9조 보고서

9월 20일에 만나서 미팅

박서영 – 문제 출제, 해석

지명화 – 문제 출제, 해석

김선호 – 보고서, PPT 작성, 발표

총 3문제를 출제해 그 중에 2문제를 선별 했으며, 각각 달팽이 문제, 미로 찾기 문제라 서술했습니다.

달팽이 문제

달팽이 배열 변형문제.

입력 받은 숫자를 n이라고 할때 n*n 달팽이 배열을 num.out 출력

1-9까지 9가지 숫자만을 갖고 출력해야 하며 n의 범위는 1-50(그 외의 숫자는 예외처리)

입력은 어떤 방식으로 받아도 좋으나, fputc를 사용해서 풀기

```
#include <stdio.h>
#include <stdlib.h>
#define _CRT_SECURE_NO_WARNINGS

int main(void){
        char Snail_Array[50][50] = {0}, numdata[10], *input;
        // 숫자 문자열을 받을 배열
        int num, size, origin_num; //각각 숫자 갯수 저장할 변수
        FILE *wfp; //writefile
        int i = 0, j = 0, k = 1;
        int t = 0, s = 0, u = 0, w = 0;

        //숫자를 입력받는다.
        printf("1-50사이의 숫자 입력 : ");
        scanf("%d",&num);

        // 개수가 정확하지 않으면 종료
        if(num < 1 || num > 50){
        printf("숫자를 잘못 입력했습니다./n");
        return 0;
        }

        if((wfp = fopen("num.out", "w")) == NULL){
        perror("fopen : num.out ");
        exit(1);
        }//파일 오픈

        // 1입력시 처리
        if(num == 1) { fputc('1', wfp); return 0;}


        // 사이즈를 정리한다.
        size = num * num;
        origin_num = num;
        // 달팽이 배열에 윗부분 첫 윗 부분
        for(j=0; j<num; j++){
        if(k == 10) k = 1; //1부터 10까지만 하기위해 설정
        sprintf(numdata, "%d", k++);
        // int형 데이터를 chat형 으로 바꾸어 준다.
        Snail_Array[i][j] = numdata[0];
```

```
        size--;
        } j--;

        if(k != 1) k--; // k개수가 1이 아닐 때만 변경한다.

        //나머지 부분을 반복해서 적는다.
        while(1) { // 오른쪽
        for(i = t; i < num; i++) {
        if(k == 10) k = 1;
        sprintf(numdata, "%d", k++);
        // int형 데이터를 chat형 으로 바꾸어 준다
        Snail_Array[i][j] = numdata[0];
        size--;

        }
        i--;t++;size++;
        if(k != 1) k--;
        if(size == 0) break;

        // 아래
        for(j=(num-1); j>=s; j--) {
        if(k == 10) k = 1;
        sprintf(numdata, "%d", k++);
        Snail_Array[i][j] = numdata[0];
                size--;
        }
        j++;s++;size++;num--;
        if(k != 1) k--;
        if(size == 0) break;

        // 왼쪽
        for(i=num; i>w; i--) {
        if(k == 10) k = 1;
        sprintf(numdata, "%d", k++);
        Snail_Array[i][j] = numdata[0];
        size--;
        }
        i++;w++;size++;
        if(k != 1) k--;
        if(size == 0)break;

        // 윗부분
        for(j=u; j<num; j++) {
        if(k == 10) k = 1;
        sprintf(numdata, "%d", k++);
        Snail_Array[i][j] = numdata[0];
        size--;
        }
        j--;u++;size++;
        if(k != 1) k--;
        if(size == 0)break;
        }

        // 배열에 저장 된것을 파일에 저장 한다.
        for(i=0; i<origin_num; i++)
        {
        for(j=0; j<origin_num; j++)
        {
        fputc(Snail_Array[i][j], wfp);
        }
        fputc('₩n', wfp);
    }

        fclose(wfp);
        return 0;
}
```

미로찾기 문제

2차원 배열의 동적할당과 해제 및 파일 입출력을 이용해 자신만의 미로찾기 게임을 만들어 보자!(단, Stack을 이용한다 -> 최단 거리를 찾기 위해서 Stack을 사용해야되기 때문이다)

맵 정보: 0(벽), 1(길), 5(입구), 9(출구)

