선배와 함께하는 실전 설계 프로그래밍
최종보고서

1. 팀 정보

구분		이름	학년	학번	학부/과	연락처
튜터		김수만	4			
튜티	1	신준호	2			
	2	은지우	2			
	3	이익준	2			

2. 프로그래밍 능력 향상을 위해 학습한 내용

C 프로그래밍 실력 향상을 위해 다음과 같이 멘토링을 진행함.
1. stack, queue, tree, graph, linked list와 같은 자료구조
2. sorting algorithm(bubble, selection, insertion, heap, merge, quick)
3. tree, graph 심화 학습
- binary tree, complete binary tree, binary search tree
- inorder, preorder, postorder
- heap 자료구조, heap에서의 삽입과 삭제
- 그래프 최단 경로 탐색 알고리즘: dijkstra, floyd
- DFS, BFS 방식
4. 간단한 dynamic programming 적용
- fibonacci 수열
- 재귀 함수 호출을 dynamic programming방식으로 바꾸기
5. 위의 내용과 관련된 백준 문제 풀이

3. 설계 프로젝트 내용

linux/unix계열의 파일시스템에서 사용 가능한 디렉토리 탐색 프로그램을 프로젝트 주제로 정하게 되었음. 트리와 같은 형태를 하고있는 파일시스템을 DFS 또는 BFS방식으로 탐색하면서 탐색할 디렉토리와 원하는 파일명을 입력받아 해당 디렉토리의 크기를 구하고, 트리 형태로 터미널에 출력하고, 원하는 파일을 찾아 해당 경로를 출력할 수 있는 기능을 담고 있음.

4. 설계 프로젝트 요구 사항 분석

1. 리눅스의 파일과 디렉토리 구조에 대해 이해하고, 이를 적용한다. 세부적으로, DIR 구조체를 사용하여 디렉토리 내의 파일을 탐색한다. (opendir, readdir, closedir 등의 함수 사용)
2. 파일의 이름, 크기 등 구체적인 정보를 얻기 위해, stat 구조체의 사용법을 파악한다.
3. 자료구조인 스택, 큐, 트리에 대해 이해한다.
4. DFS와 BFS의 개념 및 작동 원리를 이해하고, 재귀함수와 스택 또는 큐 등을 필요에 따라 각 탐색에 활용한다.
5. 접근 가능한 모든 파일들을 순회하여 탐색하며, 탐색한 내용을 바탕으로 파일 또는 디렉토리의 경로와, 디렉토리 내부 파일 크기의 총합, 내부 구조(트리 형태)를 출력한다.

5. 설계 프로젝트 상세 설계

[기능]
<공동>
[함수 헤드파일]

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <dirent.h>
#include <pwd.h>
#include <errno.h>
```

[함수 프로토타입]
<기능1 함수>
char* extract_Filename(char*);
char* set_Parentdir_Path(char*);
void before_Search(char*, char*, int);
_Bool isEmpty(int);
void enQue_for_Path(QUE*, struct NODE*);
void deQue_for_Path(QUE*);
struct NODE* create_NODE(DIR* dp, char* Nname);
void Bfs_for_Path(char* toFind, char* workDir);
void Dfs_for_Path(char*, char*);
<기능2 함수>
char* absolute(char*);
void bfs_or_dfs(char*,int);
int Bfs_for_Size(char*);
void enqueue_for_Size(Queue* , char*);
void dequeue_for_Size(Queue*);
_Bool isFull(Queue*);
void expand_Capacity(Queue*);
int Dfs_for_Size(char*);
void push(Stack* , NODE*);
NODE* initNODE(DIR*, char*, NODE*);
void pop(Stack*);
<기능3 함수>
void init(char *);
void selectmod(char *, int);
char *strrev(char *);
void Bfs_for_SearchTree(char *, char *);
int havedir(char *);
void Dfs_for_SearchTree(char *, char *);
void Dfs_for_PrintTree(int, char *);

[사용 구조체]
NODE // Stack과 QUE에 쓰일 노드
-DIR *dp : Nname 디렉토리의 readdir 진행상황을 담은 DIR*
-char Nname[max] : 디렉토리 경로
-NODE *next : 스택에서 자신보다 아래 있는 노드를 가리킴
Stack
-NODE* top : top노드의 포인터
-int size : Stack 속의 NODE 개수
Queue //argv[1] - Size
-char (*pathptr)[STR_MAX] : 디렉토리 경로를 담을 배열포인터
(문자열 최대길이 : STR_MAX)
-int front : pathptr배열의 front 인덱스
-int rear : pathptr배열의 rear 인덱스
-int capcity : pathptr이 담을 수 있는 문자열의 개수
QUE //argv[2] - Path
- struct NODE* front; : 탐색 경로를 담은 front 노드
- struct NODE* rear; : 탐색 경로를 담은 rear 노드
- int qSize: : QUE 속의 NODE 개수

<1번: path>
[함수 기능]
1. 프로그램이 위치한 디렉토리 또는 루트 디렉토리로부터 하위 디렉토리로 탐색하여 argv[2]로 입력받은 이름의 파일/디렉토리 찾기.
2. 원하는 파일/디렉토리의 이름을 찾았을 경우, 해당 경로를 출력하기.

[In/Out]
Input: 찾고 싶은 파일/디렉토리의 이름(경로도 가능)
Output: 찾은 파일/디렉토리의 절대 경로 출력

[함수 호출 관계]
main - extract_Filename
main - before_Search - set_Parentdir_Path
main - before_Search – Dfs_for_Path
main - before_Search - Bfs_for_Path - create_NODE – isEmpty
main - before_Search - Bfs_for_Path - isEmpty
main - before_Search - Bfs_for_Path - enQue_for_Path - isEmpty
main - before_Search - Bfs_for_Path - deQue_for_Path – isEmpty

[함수 작동]
1. main
- 입력받은 argv[2]에서 파일/디렉토리의 이름을 추출해(extract_Filename) toFind에 넣고, before_Search함수를 호출한다.
2. before_Search

(1) - argv[2] 중 탐색하지 않을 이름 중 현재 디렉토리(".")는 현재 경로를 불러와(getcwd) 바로 출력하고, 부모 디렉토리("..")는 현재 경로를 불러와(getcwd) 부모 경로를 설정한 후(set_Parentdir_Path) 출력한다.
- argv[2] 중 탐색할 이름은 홈에서부터 탐색할 경로와 루트부터 탐색할 경로로 구분하여, 각각 getcwd와 getpwuid 함수를 통해 탐색할 경로를 workDir에 넣는다.
(2) Dfs_for_Path 함수 또는 Bfs_for Path 함수를 호출한다.
3. Dfs_for_Path
- 원하는 파일/디렉토리를 찾을 때까지 또는 모든 디렉토리를 탐색할 때까지 Dfs_for_Path 함수를 재귀호출한다.
4. Bfs_for_Path
- 맨 처음 탐색할 경로를 create Node하고, QUE가 isEmpty하지 않을 동안, readdir을 하여 해당 dir이 디렉토리일 경우 enQue하고, 해당 dir을 다 읽었을 경우 deQue한다.
5. enQue
- isEmpty일 경우, 노드를 새롭게 추가하며, isEmpty가 아닐 경우, 노드를 노드리스트의 rear에 추가한다.
6. deQue
- isEmpty일 경우, 오류를 출력하고, isEmpty가 아닐 경우, 노드리스트의 front 노드를 삭제한다.

<2번: Size>
[함수 기능]
1. 원하는 디렉토리의 경로(argv[1])를 통해 해당 디렉토리 내에 있는 접근 가능한 모든 파일을 찾아서 그 크기의 총합을 구하고 출력하기.

[In/Out]
input : 파일의 크기를 구하고 싶은 directory 경로(절대/상대 모두 가능)
output : 입력으로 받은 directory 속 존재하는 읽을 수 있는 모든 file의 크기 총합

[함수 호출 관계]
main - absolute
main - bfs_or_dfs - Bfs_for_size - enqueue_for_Size - isFull
main - bfs_or_dfs - Bfs_for_size - enqueue_for_Size - expend_Capacity
main - bfs_or_dfs - Bfs_for_size - dequeue_for_Size - isEmpty
main - bfs_or_dfs - Dfs_for_size - initNODE
main - bfs_or_dfs - Dfs_for_size - push
main - bfs_or_dfs - Dfs_for_size - pop

[함수 작동]
1. absolute를 통해 입력받은 경로를 절대경로로 변환
2. bfs_or_dfs를 통해 main에서 받은 BFS/DFS 선택지에 따라 Bfs_for_Size, Dfs_for_Size 호출
3. Bfs_for_Size : 원형 큐를 이용해서 Bfs 전체 탐색
   1) 입력받은 디렉토리 enqueue_for_Size
   2) readdir로 front 디렉토리 읽다가 새로운 디렉토리 발견 시 enqueue_for_Size
   3) readdir로 전부 읽은 후에 다 읽은 디렉토리(front) dequeue_for_Size
4. enqueue_for_Size : isFull()에 해당할 때 expand_Capacity로 저장공간을 늘려준 후, rear에 디렉토리 경로 추가
5. isFull : 큐에서 rear 2칸 뒤가 front일 때 true
6. expand_Capacity : 큐의 내용을 복사해서 임시 저장 후 두 배의 크기를 할당 후 front=0이 되도록 임시 저장된 내용 붙여넣기
7. dequeue_for_Size : isEmpty()에 해당할 때 강제 종료하고, 이외에는 front에 있는 디렉토리 경로 제거
8. isEmpty : 큐의 현재 사용 중인 크기인 rear-front+1으로 0이 될 때 true
9. Dfs_for_Size : 스택을 이용해서 Dfs 전체 탐색
   1) 입력받은 디렉토리 push
   2) readdir로 top디렉토리 읽다가 새로운 디렉토리 발견시 push ->push된 새로운 디렉토리 읽기 시작함
   3) readdir로 전부 읽게 된다면 pop
10. initNODE : 현재 디렉토리 경로와 readdir진행상황을 담은 dp를 담은 노드 메모리 할당&리턴
11. push : top에 해당 노드 추가
12. pop : top에 있는 노드 삭제

<3번: Tree>
[함수 기능]
1. 프로그램이 위치한 디렉토리로부터 하위 디렉토리로 탐색하여 원하는 디렉토리명 또는 경로(argv[1]) 찾기.
2. 원하는 디렉토리를 찾았을 경우, 해당 디렉터리를 트리 구조로 출력하기.

[In/Out]
input : 트리 구조를 출력하고 싶은 directory 이름 또는 경로(절대/상대 모두 가능)
output : 입력으로 받은 directory의 트리 구조

[함수 호출 관계]
main – init – selectmod – Bfs_for_SearchTree – havedir – Dfs_for_PrintTree
main – init – selectmod – Dfs_for_SearchTree – Dfs_for_PrintTree

[함수 작동]
1. 디렉터리명 또는 경로를 입력받을 char 배열 전역변수를 할당
2. 프로그램이 위치한 디렉터리에서부터 시작하여 해당 디렉터리명(경로)과 일치하는 디렉터리를 탐색(BFS 또는 DFS), 디렉터리를 발견했으면 트리 형태로 출력(DFS)하고 종료
3. 일치하는 디렉터리가 없을 시 각 알고리즘에 따라 하위 디렉터리로 이동, 디렉터리 탐색 중 ".", ".."는 고려하지 않음
4. main 함수에서 디렉터리를 탐색하는 방식을 BFS와 DFS 중 택함, 탐색 시작 경로(프로그램이 위치한 경로)를 출력하고 디렉터리를 탐색, 탐색한 디렉터리를 트리 형태로 출력하고 종료

6. 설계 프로젝트 구현

