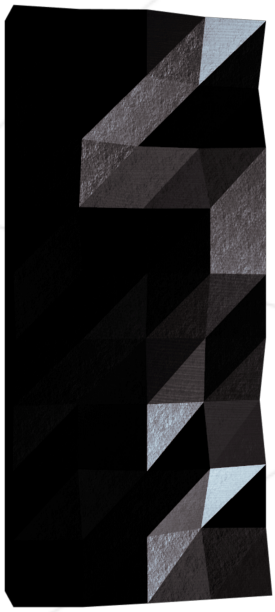

Push_swap

Çünkü Swap_push o kadar doğal değil

Özet: Bu proje sayesinde, minimum sayıda ve kısıtlı aksiyonlarla bir stack üzerinde veri sıralaması yapabileceksiniz. Bunu başarabilmek için çeşitli algoritmaları manipüle edip optimal bir veri sıralaması için birçoğu arasından en uygun olanını seçmeniz gerekecek.

Versiyon: 8.1

İçindekiler

Bölüm I

Ön Söz

- 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
 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 ()
```

Bölüm II

Giriş

`Push swap` projesi çok basit ve etkili bir algoritma projesidir: veriler sıralanmalıdır. Elinizde bir dizi integer, 2 stack ve bu stackleri manipüle etmek için bir aksiyon setiniz var.

Siz ne mi yapacaksınız? Argüman olarak alınan integerları `push_swap` aksiyonlarını kullanarak, en kısa şekilde sıralayan ve çıktısını veren bir C programı yazacaksınız.

Kolay mı geldi?

Göreceğiz...

Bölüm III

Hedefler

Bir geliştiricinin hayatında bir sıralama algoritması yazmak her zaman çok önemli bir adımdır. Genellikle complexity kavramı ile ilk karşılaşılan yer burasıdır.

Sıralama algoritmaları ve complexityleri, iş görüşmeleri sırasında sorulan klasik sorulardandır. Bir noktada bu kavramlarla yüzleşmeniz gerekeceğinden, şu an onları öğrenmek için iyi bir zaman.

Bu projenin kazanımları `C` kullanımı ve basit algoritmaların kullanımıdır. Özellikle algoritmaların complexitylerine odaklanarak.

Verileri sıralamak kolaydır. Onları en hızlı şekilde sıralamak ise daha az kolaydır. En verimli sıralama çözümü farklı integer grupları için değişkenlik gösterebilir.

Bölüm IV

Genel Talimatlar

- Projeleriniz C programlama dilinde yazılmalıdır.
- Projeleriniz Norm'a uygun olarak yazılmalıdır. Bonus dosyalarınız/fonksiyonlarınız varsa, bunlar norm kontrolüne dahil edilir ve bu dosyalarda norm hatası varsa `0` alırsınız.
- Tanımlanmamış davranışlar dışında sizin fonksiyonlarınız beklenmedik bir şekilde sonlanmamalıdır (Segmentasyon hatası, bus hatası, double free hatası, vb.) . Eğer bunlar yaşanırsa s `0` alırsınız.
- Heap'de ayırmış olduğunuz hafıza adresleri gerekli olduğu durumlarda serbest bırakılmalıdır. Hiçbir istisna tolere edilmeyecektir.
- Eğer verilen görev `Makefile` dosyasının yüklenmesini istiyorsa, sizin kaynak dosyalarınızı `-Wall`, `-Wextra` , `-Werror`, flaglarini kullanarak derleyip çıktı dosyalarını üretecek olan Makefile dosyasını oluşturmanız gerekmektedir. Makefile dosyasını oluştururken cc kullanın ve Makefile dosyanız yeniden ilişkilendirme yapmamalıdır (relink).
- `Makefile` dosyanız en azından `$(NAME)`, `all`, `clean`, `fclean` ve `re` kurallarını içermelidir.
