パイプックス

概要: このプロジェクトでは、プログラムに実装することで、すでに使い慣れている UNIX のメカニズムを詳細に調べることができます。

バージョン: 4.0

コンテンツ

私 序文 2

II 共通指示 3

III AI指示 5

IV 必須部分IV.1 例 . . . 7

。 . . . 。 。 。 。 。 。 . . . 。 . . . 。 。 。 。 。 。 . . . 。 . . . 。 8

IV.2 要件 . . . 。 。 。 。 。 。 . . . 。 . . . 。 。 。 。 。 。 . . . 。 。 。 。 。 8

V ボーナスパート 9

6 提出とピア評価 10

第1章

序文

クリスティーナ: 「どこかでサルサを踊りに行きましょう! :)」

第2章

一般的な指示

- プロジェクトは C 言語で記述する必要があります。
- プロジェクトは規格に従って記述されている必要があります。ボーナスファイルや関数がある場合は、それらも規格チェックの対象となり、規格エラーが発生した場合は0点が付与されます。
- 関数は、未定義の動作を除き、予期せず終了してはいけません（セグメンテーション違反、バスエラー、二重解放など）。予期せぬ終了が発生した場合、プロジェクトは機能しないものとみなされ、評価時に0点が与えられます。
- ヒープに割り当てられたメモリは、必要に応じて適切に解放されなければなりません。メモリリーク
  容認されません。
- 研究テーマで必要な場合は、cc を使用して、フラグ -Wall、 -Wextra、および -Werror を指定してソース ファイルを必要な出力にコンパイルする Makefile を提出する必要があります。
  さらに、Makefile では不必要な再リンクを実行してはなりません。
- Makefileには少なくとも$(NAME)、all、clean、fclean、および
  再。
- プロジェクトにボーナスを提出するには、Makefileにボーナスルールを含める必要があります。このルールは、プロジェクトのメイン部分では許可されていない様々なヘッダー、ライブラリ、または関数をすべて追加します。テーマで別途指定がない限り、ボーナスは_bonus.{c/h}ファイルに配置する必要があります。必須部分とボーナス部分の評価は別々に行われます。
- プロジェクトでlibftの使用が許可されている場合は、そのソースコードと関連するMakefileをlibftフォルダにコピーする必要があります。プロジェクトのMakefileは、ライブラリをMakefileを使用してコンパイルし、その後プロジェクトをコンパイルする必要があります。
- プロジェクトのテストプログラムを作成することを推奨します。ただし、提出は必須ではなく、採点もされません。テストプログラムを作成することで、自分の作品と他の参加者の作品を簡単にテストする機会が得られます。これらのテストは、特に論文審査の際に役立ちます。論文審査中は、自分のテスト、または評価対象となる他の参加者のテストを自由に使用できます。
- 指定されたGitリポジトリに提出してください。Gitリポジトリ内の作業のみが採点されます。Deepthoughtが採点を担当する場合は、

ピア評価の後。Deepthoughtによる採点中に、提出した課題のいずれかのセクションに誤りがあった場合、評価は中止されます。

第3章

AI指示

•文脈

学習の過程で、AIは様々なタスクを支援してくれます。AIツールの様々な機能と、それらがどのようにあなたの仕事に役立つかをじっくりと探求してください。しかし、常に慎重にアプローチし、結果を批判的に評価してください。コード、ドキュメント、アイデア、技術的な説明など、どんなものであっても、質問が適切に構成されているか、生成されたコンテンツが正確であるかを完全に確信することはできません。同僚は、間違いや盲点を回避するための貴重なリソースです。

•主なメッセージ

AI を使用して、反復的な作業や面倒な作業を削減します。

コーディングと非コーディングの両方で、あなたのビジネスに役立つプロンプトスキルを身につけましょう。
将来のキャリア。

AI システムが一般的なリスク、バイアス、倫理的問題をより適切に予測して回避する仕組みを学びます。

同僚と協力して、技術スキルとパワースキルの両方を継続的に構築します。

AI生成コンテンツは、完全に理解し、責任を負えるもののみ使用してください
のために。

•学習者のルール:

- 時間をかけて AI ツールを調べ、その仕組みを理解し、倫理的に使用して潜在的なバイアスを減らす必要があります。
- 質問する前に、自分の問題についてよく考えてください。こうすることで、より明確に、
正確な語彙を使用した、より詳細で関連性の高いプロンプト。
- AI によって生成されたものはすべて体系的にチェック、レビュー、質問、テストする習慣を身につける必要があります。
- 常にピアレビューを求める必要があります。自分自身の検証だけに頼らないでください。

•フェーズの成果:

- 汎用的なプロンプトスキルとドメイン固有のプロンプトスキルの両方を開発します。
- AI ツールを効果的に活用して生産性を向上します。
- 計算思考、問題解決能力、適応力、そして
  コラボレーション。

•コメントと例:

- 試験や評価など、真の理解を示すことが求められる状況に頻繁に遭遇するでしょう。技術スキルと対人スキルの両方を磨き続け、万全の準備をしてください。
- 自分の考えを説明し、仲間と議論することで、自分の理解のギャップが明らかになることが多い。
  理解を深める。ピアラーニングを優先しましょう。
- AIツールは、ユーザー固有のコンテキストを欠き、一般的な回答を提供する傾向があります。同じ環境にいる同僚は、より関連性の高い正確な洞察を提供できます。
- AIが最も可能性の高い回答を生成する傾向がある場合でも、同僚は別の視点や貴重なニュアンスを提供できます。品質チェックポイントとして頼りにしましょう。

