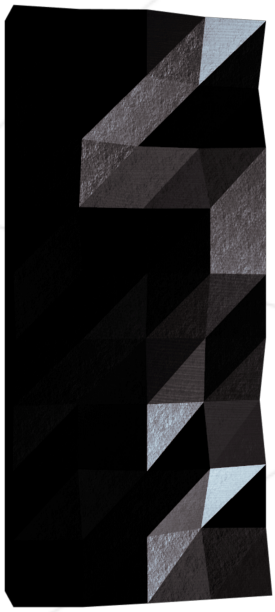

Push_swap

Swap_push no suena tan natural

Resumen: Este proyecto te hará ordenar datos en un stack, con un conjunto limitado de instrucciones, y utilizando el menor número de instrucciones. Para tener éxito, deberás trastear con algunos de los diversos algoritmos y elegir, entre todos, el más apropiado para un ordenamiento óptimo.

Versión: 5

Índice general

Capítulo I

Avance

- 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 ()
```

Capítulo II

Introducción

`Push_swap` es un proyecto de algoritmia simple y muy efectivo: tienes que ordenar datos. Tienes a tu disposición un conjunto de valores int, dos stacks y un conjunto de instrucciones para manipular ambos.

¿Que cuál es el objetivo? Simple: escribirás un programa en `C` al que llamarás `push_swap`. El programa calculará y mostrará en la salida estándar una lista de instrucciones escritas en lenguaje `push_swap`. Deberás buscar el número mínimo de instrucciones para ordenar el conjunto de enteros recibido como argumento.

¿A que es fácil? Bueno, eso ya lo veremos...

Capítulo III

Objetivos

Un importante paso en la vida de un desarrollador es escribir un algoritmo de ordenamiento. Probablemente sea la primera vez que te encuentres con el concepto de complejidad.

Los algoritmos de ordenamiento y su complejidad suponen una importante parte de las preguntas realizadas durante las entrevistas laborales. Es posiblemente un buen momento para echar un vistazo a estos conceptos porque te los reencontrarás en algún momento de tu vida.

Los objetivos a aprender son rigor, uso de `C`, y algo de algoritmia básica... Haciendo especial hincapié en la complejidad de este último punto.

Ordenar valores es simple. Ordenarlos de forma rápida es menos simple, especialmente porque de una configuración de enteros a otra, los algoritmos pueden diferir.

Capítulo IV

Instrucciones generales

- Este proyecto solo será evaluado por personas. Aunque deberás respetar los principios listados debajo, eres libre de organizar y llamar a tus archivos como mejor consideres.
- El nombre del ejecutable será `push_swap`.
- Debes entregar un `Makefile`. Ese `Makefile` deberá compilar el proyecto utilizando las reglas habituales. Solo recompilará el programa en caso de ser necesario.
- Si eres inteligente, utilizarás tu librería para este proyecto, haciendo entrega de la carpeta `libft` (`Makefile` incluído) en la raíz del repositorio. El `Makefile` de tu `push_swap` deberá compilar la librería, y luego tu proyecto.
- Las variables globales están prohibidas.
- Tu proyecto debe escribirse en `C` cumpliendo la Norma.
- Deberás gestionar los errores con cautela. Bajo ningún concepto tu programa debe terminar de forma inesperada (segmentation fault, bus error, double free, etc).
- Tu programa no puede tener `memory leaks`.
- Para la parte obligatoria tienes permitido utilizar las siguientes funciones:
  - `write`
  - `read`
  - `malloc`
  - `free`
  - `exit`
- Puedes preguntar a tus compañeros, por Slack, etc.

Capítulo V

Parte obligatoria

V.1. Las reglas del juego

- El juego se compone de dos stacks, llamados `a` y `b`.
- Para empezar:
  - En `a` tendrás números positivos y/o negativos, nunca duplicados.
  - En `b` no habrá nada.
- El objetivo es ordenar los números del stack `a` en orden ascendente.
- Para hacerlo tienes las siguientes operaciones a tu disposición:

sa : `swap a` - intercambia los dos primeros elementos encima del stack `a`. No hace nada si hay uno o menos elementos.

sb : `swap b` - intercambia los dos primeros elementos encima del stack `b`. No hace nada si hay uno o menos elementos.

ss : `swap a` y `swap b` a la vez.

pa : `push a` - toma el primer elemento del stack `b` y lo pone encima del stack `a`. No hace nada si `b` está vacío.

pb : `push b` - toma el primer elemento del stack `a` y lo pone encima del stack `b`. No hace nada si `a` está vacío.

ra : `rotate a` - desplaza hacia arriba todos los elementos del stack `a` una posición, de forma que el primer elemento se convierte en el último.

rb : `rotate b` - desplaza hacia arriba todos los elementos del stack `b` una posición, de forma que el primer elemento se convierte en el último.

rr : `rotate a` y `rotate b` - desplaza al mismo tiempo todos los elementos del stack `a` y del stack `b` una posición hacia arriba, de forma que el primer elemento se convierte en el último.

rra : `reverse rotate a` - desplaza hacia abajo todos los elementos del stack `a` una posición, de forma que el último elemento se convierte en el primero.

rrb : `reverse rotate b` - desplaza hacia abajo todos los elementos del stack `b` una posición, de forma que el último elemento se convierte en el primero.

rrr : `reverse rotate a` y `reverse rotate b` - desplaza al mismo tiempo todos los elementos del stack `a` y del stack `b` una posición hacia abajo, de forma que el último elemento se convierte en el primero.

V.2. Ejemplo

Para arrojar algo de luz sobre el funcionamiento de algunas de estas instrucciones, vamos a ordenar una lista de números aleatoria. En el siguiente ejempo, asumiremos que ambos stacks crecen por la derecha.