10 20
0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 0 1 0 1 0 1 1 1 1 1 0
1 1 0 0 1 0 0 0 1 0 1 1 1 0 0 0 0 0 1 0
5 0 1 0 1 1 1 1 1 0 1 0 1 0 1 1 1 1 1 0
0 0 1 0 1 0 0 1 0 0 1 0 1 0 1 0 0 1 0 0
0 0 1 1 1 0 0 1 1 1 1 0 1 1 1 0 0 1 1 9
0 0 0 1 0 0 1 0 0 0 1 1 0 1 0 0 1 0 0 0
0 0 0 1 0 0 1 0 1 1 0 1 0 1 0 0 1 0 0 0
0 0 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

푸는 단계

1. x좌표 및 y좌표를 가지는 구조체 Location2D를 만들자.

```
struct _Location2D {

    int row;
    int col;

};
```

2. Location2D 구조체의 데이터를 가지는 구조체 Node를 만들자.

```
struct _Node {

    Location2D data;
    Node* link;

};
```

3. 연결리스트를 노드를 변수로 사용하는 구조체 Stack을 만들자.

```
struct _Stack {

    Node* top;

};
```

4. 미로탐색을 위한 구조체 Maze를 만들자.

```
struct _Maze {

    // 멤버변수
    int width;
    int height;
    int** map;

};
```

5. 그 외 필요한 메소드

① map 2차원 배열을 동적으로 할당해주는 메소드

```
void initMaze(Maze* m, int w, int h)
```

② maze를 동적으로 해제해주는 메소드

```
void resetMaze(Maze* m)
```

③ 현재 maze를 화면에 출력해주는 메소드

```
void print(Maze* m)
```

④ 파일에서 미로 파일을 읽어오는 메소드

```
void load(Maze* m, char* fname)
```

⑤ 현재 위치가 유효한 위치인지 알려주는 메소드

```
int isValidLoc(Maze* m, int r, int c)
```

⑥ 비교할 스택과 기준 스택의 위치가 서로 이웃한지 알려주는 메소드

```
int isNeighborLoc(Location2D standardStack, Location2D compareStack)
```

⑦ 최단거리를 가기 위해 스택의 pop기능을 이용하는 메소드

```
void popRecursive()
```

⑧ 미로를 탐색하는 메스드

```
void searchExit(Maze* m)
```

6. 조건

① 미로를 단순히 찾는 것뿐만 아니라, 최단거리로 가는 방법도 구현해보자.
② main 함수는 아래와 같은 형식으로 깔끔하게 사용하자.

```
int main(void) {

    Maze m;
    load(&m, "maze.txt");
    print(&m);
    searchExit(&m);
    resetMaze(&m);

    return 0;
}
```

③ 결과는 아래와 같이 만들어보자.

해설

① Location2DLinkedStack.h

이 헤더파일은 미로 찾의 좌표를 나타내는 Location2D 구조체와 Location2D를 데이터로 가지는 연결리스트 Node 구조체, Node를 변수로 가지는 Stack 구조체를 선언한 헤더파일입니다. 우선, C언어 특성상 C++과 Java와는 다르게 클래스가 없기에 생성자를 따로 메소드 형식으로 지정해주어야 합니다. 구조체 Stack 역시나 따로 메소드 형식으로 지정해주어야 합니다.

```
#include <stdio.h>
#include <stdlib.h>

typedef struct _Location2D Location2D;
typedef struct _Node Node;
typedef struct _Stack Stack;

struct _Location2D {

        int row;
        int col;

};

struct _Node {

        Location2D data;
        Node* link;

};

struct _Stack {

        Node* top;
```

```
};

void initLocation2D(Location2D* loc, int r, int c) {

	loc->row = r;
	loc->col = c;

}

void initNode(Node* node) {

	node->data.row = 0;
	node->data.col = 0;

}

void initStack(Stack* stack) {

	stack->top = NULL;

}

int isEmpty(Stack* stack) {

	return stack->top == NULL;

}

void push(Stack* stack, Location2D data) {

	Node* now = (Node*)malloc(sizeof(Node));
	now->data = data;
	now->link = stack->top;
	stack->top = now;

}

Location2D pop(Stack* stack) {

	Node* now;
	Location2D zero;
	Location2D result;

	initLocation2D(&zero, 0, 0);

	if (isEmpty(stack)) return zero;
```

```
        now = stack->top;
        result = now->data;

        stack->top = now->link;
        free(now);

        return result;

}

Location2D peek(Stack* stack) {

        return stack->top->data;

}
```

② Maze.h

이 헤더파일은 앞에서 구현한 Location2DLinkedStack.h를 활용해서 미로의 뼈대와 틀을 만들고 탐색을 할 수 있도록 한다. Maze 역시 구조체이므로 클래스가 아니기 때문에 따로 생성자를 메소드 형식으로 구현한다.

