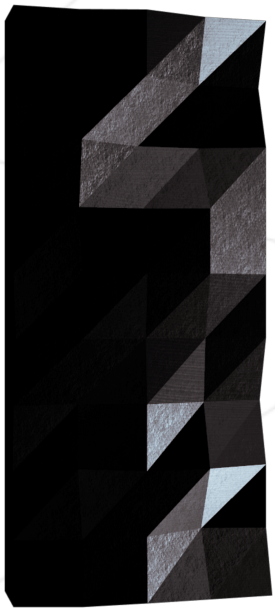

Push_swap

Porque Swap_push não parece tão natural

Preâmbulo:
Neste projeto, você irá ordenar dados em uma pilha usando um conjunto limitado de instruções, visando alcançar o menor número possível de ações. Para ter sucesso, você precisará trabalhar com vários algoritmos e escolher o mais adequado para uma ordenação de dados otimizada.

Versão: 9.0

Sumário

Capítulo I

Prefácio

- 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

Introdução

O projeto **Push swap** é um desafio algorítmico simples, mas altamente estruturado: você precisa ordenar dados.

Você tem à sua disposição um conjunto de valores inteiros, 2 pilhas e um conjunto de instruções para manipular ambas as pilhas.

Seu objetivo? Escrever um programa em C chamado `push_swap` que calcula e exibe a sequência mais curta de instruções Push_swap necessárias para ordenar os inteiros fornecidos.

Fácil?

Veremos...

Capítulo III

Objetivos

Escrever um algoritmo de ordenação é sempre um passo crucial na jornada de um desenvolvedor. É frequentemente o primeiro encontro com o conceito de complexidade.

Algoritmos de ordenação e suas complexidades são tópicos comuns em entrevistas de emprego. Agora é um ótimo momento para explorar esses conceitos, pois você provavelmente os encontrará no futuro.
Os objetivos de aprendizagem deste projeto são rigor, proficiência em `C`, e a aplicação de algoritmos básicos, com foco particular em sua complexidade.

Ordenar valores é direto, mas encontrar a maneira mais rápida de ordená-los é mais desafiador. O método de ordenação mais eficiente pode variar dependendo do arranjo dos inteiros.

Capítulo IV

Instruções Comuns

- Seu projeto deve ser escrito em C.
- Seu projeto deve ser escrito de acordo com a Norma. Se você tiver arquivos/funções bônus, eles serão incluídos na verificação da norma, e você receberá uma nota `0` se houver algum erro de norma.
- Suas funções não devem sair inesperadamente (segmentation fault, bus error, double free, etc.) exceto por comportamento indefinido. Se isso ocorrer, seu projeto será considerado não funcional e receberá uma nota `0` durante a avaliação.
- Toda memória alocada na heap deve ser liberada corretamente quando necessário. Vazamentos de memória não serão tolerados.
- Se o enunciado exigir, você deve submeter um `Makefile` que compile seus arquivos fonte para a saída requerida com as flags `-Wall`, `-Wextra`, e `-Werror`, usando `cc`. Adicionalmente, seu `Makefile` não deve realizar re-links desnecessários.
- Seu `Makefile` deve conter pelo menos as regras `$(NAME)`, `all`, `clean`, `fclean` e `re`.
- Para submeter bônus para seu projeto, você deve incluir uma regra `bonus` em seu `Makefile`, que adicionará todos os diversos headers, bibliotecas, ou funções que não são permitidas na parte principal do projeto. Os bônus devem ser colocados em arquivos `_bonus.{c/h}`, a menos que o enunciado especifique o contrário. A avaliação das partes obrigatórias e bônus é conduzida separadamente.
- Se seu projeto permitir que você use sua `libft`, você deve copiar suas fontes e seu `Makefile` associado para uma pasta `libft`. O `Makefile` do seu projeto deve compilar a biblioteca usando seu `Makefile`, e então compilar o projeto.
- Nós encorajamos você a criar programas de teste para seu projeto, mesmo que este trabalho **não precise ser submetido e não será avaliado**. Isso lhe dará a oportunidade de testar facilmente seu trabalho e o trabalho de seus colegas. Você achará esses testes especialmente úteis durante sua defesa. De fato, durante a defesa, você está livre para usar seus testes e/ou os testes do colega que você está avaliando.

