

20121277 김주호 HW4 보고서

이번 과제에서 구현한 ADT는 크게 3가지이다. Singly Linked List, Double Linked List, Circular Linked List 가 그것인데, 이에 대한 ADT를 먼저 설명하도록 한다.

- Singly Linked List

Node
<pre>typedef int element; typedef struct ListNode { element data; struct ListNode * link; }ListNode;</pre>

- ① `ListNode* insert_first(ListNode* head, element val)`
head쪽에 노드를 삽입한다.
- ② `ListNode* insert_after_pre(ListNode* head, ListNode* pre, element val)`
인자로 들어온 pre노드 뒤에 새로운 노드를 삽입한다.
- ③ `ListNode* delete_first(ListNode* head)`
head쪽에 있는 노드를 삭제한다.
- ④ `ListNode* delete_after_pre(ListNode* head, ListNode* pre)`
pre의 link가 가리키는 노드를 삭제한다.
- ⑤ `ListNode* insert_last(ListNode* head, element val)`
tail 쪽에 노드를 삽입한다.
- ⑥ `ListNode* delete_last(ListNode* head)`
tail 쪽 노드를 삭제한다.

- Circular Linked List

Node
<pre>typedef char element; typedef struct ListNode { element data; struct ListNode * link; }ListNode;</pre>

- ① `void print_list(ListNode* tail)`
연결리스트 내의 항목을 출력한다.
- ② `void print_first_node(ListNode* tail)`
첫 번째 노드의 data를 출력한다.
- ③ `ListNode* insert_first(ListNode* tail, element data)`

head 쪽에 노드를 삽입한다.

④ ListNode* insert_last(ListNode* tail, element data)

tail 쪽에 노드를 삽입한다.

⑤ ListNode* delete_first(ListNode* tail)

head 쪽에 노드를 삭제한다.

⑥ ListNode* delete_after_pre(ListNode* tail, ListNode* pre)

인자로 들어온 pre 노드의 link가 가리키는 노드를 삭제한다.

⑦ ListNode* delete_last(ListNode* tail)

tail쪽의 원소를 삭제한다.

- Double Linked List

```
Node
typedef struct {
    short int row;
    short int col;
    short int dir;
}element;

typedef struct DListNode {
    element data;
    struct DListNode * llink;
    struct DListNode * rlink;
}DListNode;
```

① void init(DListNode* phead)

이중 연결리스트를 초기화하는 함수이다. llink와 rlink가 모두 head 노드를 가리키게 한다.

② is_empty(DListNode* phead)

이중 연결리스트의 공백 여부를 알려주는 함수이다. phead->llink == phead && phead->rlink이면 해당 연결리스트는 공백이 된다.

③ void print_dlist_reverse(DListNode* phead)

이중 연결 리스트의 노드를 tail에서 시작하여 head를 끝으로 역순으로 출력해주는 함수이다.

④ void d_insert(DListNode* pre, element data)

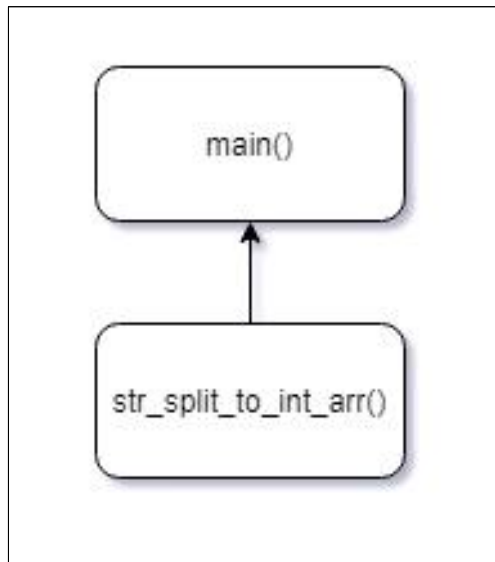
인자로 들어온 pre노드의 rlink가 가리키는 곳에 새로운 데이터를 가지는 노드를 삽입하는 함수이다.

