Push_swap

Parce que Swap_push n'est pas aussi naturel

Résuméÿ:
Ce projet vous fera trier des données sur une pile, avec un ensemble limité d'instructions, en utilisant le plus petit nombre d'actions possible. Pour réussir, vous devrez manipuler différents types d'algorithmes et choisir la solution la plus appropriée (parmi plusieurs) pour un tri optimisé des données.

Modèle : 5

Contenu

Chapitre I

Avant-propos

- C

```
#include <stdio.h>

int principal(vide) {

        printf("bonjour, monde\n"); renvoie
        0ÿ;
}
```

- 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

à droite
endp principal
cseg se termine
fin principal
```

- CODE LOL

```
HAI
PEUT A STDIO?
VISIBLE "BONJOUR LE MONDEÿ!"
KTHXBYE
```

- PHP

```
<?php
  echo "Bonjour le mondeÿ!"ÿ; ?
>
```

- BrainFuck

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

• C#

```
utiliser le systèmeÿ;

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

•ÿHTML5

```
<!DOCTYPE html>
<html>
      <tête>
            <meta charset="utf-8">
            <title>Bonjour le mondeÿ!</title> </head>
      <body> <p>Bonjour le mondeÿ!</p> </body>
      </html>
```

• YASL

```
"Bonjour le monde!"
imprimer
```

•ÿOCaml

```
let main() =
      print_endline "Bonjour le monde !"

laisser _ = principal ()
```

Chapitre II

introduction

Le projet Push_swap est un projet d'algorithme très simple et très efficace : les données devront être triées. Vous avez à votre disposition un ensemble de valeurs int, 2 piles et un ensemble d'instructions pour manipuler les deux piles.

Ton but ? Écrivez un programme en C appelé push_swap qui calcule et affiche sur la sortie standard le plus petit programme utilisant le langage d'instruction Push_swap qui trie les arguments entiers reçus.

Facile?

On verra...

Chapitre III

Buts

Ecrire un algorithme de tri est toujours une étape très importante dans la vie d'un codeur, car c'est souvent la première rencontre avec la notion de complexité.

Les algorithmes de tri, et leurs complexités font partie des questions classiques abordées lors des entretiens d'embauche. C'est probablement le bon moment pour examiner ces concepts, car vous devrez les affronter à un moment donné.

Les objectifs d'apprentissage de ce projet sont la rigueur, l'utilisation du C et l'utilisation d'algorithmes de base. Surtout en regardant la complexité de ces algorithmes de base.

Le tri des valeurs est simple. Les trier le plus rapidement possible est moins simple, notamment parce que d'une configuration d'entiers à une autre, l'algorithme de tri le plus efficace peut différer.

Chapitre IV

Instructions générales

- Ce projet ne sera corrigé que par de véritables êtres humains. Vous êtes donc libre d'organiser et de nommer vos fichiers comme vous le souhaitez, bien que vous deviez respecter certaines exigences énumérées ci-dessous.
- Le fichier exécutable doit être nommé push_swap.
- Vous devez soumettre un Makefile. Ce Makefile doit compiler le projet et doit contenir les règles habituelles. Il ne peut recompiler le programme que si nécessaire.
- Si vous êtes malin, vous utiliserez votre bibliothèque pour ce projet, soumettez aussi votre dossier libft incluant son propre Makefile à la racine de votre dépôt. Votre Makefile devra compiler la bibliothèque, puis compiler votre projet.
- Les variables globales sont interdites.
- Votre projet doit être rédigé en C conformément à la Norme.
- Vous devez gérer les erreurs avec tact. En aucun cas votre programme ne peut se fermer de manière intempestive (Défaut de segmentation, erreur de bus, double free, etc).
- Aucun programme ne peut avoir de fuites de mémoire.
- Dans votre partie obligatoire, vous êtes autorisé à utiliser les fonctions suivantesÿ:

  ÿ écrire

  ÿ lire

  ÿ malloc

  ÿ gratuit

  ÿ sortie

- Vous pouvez poser des questions sur le forum & Slack...