- Submeta seu trabalho para o repositório Git designado. Somente o trabalho no repositório Git será avaliado. Se o Deepthought for designado para avaliar seu trabalho, isso ocorrerá após as avaliações de seus colegas. Se um erro acontecer em qualquer seção do seu trabalho durante a avaliação do Deepthought, a avaliação será interrompida.

Capítulo V

Instruções para IA

Contexto

Durante sua jornada de aprendizado, a IA pode auxiliar em diversas tarefas. Dedique tempo para explorar as várias capacidades das ferramentas de IA e como elas podem apoiar seu trabalho. No entanto, sempre as utilize com cautela e avalie criticamente os resultados. Seja código, documentação, ideias ou explicações técnicas, você nunca pode ter certeza absoluta de que sua pergunta foi bem formulada ou que o conteúdo gerado é preciso. Seus colegas são um recurso valioso para ajudá-lo a evitar erros e pontos cegos.

Mensagem principal

- Use a IA para reduzir tarefas repetitivas ou tediosas.
- Desenvolva habilidades de prompt — tanto para codificação quanto para outras tarefas — que beneficiarão sua carreira futura.
- Aprenda como os sistemas de IA funcionam para melhor antecipar e evitar riscos comuns, vieses e questões éticas.
- Continue desenvolvendo habilidades técnicas e interpessoais trabalhando com seus colegas.
- Use apenas conteúdo gerado por IA que você entenda completamente e pelo qual possa se responsabilizar.

Regras para o aluno:

- Você deve dedicar tempo para explorar as ferramentas de IA e entender como elas funcionam, para que possa usá-las eticamente e reduzir potenciais vieses.
- Você deve refletir sobre seu problema antes de criar o prompt — isso ajuda você a escrever prompts mais claros, detalhados e relevantes, usando vocabulário preciso.

- Você deve desenvolver o hábito de verificar, revisar, questionar e testar sistematicamente tudo o que for gerado por IA.
- Você deve sempre buscar revisão por pares — não confie apenas em sua própria validação.

Resultados da fase:

- Desenvolver habilidades de prompt tanto para uso geral quanto para áreas específicas.
- Aumentar sua produtividade com o uso eficaz de ferramentas de IA.
- Continuar fortalecendo o pensamento computacional, resolução de problemas, adaptabilidade e colaboração.

Comentários e exemplos:

- Você encontrará regularmente situações — exames, avaliações e mais — onde você deve demonstrar compreensão real. Esteja preparado, continue desenvolvendo suas habilidades técnicas e interpessoais.
- Explicar seu raciocínio e debater com colegas frequentemente revela lacunas em sua compreensão. Priorize a aprendizagem colaborativa.
- As ferramentas de IA geralmente carecem do seu contexto específico e tendem a fornecer respostas genéricas. Seus colegas, que compartilham seu ambiente, podem oferecer insights mais relevantes e precisos.
- Onde a IA tende a gerar a resposta mais provável, seus colegas podem fornecer perspectivas alternativas e nuances valiosas. Conte com eles como um ponto de verificação de qualidade.

✓ Boa prática:

> Pergunto à IA: “Como testo uma função de ordenação?” Ela me dá algumas ideias. Eu as testo e reviso os resultados com um colega. Nós refinamos a abordagem juntos.

✗ Má prática:

> Peço à IA para escrever uma função inteira, copio e colo no meu projeto. Durante a avaliação por pares, não consigo explicar o que ela faz ou porquê. Perco credibilidade — e reprovo no meu projeto.

✓ Boa prática:

Uso a IA para ajudar a projetar um analisador sintático. Então, analiso a lógica com um colega. Detectamos dois erros e reescrevemos juntos — melhor, mais limpo e totalmente compreendido.

✗ Má prática:

Deixo o Copilot gerar meu código para uma parte importante do meu projeto. Ele compila, mas não consigo explicar como ele lida com pipes. Durante a avaliação, não consigo justificar e reprovo no meu projeto.