⑤ void d_delete(DListNode* head, DListNode* removed)

인자로 들어온 removed 노드를 삭제하는 함수이다.

1번.

- 함수 구성 다이어그램



- 함수 설명

① int str_split_to_int_arr()

입력이 “1 -2 3 4 51”처럼 띄어쓰기하여 들어오기 때문에, 각 정수에 접근하기 위해서는 입력을 공백(띄어쓰기) 기준으로 parsing하는 작업이 필요하다. 함수의 인자로 해당 string이 들어오게 되면, 이를 잘 split하여 정수 각각을 arr의 각 원소로 저장시키는 함수이다. arr에 저장된 원소의 개수를 return값으로 주도록 하여, 이를 main 함수에서 활용할 수 있도록 하였다.

input : int arr[], int max_arr_size, char* str

return : arr의 사이즈

② ListNode* rotate()

단계 1. head부터 link를 타고, 끝 2개의 노드에 접근한다.

뒤에서 두 번째 노드는 포인터 pre가 가리키도록 하고, 제일 마지막 node는 포인터 ptr이 가리키도록 한다.

단계2. pre의 link를 NULL로, ptr의 link를 head로 설정하고 head를 ptr로 업데이트 해주며 ‘회전’을 구현한다.

input : ListNode* head

return : 1회 회전을 적용시킨 연결리스트

- 코드

```
#define MAX_SIZE 300
//////////Singly Linked List ADT//////////
#include <stdio.h>
#include <stdlib.h>
typedef int element;
typedef struct ListNode {
    element data;
    struct ListNode * link;
}ListNode;
// head에 삽입
ListNode* insert_first(ListNode * head, element val) {
    ListNode* p = (ListNode *)malloc(sizeof(ListNode));

    if (p ==NULL) {
        printf("동적할당 err!!");
        exit(1);
    }
    p->data = val;
    p->link = head;
    head = p;
    return head;
}
// 노드 pre뒤에 새로운 노드 삽입
ListNode* insert_after_pre(ListNode * head, ListNode * pre, element val) {
    ListNode* p = (ListNode *)malloc(sizeof(ListNode));
    if (p ==NULL) {
        printf("동적할당 err!!");
        exit(1);
    }
    p->data = val;
    p->link = pre ->link;
    pre->link = p;
    return head;
}
// head에 있는 노드 삭제
ListNode* delete_first(ListNode * head) {
    ListNode* removed;
    if (head ==NULL) return NULL;
    removed = head;
    head = removed ->link;
    free(removed);
    return head;
}
// pre의 link가 가리키는 노드를 삭제
ListNode* delete_after_pre(ListNode * head, ListNode * pre) {
    ListNode* removed;
    removed = pre ->link;
    pre->link = removed ->link;
    free(removed);
    return head;
}
```

// 마지막 끝에 노드 삽입

```
ListNode* insert_last(ListNode * head, element val) {
    ListNode* ptr = head;
    ListNode* new_node = (ListNode *)malloc(sizeof(ListNode));
    if (new_node ==NULL) {
        return NULL;
    }
    new_node->data = val;
    new_node->link =NULL;
    if (head ==NULL) { // linked list가 비어있으면,
        head = new_node;
        return head;
    }
    // 그렇지 않으면, link 타고 맨 끝으로 간다.
    while (1) {
        if (ptr ->link ==NULL) {
            ptr->link = new_node;
            break;
        }
        ptr = ptr ->link;
    }
    return head;
}
```

// 마지막 노드 삭제

```
ListNode* delete_last(ListNode * head) {
    ListNode* removed = head;
    ListNode* ptr = head;
    if (head ==NULL) {
        return NULL;
    }
    while (1) {
        if (ptr ->link ==NULL) {
            free(ptr);
            break;
        }
        ptr = ptr ->link;
    }
    return head;
}
```