Chapitre V

Partie obligatoire

V.1 Règles du jeu

- Le jeu est composé de 2 piles nommés a et b.
- Commencer avec:
  - ÿ la pile a contient une quantité aléatoire de nombres négatifs et/ou positifs qui ne peut pas être dupliqué.
  - ÿ b est vide
- Le but est de trier par ordre croissant les nombres dans la pile a.
- Pour ce faire, vous disposez des opérations suivantes :

sa : swap a - échange les 2 premiers éléments en haut de la pile a. Ne rien faire s'il y a n'est qu'un ou aucun élément).

sb : swap b - échange les 2 premiers éléments en haut de la pile b. Ne rien faire s'il y a n'est qu'un ou aucun élément).

ss : sa et sb en même temps.

pa : appuyez sur a - prenez le premier élément en haut de b et placez-le en haut de a. Ne rien faire si b est vide.

pb : appuyez sur b - prenez le premier élément en haut de a et placez-le en haut de b. Ne rien faire si a est vide.

ra : tourner a - décaler vers le haut tous les éléments de la pile a de 1. Le premier élément devient le dernier.

rb : rotate b - décale vers le haut tous les éléments de la pile b de 1. Le premier élément devient le dernier.

rr : ra et rb en même temps.

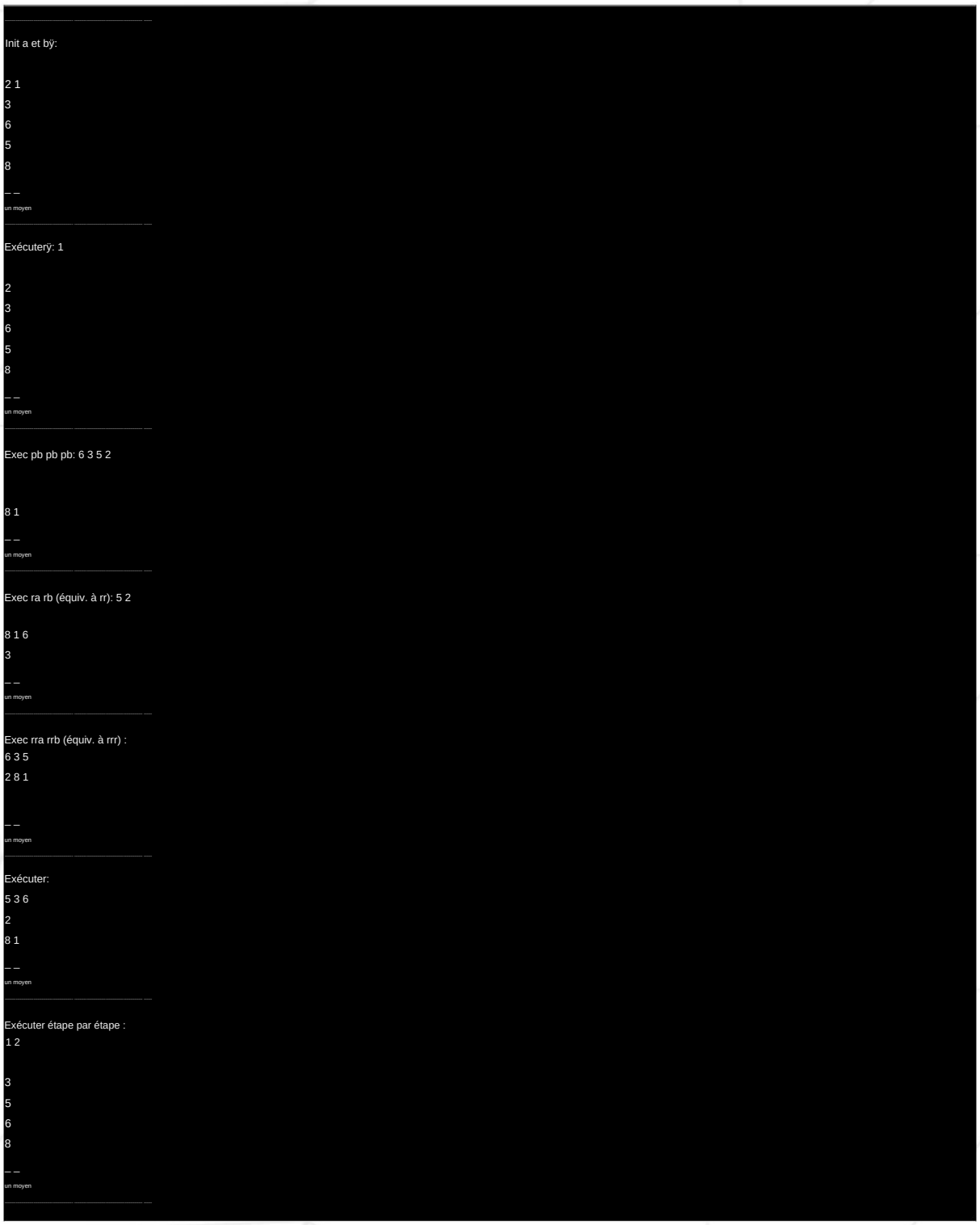
rra : reverse rotate a - décale vers le bas tous les éléments de la pile a de 1. Le dernier élément devient le premier.

rrb : reverse rotate b - décale vers le bas tous les éléments de la pile b de 1. Le dernier élément devient le premier.

rrr : rra et rrb en même temps.

