a:` 8 5 6 3 1 2
  `b:`

- sa
  `a:` 8 5 6 3 2 1
  `b:`

- pb pb pb
  `a:` 8 5 6
  `b:` 1 2 3

- ra rb (on peut donc aussi dire rr)
  `a:` 6 8 5
  `b:` 3 1 2

- rra rrb (on peut donc aussi dire rrr)
  `a:` 8 5 6
  `b:` 1 2 3

- sa
  `a:` 8 6 5
  `b:` 1 2 3

- pa pa pa
  `a:` 8 6 5 3 2 1
  `b:`

Chapitre IV

Sujet

- Vous devez faire un programme qui prend en paramètre la pile `a` sous la forme d'une liste de nombres. Le premier paramètre est au sommet de la pile (attention donc à l'ordre).
- Le programme doit afficher la suite d'opérations qui permet de trier la pile, le plus petit nombre étant au sommet. Les opérations seront affichées separées par un espace, pas d'espace au debut ni à la fin, le tout suivi d'un '\n'.
- Le but est de trier la pile avec le moins d'opérations possibles.

```
$./push_swap 2 1 3 6 5 8
sa pb pb pb sa pa pa pa
$
```

En cas d'erreur, vous afficherez "Error" suivi d'un '\n' sur la sortie d'erreur. Par exemple, certains paramètres ne sont pas des nombres, ou ne tiennent pas dans un *int* (qui est le type d'un élément sur les piles), ou encore il y a des doublons.

Chapitre V

Sujet - Partie bonus

Les bonus ne seront pris en compte que si votre partie obligatoire est pratiquement complète.

Voici quelques idées de bonus intéressants à réaliser, voire même utiles. Vous pouvez évidemment ajouter des bonus de votre invention, qui seront évalués à la discrétion de vos correcteurs.

- Ajout d'options pour le debug : -v peut afficher l'etat des piles à chaque coup, -c peut faire afficher en couleur la dernière action, etc... .

Chapitre VI

Consignes

- Ce projet doit respecter les contraintes listées ici.
- La programme doit s'appeller `push_swap`.
- Vous devez coder en C sous MacOS et avoir un Makefile.
- Votre projet doit être à la Norme.
- Vous devez gérer les erreurs de façon sensible. En aucun cas votre programme ne doit quitter de façon inattendue (Segmentation fault, etc...)
- Vous devez rendre, à la racine de votre dépôt de rendu, un fichier `auteur` contenant votre login suivi d'un '\n' :

```
$>cat -e auteur
xlogin$
$>
```

- Vous avez le droit d'utiliser les fonctions suivantes :
  - write
  - malloc
  - free
  - exit
- Vous pouvez poser vos questions sur le forum, sur jabber, IRC, ...
- Bon courage à tous !