```
/* < 기능1: path > */
/*
   isEmpty
        # 사용: enQ, deQ, BFS(while시작)
 */
_Bool isEmpty(int qSize){
        return (qSize==0); //qSize를 통해 q가 비어있는지 확인, 비어있으면 리턴 true
}

/*
   enQue_for_Path
        # 사용: BFS(firstNODE, S_ISDIR)
 */
void enQue_for_Path(QUE* q, struct NODE* newnode){

        if (isEmpty(q->qSize)){ //q가 비어있는 경우
                q->front = q->rear = newnode; //q의 front와 rear를 newnode로 일치
        }
        else { //q가 비어있지 않은 경우
                q->rear->next = newnode; //q의 끝에 newnode를 배치
                q->rear = q->rear->next; //q의 끝 설정
        }

        q->qSize++; //qSize Up
}
```

```
/*
   deQue_for_Path
    # 사용: BFS(while끝)
 */
void deQue_for_Path(QUE* q){

        if (isEmpty(q->qSize)){ //q가 비어있는 경우(deQ 불가능)
                fprintf(stderr,"delete error: QisEmpty\n");
                return;
        }

        //q의 front를 deQ
        struct NODE *tmp = q->front;
        q->front = q->front->next;
        free(tmp);

        q->qSize--; //qSize Down
}



/*
   create_NODE
     # 사용: BFS(firstNODE, S_ISDIR)
 */
struct NODE* create_NODE(DIR* dp, char* Nname){

        struct NODE* newnode = (NODE*)malloc(sizeof(NODE));

        //init newnode
        newnode->dp = dp;
        strcpy(newnode->Nname, Nname);

        return newnode;
}



/*
```


스파르탄SW교육원 | Soongsil University

```
   Bfs_for_Path
        # 인자: toFind - 찾을 파일 또는 디렉터리의 이름 / workDir - 탐색 시작 경로
        # 결과: 찾은 경우, 그 경로를 출력함 / 못 찾은 경우, 아무 것도 출력하지 않음
        # 방식: 큐를 활용해 넓이 우선 탐색
        # 사용: before_Search(끝)
 */
void Bfs_for_Path(char* toFind, char* workDir){

        //prepare
        struct dirent *dir = NULL;
        QUE q = {NULL, NULL, 0};
        struct NODE* firstNODE = create_NODE(NULL, workDir); //탐색 시작 경로를
firstNODE로 만들어 enQ
        enQue_for_Path(&q, firstNODE);
        if ((q.front->dp = opendir(workDir))==NULL){
                perror("Error Occurred!\n");
                exit(1);
        }


        //bfs
        while(!isEmpty(q.qSize)){ //q가 비어있지 않을 동안
                while((dir = readdir(q.front->dp))!=NULL){ //q의 front->dp가 끝에 도달할 때까
지 읽음
                        struct stat statbuf;

                        //check
                        if (strcmp(dir->d_name, toFind)==0){ //원하는 파일 또는 디렉터리를 찾
았을 경우
                                found++;
                                printf("PATH: %s/%s\n", q.front->Nname, dir->d_name); //
출력 후 break
                                break; //동일한 이름을 가진 파일 또는 디렉터리를 모두 찾기 위
해서 return이 아닌 break
                        }

                        //pass
                        if (strncmp(dir->d_name,".",1)==0){ //.과 ..그리고 .으로 시작하는 파일
(숨김파일 등)은 모두 건너뜀
                                continue;
                        }
```



```
                        //enque
                        char tmp[MAX]; //현재 위치를 tmp에 경로로 나타냄
                        strcpy(tmp, q.front->Nname);
                        strcat(tmp,"/");
                        strcat(tmp, dir->d_name);
                        if (lstat(tmp, &statbuf)<0){ //tmp(현재 위치)를 statbuf에 넣음
                                perror("Stat Error\n");
                                exit(1);
                        }

                        if (S_ISDIR(statbuf.st_mode)){ //dir이면 enQ(추후에 탐색)
                                struct NODE* n = create_NODE(opendir(tmp),tmp);
                                enQue_for_Path(&q,n);
                        }

                }
                closedir(q.front->dp);
                deQue_for_Path(&q); //q의 front를 deQ (새로운 q의 front를 탐색하기 위해)
        }
}



/*
        Dfs_for_Path
        # 인자: toFind - 찾을 파일 또는 디렉터리의 이름 / workDir - 탐색 시작 경로
        # 결과: 찾은 경우, 그 경로를 출력함 / 못 찾은 경우, 아무 것도 출력하지 않음
        # 방식: 재귀를 활용해 깊이 우선 탐색
        # 사용: before_Search(끝)
 */
void Dfs_for_Path(char* toFind, char* workDir) {

        //prepare
        DIR* dp = NULL;
        struct dirent *dir = NULL;
        struct stat statbuf;
        if ((dp = opendir(workDir))==NULL){
                perror("File Open Error\n");
                exit(1);
        }
```

```
        //dfs
        while((dir=readdir(dp))!=NULL){ //dp가 끝에 도달할 때까지 읽음

                //check
                if (strcmp(dir->d_name, toFind) == 0){ //원하는 파일 또는 디렉터리를 찾았을
경우
                        printf("PATH: %s/%s\n", workDir, dir->d_name); //출력 후 return
                        found++;
                        closedir(dp);
                        return; //재귀를 활용했기 때문에 break가 아닌 return
                }

                //pass
                if ((strncmp(dir->d_name,".",1)==0)){ //.과 ..그리고 .으로 시작하는 파일(숨김파일
등)은 모두 건너뜀
                        continue;
                }

                //recurse
                char tmp[MAX]; //현재 위치를 tmp에 경로로 나타냄
                strcpy(tmp,workDir);
                strcat(tmp,"/");
                strcat(tmp,dir->d_name);
                if (lstat(tmp, &statbuf)<0){ //tmp(현재 위치)를 statbuf에 넣음
                        perror("Stat Error\n");
                        exit(1);
                }

                if (S_ISDIR(statbuf.st_mode)){ //dir이면 recurse(탐색)
                        Dfs_for_Path(toFind, tmp);
                }
        }

        closedir(dp);
}



/*
```



```
   extract_Filename
        # 인자: Path - argv[2]로 입력받은 문자열 (찾을 파일/디렉토리의 경로 또는 이름)
        # 목적: toFind에 argv[2]에서 추출한 파일/디렉토리의 이름을 넣음
        # 사용: main
 */
char* extract_Filename(char* Path){

        //No slashes
        if (strstr(Path,"/")==NULL) { // argv[2]에서 파일 또는 디렉터리의 이름을 입력받은 경우
                return Path; //그대로 리턴
        }

        //Slashes
        char* temp = (char*)malloc(sizeof(char)*MAX); //파일 또는 디렉터리의 경로에서 이름만
을 추출하기 위해
        char* ptr = strtok(Path,"/"); // "/" 기준으로 token화
        while (ptr!=NULL) {
                strcpy(temp, ptr); //(결국엔) 마지막 ptr을 temp에 copy
                ptr = strtok(NULL,"/");
        }
        return temp;

}




/*
   set_Parentdir_Path
        # 목적: toFind가 ".."(부모 디렉터리)인 경우, 경로 설정을 위해
        # 사용: before_Search("..")
 */
char* set_Parentdir_Path(char* workDir){
        int i=0;
        char str[MAX], tmp[MAX][20];
        char* ptr = strtok(workDir, "/"); //getcwd로 얻은 경로를 "/"기준으로 token화
        for (i=0; ptr!=NULL; i++){
                strcpy(tmp[i], ptr);
                ptr = strtok(NULL,"/");
        }
        for (int k=0; k<i-1; k++){ //마지막 token을 제외해 경로를 재생성
                strcat(str,"/");
```

```
                strcat(str, tmp[k]);
        }
        workDir = str;
        return workDir;
}




/*
   before_Search
        # 인자: argv - 찾을 파일/디렉토리의 경로 또는 이름 / toFind - 찾을 파일/디렉토리의 이
름 / BD - BFS/DFS
        # 사용: main
 */
void before_Search(char* argv, char* toFind, int BD){

        //Prepare
        char workDir[MAX];
        struct passwd *pwd;
        errno = 0;
        if((pwd = getpwuid(getuid())) == NULL) { //사용자 정보를 불러옴 - 사용자 계정명 등 (모
든 컴퓨터에서 작동이 가능하도록)
                if(errno == 0 || errno == ENOENT || errno == ESRCH || errno == EBADF ||
errno == EPERM) {
                        fprintf(stderr,"미등록된 사용자입니다.\n");
                }
                else {
                        fprintf(stderr,"error: %s\n", strerror(errno));
                }
                exit(1);
        }

        //Branch
        if ((strcmp(".", argv) == 0)||(strcmp("./", argv) == 0)){ //탐색x (.)
                getcwd(workDir,MAX); //현위치 (작업 디렉토리)를 불러와 출력
                printf("Path: %s\n", workDir);
                return;
        }
        else if (strcmp("..", argv) == 0){ //탐색x (..)
                getcwd(workDir,MAX); //현위치를 기준으로 부모 디렉토리의 경로를 설정해 출력
```



```
                printf("Path: %s\n", set_Parentdir_Path(workDir));
                return;
        }
        else if (strncmp("./", argv, 2) == 0){ //탐색o (~)
                getcwd(workDir, MAX); //현재 디렉토리를 workDir에 넣음
        }
        else{ //탐색o (/)
                strcpy(workDir,pwd->pw_dir); //"/home/계정명"을 workDir에 넣음
        }

        //Search
        printf("탐색할 디렉터리 또는 파일 이름(경로) : %s\n", argv);
        if (BD==0){
                Bfs_for_Path(toFind, workDir);
        }
        else{
                Dfs_for_Path(toFind,workDir);
        }

