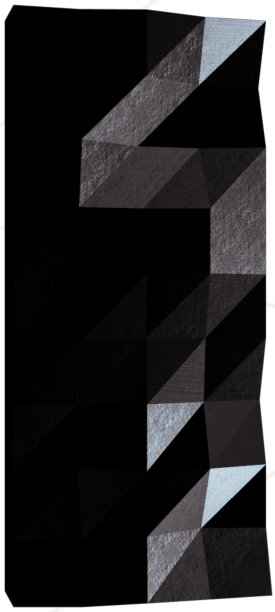

Push_swap

Parce que Swap_push, c'est moins naturel

Résumé:
Ce projet vous demande de trier des données dans une pile, en utilisant un set d'instructions limité, et avec le moins d'opérations possibles. Pour le réussir, vous devrez manipuler différents algorithmes de tri et choisir la (ou les ?) solution la plus appropriée pour un classement optimisé des données.

Version: 8.1

Table des matières

Chapitre I

Préambule

- C

```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
<?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>
```

- YASL

```
"Hello world!"
print
```

- OCaml

```
let main () =
   print_endline "Hello world !"

let _ = main ()
```

Chapitre II

Introduction

Le projet **Push swap** est un exercice d'algorithmie simple et efficace : il faut trier de la donnée.

Vous avez à votre disposition un ensemble d'entiers, deux piles et un ensemble d'instructions pour manipuler celles-ci.

Votre but ? Écrire un programme en C nommé `push_swap` qui calcule et affiche sur la sortie standard le plus petit programme, fait d'instructions du *langage Push swap*, permettant de trier les entiers passés en paramètres.

Facile ?

Et bien, c'est ce qu'on va voir...

Chapitre III

Objectifs

Écrire un algorithme de tri est toujours une étape importante dans la vie d'un programmeur débutant car il s'agit souvent de la première rencontre avec la notion de complexité.

Les algorithmes de tri et leur complexité font parti des grands classiques des entretiens d'embauche. C'est donc l'occasion rêvée pour vous pencher sérieusement sur la question car soyez certains que cela vous sera demandé.

Les objectifs de ce projet sont la rigueur, la pratique du C et l'usage d'algorithmes élémentaire. En particulier, la complexité de ces algorithmes élémentaire.

Trier des valeurs c'est simple. Les trier le plus vite possible, c'est moins simple vu que, d'une configuration des entiers à trier à une autre, un même algorithme de tri n'est pas forcément le plus efficace...

Chapitre IV

Règles communes

- Votre projet doit être écrit en C.
- Votre projet doit être codé à la Norme. Si vous avez des fichiers ou fonctions bonus, celles-ci seront inclues dans la vérification de la norme et vous aurez 0 au projet en cas de faute de norme.
- Vos fonctions ne doivent pas s'arrêter de manière inattendue (segmentation fault, bus error, double free, etc) mis à part dans le cas d'un comportement indéfini. Si cela arrive, votre projet sera considéré non fonctionnel et vous aurez 0 au projet.
- Toute mémoire allouée sur la heap doit être libéré lorsque c'est nécessaire. Aucun leak ne sera toléré.
- Si le projet le demande, vous devez rendre un Makefile qui compilera vos sources pour créer la sortie demandée, en utilisant les flags `-Wall`, `-Wextra` et `-Werror`. Votre Makefile ne doit pas relink.
- Si le projet demande un Makefile, votre Makefile doit au minimum contenir les règles `$(NAME)`, `all`, `clean`, `fclean` et `re`.
- Pour rendre des bonus, vous devez inclure une règle `bonus` à votre Makefile qui ajoutera les divers headers, librairies ou fonctions qui ne sont pas autorisées dans la partie principale du projet. Les bonus doivent être dans un fichier différent : `_bonus.{c/h}`. L'évaluation de la partie obligatoire et de la partie bonus sont faites séparément.
- Si le projet autorise votre `libft`, vous devez copier ses sources et son Makefile associé dans un dossier libft contenu à la racine. Le Makefile de votre projet doit compiler la librairie à l'aide de son Makefile, puis compiler le projet.
- Nous vous recommandons de créer des programmes de test pour votre projet, bien que ce travail **ne sera pas rendu ni noté**. Cela vous donnera une chance de tester facilement votre travail ainsi que celui de vos pairs.
- Vous devez rendre votre travail sur le git qui vous est assigné. Seul le travail déposé sur git sera évalué. Si Deepthought doit corriger votre travail, cela sera fait à la fin des peer-evaluations. Si une erreur se produit pendant l'évaluation Deepthought, celle-ci s'arrête.

