1st READ ME

이번 과제 제출 결과물은 (1) mainstream.c (2)directory_struct.c (3)directory_struct.h (4)makefile 의 총 4가지 코드들로 구성되어 있습니다.

mainstream.c	리눅스의 전반적인 명령어 실행을 위한 코드들이 담겨있는 c파일
directory_struct.c	디렉토리 구조를 표현한 c파일
directory_struct.h	Struct로 표현한 디렉토리 구조를 mainstream과 directory_struct에서 참조할 수 있게 하기 위해 만든 헤더파일

*makefile은 단순히 세 파일을 동시에 컴파일 할 수 있게 도와주는 makefile 그 자체입니다.

[개발한 리눅스 내 환경 정보]

*user name = oepickle

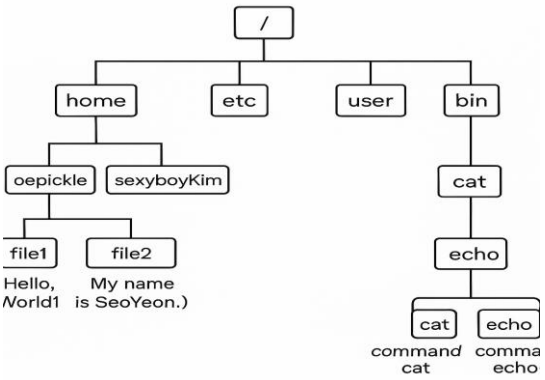
*host name = UNI-CTJ

[구현한 명령어들]

;, &&, ||, |

Cd(.., .까지), echo, pwd, exit, cat, ls

[디렉토리 구조도]



이 보고서는 ***directory_struct.h***와 ***directory_struct.c***에 대한 내용만 담고 있습니다.

directory_struct.h

```
#ifndef DIRECTORY_STRUCT_H
#define DIRECTORY_STRUCT_H

typedef struct Node {
    char name[64];
    struct Node* parent;
    struct Node* child;
    struct Node* sibling;
    struct File* file;
} Node;

extern Node* root;
```

이 파일은 mainstream과 directory_strucy에 둘 다 참조될 헤더파일이기에, 헤더 파일 중복 방지를 위해 매크로를 붙였다. (#ifndef~, #define~, #endif, #pragma once)

그 다음 디렉토리를 struct를 활용해 표현했다.
Struct로 Node를 선언한 후, 그 안에 디렉토리 이름, 부모, 자식, 이웃, 파일을 구조체 포인터로 연결해줬다.

Char name[64]	디렉토리 이름을 표현한 char 배열
Struct Node* parent	부모 노드를 가리키는 포인터
Struct Node* child	자식 노드를 가리키는 포인터
Struct Node* sibling	이웃 노드를 가리키는 포인터
Struct File* file	파일을 가리키는 포인터

```
void directoryStart();
```

*directroryStart()

이 함수는, directory_struct.c c파일에 들어있는 디렉토리 구조를 만드는 함수이다. 이 함수가 존재한 다는 것을 헤더파일로써 알려주기 위해 directory_struct.h에 선언만 해 놓았다.

```
typedef struct File {
    char name[64];
    struct Node* parentNode;
    struct File* next;
    char text[200];
} File;

#endif
#pragma once
```

디렉토리를 표현한 것과 비슷하게 파일 구조도 struct를 활용해 표현했다.

Struct로 선언한 File 안에는, 파일의 이름, 파일을 담고 있는 디렉토리 이름, 현재 파일과 연결되어 있는 이웃 파일, 파일을 속 내용을 담고 있다.

Char name[64]	File 이름을 표현한 char 배열
Struct Node* parentNode	File을 담고 있는 부모 디렉토리를 가리키는 구조체 포인터
Struct File* next	같은 디렉토리의 다른 하위 파일을 연결하는 file을 가리키는 포인터
Char text[100]	파일 내용을 담고 있는 char 배열

directory_struct.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "directory_struct.h"

void directoryStart();

Node* root = NULL;
```

Directory_struct.c는 directoryStart()라는 함수를 위해 만들었다. 이 directoryStart()라는 이름의 함수는 디렉토리 구조를 만드는 함수라서, directory_struct.c 파일에서 따로 구현 후 mainstream과 같이 컴파일 할 예정이다.

먼저 void directoryStart()로 파일을 선언해주고
Root 디렉토리의 메모리 공간을 할당해주기 위해
Node* root = NULL이라고 선언했다.

```
void directoryStart() {
    root = malloc(sizeof(Node));
    strcpy(root->name, "/");
    root->parent = NULL;
    root->child = NULL;
    root->sibling = NULL;
    root->file = NULL;

    Node* bin = malloc(sizeof(Node));
    strcpy(bin->name, "bin");
    bin->parent = root;
    bin->child = NULL;
    bin->sibling = NULL;
    bin->file = NULL;
```

malloc	Node 포인터 선언(malloc으로 동적할당)
strcpy	Name의 char 배열에 이름을 저장
O->parent	해당 Node의 parent 포인터를 미리 선언함
O->child	해당 Node의 child 포인터를 미리 선언함
O->sibling	해당 Node의 sibling 포인터를 미리 선언함
O->file	해당 Node의 file 포인터를 미리 선언함

사실 parent, child, sibling, file은 미리 NULL로 안적어줘도 됐을 것 같은데 가독성을 위해 이렇게 적어주었다.

*bin의 경우 root가 미리 선언되어 있기 때문에, bin->parent 부분만 NULL로 선언하지 않고,

"Bin->parent = root"

로 작성하여 bin에서 root를 이어주었다.

*Bin, user, ect, home, oepickle, sexyboyKim 등 모든 디렉토리를 다음과 같은 구조로 작성함.