        //전역변수 cnt : 재귀/반복의 횟수
        if (!found){
                printf("해당 디렉터리 또는 파일이 없습니다.\n");
        }

}



/* < 기능2: size > */
/*
        #함수 설명 : 경로를 받아서 절대경로로 바꿔준다.
        #변수 : char* path - 절대/상대 경로
        #리턴값 : path의 절대경로
   */
char* absolute(char* path){
        char *absPath = (char*)malloc(sizeof(char)*MAX);
        char strbuf[MAX]={};
        int intbuf;

        /*path 문자열 처리*/
        if(path[0]=='/')                                                    //path==절대
```

```
경로
		strcpy(absPath,path);
	else{
	//path==상대경로
		getcwd(absPath,MAX);
		if(strcmp(path,".")){
			for(int i=0; path[i]!='\0'; ){
				if(!strncmp(path+i,"../",3)){
					/*가장 뒤에 있는 '/'찾아서'\0'대입*/
					for(int j=0; absPath[j]!='\0';j++)
						if(absPath[j]=='/')
							intbuf = j;
					absPath[intbuf] = '\0';
					i+=3;
				}
				else if(!strncmp(path+i,"./",2)){
					i+=2;
				}
				else{
					strncat(strbuf,path+i,1);
					i++;
				}
			}
			if (strcmp(strbuf,"")){
				strcat(absPath,"/");
				strcat(absPath,strbuf);
			}
		}
	}

	return absPath;
}


/*
	#함수 설명 : 처음 BFS/DFS 선택지에 따라 fileSize함수를 실행하고 결과를 출력한다.
	#변수 : char* absPath - 크기를 구할 디렉토리의 절대 경로
		int BD - BFS/DFS 선택지 (BFS:0, DFS:1)
	#리턴값 : void
   */
void bfs_or_dfs (char* absPath,int BD){
```



```
	int totalSize = 0;

	if(BD==0)
		totalSize = Bfs_for_Size(absPath);
	else if(BD==1)
		totalSize = Dfs_for_Size(absPath);
	else
		perror("Error : unexpected value of valiable \"BD\"!!\n");

	printf("Total : %d\n",totalSize);
}

/*
	#함수 설명 : 해당 디렉토리 내의 모든 파일크기를 DFS로 탐색하여 총합을 리턴
	#변수 : char* absPath - 크기를 구할 디렉토리의 절대 경로
	#리턴값 : 해당 디렉토리 내의 모든 파일크기 합
*/
int Dfs_for_Size(char* absPath){
	struct stat stbuf;
	int totalSize = 0;

	Stack st={NULL,0};
	struct dirent *dir;
	DIR* openable;

	/*stack에 시작 디렉토리 노드 추가*/
	push(&st,initNODE(NULL,"",NULL));

	if((st.top->dp=opendir(absPath))==NULL){
		printf("Error : fail on open directory!\n");
		exit(1);
	}
	strcpy(st.top->Nname,absPath);


	/*전체 탐색 알고리즘:DFS*/
	/*
		#구현 : top에 있는 디렉토리를 readdir로 하위 디렉토리(dir) 읽고 stat을 통해 디
렉토리, 파일 판단
		#dp가 dir일 때 : 해당 dir을 stack의 top에 올려서 읽기 시작
```

```
                #dp가 file일 때 : totalsize에 해당 file사이즈 더하고 정보 출력, 다음 dir로 이동
                #dp가 NULL일 때 : 현재 top에 있는 디렉토리를 pop하고 이전 top에 있던 디렉토
리를 이어서 읽기 시작
                (하위 디렉토리를 다 읽었을 때)
         */
        while(st.size){
        //모든 디렉토리가 스택에서 pop되면 종료
                if((dir=readdir(st.top->dp))!=NULL){
                        if(strcmp(dir->d_name,".")!=0&&strcmp(dir->d_name,"..")!=0){

                        /*탐색 디렉토리 문자열 처리*/
            strcpy(absPath, st.top->Nname);
                        if(absPath[strlen(absPath)-1]!='/')
                    strcat(absPath,"/");
            strcat(absPath,dir->d_name);

            stat(absPath,&stbuf);           //탐색 디렉토리 정보 불러오기

            if((stbuf.st_mode&0xF000)==0x8000){           //dir이 파일일 경우
                                totalSize += stbuf.st_size;
             }
            else{
//dir이 디렉토리일 경우
                                if((openable = opendir(absPath))!=NULL)
                        push(&st,initNODE(openable,absPath,NULL));
                        }
                        }

                }
                else
                //dir이 NULL일 경우
                pop(&st); /*top을 이전에 읽던 디렉토리로 변경*/
        }

        return totalSize;
}

/*
        #함수 설명 : 스택의 top에 노드를 push한다.
        #변수 : Stack *s - 노드를 추가할 스택의 포인터, NODE* n - 추가할 노드의 포인터
        #리턴값 : void
*/
void push(Stack *s, NODE* n){
        NODE* tmp;

        tmp = s->top;
        s->top = n;
    s->top->next = tmp;
        s->size++;
}

/*
        #함수 설명 : 노드를 생성한다.
        #변수 : DIR* newDirp, char* newName, NODE* newNext -새로운 노드의 정보들
        #리턴값 : 해당 정보를 삽입한 새로운 노드의 포인터
*/
NODE* initNODE(DIR* newDp, char* newName, NODE* newNext){
        NODE* new = (NODE*)malloc(sizeof(NODE));

        new->dp = newDp;
        strcpy(new->Nname, newName);
        new->next = newNext;

        return new;
}

/*
        #함수 설명 : 스택의 top 노드를 pop한다.
        #변수 : Stack *s - top을 내보낼 스택의 포인터
        #리턴값 : void
*/
void pop(Stack *s){
   NODE* tmp;

   tmp = s->top;
   s->top = s->top->next;
   free(tmp);
   s->size--;
}

/*
        #함수 설명 : 해당 디렉토리 내의 모든 파일크기를 BFS로 탐색하여 총합을 리턴
```


스파르탄SW교육원 | Soongsil University

```
	#변수 : char* absPath - 크기를 구할 디렉토리의 절대 경로
	#리턴값 : 해당 디렉토리 내의 모든 파일크기 합

*/
int Bfs_for_Size(char* absPath){
	struct stat stbuf;
	int totalSize = 0;
	DIR *dirp;

	Queue Q={(char(*)[STR_MAX])malloc(sizeof(char)*STR_MAX*MAX),0,-1,MAX};

	struct dirent *dir;

	/*queue에 시작 디렉토리 경로 추가*/
	enqueue_for_Size(&Q,absPath);

	if((dirp=opendir(absPath))==NULL){
		printf("Error : fail on open directory!\n");
		exit(1);
	}


	/*전체 탐색 알고리즘:BFS*/
	/*
		#구현 : front에 있는 디렉토리를 readdir로 하위 디렉토리(dir) 읽고 stat을 통해 디렉토리, 파일 판단
		#dp가 dir일 때 : 해당 dir을 queue의 rear에 올려놓고 다음 dir로 이동
		#dp가 file일 때 : totalsize에 해당 file사이즈 더하고 정보 출력, 다음 dir로 이동
		#dp가 NULL일 때 : 현재 front에 있는 디렉토리를 dequeque하고 다음 front 디렉토리를 읽기 시작
		(하위 디렉토리를 다 읽었을 때)
	 */
	while(!isEmpty(Q.rear-Q.front+1)){
		if(dirp==NULL){
			dequeue_for_Size(&Q);
			dirp=opendir(Q.pathptr[Q.front]);
			continue;
		}
		else if((dir=readdir(dirp))!=NULL){		//다음 dir로 이동
			if(strcmp(dir->d_name,".")!=0&&strcmp(dir->d_name,"..")!=0){
```



```
				/*absPath = dir의 절대경로*/
			strcpy(absPath, Q.pathptr[Q.front]);
				if(absPath[strlen(absPath)-1]!='/')
				strcat(absPath,"/");
			strcat(absPath,dir->d_name);
				/*dir의 stat 불러오기*/
			stat(absPath,&stbuf);

			if((stbuf.st_mode&0xF000)==0x8000){		//dir이 파일일 경우
				totalSize += stbuf.st_size;
			 }
			else                                                    //dir이 디렉토리일 경우
				enqueue_for_Size(&Q,absPath);
				}

			}
		else{                                                   //dp가 NULL일 경우
				/*front갱신 후 디렉토리 변경*/
				dequeue_for_Size(&Q);
				dirp=opendir(Q.pathptr[Q.front]);
			}
		}

		return totalSize;
}

/*
		#함수 설명 : 큐의 rear에 노드를 enqueue_for_Size한다, 큐가 가득찼으면 용량을 키운다.
		#변수 : Queue *q - 경로를 집어넣을 큐,
				char* newpath - 큐에 들어갈 문자열의 포인터
		#리턴값 : void
*/
void enqueue_for_Size(Queue *q, char* newpath){

		if(isFull(q))
		    expand_Capacity(q);

		q->rear = (q->rear+1)%q->capacity;
		strcpy(q->pathptr[q->rear],newpath);
```

```
}

/*
	#함수 설명 : 큐의 front 노드를 dequeue_for_Size한다.
	#변수 : Queue *q - front를 내보낼 큐의 포인터
	#리턴값 : void
*/
void dequeue_for_Size(Queue *q){

	if(isEmpty(q->rear-q->front+1)){
		printf("queue is already empty!!\n");
		return;
	}

	q->front = (q->front+1)%q->capacity;
}

/*
	#함수 설명 : 큐가 가득찼으면 1리턴 (rear 다음다음이 front일 때 기준 Full)
	#변수 : Queue *q - 확인할 큐
	#리턴값 : void
*/
_Bool isFull(Queue * q){
	if((q->rear+2)%q->capacity==q->front)
		return 1;
	else
		return 0;
}

/*
	#함수 설명 : 큐의 용량을 증가시킨다. (default : 함수 실행 후 front=0)
	#변수 : Queue *q - 용량을 증가시킬 큐
	#리턴값 : void
*/
void expand_Capacity(Queue* q){
	char (*tmp)[STR_MAX] = (char(*)[STR_MAX])malloc(sizeof(char)*STR_MAX*q->capacity*2);

	for(int i=0; i<q->capacity; i++){
		strcpy(tmp[i],q->pathptr[(q->front+i)%(q->capacity)]);
	}

	free(q);

	q->pathptr = tmp;
	q->rear = q->rear - q->front;
	q->front = 0;
	q->capacity *= 2;
}



