Push Swap

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

Introduction

Le projet Push swap est un projet d'algorithme très simple et très direct :
des données doivent être triées.
Vous avez à votre disposition un ensemble de valeurs entières, 2 piles, et un ensemble d'instructions pour manipuler les deux piles.
Votre objectif ? Écrire un programme en C appelé push_swap qui calcule et affiche sur la sortie standard le plus petit programme, constitué d'instructions du langage Push swap,
qui trie les entiers reçus en argument.
Facile ?
On verra bien...

Objectifs

L'écriture d'un algorithme de tri est toujours une étape très importante dans le parcours d'un développeur. C'est souvent la première rencontre avec le concept de complexité. Les algorithmes de tri et leur complexité font partie des questions classiques abordées lors des entretiens d'embauche.
C'est sans doute le moment de se pencher sur ces concepts, car vous y serez un jour ou l'autre confronté.
Les objectifs d'apprentissage de ce projet sont la rigueur, l'utilisation du C, et l'utilisation des algorithmes de base. En se concentrant particulièrement sur leur complexité.
Trier des valeurs est simple. **Les trier de la manière la plus rapide possible est moins simple**. En particulier car lors d'une configuration d'entiers à une autre, la

solution de tri la plus efficace peut
différer.

Instructions communes

- Votre projet doit être écrit en C.
- Votre projet doit être écrit conformément à la norme. Si vous avez des fichiers/fonctions bonus, ils sont inclus dans la vérification de la norme et vous recevrez un 0 s'il y a une erreur de norme à l'intérieur.
- Vos fonctions ne doivent pas s'arrêter de manière inattendue (erreur de segmentation, erreur de bus, double free, etc.) en dehors des comportements indéfinis. Si cela se produit, votre projet sera considéré comme non fonctionnel et recevra un 0 lors de l'évaluation.
- Tout l'espace mémoire alloué au tas doit être correctement libéré lorsque cela est nécessaire. Aucune fuite ne sera tolérée.
- Si le sujet le requiert, vous devez soumettre un Makefile qui compilera vos fichiers sources vers la sortie requise avec les drapeaux fichiers sources vers la sortie requise avec les drapeaux -Wall, -Wextra et -Werror, utiliser cc, et votre Makefile ne doit pas être relinké.
- Votre Makefile doit au moins contenir les règles $(NAME), all, clean, fclean et re.
- Pour remettre des bonus à votre projet, vous devez inclure une règle bonus à votre Makefile,
  qui ajoutera tous les différents en-têtes, librairies ou fonctions qui sont interdits sur la partie principale du projet. Les bonus doivent être dans un fichier différent _bonus.{c/h} si
  le sujet ne spécifie rien d'autre. L'évaluation de la partie obligatoire et de la partie bonus
  se fait séparément.
- Si votre projet vous permet d'utiliser votre libft, vous devez copier ses sources et son
  Makefile associé dans un dossier libft avec son Makefile associé. Le

Makefile de votre projet doit compiler la librairie en utilisant son Makefile, puis compiler le projet.

- Nous vous encourageons à créer des programmes de test pour votre projet même si ce travail n'aura pas à être soumis et ne sera pas noté. Cela vous donnera une chance
  de tester facilement votre travail et celui de vos camarades. Vous trouverez ces tests particulièrement utiles lors de votre soutenance. En effet, lors de la soutenance, vous êtes libre d'utiliser vos tests et/ou les tests du pair que vous évaluez.
- Soumettez votre travail au dépôt git qui vous a été attribué. Seul le travail dans le dépôt git sera noté. Si Deepthought est chargé de noter votre travail, il le fera de la manière suivante après vos évaluations par les pairs. Si une erreur se produit dans l'une des sections de votre travail pendant l'évaluation de Deepthought, l'évaluation s'arrêtera.

