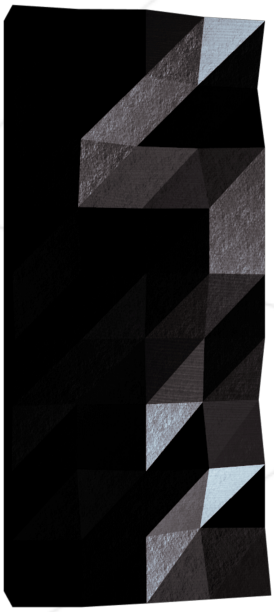

Push_swap

Porque Swap_push no es tan natural

Staff Pedagógico 42 pedago@42madrid.com

Resumen: Este proyecto le hará ordenar datos en una pila, con un conjunto limitado de instrucciones, utilizando el menor número posible de acciones. Para tener éxito, tendrá que manipular varios tipos de algoritmos y elegir la solución más adecuada (de entre muchas) para una optimización de la ordenación de datos.

Índice general

Capítulo I

Prólogo

- 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

El proyecto `Push_swap` es un proyecto de algoritmo muy simple y proyecto de algoritmo muy eficaz: habrá que ordenar los datos. Tienes a tu disposición un conjunto de valores int, 2 pilas y un conjunto de instrucciones para manipular ambas pilas.

¿Su objetivo? Escribir un programa en `C` llamado `push_swap` que calcule y muestre en la salida estándar el programa más pequeño utilizando el lenguaje de instrucciones `Push_swap` que ordena los enteros recibidos en argumentos.

¿Fácil?

Ya lo veremos...

Capítulo III

Objetivos

Escribir un algoritmo de ordenación es siempre un paso muy importante en de un programador, porque a menudo es el primer encuentro con el concepto de complejidad.

Los algoritmos de ordenación y sus complejidades forman parte de las clásicas preguntas que se discuten en las entrevistas de trabajo. Probablemente sea un buen momento de analizar estos conceptos porque en algún momento tendrás que enfrentarte a ellos. en algún momento.

Los objetivos de aprendizaje de este proyecto son el rigor, el uso de `C` y el uso de algoritmos básicos. Especialmente se busca la complejidad de estos algoritmos básicos.

Ordenar los valores es sencillo. Ordenarlos de la forma más rápida posible es menos sencillo, sobre todo porque de una configuración de enteros a otra, el algoritmo de ordenación más eficiente puede diferir.

Capítulo IV

Instrucciones generales

- Este proyecto sólo será corregido por seres humanos reales humanos. Por lo tanto, usted es libre de organizar y nombrar sus archivos como desee, aunque debe respetar algunos requisitos que se enumeran a continuación.
- El archivo ejecutable debe llamarse `push_swap`.
- Debe presentar un `Makefile`. Ese `Makefile` tiene que compilar el proyecto y debe contener debe contener las reglas habituales. Sólo puede recompilar el programa si es necesario.
- Si eres inteligente, usarás tu biblioteca para este proyecto, presentar también su carpeta `libft` incluyendo su propio `Makefile` en la raíz de su repositorio. Su `Makefile` tendrá que compilar la biblioteca, y luego compilar su proyecto.
- Las variables globales están prohibidas.
- Su proyecto debe estar escrito en `C` de de acuerdo con la Norma.
- Hay que manejar los errores de manera sensible. De ninguna manera puede su programa salir de forma inesperada (fallo de segmentación de segmentación, error de bus, doble free, etc).
- Ninguno de los dos programas puede tener `fugas de memoria`.
- Dentro de su parte obligatoria se le permite utilizar las siguientes funciones:
  - `write`
  - `read`
  - `malloc`
  - `free`
  - `exit`
- Puedes hacer preguntas en el foro & y en Slack...

Capítulo V

Parte obligatoria

V.1. Reglas del juego

- El juego se compone de 2 pilas llamados `a` y `b`.
- Para empezar:
  - `a` contiene un número aleatorio de números positivos o negativos sin duplicados.
  - `b` está vacío
- El objetivo es clasificar en orden ascendente los números en pila `a`.
- Para ello dispone de las siguientes operaciones a su disposición:

sa : `swap a` - intercambiar los primeros 2 elementos en la parte superior de la pila `a`. No hace nada si sólo hay uno o ningún elementos).

sb : `swap b` - intercambiar los primeros 2 elementos en la parte superior de la pila `b`. No hacer nada si sólo hay uno o ningún elementos).

ss : `sa` y `sb` al al mismo tiempo.

