# Cykor week2 과제 보고서

## 2025.4.10

vsc(visual studio code)에서 리눅스(wsl)를 활용해 컴파일, 실행을 하기 위해 vsc에 wsl 확장자를 설치와 각종 환경설정 (visual studio로 하려고 했으나 리눅스 환경은 vsc가 더 적합하다고 생각)
username, password를 입력 받고 <unistd.h>의 gethostname을 사용하여 hostname을 받음
username과 hostname을 이용하여 프롬프트 창을 구현하고 exit을 입력하면 종료되게 함

## 2025.4.15

scanf의 버퍼오버플로우, 공백문자, 개행문자 처리를 용이하게 하기 위해 scnaf를 fgets로 변경
문자열은 string.h 라이브러리로  다룸
unistd.h의 getcwd, chdir 함수를 사용하여 pwd, cd 구현(절대경로, 상대경로 둘다 가능)
메인함수에서 각 기능을 구현한 후 다른 함수로 분리, 메인함수에서 함수를 호출하여 사용

## 2025.4.16

dirent.h의 opendir, closedir, readdir, dirent 구조체를 사용하여 함수를 사용하여 ls 구현
ls 구현시 -a 옵션을 추가하여 '.'으로 시작하는 파일은 -a 옵션을 붙여야지 볼 수 있게함
입력받은 문자열을 확인할때 끝까지 비교하는 strcmp와 정해진 개수만 비교하는 srtncmp 사용
에러코드를 printf로 출력하는 것에서 perror로 출력하는 것으로 변경(에러의 자세한 내용 출력)
fopen, fclose,putchar로 cat구현
/hoem/username 을 ~와 동일시 하게 구현

## 2025.4.18

멀티파이프라인, 다중명령어를 처리하기 위한 방법으로 트리구조를 선택(파이프라인이 우선순위)
입력된 프롬프트를 트리구조로 표현하기 위한 CommandType과 Command 구조체 생성
아직 파싱, 구조체 트리 생성, 실행 구조를 정확히 이해하지 못해 구현은 아직 X

## 2025.4.20

input을 띄어쓰기로 구분하여 토큰화하는 tokenizer 함수 생성
파이프라인, 다중연산자 기호가 포함되지 않은 명령어 부분을 생성하는 parse_command 함수 생성
CommandType 이 파이프라인인 명령어를 생성하는 parse_pipeline 함수 생성
CommandType 이 다중연산자인 명령어를 생성하는 parse_sequence 함수 생성
Command의 트리 구조를 출력하는 print_command_tree 함수 생성
tokens 문자열 포인터 변수를 전역변수로 선언하여 parse 함수들에서 직접 접근하고, pos라는 현재 토큰 위치를 표시하는 int형 변수를 만들어 parse 함수들 끼리 주고 받음
디버깅 시 오류 방지를 위해 전에 만들어 놓았던 함수는 exit을 제외하고 주석처리
동적할당한 메모리를 제대로 해제하지 않아, left,right,type 파일이 생성되는 것을 발견하고 Command 트리구조의 메모리를 해제하는 free_command함수 생성
아직 실행은 안됨

## 2025.4.21

실제 bash에서는 ‘cat main.c|grep "error"|sort|uniq’ 이런식으로 파이프라인이나 다중연산자 앞뒤로 공백이 없어도 정상작동한다는 점을 알게되어 단순히 strtok함수를 사용하여 공백을 기준으로 토큰화하던 tokenize함수를 반복문을 통해 공백은 건너뛰고 특수문자는 특수문자 끼리 하나의 토큰을 만들고 일반 문자열은 공백이나 특수문자가 나올때까지 돌면서 하나의 토큰을 만들게 수정함
&연산자를 통한 백그라운드 프로세스를 구현(command 구조체에 is_background : int형 변수 추가) parse_commnad 단계에서 한 명령어 set의 맨 앞이나 뒤에 &기호 있을 경우 is_background를 1로 변환