// 연결리스트 인쇄

```
void print_list(ListNode * head) {
    for (ListNode * p = head; p !=NULL; p = p ->link) {
        printf("%d ", p ->data);
    }
    printf("\n");
}
```

```
ListNode* rotate(ListNode * head) {
    ListNode* ptr = head;
    ListNode* pre = head;
    if (ptr ==NULL || pre ==NULL) {
        printf("리스트가 비었습니다!\n");
        exit(1);
    }
    for (ptr = head; ptr ->link !=NULL; ptr = ptr ->link) {
```

```

        pre = ptr;
    }
    pre->link = NULL;
    ptr->link = head;
    head = ptr;
    return head;
}
////////////////////////////////////
int str_split_to_int_arr(int arr[], int max_arr_size, char * str) {
    /*공백을 포함한 str을 입력받아 공백 기준 split하여 arr에 저장하는 함수이다.*/
    int * start = arr;
    int num = 0;
    // 숫자 분리 상태를 기억 (0이면 진행 안됨, -1이면 음수 진행, 1이면 양수 진행)
    int state = 0;
    while (*str == ' ') str++; // 앞쪽 공백 제거
    for (; *str; str++) {
        if (*str != ' ') {
            if (state == 0) { // 숫자 분리가 진행되지 않았다면,
                if (*str == '-') {
                    state = -1; // 음수라는 표시,
                    str++;
                }
                else state = 1; // 양수라는 표시
            }
            // 문자를 숫자로 변경
            num = num * 10 + *str - '0';
        }
        else {
            if ((arr - start) < max_arr_size) {
                *arr += num * state;
                num = 0;
                state = 0;
                // 공백 문자 제거
                while (*(str + 1) == ' ') str++;
            }
            else return arr - start;
        }
    }
    // 숫자를 분리중에 반복문이 중단되었으면 해당 숫자를 arr에 추가한다.
    if (state) *arr += num * state;
    return arr - start;
}

int main() {
    ListNode* head = NULL;
    int size, k;
    int arr[MAX_SIZE];
    char tmp;
    char str[MAX_SIZE];
    printf("singly linked list에 저장될 정수의 크기 : ");
    if (scanf("%d", &size) == -1) {
        return 0;
    }
}

```

```

    tmp = getchar();
    printf("singly linked list에 저장될 정수들 : ");
    if (scanf("%[^Wn]s", str) == -1) {
        return 0;
    }
    printf("회전시킬 수 : ");
    if (scanf("%d", &k) == -1) {
        return 0;
    }
    str_split_to_int_arr(arr, size, str);

    for (int i = 0; i < size; i++) {
        head = insert_last(head, arr[i]);
    }
    //print_list(head);

    for (int i = 0; i < k % size; i++) {
        head = rotate(head);
    }
    print_list(head);
    return 0;
}

```

- 실행 결과

```
Microsoft Visual Studio 디버그 콘솔
singly linked list에 저장될 정수의 크기 : 5
singly linked list에 저장될 정수들 : 1 2 3 4 5
회전시킬 수 : 2
4 5 1 2 3

C:\Users\juho3\Desktop\4\LinkedList\Debug\LinkedList.exe(프로세스 29084개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...

Microsoft Visual Studio 디버그 콘솔
singly linked list에 저장될 정수의 크기 : 3
singly linked list에 저장될 정수들 : 0 1 2
회전시킬 수 : 4
2 0 1

C:\Users\juho3\Desktop\4\LinkedList\Debug\LinkedList.exe(프로세스 14972개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...

Microsoft Visual Studio 디버그 콘솔
singly linked list에 저장될 정수의 크기 : 6
singly linked list에 저장될 정수들 : 5 6 7 1 2 3
회전시킬 수 : 3
1 2 3 5 6 7

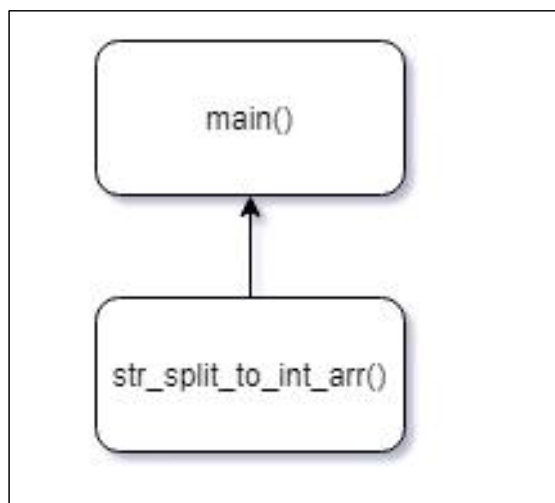
C:\Users\juho3\Desktop\4\LinkedList\Debug\LinkedList.exe(프로세스 5796개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...

Microsoft Visual Studio 디버그 콘솔
singly linked list에 저장될 정수의 크기 : 3
singly linked list에 저장될 정수들 : -1 -5 -86
회전시킬 수 : 2
-5 -86 -1

C:\Users\juho3\Desktop\4\LinkedList\Debug\LinkedList.exe(프로세스 9480개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...
```