pa : `push a` - toma el primer elemento en la parte superior de `b` y ponerlo en la parte superior de texttta. No hace nada si `b` está vacío.

pb : `push b` - toma el primer elemento en la parte superior de `a` y ponerlo en la parte superior de textttb. No hace nada si `a` está vacío.

ra : `rotate a` - desplazar hacia arriba todos los elementos de la pila `a` en 1. El primer elemento se convierte en el último.

rb : `rotate b` - desplazar hacia arriba todos los elementos de la pila `b` en 1. El primer elemento se convierte en el último.

rr : `ra` y `rb` al al mismo tiempo.

rra : `reverse rotate a` - desplazar hacia abajo todos los elementos de la pila `a` en 1. El último elemento se convierte en el primero.

rrb : `reverse rotate b` - desplazar hacia abajo todos los elementos de la pila `b` en 1. El último elemento se convierte en el primero.

rrr : `rra` y `rrb` al mismo tiempo.

V.2. Ejemplo

Para ilustrar el efecto de algunas de estas instrucciones vamos a ordenar una lista aleatoria de enteros. En este ejemplo, consideraremos que las dos pilas crecen desde 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 12 instrucciones. ¿Puedes hacerlo mejor?

V.3. El programa “push_swap”

- Tienes que escribir un programa llamado `push_swap` que recibirá como argumento la pila `a` formateada como una lista de enteros. El primer argumento debe estar en la parte superior de la pila (tenga cuidado con el orden).
- El programa debe mostrar la menor lista de instrucciones posible para ordenar la pila `a`, el número más pequeño está en la parte superior.
- Las instrucciones deben estar separadas por un ’\n’ y nada más.
- El objetivo es ordenar la pila con el mínimo número número posible de operaciones. Durante la defensa vamos a comparar el número de instrucciones que su programa encontró con el número máximo de operaciones toleradas. Si tu programa muestra una lista demasiado grande o si la lista no está bien ordenada, no obtendrá puntos.
- En caso de error, debe mostrar `Error` seguido de un ’\n’ en el error estándar. Los errores incluyen, por ejemplo: algunos argumentos no son enteros, algunos argumentos son mayores que un entero, y/o hay 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 defensa le proporcionaremos un binario para comprobar adecuadamente su programa. Funcionará de la siguiente manera:

```
$>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`, significa significa que su programa texttt-push_swap ha obtenido una lista de instrucciones que no ordena la lista. El programa `checker_OS` está disponible en los recursos del proyecto en la intranet. Usted puede encontrar en la sección de bonos de este documento una descripción de cómo funciona.

Capítulo VI

Parte de la bonificación

Nos fijaremos en su parte de bonificación si y sólo si su parte obligatoria es EXCELENTE. Esto significa que debe completar la parte parte obligatoria, de principio a fin, y su gestión de errores tiene que ser impecable, incluso en los casos de uso torcido o malo. Si no es así, sus bonificaciones serán totalmente IGNORADOS.

¿Qué tan interesante podría ser codificar tu propio verificador? ¡¡Muuuuuuy interesante!!

VI.1. El programa “checker”

- Escriba un programa llamado `checker`, que obtendrá como argumento la pila `a` formateada como una lista de enteros. El primer argumento debe estar en la parte superior de la pila (tenga cuidado con el orden). Si no se da ningún argumento `checker` se detiene y no muestra nada.
- `checker` esperará y leerá instrucciones en la entrada estándar, cada instrucción será seguida por ’\n’. Una vez leídas todas las instrucciones han sido leídas, `checker` las ejecutará en la pila recibida como argumento.
- Si después de ejecutar esas instrucciones, la pila `a` está realmente ordenada y `b` está vacía, entonces el verificador debe mostrar `."OK"` seguido de un ’\n’ en la salida estándar. En cualquier otro caso, debe mostrar el texto `"KO"` seguido de un ’\n’ en la salida estándar.
- En caso de error, debe mostrar `Error` seguido de un ’\n’ en el **error estándar**. Los errores incluyen, por ejemplo: algunos argumentos no son enteros, algunos argumentos son mayores que un entero, hay duplicados, una instrucción no existe y/o está mal formateada.

Gracias al programa checker, podrá comprobar si la lista de instrucciones que generará con el programa push_swap está realmente ordenando la pila correctamente.

```
$>./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
$>
```

Capítulo VII

Presentación y corrección por pares

Envíe su trabajo en su repositorio `GiT` como de costumbre. Sólo el trabajo en su repositorio será calificado.

¡Buena suerte a todos!