## 2025.4.23

반복문을 돌면서 한 바퀴가 끝났을때 tokens 전역변수를 초기화 되게 수정

## 2025.4.30

switch case문을 이용하여 cmd의 CommandType(enum으로 정의)로 case를 분리하고 각 타입에 맞게 실행
모든 케이스에서 프로세스는 fork함수를 사용하여 반환된 값을 pid에 저장하여 사용
CMD_NORMAL : 프로세스 생성 후 실행, 종료될 때 까지 대기
CMD_SEQUENCE : left, right를 순차적으로 무조건 실행
CMD_AND : left 실행 후 left 성공 시에만 right 실행
CMD_OR : left 실행 후 left 실패 시에만 right 실행
CMD_PIPELINE : 파일 디스크립터(fd[2]), 파이프 생성후 left에서 fd[1](쓰기)를 리다이렉션 하고 fd[0,1]닫음

파일 디스크립터(fd[2]), 파이프 생성후 right에서 fd[0](읽기)를 리다이렉션 하고 fd[0,1]닫음
switch case문에서 pid와 status 변수가 중복 선언되는 문제가 생겨 각 case 문에 중괄호를 씌워서 해결(중괄호 안에서는 중괄호 밖과 같은 형태의 변수 선언 가능)
execute_command함수에서 함수 실행시 cmd→is_background가 참일때만 waitpid를 하게 하여 백그라운드 구현

## 2025.5.2

직접 구현한 내장 명령어들은 프로세스를 생성하여 작동하지 않기 때문에 다중명령어나 파이프라인이 쓰인 명령어문 에서는 pwd,ls,cat같은 함수도 execvp 하여 실행하고 단일 명령어 일때만 직접 구현한 명령어를 실행하게 함
cd는 프로세스를 생성하는 것이 아닌 원래 쉘에서 실행해야하기에 명령어가 cd일때 따로 실행하게 함
and,or를 실행할때 cmd→left의 cd를 직접 실행하고 status에 cd 성공여부를 저장하기 위해 cd를 int형으로 바꿔 성고하면 0, 실패하면 1을 반환하게 함
파이프라인에서 cmd→left에 cd 명령어가 들어오면 오류를 내고 함수를 종료하게 함
기존의 구현한 명령어들이 input을 통으로 받아 실행하던 것을 cmd→argv를 전달하여 작동하게 바꿈
ls 에서 -a 옵션을 확인하는 과정에서 cmd→argv[1]이 -a인지를 확인했는데 이 방식이 argv[1]이 없을때 에러가 발생해, argv[1]이 존재 하는지를 판단하는 조건 추가해서 해결

## 2025.5.3

단일 함수들의 백그라운드 처리 추가
execute_command 함수에 exit 이 들어왔을 때 예외처리 추가
print_tokens와 print_tree 변수를 만들어 원할 때 토큰이나 명령어 트리구조를 출력할 수 있게 함
전역변수를 그대로 놔두고 파일을 나눠서 makefile을 하려고 했으나 전역변수 때문에 생긴 오류가 잘 해결되지 않아, 전역변수로 선언했던 것들을 main에서 선언하고 필요할 때 매개변수로 넘기는 식으로 수정
그러나 수정하고 나니 명령어를 입력했을 때 실행은 되지만 곧 바로 Segmentation fault (core dumped)오류가 나면서 프로그램이 종료됨
매인함수에서 선언한 배열들을 초기화 하여 해결(전역변수는 선언만하면 자동으로 초기화 됨)

## 2025.5.4

원래 하나의 파일에 모두 저장되어 있던 코드를 shell.h, parser.c, executor.c, utils.c, main.c로 나눠서 저장함
Makefile 을 통해 오브젝트 파일( main.o, parser.o, executor.o, utils.o)로 컴파일 하고

이 4개의 파일을 링크하여 MyBash라는 실행 파일을 만듦