V.2 Exemple

Pour illustrer l'effet de certaines de ces instructions, trions une liste aléatoire d'entiers.
Dans cet exemple, nous considérerons que les deux piles se développent à partir de la droite.



Cet exemple trie les entiers de a en 12 instructions. Pouvez-vous faire mieux ?

V.3 Le programme « push_swap »

- Vous devez écrire un programme nommé push_swap qui recevra en argument la pile a formatée comme une liste d'entiers. Le premier argument doit être en haut de la pile (attention à l'ordre).

- Le programme doit afficher la plus petite liste d'instructions possible pour trier la pile a, le plus petit nombre étant en haut.

- Les instructions doivent être séparées par un '\n' et rien d'autre.

- Le but est de trier la pile avec le minimum d'opérations possible.
  Lors de la soutenance, nous comparerons le nombre d'instructions trouvées par votre programme avec un nombre maximum d'opérations tolérées. Si votre programme affiche une liste trop longue ou si la liste n'est pas triée correctement, vous n'obtiendrez aucun point.

- Si aucun paramètre n'est spécifié, le programme ne doit rien afficher et donner la
  revenir en arrière

- En cas d'erreur, vous devez afficher Erreur suivi d'un '\n' sur l'erreur type.
  Les erreurs incluent par exempleÿ: certains arguments ne sont pas des entiers, certains arguments sont plus grands qu'un entier et/ou il y a des doublons.

```
$>./push_swap 2 1 3 6 5 8
à
pb
pb
pb
à
pa
pa
pa
$>./push_swap 0 un 2 3
Erreur $>
```

Pendant la soutenance, nous fournirons un binnaire pour bien vérifier votre programme. Cela fonctionnera comme suit :

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

Si le programme checker_OS affiche KO, cela signifie que votre push_swap est venu avec une liste d'instructions qui ne trie pas la liste. Le programme checker_OS est disponible dans les ressources du projet sur l'intranet. Vous trouverez dans la section bonus de ce document une description de son fonctionnement.

Chapitre VI

Partie bonus

Nous regarderons votre partie bonus si et seulement si votre partie obligatoire est EXCELLENTE.
Cela signifie que vous devez remplir la partie obligatoire, du début à la fin, et que votre gestion des erreurs doit être irréprochable, même en cas d'utilisation tordue ou mauvaise. Si ce n'est pas le cas, vos bonus seront totalement IGNORÉS.

Le projet Push_swap se prête peu à la création de bonus du fait de sa simplicité.
Cependant, que diriez-vous de créer votre propre vérificateur ?

VI.1 Le programme « checker »

- Écrivez un programme nommé checker, qui aura comme argument la pile a formatée comme une liste d'entiers. Le premier argument doit être en haut de la pile (attention à l'ordre). Si aucun argument n'est donné, le vérificateur s'arrête et n'affiche rien.

- checker va alors attendre et lire les instructions sur l'entrée standard, chaque instruction sera suivie de '\n'. Une fois toutes les instructions lues, checker va les exécuter sur la pile reçue en argument.

- Si après l'exécution de ces instructions, la pile a est réellement triée et b est vide, alors le vérificateur doit afficher "OK" suivi d'un '\n' sur la sortie standard. Dans tous les autres cas, le vérificateur doit afficher "KO" suivi d'un '\n' sur la sortie standard.

- En cas d'erreur, vous devez afficher Erreur suivi d'un '\n' sur l' **erreur type.**
  Les erreurs incluent par exemplеÿ: certains arguments ne sont pas des entiers, certains arguments sont plus grands qu'un entier, il y a des doublons, une instruction n'existe pas et/ou est mal formatée.

Grâce au programme de vérification, vous pourrez vérifier si la liste d'instructions que vous allez générer avec le programme push_swap est trier la pile correctement.

```
$>./vérificateur 3 2 1 0
ra
pb
à
ra
pa

OK $>./vérificateur 3 2 1 0
à
ra
pb

KO $>./checker 3 2 un 0
Erreur $>./checker ""
                        1
Erreur
$>
```

Vous N'AVEZ PAS à reproduire exactement le même comportement que le binaire que nous avons vous donnent. Il est obligatoire de gérer les erreurs mais c'est à vous de décider comment vous décidez d'analyser les arguments.

Chapitre VII

Soumission et correction par les pairs

Soumettez votre travail sur votre référentiel GiT comme d'habitude. Seul le travail sur votre référentiel sera noté.

Bonne chance à tous!