2번.

- 함수 구성 다이어그램



- 함수 설명

① int main()

단계 1. 사용자로부터 띄어쓰기로 구분된 N과 K를 입력받고 이를 parsing하여 배열의 원소에 담는다.

단계 2. Circular Linked List의 ADT인 insert_last() 함수를 활용하여, N만큼 알파벳을 순서대로 원형 연결리스트에 저장한다.

단계 3. tail의 주솟값에 해당하는 노드의 데이터를 인쇄하고, K번 만큼 링크를 타고 건너간다.

단계 4. 단계 3을 N회 반복한다.

input : 없음

return : 0

- 코드

```
#define MAX_SIZE 10
//Circular Linked List ADT////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
typedef char element;
typedef struct ListNode {
    element data;
    struct ListNode * link;
}ListNode;
// 리스트의 항목 출력
void print_list(ListNode * tail) {
    ListNode* p;
    if (tail ==NULL) return;
    p = tail ->link;
    do {
        printf("%c ", p ->data);
        p = p ->link;
    } while (p != tail);
    printf("%c\n", p ->data); // 마지막 노드 출력
}
void print_first_node(ListNode * tail) {
    ListNode* p;
    if (tail ==NULL) return;
    p = tail ->link;
    printf("%c ", p ->data);
}
ListNode* insert_first(ListNode * tail, element data) {
    ListNode* node = (ListNode *)malloc(sizeof(ListNode));
    if (node ==NULL) {
        printf("동적할당 err!");
        exit(1);
    }
    node->data = data;
```

```

        if (tail == NULL) {
            tail = node;
            node->link = tail;
        }
        else {
            node->link = tail ->link;
            tail->link = node;
        }
        return tail;
    }

ListNode* insert_last(ListNode * tail, element data) {
    ListNode* node = (ListNode *)malloc(sizeof(ListNode));
    if (node == NULL) {
        printf("동적할당 err!");
        exit(1);
    }
    node->data = data;
    if (tail == NULL) {
        tail = node;
        node->link = tail;
    }
    else {
        node->link = tail ->link;
        tail->link = node;
        tail = node;
    }
    return tail;
}

// head에 있는 노드 삭제
ListNode* delete_first(ListNode * tail) {
    ListNode* removed;
    if (tail == NULL) return NULL;
    removed = tail ->link;
    tail->link = removed ->link;
    free(removed);
    return tail;
}

// pre가 가리키는 노드의 다음 노드 삭제
ListNode* delete_after_pre(ListNode * tail, ListNode * pre) {
    ListNode* removed;
    removed = pre ->link;
    if (removed == tail) {
        pre->link = tail ->link;
        tail = pre;
        free(removed);
        return tail;
    }
    pre->link = removed ->link;
    free(removed);
    return tail;
}

// 마지막 노드 삭제
ListNode* delete_last(ListNode * tail) {
    ListNode* removed = tail;