/* < 기능3: Tree > */
/*
	#함수 설명 : 탐색할 디렉터리 이름(또는 경로), 탐색 시작 절대경로(프로그램 위치)를 출력하기 위한 함수이다.
	#변수 : char *name - 찾을 디렉터리명(경로)
	#리턴값 : void
*/
void init(char *name){ // 탐색할 디렉터리 이름(또는 경로), 탐색 시작 절대경로(프로그램 위치)출력
	printf("탐색할 디렉터리 이름(경로) : %s\n", name);
	getcwd(wd, BUFSIZ); // 프로그램 위치를 불러오기 위해 사용, wd에 저장
	return;
}

/*
	#함수 설명 : BFS/DFS 둘 중 하나를 선택하게 하는 기능을 위한 함수이다.
	#변수 : char *argv - 찾을 디렉터리명(경로)
	#리턴값 : void
*/
void selectmod(char *argv, int mod){

	if (mod == 0){
		if (strncmp("/", argv, 1) == 0) // 절대 경로로 입력했을 경우
			Bfs_for_SearchTree(argv, "..");
		else if (strncmp("..", argv, 2) == 0) // ..로 시작할 경우
			if ((strcmp("..", argv) == 0) || (strcmp("../", argv) == 0)) // 부모 디렉터리
				Dfs_for_PrintTree(0, "..");
			else if (strncmp("../.", argv, 4) == 0) // 상위 디렉터리
				Dfs_for_PrintTree(0, argv);
```


스파르탄SW교육원 | Soongsil University

```
                        else
                                Bfs_for_SearchTree(argv, ".."); // 프로그램 위치 디렉터리와 같은 깊이의 디렉터리
                else if (strncmp(".", argv, 1) == 0) // .로 시작할 경우
                        if ((strcmp(".", argv) == 0) || (strcmp("./", argv) == 0)) // 현재 디렉터리
                                Dfs_for_PrintTree(0, ".");
                        else
                                Bfs_for_SearchTree(argv, "."); // 하위 디렉터리
                else // dirname 또는 상대경로로 입력했을 경우
                        Bfs_for_SearchTree(argv, ".");
        }

        else if (mod == 1){
                if (strncmp("/", argv, 1) == 0) // 절대 경로로 입력했을 경우
                        Dfs_for_SearchTree(argv, "..");
                else if (strncmp("..", argv, 2) == 0) // ..로 시작할 경우
                        if ((strcmp("..", argv) == 0) || (strcmp("../", argv) == 0)) // 부모 디렉터리
                                Dfs_for_PrintTree(0, "..");
                        else if (strncmp("../.", argv, 4) == 0) // 상위 디렉터리
                                Dfs_for_PrintTree(0, argv);
                        else
                                Dfs_for_SearchTree(argv, ".."); // 프로그램 위치 디렉터리와 같은 깊이의 디렉터리
                else if (strncmp(".", argv, 1) == 0) // .로 시작할 경우
                        if ((strcmp(".", argv) == 0) || (strcmp("./", argv) == 0)) // 현재 디렉터리
                                Dfs_for_PrintTree(0, ".");
                        else
                                Dfs_for_SearchTree(argv, "."); // 하위 디렉터리
                else // dirname 또는 상대경로로 입력했을 경우
                        Dfs_for_SearchTree(argv, ".");
        }
}

/*
        #함수 설명 : BFS 알고리즘으로 찾고자 하는 디렉터리를 탐색하는 함수이다.
        #변수 : char *name, char *wd - 찾을 디렉터리명(경로), 탐색하기 시작할 디렉터리명(경로)
        #리턴값 : void
*/
void Bfs_for_SearchTree(char *name, char *wd){
        struct dirent *entry; struct stat buf; DIR *dp;
        NODE queue[MAX]; int front, rear; front = rear = -1; // 큐 생성

        if (chdir(wd) < 0){ // 디렉터리 이동, 실패 시 프로그램 종료
                printf("오류 발생! 프로그램 종료.\n");
                exit(1);
        }

        if ((dp = opendir(".")) == NULL){ // 디렉터리 열기, 실패 시 프로그램 종료
                printf("오류 발생! 프로그램 종료.\n");
                exit(1);
        }

        while (1){ // BFS 알고리즘

                while ((entry = readdir(dp)) != NULL){ // 현재 디렉터리 내용을 모두 읽었을 때 탈출

                        lstat(entry->d_name, &buf);

                        if (S_ISDIR(buf.st_mode)){ // 하위 디렉터리일 경우

                                char path[MAX];

                                if (strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name, "..") == 0) // ".", ".."은 고려하지 않음
                                        continue;

                                if (strncmp(strrev(name), strrev(entry->d_name), strlen(entry->d_name)) == 0){ // <입력한 이름(경로) == 탐색한 디렉터리명>인 경우

                                        strrev(name); strrev(entry->d_name);
                                        printf("\n%s----\n", entry->d_name); // 입력한(경로의) 디렉터리명 출력
                                        Dfs_for_PrintTree(0, entry->d_name); // Dfs_for_PrintTree함수 호출
                                        closedir(dp); return; // 반복문 탈출, Bfs_for_SearchTree함수 종료
                                }
```


스파르탄SW교육원 | Soongsil University

```
                                else{
                                strrev(name); strrev(entry->d_name);
                                }

                                getcwd(path, BUFSIZ);
                                strcat(path, "/");
                                strcat(path, entry->d_name);

                                if (havedir(path)){ // 탐색 중인 디렉터리가 하위 디렉터리를 보유한 경우
                                        rear++; // 저장할 공간 확보
                                        strcpy(queue[rear].Nname, path);// 대상 디렉터리 경로 저장
                                }

                                chdir("..");
                        }
                }
                // 같은 깊이의 모든 노드들의 탐색이 끝난 경우
                front++;
                closedir(dp); chdir(queue[front].Nname); // 선입선출, 비었으면 함수 종료

                if ((dp = opendir(queue[front].Nname)) == NULL){ // Dequeue한 디렉터리로 이동
                        printf("Dequeue 오류 발생. 프로그램 종료\n");
                        exit(1);
                }
        }
        return;
}

/*
        #함수 설명 : 인자로 받은 경로, 즉 디렉터리가 하위 디렉터리를 가지는지 검사하는 함수이다.
        #변수 : char *path - 검사할 디렉터리 경로
        #리턴값 : 0 (하위 디렉터리가 없을 경우), 1 (하위 디렉터리가 있을 경우)
*/
int havedir(char *path){ // 하위 디렉터리 보유 여부 체크 함수
        struct dirent *entry; struct stat buf; DIR *dp;
```



```
        chdir(path); dp = opendir(path);

        while ((entry = readdir(dp)) != NULL){

                lstat(entry->d_name, &buf);

                if (S_ISREG(buf.st_mode)) continue;

                else if (S_ISDIR(buf.st_mode)){ // 하위 디렉터리가 있을 경우

                        if (strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name, "..") == 0)
                                continue;
                        closedir(dp);
                        return 1; // 1 반환, if문 True

                }
        }

        closedir(dp);
        return 0; // 0 반환, if문 False
}

/*
        #함수 설명 : DFS 알고리즘으로 찾고자 하는 디렉터리를 탐색하는 함수이다.
        #변수 : char *name, char *wd - 찾을 디렉터리명(경로), 탐색하기 시작할 디렉터리명(경로)
        #리턴값 : void
*/
void Dfs_for_SearchTree(char *name, char *wd){ // 입력한 디렉터리를 찾는 함수, dfs 알고리즘
        struct dirent *entry; struct stat buf; DIR *dp;

        if (chdir(wd) < 0){ // 디렉터리 이동, 실패 시 프로그램 종료
                printf("오류 발생! 프로그램 종료.\n");
                exit(1);
        }

        if ((dp = opendir(".")) == NULL){ // 디렉터리 열기, 실패 시 프로그램 종료
                printf("오류 발생! 프로그램 종료.\n");
                exit(1);
```

```
        }

        while ((entry = readdir(dp)) != NULL){ // 현재 디렉터리 내용을 모두 읽었을 때 탈출

                lstat(entry->d_name, &buf);

                if (S_ISDIR(buf.st_mode)){ // 하위 디렉터리일 경우

                        if (strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name, "..") ==
0) // ".", ".."은 고려하지 않음
                                continue;

                        if          (strncmp(strrev(name),          strrev(entry->d_name),
strlen(entry->d_name)) == 0){ // <입력한 이름(경로) == 탐색한 디렉터리명>인 경우
                                strrev(name); strrev(entry->d_name); // 비교 위해 뒤집은 문
자열 원상태로 복구
                                printf("%s----\n", entry->d_name); // 입력한(경로의) 디렉터리
명 출력
                                Dfs_for_PrintTree(0, entry->d_name); // Dfs_for_PrintTree함
수 호출
                                chdir(".."); closedir(dp); return; // Dfs_for_SearchTree 함수
종료
                        }

                        else{
                                strrev(name); strrev(entry->d_name); // 비교 위해 뒤집은 문
자열 원상태로 복구
                        }

                        Dfs_for_SearchTree(name, entry->d_name); // 없을 경우, 재귀 호출
(더 하위 디렉터리로 이동)
                }
        }

        chdir(".."); closedir(dp); // 백트래킹(부모 디렉터리로 올라감)
}

/*
        #함수 설명 : DFS 알고리즘으로 찾은 디렉터리의 하위 파일과 디렉터리들을 트리 구조로 출
력하는 함수이다. (BFS 방식은 트리 구조를 출력하는 것이 까다롭기 때문에 DFS 방식으로 만듬)
        #변수 : int tmp, char *wd - depth 구분하기 위한 정수, 트리 구조를 출력할 디렉터리
```



```
        #리턴값 : void
*/
void Dfs_for_PrintTree(int tmp, char *wd){ // 디렉터리를 트리구조로 출력하는 함수, dfs 알고리
즘
        struct dirent *entry; struct stat buf; DIR *dp; int count = 0;

        count = tmp; // \t 횟수 구분(깊이 동일한 노드들 같은 열에 출력하기 위함)

        if (chdir(wd) < 0){ // dfs()와 동일
                printf("오류 발생! 프로그램 종료.\n");
                exit(1);
        }

        if ((dp = opendir(".")) == NULL){
                printf("오류 발생! 프로그램 종료.\n");
        }

        while ((entry = readdir(dp)) != NULL){ // dfs()와 동일

                lstat(entry->d_name, &buf);

                if (S_ISREG(buf.st_mode)){ // 탐색한 것이 파일일 경우

                        for (int i = 0 ; i <= count ; i++)
                                printf("\t");
                        printf("-%s\n", entry->d_name);

                }

                else if (S_ISDIR(buf.st_mode)){ // 탐색한 것이 디렉터리일 경우
                        if (strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name, "..") ==
0) // dfs()와 동일
                                continue;