1. initMaze() 메소드는 이차원 배열을 동적할당을 하여 뼈대와 틀을 만들어 준다. (malloc을 사용)

2. resetMaze() 메소드는 이차원 배열 동적할당을 해제해준다. (free를 사용)

3. print() 메소드는 현재 구축되어 있는 미로를 보여준다. 이때, 파일입출력을 통해 얻은 미로 파일의 숫자에 따라 다양한 모양이 나타나도록 한다.

4. load() 메소드는 틀이 갖춰진 미로의 속을 채워주는 역할을 한다. 위의 문제에서 제시된 것과 같이, 0은 벽, 1은 길, 5는 입구, 9는 출구이다. 파일입출력을 여기서 사용하는데 이때, 유용하게 사용할 수 있는 함수가 ungetc() 메소드이다. 이 메소드는 스트림에서 읽은 하나의 문자를 다시 되돌려준다. 즉, 버퍼에 쌓이는 문자들 가운데 가장 마지막에 들어온 문자가 읽혀진다. 약간 stack과 비슷한 개념이라고 생각하면 된다. 읽어들이는 문자를 fscanf() 메소드를 사용해서 정수로 변환해주어 Maze의 맵을 구성해준다. 또 하나 중요한 것은 나중에 사용될 입구의 좌표를 기억하기 위해서 읽어들인 문자의 정수 값이 5라면 전역변수로 선언해준 entryCoordinateXpos와 entryCoordinateYpos의 값을 그 위치의 w, h로 저장해준다.