Capítulo VI

Parte Obrigatória

VI.1 As regras

- Você tem 2 pilhas nomeadas `a` e `b`.
- No início:
  - A pilha `a` contém um número aleatório de inteiros únicos negativos e/ou positivos.
  - A pilha `b` está vazia.
- O objetivo é ordenar os números na pilha `a` em ordem crescente. Para alcançar isso, você tem as seguintes operações à sua disposição:

  sa (`swap a`): Troca os 2 primeiros elementos no topo da pilha `a`.
  Não faz nada se houver apenas um elemento ou nenhum.

  sb (`swap b`): Troca os 2 primeiros elementos no topo da pilha `b`.
  Não faz nada se houver apenas um elemento ou nenhum.

  ss : `sa` e `sb` ao mesmo tempo.

  pa (`push a`): Pega o primeiro elemento no topo de `b` e o coloca no topo de `a`.
  Não faz nada se `b` estiver vazia.

  pb (`push b`): Pega o primeiro elemento no topo de `a` e o coloca no topo de `b`.
  Não faz nada se `a` estiver vazia.

  ra (`rotate a`): Move todos os elementos da pilha `a` para cima em 1.
  O primeiro elemento se torna o último.

  rb (`rotate b`): Move todos os elementos da pilha `b` para cima em 1.
  O primeiro elemento se torna o último.

  rr : `ra` e `rb` ao mesmo tempo.

  rra (`reverse rotate a`): Move todos os elementos da pilha `a` para baixo em 1.
  O último elemento se torna o primeiro.

  rrb (`reverse rotate b`): Move todos os elementos da pilha `b` para baixo em 1.
  O último elemento se torna o primeiro.

  rrr : `rra` e `rrb` ao mesmo tempo.

VI.2 Exemplo

Para ilustrar o efeito de algumas dessas instruções, vamos ordenar uma lista aleatória de inteiros. Neste exemplo, consideraremos que ambas as pilhas crescem a partir da direita.

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

d Os inteiros na pilha `a` são ordenados em 12 instruções. Você consegue fazer melhor?

VI.3 O programa "push_swap".

d

Nome do programa	`push_swap`
Arquivos para entregar	`Makefile, *.h, *.c`
Makefile	`NAME, all, clean, fclean, re`
Argumentos	`pilha a: Uma lista de inteiros`
Funções externas autorizadas	`d` • `read, write, malloc, free, exit` • `ft_printf ou qualquer equivalente que VOCÊ codificou`
Libft autorizada	`Sim`
Descrição	`Ordena pilhas`

d

Seu projeto deve cumprir as seguintes regras:

- Você deve entregar um `Makefile` que irá compilar seus arquivos de origem. Ele não deve dar relink.
- Variáveis globais são proibidas.
- Você deve escrever um programa chamado `push_swap` que recebe como argumento a pilha `a` formatada como uma lista de inteiros. O primeiro argumento deve estar no topo da pilha (tome cuidado com a ordem).
- O programa deve exibir a sequência mais curta de instruções necessárias para ordenar a pilha `a` com o menor número no topo.
- As instruções devem ser separadas por um '\n' e nada mais.
- O objetivo é ordenar a pilha com o menor número possível de operações. Durante o processo de avaliação, o número de instruções encontradas pelo seu programa será comparado com um limite: o número máximo de operações toleradas. Se seu programa exibir uma lista maior ou se os números não estiverem ordenados corretamente, sua nota será 0.
- Se nenhum parâmetro for especificado, o programa não deve exibir nada e deve retornar ao prompt.

- Em caso de erro, ele deve exibir `"Error"` seguido de um '\n' no erro padrão. Os erros incluem, por exemplo: alguns argumentos não sendo inteiros, alguns argumentos excedendo os limites de inteiros e/ou a presença de duplicatas.

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

Durante o processo de avaliação, um binário será fornecido para verificar corretamente seu programa.