                        for (int i = 0 ; i < 3 ; i++){ // 칸을 벌려서 겹치는 경우 예방하기 위함
                                for (int j = 0 ; j <= count ; j++)
                                        printf("\t");
                                printf("|\n");
                        }

                        for (int i = 0 ; i <= count ; i++) printf("\t");
```

```
                        printf("-%s----\n", entry->d_name); // (하위)디렉터리명 출력
                        tmp = count + 1;
                        Dfs_for_PrintTree(tmp, entry->d_name); // 재귀 호출 (더 하위 디렉터
리로 이동)

                        for (int i = 0 ; i < 3 ; i++){ // 위와 동일
                                for (int j = 0 ; j <= count ; j++)
                                        printf("\t");
                                printf("|\n");
                        }

                }
        }

        chdir(".."); closedir(dp); // 백트래킹(부모 디렉터리로 올라감)
}

char *strrev(char *str){ // (문자열 뒤집는 함수(string.h_strrev는 리눅스 사용 불가))
        char *p1, *p2;
        if (!str || !*str)
                return str;
        for (p1 = str, p2 = str + strlen(str)-1; p2 > p1 ; ++p1, --p2){
                *p1 ^= *p2;
                *p2 ^= *p1;
                *p1 ^= *p2;
        }
        return str;
}
```

7. 설계 프로젝트 소스 코드

<함수 부분>

```
#include "head.h"
char wd[MAX];
_Bool found = 0;
```



```
/* < 기능1: path > */
/*
   isEmpty
        # 사용: enQ, deQ, BFS(while시작)
 */
_Bool isEmpty(int qSize){
        return (qSize==0); //qSize를 통해 q가 비어있는지 확인, 비어있으면 리턴 true
}



/*
   enQue_for_Path
        # 사용: BFS(firstNODE, S_ISDIR)
 */
void enQue_for_Path(QUE* q, struct NODE* newnode){

        if (isEmpty(q->qSize)){ //q가 비어있는 경우
                q->front = q->rear = newnode; //q의 front와 rear를 newnode로 일치
        }
        else { //q가 비어있지 않은 경우
                q->rear->next = newnode; //q의 끝에 newnode를 배치
                q->rear = q->rear->next; //q의 끝 설정
        }

        q->qSize++; //qSize Up
}



/*
   deQue_for_Path
    # 사용: BFS(while끝)
 */
void deQue_for_Path(QUE* q){

        if (isEmpty(q->qSize)){ //q가 비어있는 경우(deQ 불가능)
                fprintf(stderr,"delete error: QisEmpty₩n");
                return;
        }

        //q의 front를 deQ
        struct NODE *tmp = q->front;
        q->front = q->front->next;
        free(tmp);

        q->qSize--; //qSize Down
}



/*
   create_NODE
     # 사용: BFS(firstNODE, S_ISDIR)
```

```
 */
struct NODE* create_NODE(DIR* dp, char* Nname){

	struct NODE* newnode = (NODE*)malloc(sizeof(NODE));

	//init newnode
	newnode->dp = dp;
	strcpy(newnode->Nname, Nname);

	return newnode;
}



/*
   Bfs_for_Path
	# 인자: toFind - 찾을 파일 또는 디렉터리의 이름 / workDir - 탐색 시작 경로
	# 결과: 찾은 경우, 그 경로를 출력함 / 못 찾은 경우, 아무 것도 출력하지 않음
	# 방식: 큐를 활용해 넓이 우선 탐색
	# 사용: before_Search(끝)
 */
void Bfs_for_Path(char* toFind, char* workDir){

	//prepare
	struct dirent *dir = NULL;
	QUE q = {NULL, NULL, 0};
	struct NODE* firstNODE = create_NODE(NULL, workDir); //탐색 시작 경로를
firstNODE로 만들어 enQ
	enQue_for_Path(&q, firstNODE);
	if ((q.front->dp = opendir(workDir))==NULL){
		perror("Error Occurred!₩n");
		exit(1);
	}


	//bfs
	while(!isEmpty(q.qSize)){ //q가 비어있지 않을 동안
		while((dir = readdir(q.front->dp))!=NULL){ //q의 front->dp가 끝에 도달할
때까지 읽음
			struct stat statbuf;

			//check
			if (strcmp(dir->d_name, toFind)==0){ //원하는 파일 또는 디렉터리를
찾았을 경우
				found++;
				printf("PATH: %s/%s₩n", q.front->Nname, dir->d_name);
//출력 후 break
				break; //동일한 이름을 가진 파일 또는 디렉터리를 모두 찾기
위해서 return이 아닌 break
			}

			//pass
			if (strncmp(dir->d_name,".",1)==0){ //.과 ..그리고 .으로 시작하는
파일(숨김파일 등)은 모두 건너뜀
				continue;
```



```
			}

			//enque
			char tmp[MAX]; //현재 위치를 tmp에 경로로 나타냄
			strcpy(tmp, q.front->Nname);
			strcat(tmp,"/");
			strcat(tmp, dir->d_name);
			if (lstat(tmp, &statbuf)<0){ //tmp(현재 위치)를 statbuf에 넣음
				perror("Stat Error₩n");
				exit(1);
			}

			if (S_ISDIR(statbuf.st_mode)){ //dir이면 enQ(추후에 탐색)
				struct NODE* n = create_NODE(opendir(tmp),tmp);
				enQue_for_Path(&q,n);
			}

		}
		closedir(q.front->dp);
		deQue_for_Path(&q); //q의 front를 deQ (새로운 q의 front를 탐색하기 위해)
	}
}



/*
	Dfs_for_Path
	# 인자: toFind - 찾을 파일 또는 디렉터리의 이름 / workDir - 탐색 시작 경로
	# 결과: 찾은 경우, 그 경로를 출력함 / 못 찾은 경우, 아무 것도 출력하지 않음
	# 방식: 재귀를 활용해 깊이 우선 탐색
	# 사용: before_Search(끝)
 */
void Dfs_for_Path(char* toFind, char* workDir) {

	//prepare
	DIR* dp = NULL;
	struct dirent *dir = NULL;
	struct stat statbuf;
	if ((dp = opendir(workDir))==NULL){
		perror("File Open Error₩n");
		exit(1);
	}


	//dfs
	while((dir=readdir(dp))!=NULL){ //dp가 끝에 도달할 때까지 읽음

		//check
		if (strcmp(dir->d_name, toFind) == 0){ //원하는 파일 또는 디렉터리를 찾았을
경우
			printf("PATH: %s/%s₩n", workDir, dir->d_name); //출력 후 return
			found++;
			closedir(dp);
			return; //재귀를 활용했기 때문에 break가 아닌 return
		}
```

```
                //pass
                if ((strncmp(dir->d_name,".",1)==0)){ //.과 ..그리고 .으로 시작하는
파일(숨김파일 등)은 모두 건너뜀
                        continue;
                }

                //recurse
                char tmp[MAX]; //현재 위치를 tmp에 경로로 나타냄
                strcpy(tmp,workDir);
                strcat(tmp,"/");
                strcat(tmp,dir->d_name);
                if (lstat(tmp, &statbuf)<0){ //tmp(현재 위치)를 statbuf에 넣음
                        perror("Stat Error₩n");
                        exit(1);
                }

                if (S_ISDIR(statbuf.st_mode)){ //dir이면 recurse(탐색)
                        Dfs_for_Path(toFind, tmp);
                }
        }

        closedir(dp);
}



/*
   extract_Filename
        # 인자: Path - argv[2]로 입력받은 문자열 (찾을 파일/디렉토리의 경로 또는 이름)
        # 목적: toFind에 argv[2]에서 추출한 파일/디렉토리의 이름을 넣음
        # 사용: main
 */
char* extract_Filename(char* Path){

        //No slashes
        if (strstr(Path,"/")==NULL) { // argv[2]에서 파일 또는 디렉터리의 이름을 입력받은 경우
                return Path; //그대로 리턴
        }

        //Slashes
        char* temp = (char*)malloc(sizeof(char)*MAX); //파일 또는 디렉터리의 경로에서
이름만을 추출하기 위해
        char* ptr = strtok(Path,"/"); // "/" 기준으로 token화
        while (ptr!=NULL) {
                strcpy(temp, ptr); //(결국엔) 마지막 ptr을 temp에 copy
                ptr = strtok(NULL,"/");
        }
        return temp;

}



/*
```



```
   set_Parentdir_Path
        # 목적: toFind가 ".."(부모 디렉터리)인 경우, 경로 설정을 위해
        # 사용: before_Search("..")
 */
char* set_Parentdir_Path(char* workDir){
        int i=0;
        char str[MAX], tmp[MAX][20];
        char* ptr = strtok(workDir, "/"); //getcwd로 얻은 경로를 "/"기준으로 token화
        for (i=0; ptr!=NULL; i++){
                strcpy(tmp[i], ptr);
                ptr = strtok(NULL,"/");
        }
        for (int k=0; k<i-1; k++){ //마지막 token을 제외해 경로를 재생성
                strcat(str,"/");
                strcat(str, tmp[k]);
        }
        workDir = str;
        return workDir;
}



/*
   before_Search
        # 인자: argv - 찾을 파일/디렉토리의 경로 또는 이름 / toFind - 찾을 파일/디렉토리의
이름 / BD - BFS/DFS
        # 사용: main
 */
void before_Search(char* argv, char* toFind, int BD){

        //Prepare
        char workDir[MAX];
        struct passwd *pwd;
        errno = 0;
        if((pwd = getpwuid(getuid())) == NULL) { //사용자 정보를 불러옴 - 사용자 계정명 등
(모든 컴퓨터에서 작동이 가능하도록)
                if(errno == 0 || errno == ENOENT || errno == ESRCH || errno == EBADF ||
errno == EPERM) {
                        fprintf(stderr,"미등록된 사용자입니다.₩n");
                }
                else {
                        fprintf(stderr,"error: %s₩n", strerror(errno));
                }
                exit(1);
        }

        //Branch
        if ((strcmp(".", argv) == 0)||(strcmp("./", argv) == 0)){ //탐색x (.)
                getcwd(workDir,MAX); //현위치 (작업 디렉토리)를 불러와 출력
                printf("Path: %s₩n", workDir);
                return;
        }
        else if (strcmp("..", argv) == 0){ //탐색x (..)
                getcwd(workDir,MAX); //현위치를 기준으로 부모 디렉토리의 경로를 설정해 출력
```

```
                printf("Path: %s₩n", set_Parentdir_Path(workDir));
                return;
        }
        else if (strncmp("./", argv, 2) == 0){ //탐색o (~)
                getcwd(workDir, MAX); //현재 디렉토리를 workDir에 넣음
        }
        else{ //탐색o (/)
                strcpy(workDir,pwd->pw_dir); //"/home/계정명"을 workDir에 넣음
        }