✓良い実践例:

> AIに「ソート関数をテストするにはどうすればいいですか？」と尋ねると、いくつかのアイデアが浮かびました。それを試してみて、同僚と結果を確認します。そして、一緒にアプローチを改良していきます。

悪い習慣:

> AIに関数全体を書いてもらい、それをプロジェクトにコピー&ペーストしました。しかし、ピア評価の際に、それが何をするのか、なぜそうするのかを説明できませんでした。信頼を失い、プロジェクトは失敗に終わりました。

✓良い実践例:

> AIを活用してパーサーの設計を支援します。その後、同僚と一緒にロジックを検証します。2つのバグを見つけ、一緒に書き直します。より良く、よりクリーンで、完全に理解できるものになります。

悪い習慣:

> プロジェクトの重要な部分のコードをCopilotで生成しました。コンパイルは通るものの、パイプ処理の仕組みが理解できません。評価時に説明がつかず、プロジェクトは失敗に終わりました。

第4章

必須部分

プログラム名	pipex
ファイルを提出する	Makefile、*.h、*.c NAME、all、clean、
Makefile 引	fclean、re file1 cmd1 cmd2 file2
数外部関数。	
	• open、close、read、write、malloc、free、 perror、strerror、access、dup、 dup2、execve、exit、fork、pipe、unlink、wait、 waitpid • ft_printf または同等の関数 あなたがコーディングした
Libft認定	はい
説明	このプロジェクトはパイプの取り扱いに重点を置いています。

プログラムは次のように実行される必要があります。

./pipex ファイル1 コマンド1 コマンド2 ファイル2

4 つの引数が必要です:

• file1 と file2 はファイル名です。

• cmd1 と cmd2 はパラメータを持つシェル コマンドです。

次のシェル コマンドとまったく同じように動作する必要があります。

```
$> < ファイル1 コマンド1 | コマンド2 > ファイル2
```

IV.1 例

```
$> ./pipex 入力ファイル "ls -l" "wc -l" 出力ファイル
```

その動作は次と同等です: < infile ls -l | wc -l > outfile

```
$> ./pipex 入力ファイル "grep a1" "wc -w" 出力ファイル
```

その動作は次のものと同等です: < infile grep a1 | wc -w > outfile

IV.2 要件

プロジェクトは次のルールに準拠する必要があります。

- ソースファイルをコンパイルするMakefileを提出してください。
  不要な再リンク。
- プログラムは予期せず終了してはなりません（例 :セグメンテーション違反、バス エラー、ダブル フリーなど）。
- プログラムにはメモリリークがあってはなりません。
- 不明な場合は、シェルコマンドと同じ方法でエラーを処理します: < file1 cmd1 | cmd2 > file2

第5章

ボーナス部分

以下の場合、追加ポイントを獲得できます。

- 複数のパイプを処理します。

これ：

```
$> ./pipex ファイル1 コマンド1 コマンド2 コマンド3 ... コマンド2 ファイル2
```

次のように動作するはずです：

```
< ファイル1 コマンド1 | コマンド2 | コマンド3 ... | コマンドn > ファイル2
```

- 最初のパラメータが「here_doc」の場合、「and」をサポートします。

これ：

```
$> ./pipex here_doc LIMITER cmd cmd1 ファイル
```

次のように動作するはずです：

```
cmd << LIMITER | cmd1 >> ファイル
```

ボーナス部分は、必須部分が「完璧」な場合にのみ評価されます。「完璧」とは、必須部分が全体として完成し、不具合なく機能することを意味します。必須要件をすべて満たしていない場合、ボーナス部分は全く評価されません。

第6章

提出とピア評価

課題は通常通りGitリポジトリに提出してください。審査ではリポジトリ内の作業のみが評価されます。ファイル名が正しいか、必ず再確認してください。

評価中に、プロジェクトの簡単な変更が求められる場合があります。これには、軽微な動作変更、数行のコードの追加または書き換え、あるいは簡単に追加できる機能などが含まれる場合があります。

この手順はすべてのプロジェクトに当てはまるとは限りませんが、評価ガイドラインに記載されている場合は準備しておく必要があります。

このステップは、プロジェクトの特定の部分に関する実際の理解を確認することを目的としています。
変更は、選択した任意の開発環境 (通常のセットアップなど) で実行でき、評価の一部として特定の時間枠が定義されていない限り、数分以内に実行できるはずです。

たとえば、関数やスクリプトに小さな更新を加えたり、表示を変更したり、新しい情報を保存するためにデータ構造を調整したりすることが求められる場合があります。

詳細（範囲、対象など）は評価ガイドラインに明記され、同じプロジェクトでも評価ごとに異なる場合があります。

file.bfe:VACsSfsWN1cy33ROeASsmsgnY0o0sDMJev7zFHhw
QS8mvM8V5xQQpLc6cDCFXDWTiFzZ2H9skYkiJ/DpQtnM/uZ0