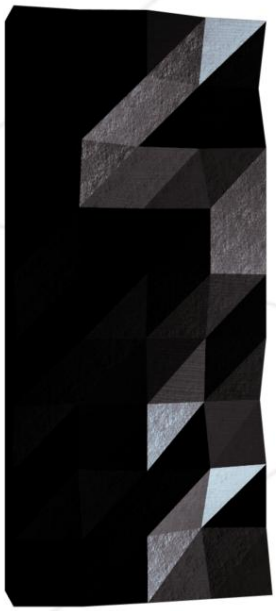

Obter próxima linha

Ler uma linha de um fd é muito tedioso

Resumo:
Este projeto trata da programação de uma função que retorna uma linha lida de um descritor de arquivo.

Versão: 11

Conteúdo

Capítulo I

Metas

Este projeto não apenas permitirá que você adicione uma função muito conveniente à sua coleção, mas também fará com que você aprenda um novo conceito muito interessante em programação C: variáveis estáticas.

Capítulo II

Instruções Comuns

- Seu projeto deve ser escrito em C.
- Seu projeto deve ser redigido de acordo com a Norma. Se você tiver arquivos/funções de bônus, eles serão incluídos na verificação da norma e você receberá um 0 se houver um erro de norma dentro.
- Suas funções não devem ser encerradas inesperadamente (falha de segmentação, erro de barramento, double free, etc.) além de comportamentos indefinidos. Se isso acontecer, seu projeto será considerado não funcional e receberá nota 0 na avaliação.
- Todo o espaço de memória alocado no heap deve ser liberado adequadamente quando necessário. Sem vazamentos será tolerado.
- Se o assunto exigir, você deve enviar um Makefile que irá compilar seus arquivos de origem para a saída necessária com os sinalizadores -Wall, -Wextra e -Werror, use cc e seu Makefile não deve revincular.
- Seu Makefile deve conter pelo menos as regras $(NAME), all, clean, fclean e ré.
- Para ativar bônus em seu projeto, você deve incluir uma regra de bônus em seu Makefile, que adicionará todos os vários cabeçalhos, bibliotecas ou funções que são proibidas na parte principal do projeto. Os bônus devem estar em um arquivo diferente _bonus.{c/h} se o assunto não especificar mais nada. A avaliação da parte obrigatória e bônus é feita separadamente.
- Se seu projeto permite que você use sua libft, você deve copiar suas fontes e seu Makefile associado em uma pasta libft com seu Makefile associado. O Makefile do seu projeto deve compilar a biblioteca usando seu Makefile e, em seguida, compilar o projeto.
- Incentivamos você a criar programas de teste para o seu projeto, mesmo que este trabalho **não precise ser enviado e não seja avaliado.** Isso lhe dará a chance de testar facilmente seu trabalho e o de seus colegas. Você achará esses testes especialmente úteis durante sua defesa. De fato, durante a defesa, você é livre para usar seus testes e/ou os testes do par que está avaliando.
- Envie seu trabalho para o repositório git atribuído. Somente o trabalho no repositório git será avaliado. Se o Deepthought for designado para avaliar seu trabalho, isso será feito

após suas avaliações de pares. Se ocorrer um erro em qualquer seção do seu trabalho durante a avaliação do Deepthought, a avaliação será interrompida.

Capítulo III

parte obrigatória

Nome da função	get_next_line char
Protótipo	*get_next_line(int fd); get_next_line.c,
Entregar arquivos	get_next_line_utils.c, get_next_line.h fd: O descritor de arquivo para ler da linha de
Parâmetros	leitura: comportamento correto
Valor de retorno	NULL: não há mais nada para ler ou ocorreu um erro
Funções externas.	ler, malloc, grátis
Descrição	Escreva uma função que retorne uma linha lida de um descritor de arquivo

- Chamadas repetidas (por exemplo, usando um loop) para sua função get_next_line() devem permitir que você leia o arquivo de texto apontado pelo descritor de arquivo, **uma linha por vez.**
- Sua função deve retornar a linha que foi lida.
  Se não houver mais nada para ler ou se ocorreu um erro, ele deve retornar NULL.
- Certifique-se de que sua função funcione conforme o esperado ao ler um arquivo e ao leitura da entrada padrão.
- **Observe** que a linha retornada deve incluir o caractere final \n, exceto se o final do arquivo for alcançado e não terminar com um caractere \n.
- Seu arquivo de cabeçalho get_next_line.h deve conter pelo menos o protótipo da função get_next_line().
- Adicione todas as funções auxiliares necessárias no arquivo get_next_line_utils.c.

