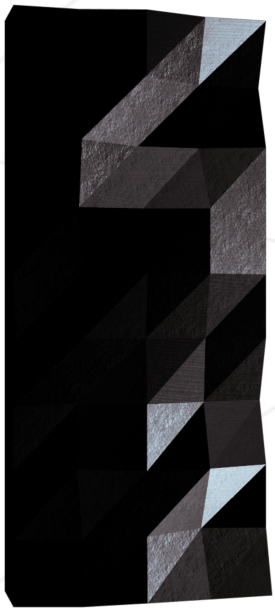

Minishell

Tão belo quanto uma concha

Preâmbulo:
Este projeto consiste em criar um shell simples.
Sim, seu próprio pequeno Bash.
Você ganhará amplo conhecimento sobre processos e descritores de arquivos.

Versão: 9.0

Sumário

Capítulo I

Introdução

Shells existem desde o início da TI.

Naquela época, todos os desenvolvedores concordavam que se comunicar com um computador via interruptores de entrada/saída alinhados era extremamente frustrante.

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

Com o `Minishell`, você viajará no tempo e experimentará os desafios que os desenvolvedores enfrentaram antes da existência do *Windows*.

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 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 III

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 IV

Parte Obrigatória

Nome do programa	`minishell`
Arquivos para entregar	`Makefile, *.h, *.c`
Makefile	`NAME, all, clean, fclean, re`
Argumentos	
Funções externas autorizadas	`readline, rl_clear_history, rl_on_new_line, rl_replace_line, rl_redisplay, add_history, printf, malloc, free, write, access, open, read, close, fork, wait, waitpid, wait3, wait4, signal, sigaction, sigemptyset, sigaddset, kill, exit, getcwd, chdir, stat, lstat, fstat, unlink, execve, dup, dup2, pipe, opendir, readdir, closedir, strerror, perror, isatty, ttyname, ttyslot, ioctl, getenv, tcsetattr, tcgetattr, tgetent, tgetflag, tgetnum, tgetstr, tgoto, tputs`
Libft autorizada	`Sim`
Descrição	`Escreva um shell`

Seu shell deve:

- Exibir um **prompt** enquanto aguarda um novo comando.
- Ter um **histórico** funcional.
- Pesquisar e iniciar o executável correto (com base na variável `PATH` ou usando um caminho relativo ou absoluto).
- Usar no máximo **uma variável global** para indicar um sinal recebido. Considere as implicações: essa abordagem garante que seu manipulador de sinal não acessará suas estruturas de dados principais.

Cuidado. Essa variável global deve armazenar apenas o número do sinal e não deve fornecer informações adicionais ou acesso a dados. Portanto, o uso de estruturas do tipo "norm"no escopo global é proibido.

- Não interpretar aspas não fechadas ou caracteres especiais que não são necessários pelo enunciado, como \ (barra invertida) ou ; (ponto e vírgula).
- Lidar com ' (aspas simples), que deve impedir o shell de interpretar os metacaracteres na sequência entre aspas.
- Lidar com " (aspas duplas), que deve impedir o shell de interpretar os metacaracteres na sequência entre aspas, exceto para $ (sinal de dólar).
- Implementar os seguintes **redirecionamentos**:
  - < deve redirecionar a entrada.
  - > deve redirecionar a saída.
  - << deve receber um delimitador, então ler a entrada até que uma linha contendo o delimitador seja encontrada. No entanto, não precisa atualizar o histórico!
  - >> deve redirecionar a saída no modo de anexação.
- Implementar **pipes** (caractere |). A saída de cada comando no pipeline é conectada à entrada do próximo comando via um pipe.
- Lidar com **variáveis de ambiente** ($ seguido de uma sequência de caracteres), que devem ser expandidas para seus valores.
- Lidar com $? que deve expandir para o status de saída do pipeline em primeiro plano mais recentemente executado.
- Lidar com ctrl-C, ctrl-D e ctrl-\ que devem se comportar como em 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 comandos **built-in**:
  - echo com a opção -n

- `cd` com apenas um caminho relativo ou absoluto
- `pwd` sem opções
- `export` sem opções
- `unset` sem opções
- `env` sem opções ou argumentos
- `exit` sem opções

A função `readline()` pode causar vazamentos de memória, mas você não é obrigado a corrigi-los. No entanto, 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 enunciado. Qualquer coisa que não seja solicitada não é necessária.
Se tiver alguma dúvida sobre um requisito, tome bash como uma referência.

Capítulo V

Parte Bônus

Seu programa deve implementar:

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

A parte bônus só será avaliada se a parte obrigatória estiver concluída perfeitamente. Perfeito significa que a parte obrigatória está totalmente implementada e funciona sem nenhum problema. Se você não tiver passado TODOS os requisitos obrigatórios, sua parte bônus não será avaliada.

Capítulo VI

Submissão e avaliação por pares

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

During the evaluation, a brief **modification of the project** may occasionally be requested. This could involve a minor behavior change, a few lines of code to write or rewrite, or an easy-to-add feature.

While this step may **not be applicable to every project**, you must be prepared for it if it is mentioned in the evaluation guidelines.

This step is meant to verify your actual understanding of a specific part of the project. The modification can be performed in any development environment you choose (e.g., your usual setup), and it should be feasible within a few minutes — unless a specific timeframe is defined as part of the evaluation.
You can, for example, be asked to make a small update to a function or script, modify a display, or adjust a data structure to store new information, etc.

The details (scope, target, etc.) will be specified in the **evaluation guidelines** and may vary from one evaluation to another for the same project.