1. Inicjalizacja programu

1.1. **Uruchomienie programu:**

- Przyjmij argumenty z linii poleceń.
- Inicjalizuj środowisko programu:
    - Pobierz zmienne środowiskowe.
    - Przygotuj historię poleceń.
- Skonfiguruj sygnały (`SIGINT`, `SIGQUIT`) za pomocą funkcji `signal` lub `sigaction`.

1.2. **Przygotowanie prompta:**

- Stwórz funkcję generującą prompt.
- Wywołuj prompt w pętli głównej programu.

2. Przyjęcie danych wejściowych od użytkownika

2.1. **Obsługa wejścia:**

- Wywołaj `readline` w celu przyjęcia wiersza polecenia od użytkownika.
- Dodaj wpisane polecenie do historii (`add_history`).

2.2. **Weryfikacja danych:**

- Sprawdź, czy użytkownik wpisał dane.
- Obsłuż specjalne przypadki, np. pustą linię, `ctrl-D` (wyjście z programu) i `ctrl-C` (anulowanie aktualnego polecenia).

3. Parsowanie i wstępne przetwarzanie danych

3.1. **Podział polecenia:**

- Rozbij wiersz polecenia na tokeny, biorąc pod uwagę cudzysłowy (pojedyncze i podwójne) oraz znaki specjalne (np. `|`, <, >).

3.2. **Analiza składni:**

- Sprawdź poprawność składni, np. brakujące zamknięcia cudzysłowów, nieoczekiwane znaki.
- Zbuduj strukturę danych reprezentującą komendę:
    - Nazwa komendy.
    - Argumenty.
    - Informacje o redirekcjach (<, >, >>, <<).
    - Informacje o potokach (`|`).

4. Wykonanie przetworzonego polecenia

4.1. **Rozpoznanie rodzaju komendy:**

- Sprawdź, czy komenda to wbudowana funkcja (np. `cd`, `echo`, `pwd`).
- Jeśli to zewnętrzny program, znajdź jego lokalizację przy użyciu zmiennej `PATH`.

4.2. **Obsługa wbudowanych komend:**

- `cd`: zmiana katalogu.
- `pwd`: wyświetlenie bieżącego katalogu.
- `echo`: wyświetlenie tekstu.
- `export`/`unset`: zarządzanie zmiennymi środowiskowymi.

- `env`: wyświetlenie wszystkich zmiennych środowiskowych.
- `exit`: zakończenie działania programu.

4.3. **Uruchamianie zewnętrznych programów:**

- Utwórz proces potomny za pomocą `fork`.
- W procesie potomnym wykonaj komendę za pomocą `execve`.
- W procesie nadrzędnym czekaj na zakończenie procesu potomnego za pomocą `wait` lub `waitpid`.

5. Obsługa potoków i redirekcji

5.1. **Redirekcje:**

- Wykryj redirekcje w strukturze danych:
  - <: Przekierowanie wejścia za pomocą `dup2` na plik.
  - > i >>: Przekierowanie wyjścia do pliku w trybie nadpisywania lub dopisywania.
  - <<: Obsługa heredoca – czytaj dane do momentu napotkania delimitera.
- Skonfiguruj deskryptory plików za pomocą funkcji systemowych (`open`, `dup2`).

5.2. **Potoki:**

- Wykryj operator `|` w poleceniu.
- Utwórz potok za pomocą `pipe` i skonfiguruj wyjścia/wejścia procesów, aby połączyć je w kolejności.
- Obsłuż wiele potoków w jednej komendzie, np. `ls -l | grep txt | wc -l`.

6. Zarządzanie wynikami i statusami

6.1. **Wyświetlenie wyników:**

- Upewnij się, że wynik komendy (stdout lub stderr) jest prawidłowo wyświetlany w terminalu lub przekierowywany zgodnie z redirekcjami.

6.2. **Status zakończenia:**

- Zapisz status zakończenia ostatniego procesu do zmiennej `$?`, aby można było go wykorzystać w kolejnych komendach.

7. Zarządzanie błędami

7.1. **Weryfikacja błędów składni:**

- Informowanie użytkownika o błędach w wierszu poleceń (np. "syntax error near unexpected token").

7.2. **Obsługa błędów systemowych:**

- Sprawdzenie błędów funkcji systemowych (np. `execve`, `fork`) za pomocą `errno` i wyświetlenie komunikatów (`perror`).

8. Pętla główna programu

8.1. **Zapętlenie pracy:**

- Po przetworzeniu i wykonaniu komendy wyświetl nowy prompt.

- Kontynuuj pracę do momentu wywołania `exit` lub `ctrl-D`.

9. Zakończenie programu

9.1. **Czyszczenie pamięci:**

- Zwolnij wszystkie alokowane zasoby, w tym historię poleceń, struktury danych i zmienne środowiskowe.
- Zamknij wszystkie otwarte deskryptory plików.