```

```

        ListNode* pre = tail;
        for (pre = tail; pre ->link != tail; pre = pre ->link);
        tail = delete_after_pre(tail, pre);
        return tail;
    }
    //////////////////////////////////////
    int str_split_to_int_arr(int arr[], int max_arr_size, char * str) {
        /*공백을 포함한 str을 입력받아 공백 기준 split하여 arr에 저장하는 함수이
        다.*/
        int * start = arr;
        int num =0;
        // 숫자 분리 상태를 기억 (0이면 진행 안됨, -1이면 음수 진행, 1이면 양수 진
        행)
        int state =0;
        while (*str == ' ') str ++;; // 앞쪽 공백 제거
        for (; *str; str ++) {
            if (*str != ' ') {
                if (state ==0) { // 숫자 분리가 진행되지 않았다면,
                    if (*str == '-') {
                        state =-1; // 음수라는 표시,
                        str++;
                    }
                    else state =1; // 양수라는 표시
                }
                // 문자를 숫자로 변경
                num = num *10 +*str -'0';
            }
            else {
                if ((arr - start) < max_arr_size) {
                    *arr += num * state;
                    num =0;
                    state =0;
                    // 공백 문자 제거
                    while (*(str +1) == ' ') str ++;
                }
                else return arr - start;
            }
        }
        // 숫자를 분리중에 반복문이 중단되었으면 해당 숫자를 arr에 추가한다.
        if (state)*arr += num * state;;
        return arr - start;
    }
}

int main() {
    ListNode* tail =NULL;
    ListNode* pre;
    char str[MAX_SIZE];
    int arr[2];
    int N, K;
    printf("N과 K를 띄어쓰기로 구분하여 입력하세요. : ");
    if (scanf("%[^Wn]s", str) ==-1) {
        return 0;
    }
    str_split_to_int_arr(arr, 2, str);
    N = arr[0];

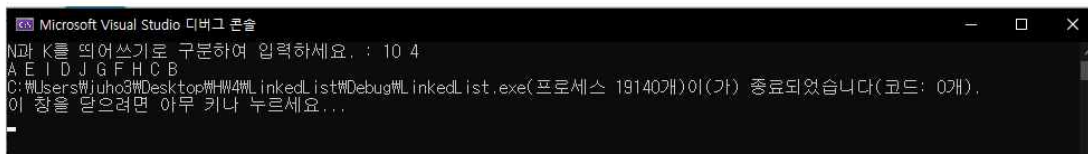
```

```

    K = arr[1];
    for (int i =0; i < N; i ++) {
        tail = insert_last(tail, i +65);
    }
    int i;
    for (i =0; i < N; i ++) {
        print_first_node(tail);
        tail = delete_first(tail);
        if (i == N -1) break;
        for (int j =0; j < K -1; j ++) {
            tail = tail ->link;
        }
    }
    printf("%n");
    return 0;
}

```

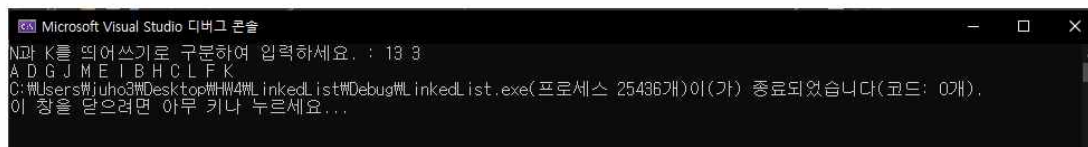
– 실행 결과



```

Microsoft Visual Studio 디버그 콘솔
N과 K를 띄어쓰기로 구분하여 입력하세요. : 10 4
A E I D J G F H C B
C:\Users\juho3\Desktop\LinkedList\Debug\LinkedList.exe(프로세스 19140개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...

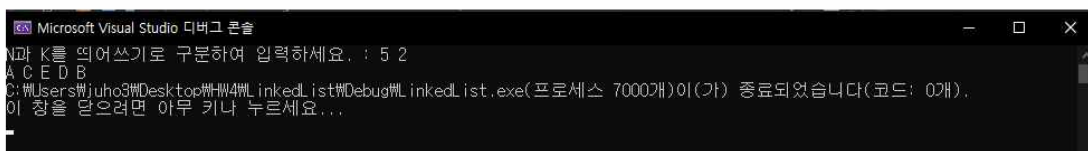
```



```

Microsoft Visual Studio 디버그 콘솔
N과 K를 띄어쓰기로 구분하여 입력하세요. : 13 3
A D G J M E I B H C L F K
C:\Users\juho3\Desktop\LinkedList\Debug\LinkedList.exe(프로세스 25436개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...