Ele funcionará da seguinte 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
$>
```

Se o programa `checker_OS` exibir `"KO"`, significa que seu `push_swap` criou uma lista de instruções que não ordena os números.

O programa checker_OS está disponível nos recursos do projeto na intranet.

Você pode encontrar uma descrição de como ele funciona na Parte de Bônus deste documento.

VI.4 Benchmark

Para validar este projeto, você deve realizar algumas ordenações com um número mínimo de operações:

- Para **validação máxima do projeto** (100%) e elegibilidade para bônus, você deve:
  - Ordenar **100 números aleatórios em menos de 700 operações**.
  - Ordenar **500 números aleatórios em no máximo 5500 operações**.
- Para **validação mínima do projeto** (que implica uma nota mínima de 80%), você pode ter sucesso com médias diferentes:
  - **100 números em menos de 1100 operações** e **500 números em menos de 8500 operações**
  - **100 números em menos de 700 operações** e **500 números em menos de 11500 operações**
  - **100 números em menos de 1300 operações** e **500 números em menos de 5500 operações**

...

Tudo isso será verificado durante sua avaliação.

`Se você deseja completar a parte de bônus, deve validar completamente o projeto com cada etapa de benchmark atingindo a pontuação mais alta possível.`

Capítulo VII

Parte Bônus

Devido à sua simplicidade, este projeto oferece oportunidades limitadas para recursos adicionais. No entanto, por que não criar seu próprio verificador?

Graças ao programa checker, você poderá verificar se a lista de instruções gerada pelo programa push_swap realmente ordena a pilha corretamente.

A parte bônus só será avaliada se a parte obrigatória estiver perfeita. Perfeita significa que a parte obrigatória foi totalmente concluída e funciona sem erros. Neste projeto, isso implica validar todos os benchmarks sem exceção. Se você não passou TODOS os requisitos obrigatórios, sua parte bônus não será avaliada.

VII.1 O programa "checker".

Nome do programa	checker
Arquivos para entregar	*.h, *.c
Makefile	bonus
Argumentos	pilha a: Uma lista de inteiros
Funções externas autorizadas	• read, write, malloc, free, exit • ft_printf ou qualquer equivalente que VOCÊ codificou
Libft autorizada	Sim
Descrição	Executa as instruções de ordenação

- Escreva um programa chamado `checker` que recebe como argumento a pilha `a` formatada como uma lista de inteiros. O primeiro argumento deve estar no topo da pilha (tome cuidado com a ordem). Se nenhum argumento for dado, ele para e não exibe nada.

- Ele então esperará e lerá instruções da entrada padrão, com cada instrução seguida por '\n'. Depois que todas as instruções forem lidas, o programa deve executá-las na pilha recebida como argumento.

- Se, após executar essas instruções, a pilha `a` estiver realmente ordenada e a pilha `b` estiver vazia, então o programa deve exibir `"OK"` seguido de um '\n' na saída padrão.

- Em todos os outros casos, ele deve exibir `"KO"` seguido de um '\n' na saída padrão.

- Em caso de erro, você deve exibir `"Error"` seguido de um '\n' no **erro padrão**. Os erros incluem, por exemplo: alguns argumentos não são inteiros, alguns argumentos são maiores que um inteiro, existem duplicatas, uma instrução não existe e/ou está incorretamente formatada.

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

Você NÃO precisa reproduzir o mesmo comportamento exato do binário fornecido. É obrigatório gerenciar erros, mas cabe a você decidir como deseja analisar os argumentos.

Capítulo VIII

Submissão e avaliação por pares

Envie sua tarefa em seu repositório `Git` como de costume. Somente o trabalho dentro do seu repositório será avaliado durante a defesa. Não hesite em verificar novamente os nomes dos seus arquivos para garantir que estejam corretos.

Como essas tarefas não são verificadas por um programa, sinta-se à vontade para organizar seus arquivos como desejar, desde que você entregue os arquivos obrigatórios e cumpra os requisitos.

Durante a avaliação, uma breve **modificação do projeto** pode ser ocasionalmente solicitada. Isso pode envolver uma pequena mudança de comportamento, algumas linhas de código para escrever ou reescrever, ou um recurso fácil de adicionar.

Embora esta etapa possa **não ser aplicável a todos os projetos**, você deve estar preparado para ela se for mencionada nas diretrizes de avaliação.

Esta etapa visa verificar sua compreensão real de uma parte específica do projeto. A modificação pode ser realizada em qualquer ambiente de desenvolvimento que você escolher (por exemplo, sua configuração usual), e deve ser viável em poucos minutos — a menos que um prazo específico seja definido como parte da avaliação.
Você pode, por exemplo, ser solicitado a fazer uma pequena atualização em uma função ou script, modificar uma exibição ou ajustar uma estrutura de dados para armazenar novas informações, etc.

Os detalhes (escopo, alvo, etc.) serão especificados nas **diretrizes de avaliação** e podem variar de uma avaliação para outra para o mesmo projeto.

file.bfe:VABB7yO9xm7xWXROeASsmsgnYOoOsDMJev7zFHhwQS8mvM8V5xQQp
Lc6cDCFXDWTiFzZ2H9skYkiJ/DpQtnM/uZO