Um bom começo seria saber o que é uma variável estática é.

• Como você terá que ler arquivos em get_next_line(), adicione esta opção ao seu
chamada do compilador: -D
BUFFER_SIZE=n Definirá o tamanho do buffer para read().
O valor do tamanho do buffer será modificado por seus pares avaliadores e pela Moulinette para testar seu código.

Devemos ser capazes de compilar este projeto com e sem o sinalizador -D BUFFER_SIZE além dos sinalizadores usuais. Você pode escolher o valor padrão de sua escolha.

• Você compilará seu código da seguinte maneira (um tamanho de buffer de 42 é usado como exemplo):
cc -Wall -Wextra -Werror -D BUFFER_SIZE=42 <arquivos>.c

• Consideramos que get_next_line() tem comportamento indefinido se o arquivo apontado pelo descritor de arquivo mudou desde a última chamada enquanto read() não chegou ao final do arquivo.

• Também consideramos que get_next_line() tem um comportamento indefinido ao ler um arquivo binário.
No entanto, você pode implementar uma maneira lógica de lidar com esse comportamento, se desejar.

Sua função ainda funciona se o valor de BUFFER_SIZE for 9999? Se for 1? 10000000?
Você sabe por quê?

Tente ler o mínimo possível cada vez que get_next_line() for chamado. Se você encontrar uma nova linha, deverá retornar a linha atual.

Não leia o arquivo inteiro e depois processe cada linha.

Proibido

• Você não tem permissão para usar sua libft neste projeto.

• lseek() é proibido.

• Variáveis globais são proibidas.

Capítulo IV

Parte bônus

Este projeto é direto e não permite bônus complexos. No entanto, confiamos na sua criatividade. Se você concluiu a parte obrigatória, experimente esta parte bônus.

Aqui estão os requisitos da parte bônus:

- Desenvolva get_next_line() usando apenas uma variável estática.
- Seu get_next_line() pode gerenciar vários descritores de arquivo ao mesmo tempo.
  Por exemplo, se você puder ler os descritores de arquivo 3, 4 e 5, poderá ler de um fd diferente por chamada sem perder o thread de leitura de cada descritor de arquivo ou retornar uma linha de outro fd.
  Isso significa que você deve ser capaz de chamar get_next_line() para ler de fd 3, depois fd 4, depois 5, depois novamente 3, mais uma vez 4 e assim por diante.

Anexe o sufixo _bonus.[c\h] aos arquivos de peças de bônus.
Isso significa que, além dos arquivos obrigatórios da peça, você deverá entregar os 3 seguintes arquivos:

- get_next_line_bonus.c
- get_next_line_bonus.h
- get_next_line_utils_bonus.c

A parte bônus só será avaliada se a parte obrigatória for PERFEITA. Perfeito significa que a peça obrigatória foi executada integralmente e funciona sem avarias. Se você não passou em TODAS as
requisitos obrigatórios, sua parte de bônus não será avaliada.

Capítulo V

Submissão e avaliação por pares

Entregue sua atribuição em seu repositório Git como de costume. Apenas o trabalho dentro do seu repositório será avaliado durante a defesa. Não hesite em verificar novamente os nomes de seus arquivos para garantir que estejam corretos.

Ao escrever seus testes, lembre-se que: 1) Tanto o tamanho do buffer quanto o tamanho da linha podem ter valores muito diferentes.

2) Um descritor de arquivo não aponta apenas para arquivos comuns.
Seja inteligente e verifique com seus colegas. Prepare um conjunto completo de diversos testes para defesa.

Uma vez aprovado, não hesite em adicionar seu get_next_line() ao seu libft.