```
--------------------------------------------------------------------------------------------------------
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
--------------------------------------------------------------------------------------------------------
```

Este ejemplo ordena los enteros de `a` en doce instrucciones. ¿Puedes hacerlo mejor?

V.3. El programa: “push_swap”

- Debes escribir un programa llamado `push_swap` que recibirá como argumento el stack `a` con el formato de una lista de enteros. El primer argumento debe ser el que esté encima del stack (cuidado con el orden).
- El programa debe mostrar la lista de instrucciones (más corta posible) para ordenar el stack `a`, de menor a mayor donde el menor se sitúe en la cima del stack.
- Las instrucciones deben separarse utilizando un “\n” y nada más.
- El objetivo es ordenar el stack con el mínimo número de operaciones posible. Durante la evaluación compararemos el número de instrucciones obtenido por tu programa con un rango de instrucciones máximo.
- Tu programa no recibirá puntos tanto si tu programa muestra una lista demasiado larga como si el resultado no es correcto.
- En caso de error, deberás mostrar `Error` seguido de un “\n” en la salida de errores estándar. Algunos de los posibles errores son: argumentos que no son enteros, argumentos superiores a un int, y/o encontrar números duplicados.

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

Durante la evaluación tendrás a tu disposición un binario para verificar el correcto funcionamiento de tu programa. Funciona de la siguiente forma:

```
$>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 el programa `checker_OS` muestra `KO`, implicará que tu programa `push_swap` encontró una solución inválida (lo que quiere decir que no ordenaba la lista). El programa `checker_OS` está disponible en la parte “recursos del proyecto” en la Intranet.

Puedes encontrar su funcionamiento descrito debajo, en la “parte bonus”.

Capítulo VI

Parte bonus

Tus bonus serán evaluados exclusivamente si la parte obligatoria es EXCELENTE. Esto quiere decir, evidentemente, que debes completar la parte obligatoria, de principio a fin, y que tu gestión de errores debe ser impecable aunque el programa se utilice incorrectamente. De no ser así, esta parte será IGNORADA.

¿Te animas a hacerte tu propio checker? ¡Suena interesante!

VI.1. El programa “checker”

- Escribe un programa llamado `checker`, que tomará como argumento el stack `a` formateado como una lista de enteros. El primer argumento debe estar encima del stack (cuidado con el orden). Si no se da argumento, `checker` termina y no muestra nada.
- Durante la ejecución de `checker` se esperará y leerá una lista de instrucciones, separadas utilizando ’\n’. Cuando todas las instrucciones se hayan leído, `checker` las ejecutará utilizando el stack recibido como argumento.
- Tras ejecutar todas las instrucciones, el stack `a` deberá estar ordenado y el stack `b` deberá estar vacío. Si es así, `checker` imprimirá `"OK"` seguido de un ’\n’ en el stdout. De no ser así, mostrará `"KO"` seguido de un ’\n’ en el stdout.
- En caso de error, deberás mostrar `Error` seguido de un ’\n’ en la **stderr**. Los errores son:
  - Algunos o todos los argumentos no son enteros
  - Algunos o todos los argumentos son más grandes que un int
  - Hay uno o varios números duplicados
  - Una instrucción no existe y/o no está bien formateada

Gracias al programa checker, podrás verificar si la lista de instrucciones que generarás con tu push_swap ordena realmente el stack o no.

```
$>./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
$>./checker "" 1
Error
$>
```

No tienes que clonar exactamente el comportamiento exacto del binario que te damos. Es obligatorio gestionar errores pero es decisión tuya cómo gestionar los argumentos.

Capítulo VII

Entrega y evaluación

Entrega tu trabajo en tu repositorio `git` como de costumbre. Solo el trabajo en tu repositorio será evaluado.

¡Buena suerte!