ALOC

ALOC 2025 Final Presentation

MiniShell Project

로우레벨 프로그래밍 근데 이제 백엔드를 곁들인

TABLE OF CONTENTS

CHAPTER 01

프로젝트의 목적과 기대 효과

전공자와 비전공자의
격차 감소

CS적인 깊이와 기반
지식이 중요

쉘(Shell)의 작동 원리를
이해하고 직접 구현

CHAPTER1 프로젝트의 목적과 기대 효과

1. 나만의 Unix Shell을 직접 구현 

2. echo, cd, pwd, export, unset, env, exit 같은 내장 명령어 직접 구현 

3. |, <, >, >>, ^C, ^D 등의 파이프, 리디렉션 

CHAPTER 02

스터디 진행 방식

Week 1~4

JAVA 언어 공부

- 자바 기초 학습
- 개발 환경 세팅

Week 5

스터디 목표 변경

- JAVA로 구현에 한계를 느낌
- C언어를 이용하기로 결정
- 프로젝트의 방향을 변경

Week 6~8

프로세스 동작 원리 학습

- 유닉스 시스템 구조 학습
- 프로세스 동작 원리 학습

Week 9~13

Minishell 구현

- **주차 별로 파트를 하나 정하여, 각자 코드를 구현**
- 후에 만나서 코드 리뷰를 통한 학습

CHAPTER 03

구조 설계 및 기능 구현

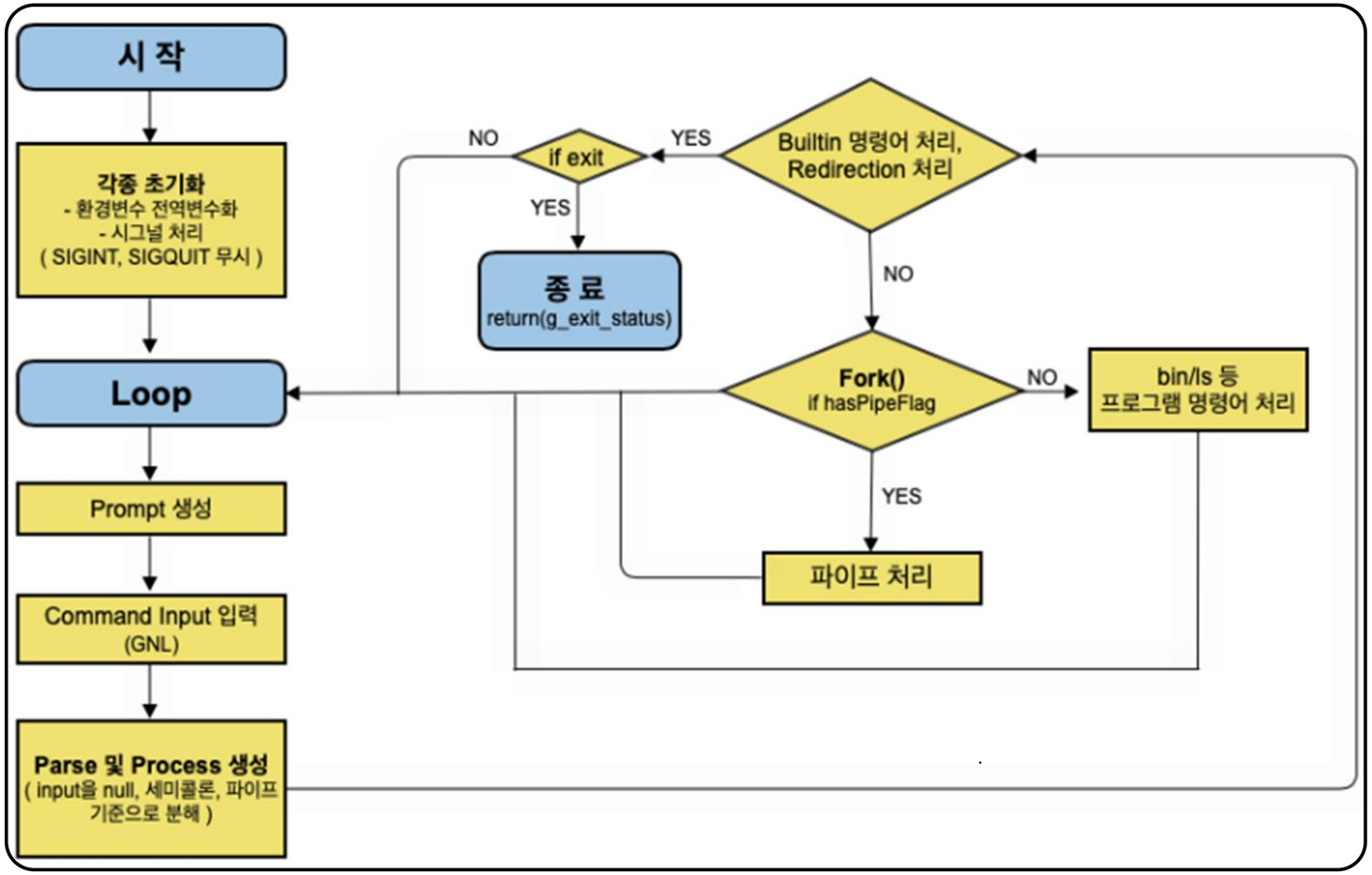
```
minishell/
├── include/              # 전역 헤더 파일
│   └── minishell.h
├── lib/                  # 외부 라이브러리, 공용 유틸 함수
├── src/
│   └── shell/            # Shell 관련 로직
│       ├── parser/       # 명령어 파싱
│       ├── builtin/      # 내장 명령어
│       ├── executor/     # 명령 실행, 리디렉션, 파이프
│       ├── utils/        # 문자열 유틸, 에러 처리
│       └── main.c        # Shell 진입점 (REPL)
│
│
│
│
│
├── Makefile              # 빌드 설정
```