        //Search
        printf("탐색할 디렉터리 또는 파일 이름(경로) : %s₩n", argv);
        if (BD==0){
                Bfs_for_Path(toFind, workDir);
        }
        else{
                Dfs_for_Path(toFind,workDir);
        }

        //전역변수 cnt : 재귀/반복의 횟수
        if (!found){
                printf("해당 디렉터리 또는 파일이 없습니다.₩n");
        }

}



/* < 기능2: size > */
/*
        #함수 설명 : 경로를 받아서 절대경로로 바꿔준다.
        #변수 : char* path - 절대/상대 경로
        #리턴값 : path의 절대경로
   */
char* absolute(char* path){
        char *absPath = (char*)malloc(sizeof(char)*MAX);
        char strbuf[MAX]={};
        int intbuf;

        /*path 문자열 처리*/
        if(path[0]=='/')
//path==절대경로
                strcpy(absPath,path);
        else{
        //path==상대경로
                getcwd(absPath,MAX);
                if(strcmp(path,".")){
                        for(int i=0; path[i]!='₩0'; ){
                                if(!strncmp(path+i,"../",3)){
                                        /*가장 뒤에 있는 '/'찾아서'₩0'대입*/
                                        for(int j=0; absPath[j]!='₩0';j++)
                                                if(absPath[j]=='/')
                                                        intbuf = j;
                                        absPath[intbuf] = '₩0';
                                        i+=3;
                                }
                                else if(!strncmp(path+i,"./",2)){
                                        i+=2;
                                }
                                else{
                                        strncat(strbuf,path+i,1);
                                        i++;
                                }
                        }
                        if (strcmp(strbuf,"")){
                                strcat(absPath,"/");
                                strcat(absPath,strbuf);
                        }
                }
        }

        //printf("absPath: %s₩n", absPath);
        return absPath;
}


/*
        #함수 설명 : 처음 BFS/DFS 선택지에 따라 fileSize함수를 실행하고 결과를 출력한다.
        #변수 : char* absPath - 크기를 구할 디렉토리의 절대 경로
                        int BD - BFS/DFS 선택지 (BFS:0, DFS:1)
        #리턴값 : void
   */
void bfs_or_dfs (char* absPath,int BD){

        int totalSize = 0;

        if(BD==0)
                totalSize = Bfs_for_Size(absPath);
        else if(BD==1)
                totalSize = Dfs_for_Size(absPath);
        else
                perror("Error : unexpected value of valiable ₩"BD₩"!!₩n");

        printf("Total : %d₩n",totalSize);
}

/*
        #함수 설명 : 해당 디렉토리 내의 모든 파일크기를 DFS로 탐색하여 총합을 리턴
        #변수 : char* absPath - 크기를 구할 디렉토리의 절대 경로
        #리턴값 : 해당 디렉토리 내의 모든 파일크기 합
*/
int Dfs_for_Size(char* absPath){
        struct stat stbuf;
        int totalSize = 0;

        Stack st={NULL,0};
        struct dirent *dir;
        DIR* openable;

        /*stack에 시작 디렉토리 노드 추가*/
```


스파르탄SW교육원 | Soongsil University

```
	push(&st,initNODE(NULL,"",NULL));

	if((st.top->dp=opendir(absPath))==NULL){
		printf("Error : fail on open directory!₩n");
		exit(1);
	}
	strcpy(st.top->Nname,absPath);


	/*전체 탐색 알고리즘:DFS*/
	/*
		#구현 : top에 있는 디렉토리를 readdir로 하위 디렉토리(dir) 읽고 stat을 통해
디렉토리, 파일 판단
		#dp가 dir일 때 : 해당 dir을 stack의 top에 올려서 읽기 시작
		#dp가 file일 때 : totalsize에 해당 file사이즈 더하고 정보 출력, 다음 dir로 이동
		#dp가 NULL일 때 : 현재 top에 있는 디렉토리를 pop하고 이전 top에 있던
디렉토리를 이어서 읽기 시작
		(하위 디렉토리를 다 읽었을 때)
	 */
	while(st.size){
	//모든 디렉토리가 스택에서 pop되면 종료
		//printf("st.top->Nname: %s, st.size: %d₩n", st.top->Nname, st.size);
		if((dir=readdir(st.top->dp))!=NULL){
			if(strcmp(dir->d_name,".")!=0&&strcmp(dir->d_name,"..")!=0){

			/*탐색 디렉토리 문자열 처리*/
		strcpy(absPath, st.top->Nname);
			if(absPath[strlen(absPath)-1]!='/')
		    strcat(absPath,"/");
		strcat(absPath,dir->d_name);

		stat(absPath,&stbuf);//탐색 디렉토리 정보 불러오기

		if((stbuf.st_mode&0xF000)==0x8000){		//dir이 파일일 경우
//				printf("File₩n");
				totalSize += stbuf.st_size;
		 }
		else{
//dir이 디렉토리일 경우
//				printf("Dir₩n");
				if((openable = opendir(absPath))!=NULL)
			push(&st,initNODE(openable,absPath,NULL));
			}
			}

		}
		else
		//dir이 NULL일 경우
		pop(&st); /*top을 이전에 읽던 디렉토리로 변경*/
	}

	return totalSize;
}

/*
```



```
	#함수 설명 : 스택의 top에 노드를 push한다.
	#변수 : Stack *s - 노드를 추가할 스택의 포인터, NODE* n - 추가할 노드의 포인터
	#리턴값 : void
*/
void push(Stack *s, NODE* n){
	NODE* tmp;

	tmp = s->top;
	s->top = n;
    s->top->next = tmp;
	s->size++;
	//printf("Push! %s₩n", s->top->Nname);
}

/*
	#함수 설명 : 노드를 생성한다.
	#변수 : DIR* newDirp, char* newName, NODE* newNext -새로운 노드의 정보들
	#리턴값 : 해당 정보를 삽입한 새로운 노드의 포인터
*/
NODE* initNODE(DIR* newDp, char* newName, NODE* newNext){
	NODE* new = (NODE*)malloc(sizeof(NODE));

	new->dp = newDp;
	strcpy(new->Nname, newName);
	new->next = newNext;

	return new;
}

/*
	#함수 설명 : 스택의 top 노드를 pop한다.
	#변수 : Stack *s - top을 내보낼 스텍의 포인터
	#리턴값 : void
*/
void pop(Stack *s){
   NODE* tmp;

   tmp = s->top;
   s->top = s->top->next;
	//printf("Pop! %s₩n₩n", tmp->Nname);
   free(tmp);
   s->size--;
}

/*
	#함수 설명 : 해당 디렉토리 내의 모든 파일크기를 BFS로 탐색하여 총합을 리턴
	#변수 : char* absPath - 크기를 구할 디렉토리의 절대 경로
	#리턴값 : 해당 디렉토리 내의 모든 파일크기 합

*/
int Bfs_for_Size(char* absPath){
	struct stat stbuf;
	int totalSize = 0;
	DIR *dirp;
```

```
        Queue Q={(char(*)[STR_MAX])malloc(sizeof(char)*STR_MAX*MAX),0,-1,MAX};

        struct dirent *dir;

        /*queue에 시작 디렉토리 경로 추가*/
        enqueue_for_Size(&Q,absPath);

        if((dirp=opendir(absPath))==NULL){
                printf("Error : fail on open directory!₩n");
                exit(1);
        }


        /*전체 탐색 알고리즘:BFS*/
        /*
                #구현 : front에 있는 디렉토리를 readdir로 하위 디렉토리(dir) 읽고 stat을 통해
디렉토리, 파일 판단
                #dp가 dir일 때 : 해당 dir을 queue의 rear에 올려놓고 다음 dir로 이동
                #dp가 file일 때 : totalsize에 해당 file사이즈 더하고 정보 출력, 다음 dir로 이동
                #dp가 NULL일 때 : 현재 front에 있는 디렉토리를 dequeque하고 다음 front
디렉토리를 읽기 시작
                (하위 디렉토리를 다 읽었을 때)
         */
        while(!isEmpty(Q.rear-Q.front+1)){
                if(dirp==NULL){
                        dequeue_for_Size(&Q);
                        dirp=opendir(Q.pathptr[Q.front]);
                        continue;
                }
                else if((dir=readdir(dirp))!=NULL){                     //다음 dir로 이동
                        if(strcmp(dir->d_name,".")!=0&&strcmp(dir->d_name,"..")!=0){

                        /*absPath = dir의 절대경로*/
            strcpy(absPath, Q.pathptr[Q.front]);
                        if(absPath[strlen(absPath)-1]!='/')
                    strcat(absPath,"/");
            strcat(absPath,dir->d_name);
                        /*dir의 stat 불러오기*/
            stat(absPath,&stbuf);

            if((stbuf.st_mode&0xF000)==0x8000){         //dir이 파일일 경우
                                totalSize += stbuf.st_size;
                 }
                else                                                                    //dir이
디렉토리일 경우
                        enqueue_for_Size(&Q,absPath);
                        }

                }
        else{
//dp가 NULL일 경우
                        /*front갱신 후 디렉토리 변경*/
                        dequeue_for_Size(&Q);
                        dirp=opendir(Q.pathptr[Q.front]);
                }
```



```
        }

        return totalSize;
}

/*
        #함수 설명 : 큐의 rear에 노드를 enqueue_for_Size한다, 큐가 가득찼으면 용량을 키운다.
        #변수 : Queue *q - 경로를 집어넣을 큐,
                        char* newpath - 큐에 들어갈 문자열의 포인터
        #리턴값 : void
*/
void enqueue_for_Size(Queue *q, char* newpath){

        if(isFull(q))
            expand_Capacity(q);

        q->rear = (q->rear+1)%q->capacity;
        strcpy(q->pathptr[q->rear],newpath);

}

/*
        #함수 설명 : 큐의 front 노드를 dequeue_for_Size한다.
        #변수 : Queue *q - front를 내보낼 큐의 포인터
        #리턴값 : void
*/
void dequeue_for_Size(Queue *q){

        if(isEmpty(q->rear-q->front+1)){
                printf("queue is already empty!!₩n");
                return;
        }

        q->front = (q->front+1)%q->capacity;
}

/*
        #함수 설명 : 큐가 가득찼으면 1리턴 (rear 다음다음이 front일 떄 기준 Full)
        #변수 : Queue *q - 확인할 큐
        #리턴값 : void
*/
_Bool isFull(Queue * q){
        if((q->rear+2)%q->capacity==q->front)
                return 1;
        else
                return 0;
}

/*
        #함수 설명 : 큐의 용량을 증가시킨다. (default : 함수 실행 후 front=0)
        #변수 : Queue *q - 용량을 증가시킬 큐
        #리턴값 : void
*/
void expand_Capacity(Queue* q){
        char (*tmp)[STR_MAX] =
```