- Projenize bonusu dahil etmek için Makefile dosyanıza `bonus` kuralını dahil etmeniz gerekmektedir. Bonus kuralının dahil edilmesi bu projenin ana kısmında kullanılması yasak olan bazı header dosyaları, kütüphaneler ve fonksiyonların eklenmesini sağlayacaktır. Eğer projede farklı bir tanımlama yapılmamışsa, bonus projeleri `_bonus.{c/h}` dosyaları içerisinde olmalıdır. Ana proje ve bonus proje değerlendirmeleri ayrı ayrı gerçekleştirilmektedir.
- Eğer projeniz kendi yazmış olduğunuz `libft` kütüphanesini kullanmanıza izin veriyorsa, bu kütüphane ve ilişkili `Makefile` dosyasını proje dizinindeki `libft` klasörüne ilişkili `Makefile` dosyası ile kopyalamanız gerekmektedir. Projenizin `Makefile` dosyası öncelikle `libft` kütüphanesini kütüphanenin `Makefile` dosyasını kullanarak derlemeli ardından projeyi derlemelidir.
- Test programları sisteme yüklenmek zorunda değildir ve puanlandırılmayacaktır. Buna rağmen test programları yazmanızı şiddetle önermekteyiz. Test programları

sayesinde kendinizin ve arkadaşlarınız projelerinin çıktılarını kolaylıkla gözlemleyebilirsiniz. Bu test dosyalarından özellikle savunma sürecinde çok faydalanacaksınız. Savunma sürecinde kendi projeleriniz ve arkadaşlarınızın projeleri için test programlarını kullanmakta özgürsünüz.

- Çalışmalarınız atanmış olan git repolarına yüklemeniz gerekmektedir. Sadece git reposu içerisindeki çalışmalar notlandırılacaktır. Eğer Deepthought sizin çalışmanızı değerlendirmek için atanmışsa, bu değerlendirmeyi arkadaşlarınızın sizin projenizi değerlendirmesinden sonra gerçekleştirecektir. Eğer Deepthought değerlendirme sürecinde herhangi bir hata ile karşılaşılırsa değerlendirme durdurulacaktır.

Bölüm V

Zorunlu Bölüm

V.1 Kurallar

- `a` ve `b` isimli 2 stackiniz var.
- Başlangıçta:
  - `a` stacki rastgele sayıda, birbirinin aynısı olmayan negatif ve/veya pozitif sayıdan oluşmaktadır.
  - `b` stacki boştur.
- Amaç stacktaki sayıları artan şekilde `a` stackine sıralamaktır. Bunu başarabilmek için aşağıdaki aksiyonları kullanabilirsiniz:

sa (`swap a`): `a` stackinin en üstteki iki elemanını birbirileri ile yer değiştirin.
Eğer yalnızca bir eleman var ise veya hiç eleman yoksa bir şey yapmayın.

sb (`swap b`): `b` stackinin en üstteki iki elemanını birbirileri ile yer değiştirin.
Eğer yalnızca bir eleman var ise veya hiç eleman yoksa bir şey yapmayın.

ss : `sa` ve `sb` aksiyonlarının ikisini de çalıştır.

pa (`push a`): `b` stackinin en üstündeki elemanı `a` stackinin en üstüne koyun.
Eğer `b` stacki boş ise bir şey yapmayın.

pb (`push b`): `a` stackinin en üstündeki elemanı `b` stackinin en üstüne koyun.
Eğer `a` stacki boş ise bir şey yapmayın.

ra (`rotate a`): `a` stackinin bütün elemanlarını bir yukarı kaydırın.
İlk eleman artık son eleman olacaktır.