Directory structure

디렉터리 구조

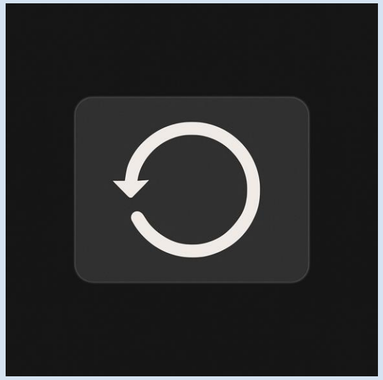
CHAPTER 03

구조 설계 및 기능 구현



CHAPTER 03

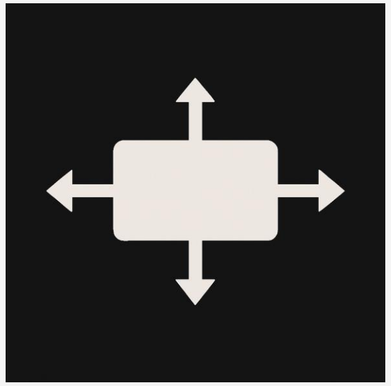
주요 구성



01

입력 루프 (Loop)

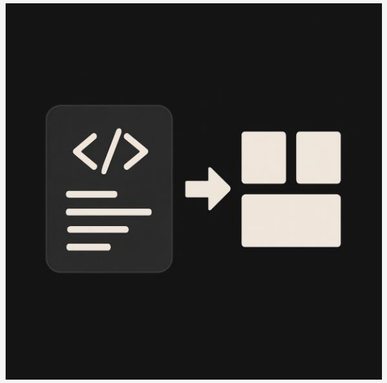
사용자로부터 명령어를 입력받고 반복 실행



02

runAPP

애플리케이션 전체적인 동작을 컨트롤(분기)



03

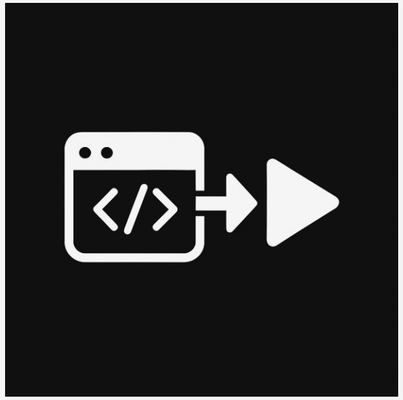
parser

명령어 문자열 파싱 및 토큰화

04

builtin

내부 명령어 처리 (cd, exit, export 등)



05

executer

명령 실행 흐름 제어 (fork, exec, wait)

CHAPTER 03

구조 설계 및 기능 구현

```c
// 파이프 포함한 명령어 한 줄 모두 parsing
ParsedCommand* parseCommand(char *input) {
  // 이중 연결 리스트로 파싱된 명령어를 저장
  ParsedCommand* head = NULL;
  ParsedCommand* tail = NULL;

  char* saveptr;
  char* commandStr = strtok_r(input, "|", &saveptr);

  while (commandStr != NULL) {
    ParsedCommand* command = parseSingleCommand(commandStr);
    if (command == NULL) {
      fprintf(stderr, "parse error\n");
      freeParsedCommand(head);
      return NULL;
    }

    if (head == NULL) {
      head = command;
      tail = command;
    } else {
      tail->next = command;
      tail = command;
    }

    commandStr = strtok_r(NULL, "|", &saveptr);
  }

  return head;
}
```

Pipeline

파이프 라인

CHAPTER 03

구조 설계 및 기능 구현

```c
if (strcmp(token, "<") == 0) {
    command->isAppend = 0;
    token = strtok(NULL, " ");
    if (token) {
        free(command->inputFile); // 기존 inputFile 있다면 해제
        command->inputFile = strdup(token);
        if (!command->inputFile) {
            fprintf(stderr, "Failed to strdup inputFile\n");
            freeParsedCommand(command);
            return NULL;
        }
    }
} else if (strcmp(token, ">") == 0) {
    command->isAppend = 0;
    token = strtok(NULL, " ");
    if (token) {
        free(command->outputFile);
        command->outputFile = strdup(token);
        if (!command->outputFile) {
            fprintf(stderr, "Failed to strdup outputFile\n");
            freeParsedCommand(command);
            return NULL;
        }
    }
} else if (strcmp(token, ">>") == 0) {
    command->isAppend = 1;
    token = strtok(NULL, " ");
    if (token) {
        free(command->outputFile);
        command->outputFile = strdup(token);
        if (!command->outputFile) {
            fprintf(stderr, "Failed to strdup outputFile for append\n");
            freeParsedCommand(command);
            return NULL;
        }
    }
```

Redirection

리다이렉션 (<)

CHAPTER 05

문제점과 해결 과정

문제점

- 시간부족으로 인한 DBMS 연동 구현 실패
- 협업 경험이 부족 GitHub 사용이 익숙하지 않음
- C언어 경험부족으로 인한 어려움
- 구조를 분리하거나 함수 단위를 처리하는데 어려움

해결 방안

- 시간여유가 될 때 따로 DBMS구현해보기
- 일요일 코드 리뷰 시간에 코드를 함께 공유하며 개선
- 리팩토링을 통해 코드 품질을 개선
- Makefile 작성으로 빌드 효율을 높이고 생산성도 향상

QNA