```



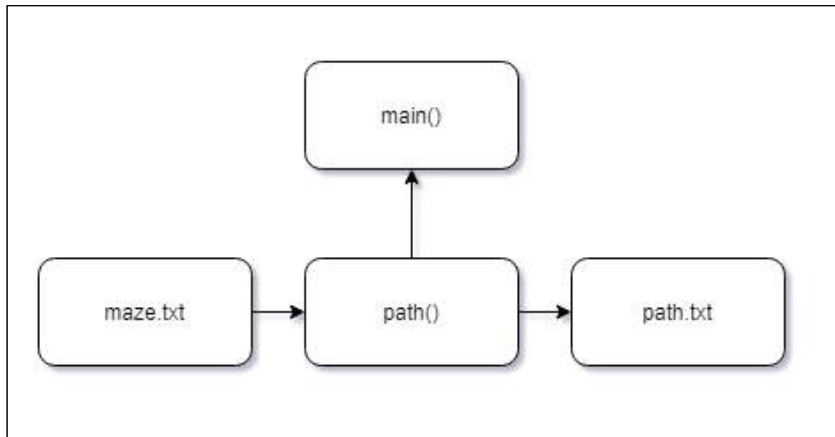
```

Microsoft Visual Studio 디버그 콘솔
N과 K를 띄어쓰기로 구분하여 입력하세요. : 5 2
A C E D B
C:\Users\juho3\Desktop\LinkedList\Debug\LinkedList.exe(프로세스 7000개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...

```

3번

– 함수 구성 다이어그램



- Double Linked List로 구현한 Stack의 연산

① void push()

이중 연결 리스트에서 새로운 데이터를 노드 pre의 오른쪽에 삽입하는 함수인 d_insert()를 이용하여, head 쪽에 새로운 정보를 밀어 넣는 방향으로 스택 자료구조의 push연산을 구현한다.

input : DListNode* head, element data

return : 없음

② element pop()

이중 연결리스트에서 타겟 노드를 삭제하는 d_delete()를 활용하여, 스택 자료구조의 pop연산을 구현한다. 첫 번째 노드를 임시저장했다가 return하고, 첫 번째 노드를 삭제하는 방식으로 pop()연산을 구현한다. 정리하면, 노드의 삽입과 삭제를 모두 head쪽에서 해내는 방식으로 스택 자료구조를 구현한 것이다.

input : DListNode* head

return : 삭제한 원소 tmp

- 함수 설명

① void path()

단계 1. maze.txt 파일을 읽어와서, 이 정보를 maze 이차원 array에 저장한다.

단계 2. stack 자료구조를 이용하여, 방문한 자리를 기록하는 기능, 막다른 길에서 최근 분기점으로 돌아가는 기능을 수행한다.

외부에서 stack 자료구조를 활용하는 방식은 이전 maze 실습과 거의 동일하다. 이중 연결리스트에서 push하고 pop할 수 있도록 parameter만 조정하면 된다.

단계 3. 출구에 도달할 수 있다면, 이를 path.txt 파일에 쓴다. (쓰기모드 “w”)

지나온 경로를 처음부터 출력하고자 한다면, stack에 저장되어 있는 자료들을 하나

씩 pop()하는 방식으로 접근하여서는 안된다. 가장 최근에 방문한 지점이 pop()되기 때문이다. stack 자료구조에 저장된 내용의 맨 바닥(처음)에 접근하기 위해서 이중 연결리스트의 left link를 활용한다. head 노드에서 llink를 타고 한번만 이동한다면, 곧바로 stack 자료구조의 바닥에 도달할 수 있다. 이는 단순 연결리스트 자료구조로 구현한 stack이었다면, 불가능한 접근이 된다.