5. 마지막으로 중요한 것은 popRecursive() 메소드이다. 이 메소드는 미로 탐색의 최단경로를 구하기 위해 반드시 있어야 한다. 맨 처음 선언한 standardStack과 compareStack이 여기서 사용된다. 처음 미로 탐색을 할 때 스택을 사용하면 깊이 우선 탐색을 하면서 스택에 값이 들어가고 나가고를 반복한다. 이때, compareStack이라는 변수를 하나 더 생성하여 기준 스택과 똑같이 값을 넣어주고 빼주고를 반복한다. 하지만 다른 점이 하나가 있다. 바로, compareStack의 경우, 들어오는 값이 스택 안에 가장 위에 있는 top변수와 이웃하지 않을 경우 값을 넣지 않고 이웃할 경우에만 값을 넣어준다. 이렇게 되면 결국 compareStack 안에는 서로 이웃한 값들끼리만 쌓여있으므로 나중에 출력해보면 최단 거리가 나오게 된다.

```
#include "Location2DLinkedStack.h"

typedef struct _Maze Maze;
Stack standardStack;
```

```
Stack compareStack;
int entryCoordinateXpos;
int entryCoordinateYpos;

struct _Maze {

	// 멤버변수
	int width;
	int height;
	int** map;

};

void initMaze(Maze* m, int w, int h) {

	int i;
	m->map = (int**)malloc(sizeof(int*) * m->width); // 이차원 동적 할당

	for (i = 0; i < m->width; i++)
		m->map[i] = (int*)malloc(sizeof(int) * m->height);

}

void resetMaze(Maze* m) {

	int i;
	for (i = 0; i < m->width; i++)
		free(m->map[i]);
	free(m->map);

}

void print(Maze* m) {

	int w, h;

	printf("Maze Size = %d x %d\n", m->width, m->height);
	for (w = 0; w < m->width; w++) {

		for (h = 0; h < m->height; h++) {
```

```
                        if (m->map[w][h] == 0) printf("□");
                        else if (m->map[w][h] == 1) printf("  ");
                        else if (m->map[w][h] == 5) printf("○");
                        else if (m->map[w][h] == 9) printf("◎");
                        else if (m->map[w][h] == 777) printf("☆");
                        else printf("■");

                } printf("\n");
        }

}

void load(Maze* m, char* fname) {

        char c;
        int val;
        int Line = 0;
        int w = -1;  // 가로
        int h = 0; // 세로

        FILE* fp = fopen(fname, "r");
        if (fp == NULL) {
                printf("File Open Fail\n");
                return;
        }

        while ((c = getc(fp)) != EOF) {
                // 첫 번째 라인에서 맵의 크기를 받아온다
                if (Line == 0) {

                        ungetc(c, fp);
                        fscanf(fp, "%d %d", &m->width, &m->height);
                        // 받아온 가로와 세로를 이용해 초기화
                        initMaze(m, m->width, m->height);
                        // 이 조건문을 빠져나가면 쓸모가 없어진다
                        Line++;

                } else {
                        // '\n'를 만나게 되기 전까지
                        if(c != '\n') {
```

```
                                // 열에서 한 번씩 이전 버퍼에 문자를 담아준다
                                ungetc(c,fp);
                                fscanf(fp, "%d", &val);

                                if (val == 5) {

                                        entryCoordinateXpos = w;
                                        entryCoordinateYpos = h;

                                }

                                m->map[w][h] = val;
                                // 원형 큐에서와 같은 원리로 h가 증가할 때 값이 범위를
                                   벗어나지 않도록 해준다.
                                h = (h + 1) % m->height;

                        } else {

                                w = (w + 1) % m->width;

                        }
                }
        }

        fclose(fp);

}

int isValidLoc(Maze* m, int r, int c) { // 좌표 (r, c)가 맵 안에 위치해 있는지

        if (r < 0 || c < 0 || r >= m->width || c >= m->height) return 0;
        else return m->map[r][c] == 1 || m->map[r][c] == 9;

}
// 기존 스택과 비교할 스택의 좌표가 이웃해 있는지
int isNeighborLoc(Location2D standardStack, Location2D compareStack) {

        int r1 = standardStack.row;
        int c1 = standardStack.col;
        int r2 = compareStack.row;
        int c2 = compareStack.col;
```

```
        if ((r1 == r2 && ((c1 == c2 + 1) || (c1 == c2 - 1))) || (c1 == c2 && ((r1 == r2 + 1) || (r1 == r2 - 1)))) return 1;
        else if (r1 == r2 && c1 == c2) return 1;
        else return 0;

}
// 길만 따라서 갈 수 있도록 pop을 반복해서 사용
void popRecursive() {

        Location2D loc1;
        Location2D loc2;

        if (!isEmpty(&standardStack)) {

                initLocation2D(&loc1, peek(&standardStack).row, peek(&standardStack).col);
                initLocation2D(&loc2, peek(&compareStack).row, peek(&compareStack).col);

                while (!isNeighborLoc(loc1, loc2) && !isEmpty(&compareStack)) {

                        pop(&compareStack);
                        initLocation2D(&loc2, peek(&compareStack).row, peek(&compareStack).col);

                }
        }

}

void searchExit(Maze* m) { // 미로 탐색 메소드

        Location2D entry;
        Location2D sample;

        Location2D checkLoc;
        Location2D checkStack;
        Location2D ele;

        Location2D directionUp;
```

```
Location2D directionDown;
Location2D directionLeft;
Location2D directionRight;

int r;
int c;

int ele_Row;
int ele_Col;

printf("\n=============미로 탐색 시작=============\n");

initLocation2D(&entry, entryCoordinateXpos, entryCoordinateYpos);
initLocation2D(&sample, entryCoordinateXpos, entryCoordinateYpos);

push(&standardStack, entry);
push(&compareStack, sample);

while (!isEmpty(&standardStack)) {

        initLocation2D(&checkLoc, peek(&standardStack).row, peek(&standardStack).col);
        initLocation2D(&checkStack, peek(&compareStack).row, peek(&compareStack).col);

        if (isNeighborLoc(checkLoc, checkStack)) push(&compareStack, checkLoc);
        else {

                popRecursive();
                push(&compareStack, checkLoc);

        }

        pop(&standardStack);

        r = checkLoc.row;
        c = checkLoc.col;

        if (m->map[r][c] == 9) {
                print(m);

                printf("\n=============최단 경로 검색=============\n");
                while(!isEmpty(&compareStack)) {
```

```
                                initLocation2D(&ele, peek(&compareStack).row,
peek(&compareStack).col);
                                pop(&compareStack);
                                ele_Row = ele.row;
                                ele_Col = ele.col;
                                m->map[ele_Row][ele_Col] = 777;

                        }
                        print(m);

                        return;

                } else {

                        m->map[r][c] = '.';

                        initLocation2D(&directionUp, r - 1, c);
                        initLocation2D(&directionDown, r + 1, c);
                        initLocation2D(&directionLeft, r, c - 1);
                        initLocation2D(&directionRight, r, c + 1);

                        if (isValidLoc(m, r, c + 1)) push(&standardStack,
directionRight); // 우
                        if (isValidLoc(m, r, c - 1)) push(&standardStack,
directionLeft); // 좌
                    if (isValidLoc(m, r - 1, c)) push(&standardStack, directionUp);
// 상
                   if (isValidLoc(m, r + 1, c)) push(&standardStack, directionDown);
// 하

                }
        }
        printf("미로 탐색 실패\n");

}
```