```
(char(*)[STR_MAX])malloc(sizeof(char)*STR_MAX*q->capacity*2);

        for(int i=0; i<q->capacity; i++){
                strcpy(tmp[i],q->pathptr[(q->front+i)%(q->capacity)]);
        }

        free(q);

        q->pathptr = tmp;
        q->rear = q->rear - q->front;
        q->front = 0;
        q->capacity *= 2;
}



/* < 기능3: Tree > */
/*
        #함수 설명 : 탐색할 디렉터리 이름(또는 경로)을 출력하기 위한 함수이다.
        #변수 : char *name - 찾을 디렉터리명(경로)
        #리턴값 : void
*/
void init(char *name){
        printf("탐색할 디렉터리 이름(경로) : %s₩n", name);
        return;
}

/*
        #함수 설명 : BFS/DFS 둘 중 하나를 선택하게 하는 기능을 위한 함수이다.
        #변수 : char *argv - 찾을 디렉터리명(경로)
        #리턴값 : void
*/
void selectmod(char *argv, int mod){

        if (mod == 0){
                if (strncmp("/", argv, 1) == 0) // 절대 경로로 입력했을 경우
                        Bfs_for_SearchTree(argv, "..");
                else if (strncmp("..", argv, 2) == 0) // ..로 시작할 경우
                        if ((strcmp("..", argv) == 0) || (strcmp("../", argv) == 0)) // 부모
디렉터리
                                Dfs_for_PrintTree(0, "..");
                        else if (strncmp("../.", argv, 4) == 0) // 상위 디렉터리
                                Dfs_for_PrintTree(0, argv);
                        else
                                Bfs_for_SearchTree(argv, ".."); // 프로그램 위치 디렉터리와
같은 깊이의 디렉터리
                else if (strncmp(".", argv, 1) == 0) // .로 시작할 경우
                        if ((strcmp(".", argv) == 0) || (strcmp("./", argv) == 0)) // 현재
디렉터리
                                Dfs_for_PrintTree(0, ".");
                        else
                                Bfs_for_SearchTree(argv, "."); // 하위 디렉터리
                else // dirname 또는 상대경로로 입력했을 경우
                        Bfs_for_SearchTree(argv, ".");
        }
```



```
        else if (mod == 1){
                if (strncmp("/", argv, 1) == 0) // 절대 경로로 입력했을 경우
                        Dfs_for_SearchTree(argv, "..");
                else if (strncmp("..", argv, 2) == 0) // ..로 시작할 경우
                        if ((strcmp("..", argv) == 0) || (strcmp("../", argv) == 0)) // 부모
디렉터리
                                Dfs_for_PrintTree(0, "..");
                        else if (strncmp("../.", argv, 4) == 0) // 상위 디렉터리
                                Dfs_for_PrintTree(0, argv);
                        else
                                Dfs_for_SearchTree(argv, ".."); // 프로그램 위치 디렉터리와
같은 깊이의 디렉터리
                else if (strncmp(".", argv, 1) == 0) // .로 시작할 경우
                        if ((strcmp(".", argv) == 0) || (strcmp("./", argv) == 0)) // 현재
디렉터리
                                Dfs_for_PrintTree(0, ".");
                        else
                                Dfs_for_SearchTree(argv, "."); // 하위 디렉터리
                else // dirname 또는 상대경로로 입력했을 경우
                        Dfs_for_SearchTree(argv, ".");
        }
}

/*
        #함수 설명 : BFS 알고리즘으로 찾고자 하는 디렉터리를 탐색하는 함수이다.
        #변수 : char *name, char *wd - 찾을 디렉터리명(경로), 탐색하기 시작할
디렉터리명(경로)
        #리턴값 : void
*/
void Bfs_for_SearchTree(char *name, char *wd){
        struct dirent *entry; struct stat buf; DIR *dp;
        NODE queue[MAX]; int front, rear; front = rear = -1; // 큐 생성

        if (chdir(wd) < 0){ // 디렉터리 이동, 실패 시 프로그램 종료
                printf("오류 발생! 프로그램 종료.₩n");
                exit(1);
        }

        if ((dp = opendir(".")) == NULL){ // 디렉터리 열기, 실패 시 프로그램 종료
                printf("오류 발생! 프로그램 종료.₩n");
                exit(1);
        }

        while (1){ // BFS 알고리즘

                while ((entry = readdir(dp)) != NULL){ // 현재 디렉터리 내용을 모두 읽었을 때
탈출

                        lstat(entry->d_name, &buf);

                        if (S_ISDIR(buf.st_mode)){ // 하위 디렉터리일 경우

                                char path[MAX];
```

```
                                if (strcmp(entry->d_name, ".") == 0 ||
strcmp(entry->d_name, "..") == 0) // ".", ".."은 고려하지 않음
                                        continue;

                                if (strncmp(strrev(name), strrev(entry->d_name),
strlen(entry->d_name)) == 0){ // <입력한 이름(경로) == 탐색한 디렉터리명>인 경우

                                        strrev(name); strrev(entry->d_name);
                                        printf("₩n%s----₩n", entry->d_name); //
입력한(경로의) 디렉터리명 출력
                                        Dfs_for_PrintTree(0, entry->d_name); //
Dfs_for_PrintTree함수 호출
                                        closedir(dp); return; // 반복문 탈출,
Bfs_for_SearchTree함수 종료
                                }

                                else{
                                strrev(name); strrev(entry->d_name);
                                }

                                getcwd(path, BUFSIZ);
                                strcat(path, "/");
                                strcat(path, entry->d_name);

                                if (havedir(path)){ // 탐색 중인 디렉터리가 하위 디렉터리를
보유한 경우

                                        rear++; // 저장할 공간 확보
                                        strcpy(queue[rear].Nname, path);// 대상 디렉터리
경로 저장
                                }

                                chdir("..");
                        }
                }
                // 같은 깊이의 모든 노드들의 탐색이 끝난 경우
                if (front == rear){
                        printf("Queue us Empty.(DIR not found) Program terminated!₩n");
                        return;
                }
                else
                        front++;
                closedir(dp); chdir(queue[front].Nname); // 선입선출, 비었으면 함수 종료

                if ((dp = opendir(queue[front].Nname)) == NULL){ // Dequeue한 디렉터리로
이동
                        printf("Dequeue 오류 발생. 프로그램 종료₩n");
                        exit(1);
                }
        }
        return;
}

/*
        #함수 설명 : 인자로 받은 경로, 즉 디렉터리가 하위 디렉터리를 가지는지 검사하는
함수이다.
```



```
        #변수 : char *path - 검사할 디렉터리 경로
        #리턴값 : 0 (하위 디렉터리가 없을 경우), 1 (하위 디렉터리가 있을 경우)
*/
int havedir(char *path){ // 하위 디렉터리 보유 여부 체크 함수
        struct dirent *entry; struct stat buf; DIR *dp;

        chdir(path); dp = opendir(path);

        while ((entry = readdir(dp)) != NULL){

                lstat(entry->d_name, &buf);

                if (S_ISREG(buf.st_mode)) continue;

                else if (S_ISDIR(buf.st_mode)){ // 하위 디렉터리가 있을 경우

                        if (strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name, "..")
== 0)
                                continue;
                        closedir(dp);
                        return 1; // 1 반환, if문 True

                }
        }

        closedir(dp);
        return 0; // 0 반환, if문 False
}

/*
        #함수 설명 : DFS 알고리즘으로 찾고자 하는 디렉터리를 탐색하는 함수이다.
        #변수 : char *name, char *wd - 찾을 디렉터리명(경로), 탐색하기 시작할
디렉터리명(경로)
        #리턴값 : void
*/
void Dfs_for_SearchTree(char *name, char *wd){ // 입력한 디렉터리를 찾는 함수, dfs 알고리즘
        struct dirent *entry; struct stat buf; DIR *dp;

        if (chdir(wd) < 0){ // 디렉터리 이동, 실패 시 프로그램 종료
                printf("오류 발생! 프로그램 종료.₩n");
                exit(1);
        }

        if ((dp = opendir(".")) == NULL){ // 디렉터리 열기, 실패 시 프로그램 종료
                printf("오류 발생! 프로그램 종료.₩n");
                exit(1);
        }

        while ((entry = readdir(dp)) != NULL){ // 현재 디렉터리 내용을 모두 읽었을 때 탈출

                lstat(entry->d_name, &buf);

                if (S_ISDIR(buf.st_mode)){ // 하위 디렉터리일 경우

                        if (strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name, "..")
```

```
== 0) // ".", ".."은 고려하지 않음
                                continue;

                        if (strncmp(strrev(name), strrev(entry->d_name),
strlen(entry->d_name)) == 0){ // <입력한 이름(경로) == 탐색한 디렉터리명>인 경우
                                strrev(name); strrev(entry->d_name); // 비교 위해 뒤집은
문자열 원상태로 복구
                                printf("%s----₩n", entry->d_name); // 입력한(경로의)
디렉터리명 출력
                                Dfs_for_PrintTree(0, entry->d_name); // Dfs_for_PrintTree함수
호출
                                chdir(".."); closedir(dp); return; // Dfs_for_SearchTree 함수
종료
                        }

                        else{
                                strrev(name); strrev(entry->d_name); // 비교 위해 뒤집은
문자열 원상태로 복구
                        }

                        Dfs_for_SearchTree(name, entry->d_name); // 없을 경우, 재귀 호출(더
하위 디렉터리로 이동)
                }
        }

        chdir(".."); closedir(dp); // 백트래킹(부모 디렉터리로 올라감)
}

/*
        #함수 설명 : DFS 알고리즘으로 찾은 디렉터리의 하위 파일과 디렉터리들을 트리 구조로
출력하는 함수이다. (BFS 방식은 트리 구조를 출력하는 것이 까다롭기 때문에 DFS 방식으로 만듬)
        #변수 : int tmp, char *wd - depth 구분하기 위한 정수, 트리 구조를 출력할 디렉터리
        #리턴값 : void
*/
void Dfs_for_PrintTree(int tmp, char *wd){ // 디렉터리를 트리구조로 출력하는 함수, dfs
알고리즘
        struct dirent *entry; struct stat buf; DIR *dp; int count = 0;

        count = tmp; // ₩t 횟수 구분(깊이 동일한 노드들 같은 열에 출력하기 위함)

        if (chdir(wd) < 0){ // dfs()와 동일
                printf("오류 발생! 프로그램 종료.₩n");
                exit(1);
        }

        if ((dp = opendir(".")) == NULL){
                printf("오류 발생! 프로그램 종료.₩n");
        }

        while ((entry = readdir(dp)) != NULL){ // dfs()와 동일

                lstat(entry->d_name, &buf);

                if (S_ISREG(buf.st_mode)){ // 탐색한 것이 파일일 경우
```



```
                        for (int i = 0 ; i <= count ; i++)
                                printf("₩t");
                        printf("-%s₩n", entry->d_name);