Chapitre V

Partie obligatoire

V.1 Les règles

- Le jeu est constitué de 2 piles nommées `a` et `b`.
- Au départ :
  - La pile `a` contient une quantité aléatoire de négatif et/ou des nombres positifs qui ne peuvent pas être dupliqués.
  - La pile `b` est vide.

- Le but du jeu est de trier les nombres de la pile `a` par ordre croissant. Pour ce faire, vous disposez des instructions 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 vers le haut tous les élements de la pile `a`. Le premier élément devient le dernier.

  rb (`rotate b`) : Décale d'une position vers le haut tous les élements de la pile `b`. Le premier élément devient le dernier.

  rr : `ra` et `rb` en même temps.

  rra (`reverse rotate a`) : Décale d'une position vers le bas tous les élements de la pile `a`. Le dernier élément devient le premier.

  rrb (`reverse rotate b`) : Décale d'une position vers le bas tous les élements de la pile `b`. Le dernier élément devient le premier.

  rrr : `rra` et `rrb` en même temps.

V.2 Exemple

Pour illustrer l'effet de ces instructions, trions une liste d'entiers. Dans cet exemple, nous considérerons que les deux piles se développent 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
--------------------------------------------------------------------------------------------------------
```

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

V.3 Le programme "push_swap"

Nom du programme	`push_swap`
Fichiers de rendu	`Makefile, *.h, *.c`
Makefile	`NAME, all, clean, fclean, re`
Arguments	`pile a : une liste d'entiers`
Fonctions externes autorisées	• `read, write, malloc, free, exit` • `ft_printf et tout équivalent que VOUS avez codé`
Libft autorisée	`Oui`
Description	`Trier les piles`

Votre projet doit respecter les règles suivantes :

- Vous devez rendre un `Makefile` qui compilera vos fichiers sources. Il ne doit pas `relink`.
- Les variables globales sont interdites.
- Vous devez écrire un programme nommé `push_swap` qui prend en paramètre la pile `a` sous la forme d'une liste d'entiers. Le premier paramètre est au sommet de la pile (attention donc à l'ordre).
- Le programme doit afficher un programme composé de la plus courte suite d'instructions possible qui permet de trier la pile `a`, le plus petit nombre étant au sommet de la pile.
- Les instructions doivent être separées par un '\n' et rien d'autre.
- Le but est de trier les entiers avec le moins d'opérations possible. En évaluation, le nombre d'instructions calculé par votre programme sera comparé avec un nombre d'opérations maximum toléré. Si votre programme sort un programme trop long, ou si la liste d'entiers n'est pas triée, vous aurez 0.
- Si aucun paramètre n'est spécifié, le programme ne doit rien afficher et rendre l'invite de commande.
- En cas d'erreur, vous devez afficher `"Error"` suivi d'un '\n' sur la **sortie d'erreur**. Par exemple, si certains paramètres ne sont pas des nombres, ne tiennent pas dans un `int`, ou encore, s'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
$>
```

Pendant l'évaluation, un binaire sera fourni afin de tester votre programme correctement.