rb (`rotate b`): `b` stackinin bütün elemanlarını bir yukarı kaydırın.
İlk eleman artık son eleman olacaktır.

rr : `ra` ve `rb` aksiyonlarının ikisini de çalıştır.

rra (`reverse rotate a`): `a` stackinin bütün elemanlarını bir aşağı kaydırın.
Son eleman artık ilk eleman olacaktır.

rrb (`reverse rotate b`): `b` stackinin bütün elemanlarını bir aşağı kaydırın.
Son eleman artık ilk eleman olacaktır.

rrr : `rra` ve `rrb` aksiyonlarının ikisini de çalıştırır.

V.2 Örnek

Tanımlanan aksiyonların davranışlarını gösterebilmek için, rastgele bir integer listesini sıralamayı deneyelim. Bu örnekte iki stackin de sağdan büyüdüğünü varsayacağız.

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

a stackindeki integerlar 12 aksiyon ile sıralandı. Daha iyisini yapabilir misin?

V.3 "push_swap" Programı

Program adı	`push_swap`
Teslim edilecek dosyalar	`Makefile, *.h, *.c`
Makefile	`NAME, all, clean, fclean, re`
Argümanlar	`stack a: Bir integer listesi`
Harici fonksiyonlar.	• `read, write, malloc, free, exit` • `ft_printf veya SİZİN kodladığınız bir alternatif`
Libft kullanılabilir mi?	`Evet`
Açıklama	`Stackleri Sırala`

Projeniz aşağıdaki kurallara uymalıdır.

- Source dosyalarınızı compile edecek bir `Makefile` dosyası oluşturmalı ve proje repositorynize yüklemelisiniz. Makefile relink yapmamalıdır.
- Global değişkenler yasaktır.
- You have to write a program named `push_swap` that takes as an argument the stack `a` formatted as a list of integers. The first argument should be at the top of the stack (be careful about the order).
- Program, en küçük sayı en üstte olacak şekilde, `a` stackini sıralamak için gereken minimum aksiyonların listesini çıktı olarak vermelidir.
- Aksiyonlar yalnızca '\n' ile ayrılmalıdır. Başka/Ekstra hiçbir karakter kabul edilmeyecektir.
- Hedef stacki mümkün olan minimum sayıda aksiyon ile sıralamaktır. Değerlendirme sürecinde programınızın verdiği aksiyon çıktılarını bir limite göre karşılaştırılacaktır: izin verilen maksimum aksiyon sayısı. Eğer programınız daha uzun bir liste çıktısı veriyorsa veya integerlar doğru bir şekilde sıralanmamışsa notunuz 0 olacaktır.
- Eğer program çalıştırılırken hiçbir argüman verilmezse, program hiçbir çıktı vermemeli ve çıkış yapmalıdır.
- Herhangi bir hata durumunda, standard error'a `"Error"` ve takiben '\n' çıktısı verilmelidir. Hata örnekleri: bazı argümanların integer olmaması, bazı argümanların integerdan daha büyük olması ve/veya aynı integerdan birden çok kez bulunması.

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

Değerlendirme süreci boyunca, programınızı test etmek için başka bir program sağlanacaktır.

Kontrolcü programı aşağıdaki gibi çalıştırılacaktır:

```
$>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
$>
```

Eğer `checker_OS` programı `"KO"` çıktısı verirse, bu sizin `push_swap` programınızın verilen integerları yanlış sıralayan bir aksiyon listesi oluşturduğunu gösterir.

`checker_OS` programı intranet üzerindeki projenizin kaynaklarında bulunmaktadır.
Kontrolcü programın nasıl çalıştığını öğrenmek istiyorsanız bu dokümanın bonus bölümüne bakabilirsiniz.

V.4 Performans kontrolü

Bu projeyi doğrulamak için, minimum sayıda işlemle belirli sıralamaları gerçekleştirmelisiniz:

- **Minimum geçer puanlı** bir doğrulama için (yani en az 80 puan) **100 rastgele sayıyı 700'den az işlemde** sıralayabilmeniz gerekir.
- **Tam puanlı geçer** bir doğrulama için ve dolayısıyla bonuslarıda puanlandırmak için yukarıdaki ilk adımı yerine getirmeniz gerekir ve ayrıca **500 rastgele sayıyı 5500'den az işlemde** sıralayabilmeniz gerekir.

Tüm bunlar değerlendirmeniz sırasında kontrol edilecektir.

`Bonus kısmını tamamlamak istiyorsanız, her kontrol adımında mümkün olan en yüksek puanı alarak projeyi doğrulamanız gerekir.`

Bölüm VI

Bonus Bölüm

Bu proje basitliği nedeniyle üzerine ekstra özellik eklenmesine pek fırsat bırakmamaktadır. O zaman, kendi kontrolcü programınızı yazmaya ne dersiniz?

checker programı sayesinde, push_swap programı tarafından belirtilen aksiyonların integer listesini doğru sıralayıp sıralamadığını test edebileceksiniz.

VI.1 "checker" Programı

Program adı	checker
Teslim edilecek dosyalar	*.h, *.c
Makefile	bonus
Argümanlar	stack a: Bir integer listesi
Harici fonksiyonlar.	• read, write, malloc, free, exit • ft_printf veya SİZİN kodladığınız bir alternatif
Libft kullanılabilir mi?	Evet
Açıklama	Sıralama aksiyonlarını çalıştır

- checker isimli ve argüman olarak a stackini (integer listesi) alan bir program yazın. İlk argüman stackin en tepesinde bulunacaktır (sırasına dikkat edin). Eğer hiçbir argüman girilmemişse program hiçbir çıktı yapmaz ve çıkış yapar.

- Program çalışmaya başladığında bekleyecek ve standart inputtan aksiyon girişlerini okuyacaktır. Her aksiyon '\n' ile bitirilir. Bütün aksiyonlar okunduktan sonra program stack üzerine bu aksiyonları uygulayacaktır.
- Eğer bu aksiyonlardan sonra `a` stacki doğru sıralanmışsa ve `b` stacki boş ise, program `"OK"` ve '\n' çıktısını standart output'a vermelidir.
- Diğer bütün durumlarda `"KO"` ve '\n' çıktısını standart output'a vermelidir.
- Herhangi bir hata durumunda `"Error"` ve '\n' çıktısını **standard error**'a vermelidir. Hata örnekleri: bazı argümanların integer veri tipinde olmaması, bazı argümanların integer değerden büyük olması, birden fazla aynı argümandan bulunması, girilen bir aksiyonun var olmaması veya yanlış formatta yazılması.

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

Yazdığınız program size sağlanan kontrolcü program ile tamamen aynı çalışmak zorunda değildir. Hataları yönetmek zorundasınız fakat argümanları nasıl parse edeceğiniz size kalmıştır.

Bonus bölüm, yalnızca zorunlu bölüm KUSURSUZ ise değerlendirilecektir. Kusursuz, zorunlu bölümün tamamen yapıldığı ve sorunsuz çalıştığı anlamına gelir. TÜM zorunlu gereksinimleri tamamlamadıysanız, bonus bölüm hiçbir şekilde değerlendirilmeyecektir.

Bölüm VII

Submission and peer-evaluation

Projenizi her zamanki gibi `Git` repositorynize gönderin. Savunma sırasında yalnızca reponuzdaki çalışmalar değerlendirilecektir. Dosyalarınızın adlarının doğru olduklarından emin olmak için adlarını iki kez kontrol etmekten çekinmeyin.

Bu proje Moulinette tarafından kontrol edilmeyeceği için, zorunlu dosyalar gereksinimleri karşıladığı sürece dosyalarınızı istediğiniz gibi organize edebilirsiniz.

file.bfe:VABB7yO9xm7xWXROeASsmsgnYOoOsDMJev7zFHhwQS8mvM8V5xQQp
Lc6cDCFXDWTiFzZ2H9skYkiJ/DpQtnM/uZO