                }

                else if (S_ISDIR(buf.st_mode)){ // 탐색한 것이 디렉터리일 경우
                        if (strcmp(entry->d_name, ".") == 0 || strcmp(entry->d_name, "..")
== 0) // dfs()와 동일
                                continue;

                        for (int i = 0 ; i < 3 ; i++){ // 칸을 벌려서 겹치는 경우 예방하기 위함
                                for (int j = 0 ; j <= count ; j++)
                                        printf("₩t");
                                printf("|₩n");
                        }

                        for (int i = 0 ; i <= count ; i++) printf("₩t");
                        printf("-%s----₩n", entry->d_name); // (하위)디렉터리명 출력
                        tmp = count + 1;
                        Dfs_for_PrintTree(tmp, entry->d_name); // 재귀 호출 (더 하위
디렉터리로 이동)

                        for (int i = 0 ; i < 3 ; i++){ // 위와 동일
                                for (int j = 0 ; j <= count ; j++)
                                        printf("₩t");
                                printf("|₩n");
                        }

                }
        }

        chdir(".."); closedir(dp); // 백트래킹(부모 디렉터리로 올라감)
}

char *strrev(char *str){ // (문자열 뒤집는 함수(string.h_strrev는 리눅스 사용 불가))
        char *p1, *p2;
        if (!str || !*str)
                return str;
        for (p1 = str, p2 = str + strlen(str)-1; p2 > p1 ; ++p1, --p2){
                *p1 ^= *p2;
                *p2 ^= *p1;
                *p1 ^= *p2;
        }
        return str;
}

int main(int argc, char *argv[]){

        int BD; //BFS or DFS
        char* toFind; //File or Dir Name to Find
        char* absDir;

        if(argc != 3){ //Need more data
                fprintf(stderr, "Usage : Program [-Dir] [-Dir/-File]₩n");
```

```
                exit(1);
        }

        printf("Search Option(BFS:0, DFS:1): ");
        scanf("%d",&BD);

        if((BD==0)||(BD==1)){

                if (BD==0)
                        printf("BFS Search₩n");
                else
                        printf("DFS Search₩n");
                printf("₩n");

                //path
                char tmp_Path[MAX];
                strcpy(tmp_Path,argv[2]);
                char* toFind = extract_Filename(tmp_Path); //입력받은 argv[2]에서 파일 또는
디렉터리 이름만을 추출해 toFind에 넣음
                before_Search(argv[2], toFind, BD);
                printf("₩n");

                //size && tree
                init(argv[1]);
                absDir = absolute(argv[1]); //size
                bfs_or_dfs(absDir,BD);
                printf("₩n");

                printf("Tree:"); //tree
                selectmod(argv[1], BD);
        }
        else
                printf("Select Error(BFS:0, DFS:1)₩n");

        return 0;
}
```

<헤드 파일>

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <dirent.h>
#include <pwd.h>
#include <errno.h>



#define MAX 1000
#define STR_MAX 255
```



```


/*NODE && QUE*/
typedef struct NODE{
        DIR *dp;
        char Nname[MAX];
        struct NODE* next;
}NODE;

typedef struct{ //argv[1]
        NODE* top;
        int size;
}Stack;

typedef struct{ //argv[1]
        char (*pathptr)[STR_MAX];
        int front;
        int rear;
        int capacity;
}Queue;

typedef struct Queue{ //argv[2]
        struct NODE* front;
        struct NODE* rear;
        int qSize;
}QUE;



/*functions*/

//익준
//CO
void init(char *);
void selectmod(char *, int);
char *strrev(char *);
//BFS
void Bfs_for_SearchTree(char *, char *);
int havedir(char *);
//DFS
void Dfs_for_SearchTree(char *, char *);
void Dfs_for_PrintTree(int, char *);

//준호
//CO
char* absolute(char*);
void bfs_or_dfs(char*,int);
//BFS
int Bfs_for_Size(char*);
void enqueue_for_Size(Queue* , char*);
void dequeue_for_Size(Queue*);
_Bool isFull(Queue*);
void expand_Capacity(Queue*);
//DFS
int Dfs_for_Size(char*);
```

```
void push(Stack* , NODE*);
NODE* initNODE(DIR*, char*, NODE*);
void pop(Stack*);

//지우
//CO
char* extract_Filename(char*);
char* set_Parentdir_Path(char*);
void before_Search(char*, char*, int);
//BFS
_Bool isEmpty(int);
void enQue_for_Path(QUE*, struct NODE*);
void deQue_for_Path(QUE*);
struct NODE* create_NODE(DIR* dp, char* Nname);
void Bfs_for_Path(char* toFind, char* workDir);
//DFS
void Dfs_for_Path(char*, char*);
```

8. 최종 결과

```
smkim@smkim-virtual-machine:~/Mentoring$ ./test . 1.c
Search Option(BFS:0, DFS:1): 0
BFS Search

탐색할 디렉터리 이름(경로) : 1.c
PATH: /home/smkim/Mentoring/1.c
PATH: /home/smkim/Mentoring/dir2/1.c
PATH: /home/smkim/Mentoring/dir1/dir3/1.c

탐색할 디렉터리 이름(경로) : .
Total : 122974
Tree:   -test.c
        |
        |
        |
        -dir1----
                -1.txt
                -2.txt
                |
                |
                |
                -dir3----
                        -3.c
                        -1.c
                        -2.c
                |
                |
                |
        |
        |
        |
        |
        |
        |
        -dir2----
                -1.c
                -head.h
                -a.txt
                -dfs
                -test
                -b.txt
        |
        |
        |
        -final
        -1.c
        -head.h
        -test
        -final.c
```


스파르탄SW교육원 | Soongsil University

```
smkim@smkim-virtual-machine:~/Mentoring$ ./test . 1.c
Search Option(BFS:0, DFS:1): 1
DFS Search

탐색할 디렉터리 이름(경로) : 1.c
PATH: /home/smkim/Mentoring/dir1/dir3/1.c
PATH: /home/smkim/Mentoring/dir2/1.c
PATH: /home/smkim/Mentoring/1.c

탐색할 디렉터리 이름(경로) : .
Total : 122974
Tree:    -test.c
         |
         |
         |
         -dir1----
                 -1.txt
                 -2.txt
                 |
                 |
                 |
                 -dir3----
                          -3.c
                          -1.c
                          -2.c
                 |
                 |
                 |
         |
         |
         |
         |
         |
         |
         -dir2----
                 -1.c
                 -head.h
                 -a.txt
                 -dfs
                 -test
                 -b.txt
         |
         |
         |
         -final
         -1.c
         -head.h
         -test
         -final.c
```



```
smkim@smkim-virtual-machine:~/Mentoring$ ./test ./dir2 1.c
Search Option(BFS:0, DFS:1): 0
BFS Search

탐색할 디렉터리 이름(경로) : 1.c
PATH: /home/smkim/Mentoring/1.c
PATH: /home/smkim/Mentoring/dir2/1.c
PATH: /home/smkim/Mentoring/dir1/dir3/1.c

탐색할 디렉터리 이름(경로) : ./dir2
Total : 42539

Tree:
dir2----
        -1.c
        -head.h
        -a.txt
        -dfs
        -test
        -b.txt
```

```
smkim@smkim-virtual-machine:~/Mentoring$ ./test ./dir2 1.c
Search Option(BFS:0, DFS:1): 1
DFS Search

탐색할 디렉터리 이름(경로) : 1.c
PATH: /home/smkim/Mentoring/dir1/dir3/1.c
PATH: /home/smkim/Mentoring/dir2/1.c
PATH: /home/smkim/Mentoring/1.c

탐색할 디렉터리 이름(경로) : ./dir2
Total : 42539

Tree:dir2----
        -1.c
        -head.h
        -a.txt
        -dfs
        -test
        -b.txt
```

9. 기타

본 프로그램의 장단점 (멘토와 멘티가 별도 작성)	[멘토] 코로나 시국이니만큼 8할은 비대면으로 진행하였는데 목표한 프로젝트를 완성하는 것에는 큰 문제가 되진 않았지만 소통하는 것에 답답함은 있었고, 모든 멤버가 한자리에 모이는 것이 힘들어 선후배 간에 친목도모 차원에서는 다소 아쉬웠다. 정해진 커리큘럼 없이 스스로 멘토링을 계획해야 하는 부분 때문에 정해진 시간 내에 어떻게 지식을 전달할지 고민했고, 설명하는 과정에서 기존의 지식도 공고히 하고 말로써 내 지식을 표출할 수 있는 능력도 기르는 데에 도움이 되었고, 코로나 시대에 후배들과 함께 할 수 있는 좋은 기회가 된 것 같다. [멘티] 1. 프로젝트를 진행하며, 새로운 학우와 선배를 알 수 있고, 이들로부터 여러 도움을 받을 수 있다. 2. 잊고 있었던 프로그래밍 개념을 복습하거나 새로운 지식을 배워, 이를 프로젝트에 적용할 수 있다. 3. 직접 프로젝트를 설계 및 구현하는 방법을 익히고, git을 활용해 팀 프로젝트를 하는 방법을 배울 수 있다. 4. 다양한 환경에서 실행했을 때 생기는 에러를 경험하고 대처할 수 있는 능력을 기를 수 있다.
학습 사진 (2장 이상)	
추천합니다!!	▪ 구체적으로 어떤 친구들에게 추천하고 싶은지 적어주세요. 1. 지난 학기에서 배운 프로그래밍 학습 내용을 복습하고 싶은 학우 2. 다음 학기에서 배울 또는 그 후에 알아야 할 프로그래밍 개념과 지식에 관해 미리 학습하고 싶은 학우 3. 선배로부터 코딩 또는 학습 및 대학 생활에 관한 팁을 얻고 싶은 학우 ▪ 그 이유는 무엇인가요?

	1. 지난 학기에서 배웠으나 잊고 있었던, 또는 활용하지 못했던 프로그래밍 학습 내용을 다시 익히고, 이를 프로젝트 설계에 활용하여 확실히 이해할 수 있다. 2. 앞으로 알아야 하는 개념과 지식을 간단히 배우고, 이를 프로젝트 설계에 활용하여 이해할 수 있다. 3. 프로젝트를 약 2개월 진행하며 코딩뿐만 아니라 대학과 관련된 여러 팁들을 선배로부터 얻을 수 있다.
본 프로그램에서 보완할 점	비대면을 통한 진행을 할 때는 상호간의 의사소통이 원활하지 못했던 점이 아쉬웠다.