



Miniconcha

Tão linda quanto uma concha

Resumo:

Este projeto é sobre criar um shell simples.

Sim, sua própria pequena festa.

Você aprenderá muito sobre processos e descritores de arquivos.

Versão: 7.1

Conteúdo

EU	Introdução	2
II	Instruções comuns	3
III	Parte obrigatória	5
IV	Parte bônus	8
V	Submissão e avaliação por pares	9

Capítulo I

Introdução

A existência de shells está ligada à própria existência de TI.

Na época, todos os desenvolvedores concordaram que *a comunicação com um computador usando alinhados Interruptores 1/0 eram muito irritantes.*

Era lógico que eles tivessem a ideia de criar um software para se comunicar com um computador usando linhas interativas de comandos em uma linguagem um pouco próxima da linguagem humana.

Graças ao Minishell, você poderá viajar no tempo e retornar aos problemas as pessoas enfrentavam quando o *Windows* não existia.

Capítulo II

Instruções comuns

- Seu projeto deve ser escrito em C.
- Seu projeto deve ser escrito 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 sair 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 um 0 durante a 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 compilará seus arquivos de origem para a saída necessária com os sinalizadores -Wall, -Wextra e -Werror, use cc e seu Makefile não deve ser vinculado novamente.
- Seu Makefile deve conter pelo menos as regras \$(NAME), all, clean, fclean e ré.
- Para entregar bônus ao seu projeto, você deve incluir um bônus de regra ao 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 nada mais. A avaliação da parte obrigatória e bônus é feita separadamente.
- Se seu projeto permitir 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.
- Nós o encorajamos a criar programas de teste para seu projeto, mesmo que esse trabalho **não precise ser enviado e não seja classificado**. Isso lhe dará uma chance 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 é livre para usar seus testes e/ou os testes do colega que está avaliando.
- Envie seu trabalho para o repositório git designado. Somente o trabalho no repositório git será classificado. Se o Deepthought for designado para classificar seu trabalho, isso será feito

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

Capítulo III

Parte obrigatória

Nome do programa	miniconcha
Entregar arquivos	Makefile, *.h, *.c NOME,
Argumentos	todos, limpo, fclean, re
do Makefile	
Funções externas.	readline, rl_clear_history, rl_on_new_line, rl_replace_line, rl_redisplay, add_history, printf, malloc, liberar, escrever, acessar, abrir, ler, fechar, bifurcar, esperar, waitpid, wait3, wait4, sinalizar, sigaction, sigemptyset, sigaddset, matar, sair, getcwd, chdir, stat, lstat, fstat, desvincular, execve, dup, dup2, pipe, opendir, readdir, closedir, strerror, perror, isatty, ttyname, ttyslot, ioctl, getenv, tcsetattr, tcgetattr, tgetent, tgetflag, tgetnum, tgetstr, tgoto, tputs
Autorizado pela Libft	Sim
Descrição	Escreva um shell

Seu shell deve:

- Exibir um **prompt** ao aguardar um novo comando.
- Ter **histórico de trabalho**.
- Pesquise e inicie o executável correto (com base na variável PATH ou usando um caminho relativo ou absoluto).
- Evite usar mais de **uma variável global** para indicar um sinal recebido. Considere as implicações: essa abordagem garante que seu manipulador de sinal não acessará suas principais estruturas de dados.



Tenha cuidado. Esta variável global não pode fornecer nenhuma outra informação ou acesso a dados além do número de um sinal recebido.

Portanto, usar estruturas do tipo "norma" no escopo global é proibido.

- Não interpretar aspas não fechadas ou caracteres especiais que não sejam exigidos pelo assunto como \ (barra invertida) ou ; (ponto e vírgula).
- Manipular ' (aspas simples) que deve impedir o shell de interpretar o meta-caracteres na sequência citada.
- Manipule " (aspas duplas) que deve impedir que o shell interprete os metacaracteres na sequência entre aspas, exceto \$ (cifrão).
- Implementar **redirecionamentos**:

• `<` deve redirecionar a entrada.

• `>` deve redirecionar a saída.

• `<<` deve receber um delimitador, então leia a entrada até que uma linha contendo o delimitador seja vista. No entanto, não precisa atualizar o histórico!

• `>>` deve redirecionar a saída no modo de acréscimo.

- Implementar **pipes** (caractere `|`). A saída de cada comando no pipeline é conectado à entrada do próximo comando por meio de um pipe.
- Manipular **variáveis de ambiente** (\$ seguido por uma sequência de caracteres) que devem se expandir para seus valores.
- Identificador `$?` que deve expandir para o status de saída do pipeline de primeiro plano executado mais recentemente.
- Manipule `ctrl-C`, `ctrl-D` e `ctrl-\`, que devem se comportar como no bash.
- No modo interativo:

• `ctrl-C` exibe um novo prompt em uma nova linha.

• `ctrl-D` sai do shell.

• `ctrl-\` não faz nada.

- Seu shell deve implementar os seguintes **builtins**:

• `echo` com opção `-n`

• `cd` com apenas um caminho relativo ou absoluto

• `pwd` sem opções

• `export` sem opções

• `set` não definido e sem opções

• `env` sem opções ou argumentos

• `exit` sem opções

A função `readline()` pode causar vazamentos de memória. Você não precisa consertá-los. Mas isso **não significa** que seu próprio código, sim, o código que você escreveu, pode ter vazamentos de memória.



Você deve se limitar à descrição do assunto. Qualquer coisa que não seja perguntada não é necessária.

Se você tiver alguma dúvida sobre um requisito, use [o bash](#) como referência.

Capítulo IV

Parte bônus

Seu programa deve implementar:

- && e || com parênteses para prioridades.
- Os curingas * devem funcionar para o diretório de trabalho atual.



A parte bônus só será avaliada se a parte obrigatória for PERFEITA. Perfeito significa que a parte obrigatória foi feita integralmente e funciona sem apresentar mau funcionamento. Se você não passou em TODOS os requisitos obrigatórios, sua parte bônus não será avaliada.

Capítulo V

Submissão e avaliação por pares

Entregue sua tarefa no seu repositório Git como de costume. Apenas o trabalho dentro do seu repositório será avaliado durante a defesa. Não hesite em verificar novamente o nomes dos seus arquivos para garantir que estejam corretos.