Lab. 3

Buffer IO

9조

박서영, 지명화, 김선호

Lab 3-3. Integer Values in 3 Bytes

- **Binary 형태의 파일을 읽어, Ascii 형태로 출력**
- Input file : 3 byte 정수 숫자 들 (Big-endian)
  - System call을 통해 읽을 것
- Output file : 공백으로 구분된 정수 들
  - Standard IO를 통해 출력 할 것

Lab 3-3. Integer Values in 3 Bytes

- 실행의 예

```
laply@DESKTOP-T9L7HIK:~$ cat output
9127378 8070136 3971277 3360985 11590853 4807960 15242356 5593422 9314124 15236371 1792247 8312827
499 921482 5383565 10287422 15561328 14467762 15718330 10994954 6817797 7011252 14966103 10178654
660169 15580595 8621355 5493008 254161 10122931 14982920 15934661 11044413 3589142 9444740 9828526
7820 295542 4611960 2410229 4729587 6535694 8932154 15389628 5339075 776167 4105293 5593236 108990
42 16775476 4868226 9730389 10140599 laply@DESKTOP-T9L7HIK:~$ cat output
```

```
8312827 11616800 12001084 14854748 4579771 6226125 13656861 2099302 11610181 1450781 1501703 9489912 15680
178654 1824762 2996848 8643796 7418056 12310972 7102824 9210303 3846456 1942409 4434044 1923862 6522053 10
9828526 1279561 8385855 4046265 8097359 15397107 2235026 1498670 444526 5231874 10142338 7862582 765504 46
10899098 2310998 4750554 5166168 5900141 14195294 14994694 7179702 5803935 2263616 15276933 4423699 44986
```

Lab 3-3. Integer Values in 3 Bytes

```
int read_file(char* argv[]){
	int rfd = open(argv[1], O_RDONLY); // 파일 input은 system call을 이용
	if (rfd == -1) { perror("Open input"); exit(1); }
	return rfd;
}

FILE* write_file(char* argv[], FILE* wfp){
	wfp = fopen(argv[2], "wb"); // 파일 output은 stand library를 이용
	if (wfp == NULL) { perror("Open output"); exit(1); }
	return wfp;
}
```

Lab 3-3. Integer Values in 3 Bytes

```
i int main(int argc, char* argv[]) {
                int rfd, n, i = 0, sum = 0, num;
                FILE* wfp;
                unsigned char buf;
                rfd = read_file(argv); // read파일 open
                wfp = write_file(argv, wfp); //write 파일 open
                while ((n = read(rfd, &buf, 1)) > 0) { // 한 바이트씩 읽어온다.총 3개의 바이트가 필요하다.
                 // 총 3가지 조건이 있는데 빅 엔디언 이므로 처음 read한 값에 비트연산자를 이용해서 - 16자리
이용
                 // 두번째 read값 에 비트연산자를 이용해서 - 8자리 이동
                 // 3번째 read 값은 그대로 출력한다.
                if (i == 0) num = (buf << 16);
                else if (i == 1) num = (buf << 8);
                else num = buf;
                i++; sum += num; // 수행 1번 했으므로 증가 처리한 값으로 다 더한다.
                 // 3번read작업을 다했다면 저장한다.
                if (i == 3) {
                fprintf(wfp, "%d ", sum);
                sum = 0; i = 0;
                }
                }
                close(rfd);
                fclose(wfp);
                return 0;
}
```

해설에 있는 코드 사용시

- **해설의 코드를 사용했음을 명시하고,**
- **3회 타이핑 하는 동영상 첨부**
  - Youtube 업로드 후, 링크 제출