Partie obligatoire

V.1 Les règles

- Vous disposez de 2 piles nommées a et b.
- Au début :
  ◦ La pile a contient une quantité aléatoire de nombres négatifs et/ou positifs
  qui ne peuvent pas être dupliqués.
  ◦ La pile b est vide.
- Le but est de trier par ordre croissant les nombres dans la pile a. Pour ce faire, vous avez les
  opérations suivantes à votre disposition :
  sa (swap a) : Intervertir les 2 premiers éléments en haut de la pile a.
  Ne fait rien s'il n'y a qu'un seul ou aucun élément.
  sb (swap b) : Echanger les 2 premiers éléments au sommet de la pile b.
  Ne faites rien s'il n'y a qu'un seul ou aucun élément.
  ss : sa et sb en même temps.
  pa (push a) : Prendre le premier élément au sommet de b et le mettre au sommet de a.
  Ne fait rien si b est vide.

pb (push b) : Prend le premier élément au sommet de a et le met au sommet de b.
Ne fait rien si a est vide.
ra (rotate a) : Décale vers le haut tous les éléments de la pile a de 1.
Le premier élément devient le dernier.
rb (rotation de b) : Déplace 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 croissent à partir de la droite.

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

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

- -
a b
---------------------------------------------------------------------------------------------------
Exec ra rb (equiv. to rr):
5 2
8 1
6 3

- -
a b
---------------------------------------------------------------------------------------------------
Exec rra rrb (equiv. to rrr):
6 3
5 2
8 1

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

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

- -
a b
---------------------------------------------------------------------------------------------------
```

Les nombres entiers de a sont triés en 12 instructions. Pouvez-vous faire mieux ?

Nom du programme	push_swap

Fichiers à rendre	Makefile, *.h, *.c
Makefile	NAME, all, clean, fclean, re
Argument	stack a : une liste de int
Fonctions externes	- read, write, malloc, free, exit. - ft_printf et toute fonction codée
Libft autorisée	oui
Description	trier les piles

Votre projet doit respecter les règles suivantes :

- Vous devez fournir un Makefile qui compilera vos fichiers sources. Il ne doit pas relier.
- Les variables globales sont interdites.
- Vous devez écrire un programme nommé push_swap qui prend 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).**
- Le programme doit afficher la plus petite liste d'instructions possible pour trier la pile a, le plus petit nombre se trouvant en haut.
- Les instructions doivent être séparées par un '\n' et rien d'autre.
- Le but est de trier la pile avec le plus petit nombre d'opérations possible. Pendant l'évaluation, le nombre d'instructions trouvées par votre programme sera comparé à une limite : le nombre maximal d'opérations toléré. Si votre programme affiche une liste plus longue ou si les nombres ne sont pas triés correctement, votre note sera de 0.
- Si aucun paramètre n'est spécifié, le programme ne doit rien afficher et renvoyer l'invite
  en retour.
- En cas d'erreur, il doit afficher "Error" suivi d'un '\n' sur l'erreur standard.
  Les erreurs comprennent par exemple : certains arguments ne sont pas des

entiers, certains arguments sont plus grands qu'un nombre entier et/ou il y a des doublons.

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

Au cours du processus d'évaluation, un binaire sera fourni afin de vérifier correctement votre programme.
Il fonctionnera de la manière suivante :

```
$>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
$>
```

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

Soumission et évaluation par les pairs
Remettez votre travail dans votre dépôt Git comme d'habitude. Seul le travail contenu dans votre dépôt sera évalué lors de la soutenance. N'hésitez pas à vérifier les noms de vos fichiers
vos fichiers pour vous assurer qu'ils sont corrects.
Comme ces travaux ne sont pas vérifiés par un programme, n'hésitez pas à organiser vos fichiers comme vous le souhaitez, tant que vous rendez votre travail dans le dépôt Git.

comme vous le souhaitez, du moment que vous rendez les fichiers obligatoires et que vous respectez les exigences.

Liste chainées

Gestion d’erreur

Push_swap étapes