Il fonctionnera ainsi :

```
$>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` calcule un programme qui ne trie pas la liste.

Ce programe checker_OS est disponible dans les ressources du projet sur l'intranet. Le détail de son fonctionnement est présenté dans la partie bonus de ce document.

V.4 Performance

Pour valider ce projet, vous devez réaliser certains tris avec un nombre minimal d'opérations :

- Pour une **validation minimaliste** (et une note de 80), vous devez pouvoir **trier 100 nombres aléatoires en moins de 700 opérations**. À noter ici que nous parlons d'une moyenne, mais de manière générale, vous devriez toujours avoir au maximum moins de 700 opérations pour effectuer le tri complet.
- Pour une **validation optimale et maximale du projet** et donc pouvoir accéder aux bonus, vous devez remplir la condition ci-haut, mais aussi trier **500 nombres aléatoires**, le tout devrait se faire en un **maximum de 5500 opérations**.
- Le tout sera vérifié lors de votre évaluation.

`Si vous voulez compléter la partie bonus, chaque étape de performance du projet (benchmark) doit recevoir la plus haute note possible.`

Chapitre VI

Partie bonus

Ce projet se prête peu à la création de bonus de par sa simplicité. Cependant, que diriez-vous de créer votre propre `checker` ?

Grâce au programme checker, vous allez pouvoir vérifier que la liste d'instructions générée par le programme push_swap trie bien la pile passée en paramètre.

La partie bonus ne sera évaluée que si la partie obligatoire est PARFAITE. Parfaite signifie que la partie obligatoire du projet a été faite en totalité et fonctionne sans problèmes. Dans ce projet, cela signifie de réussir tous les indicatifs de performance avec une note parfaite, sans exception. Si vous ne passez pas TOUTES les exigences à la perfection, votre partie bonus sera ignorée.

VI.1 Le programme "checker"

Nom du programme	`checker`
Fichiers de rendu	`*.h, *.c`
Makefile	`bonus`
Arguments	`pile a : une liste d'entiers`
Fonctions externes autorisées	• `read, write, malloc, free, exit` • `ft_printf et tout équivalent que VOUS avez codé`
Libft autorisée	`Oui`
Description	`Exécuter les instructions de tri`

- Vous devez écrire un programme nommé `checker` qui prend en paramètre la pile `a` sous la forme d'une liste d'entiers. Le premier paramètre est au sommet de la pile (attention donc à l'ordre). Si aucun argument n'est donné, le programme s'arrête et n'affiche rien.
- Il doit ensuite attendre et lire des instructions sur l'entrée standard, chaque instruction suivie par un '\n'. Une fois toutes les instructions lues, le programme va les exécuter sur la pile d'entiers passée en paramètre.
- Si à la suite de l'exécution la pile `a` est bien triée et la pile `b` est vide, alors le programme doit afficher `"OK"` suivi par un '\n' sur la sortie standard.
- Sinon, il doit afficher `"KO"` suivi par un '\n' sur la sortie standard.
- En cas d'erreur, vous devez afficher `"Error"` suivi d'un '\n' sur la **sortie d'erreur**. Par exemple, si certains paramètres ne sont pas des nombres, ne tiennent pas dans un `int`, s'il y a des doublons ou, bien sûr, si une instruction n'existe pas ou est mal formatée.

```
$>./checker 3 2 1 0
rra
pb
sa
rra
pa
OK
$>./checker 3 2 1 0
sa
rra
pb
KO
$>./checker 3 2 one 0
Error
$>
```

Vous N'AVEZ PAS à reproduire exactement le même comportement que le binaire qui est fourni. Il est obligatoire de gérer les erreurs mais c'est à vous de décider comment vous souhaitez analyser les arguments.

La partie bonus ne sera évaluée que si la partie obligatoire est PARFAITE. Parfaite signifie que la partie obligatoire du projet a été faite en totalité et fonctionne sans problèmes. Dans ce projet, cela signifie de réussir tous les indicatifs de performance avec une note parfaite, sans exception. Si vous ne passez pas TOUTES les exigences à la perfection, votre partie bonus sera ignorée.

Chapitre VII

Rendu et peer-evaluation

Rendez votre travail sur votre dépot `Git` comme d'habitude. Seul le travail présent sur votre dépot sera évalué en soutenance. Vérifiez bien les noms de vos dossiers et de vos fichiers afin que ces derniers soient conformes aux demandes du sujet.

Vu que votre travail ne sera pas évalué par un programme, organisez vos fichiers comme bon vous semble du moment que vous rendez les fichiers obligatoires et respectez les consignes du sujet.

file.bfe:VABB7yO9xm7xWXROeASsmsgnY0o0sDMJev7zFHhwQS8mvM8V5xQQp
Lc6cDCFXDWTiFzZ2H9skYkiJ/DpQtnM/uZ0