```c
Node* user = malloc(sizeof(Node));
strcpy(user->name, "user");
user->parent = root;
user->child = NULL;
user->sibling = bin;
user->file = NULL;

Node* ect = malloc(sizeof(Node));
strcpy(ect->name, "ect");
ect->parent = root;
ect->child = NULL;
ect->sibling = user;
ect->file = NULL;

Node* home = malloc(sizeof(Node));
strcpy(home->name, "home");
home->parent = root;
home->child = NULL;
home->sibling = ect;
home->file = NULL;

root->child = home;
```

위에서 언급했던 것처럼 위 구조와 같이 작성함.

*user도 같은 구조로 작성했는데
Root와 bin은 미리 작성해 놓았기 때문에, User->parent와 user->sibling은 NULL이 아닌 값으로 이어주었다.

*ect와 home도 비슷한 논리로, 미리 선언되어 있는 포인터들과 연결할 만하면 NULL이 아닌 그 값들로 미리 연결해주었다.

이런식으로 bin, user, ect, home을 다 선언했다...

[sibling struct 연결 구조도]
Home -> ect -> user -> bin

일부러 bin -> home은 sibling으로 이어주지 않았다.
(Cd를 돌릴 때, 무한루프가 돌지 않도록 하기 위함)

```c
Node* user1 = malloc(sizeof(Node));
strcpy(user1->name,"oepickle");
user1->parent = home;
user1->child = NULL;
user1->sibling = NULL;
user1->file = NULL;

Node* user2 = malloc(sizeof(Node));
strcpy(user2->name, "sexyboyKim");
user2->parent = home;
user2->child = NULL;
user2->sibling = NULL;
user2->file = NULL;

user1->sibling = user2;
home->child = user1;
```

Home 아래 들어갈 디렉토리인 user1과 user2도 같은 원리로 만들어주었다.

Home
ㄴoepickle
ㄴsexyboyKim

그 후 User1->sibling = user2로 이어줬다.

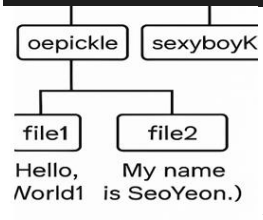
[home 속의 sibling 디렉토리 구조도]

User1 -> user2

얘도 user2->user1은 일부러 이어주지 않았다. Cat을 돌릴 때 이렇게 이어버리면 무한루프에 빠지게 될 것 같아서.

```c
//----------file make--------------
File* lib1 = malloc(sizeof(File));
strcpy(lib1->name, "file1");
lib1->parentNode = user1;
lib1->next = NULL;
strcpy(lib1->text, "Hello, World1");

File* lib2 = malloc(sizeof(File));
strcpy(lib2->name, "file2");
lib2->parentNode = user1;
lib2->next = NULL;
strcpy(lib2->text, "My name is SeoYeon.");
```



<- [구조도]

이 다음부터 이제 디렉토리가 아닌 파일을 만드는 파트이다.

malloc	file 포인터 선언(malloc으로 동적할당)
Strcpy(1)	File의 이름을 char 배열에 저장
O->parentNode	해당 file의 parent 포인터 잇기
O->next	해당 file의 이웃 file 잇기
Strcpy(2)	File의 내용을 char 배열에 저장(text)

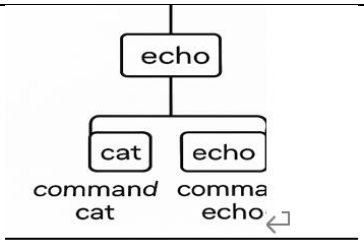
이런 구조를 띄게끔 코드를 작성하며 lib1과 lib2를 만들고, 이 두 파일을 user1(oepickle)밑으로 넣어주었다.

```
File* commandCat = malloc(sizeof(File));
strcpy(commandCat->name, "cat");
commandCat->parentNode = bin;
commandCat->next = NULL;
strcpy(commandCat->text, "command cat");

File* commandEcho = malloc(sizeof(File));
strcpy(commandEcho->name, "echo");
commandEcho->parentNode = bin;
commandEcho->next = NULL;
strcpy(commandEcho->text, "command echo");

commandCat->next = commandEcho;
bin->file = commandCat;

lib1->next = lib2;
user1->file = lib1;
```



<- 이 부분에 해당되는 file들을 만든 코드이다.

위에서 언급한 구조대로 모두 코드를 짰다.

Cat 실행 시 무한루프에 빠지지 않도록 여기서도

Lib1-> lib2

이렇게만 이어주었다.