출구에 도달할 수 없다면, 콘솔에 실패 사실을 메시지로 알려주게 된다.

input : DListNode** head

output : 없음

- 코드

```
#include <stdio.h >
#include <stdlib.h >
#include <time.h >
#define MAX_ROW 10
#define MAX_COL 10
#define TRUE 1
#define FALSE 0
#define EXIT_ROW MAX_ROW -1
#define EXIT_COL MAX_COL -1
////////////////////Doubly Linked List ADT////////////////////
#include <stdio.h >
#include <stdlib.h >
typedef struct {
    short int row;
    short int col;
    short int dir;
}element;
typedef struct DListNode {
    element data;
    struct DListNode * llink;
    struct DListNode * rlink;
}DListNode;
// 이중 연결 리스트를 초기화
void init(DListNode * phead) {
    phead->llink = phead;
    phead->rlink = phead;
}
int is_empty(DListNode * phead) {
    return phead ->llink == phead && phead ->rlink == phead;
}
// 이중 연결 리스트의 노드를 출력
void print_dlist_reverse(DListNode * phead) {
    DListNode* p;
    for (p = phead ->llink; p != phead; p = p ->llink) {
        printf("%2d%5dWn", p ->data.row, p ->data.col);
    }
}
```

```

// 새로운 데이터를 노드 pre의 오른쪽에 삽입
void d_insert(DListNode * pre, element data) {
    DListNode* newNode = (DListNode *)malloc(sizeof(DListNode));
    if (newNode == NULL) {
        printf("동작할당 err!");
        exit(1);
    }
    newNode->data = data;
    newNode->llink = pre;
    newNode->rlink = pre ->rlink;
    pre->rlink ->llink = newNode;
    pre->rlink = newNode;
}

// 노드 removed를 삭제
void d_delete(DListNode * head, DListNode * removed) {
    if (removed == head) return;
    removed->llink ->rlink = removed ->rlink;
    removed->rlink ->llink = removed ->llink;
    free(removed);
}

////////////////////////////////////
// 전역 변수
typedef struct {
    short int vert;
    short int horiz;
}offsets;
offsets move[8] = { {-1, 0}, {-1,1}, {0,1}, {1,1}, {1,0}, {1,-1}, {0,-1}, {-1,-1} };
int maze[MAX_ROW][MAX_COL];
int mark[MAX_ROW][MAX_COL];
// 함수 선언
void path(DListNode ** head);
void push(DListNode * head, element data);
element pop(DListNode * head);
////////////////////////////////////Stack using Doubly linked list ADT////////////////////////////////////
void push(DListNode * head, element data) {
    d_insert(head, data);
}
element pop(DListNode * head) {
    element tmp = head ->rlink ->data;
    d_delete(head, head->rlink);
    return tmp;
}

////////////////////////////////////
void path(DListNode ** head)
{
    int row, col, next_row, next_col, dir, found = FALSE;
    element position, initial;
    FILE* fp1, *fp2;
    if ((fp1 = fopen("maze.txt", "r")) == NULL) {
        printf("파일 읽기 오류! Wn");
        exit(1);
    }
    for (int i =0; i < MAX_ROW; i ++) {

```

```

        for (int j =0; j < MAX_COL; j++) {
            if (fscanf(fp1, "%d", &maze[i][j]) == -1) {
                return ;
            }
        }
    }
    fclose(fp1);
    mark[1][1] =1;
    initial.row =1;
    initial.col =1;
    initial.dir =1;
    push(*head, initial);
    while (!is_empty(*head) &&!found) {
        position =pop(*head);
        row = position.row;
        col = position.col;
        dir = position.dir;
        while (dir <8 &&!found) {
            next_row = row + move[dir].vert;
            next_col = col + move[dir].horiz;
            if (next_row == EXIT_ROW && next_col == EXIT_COL) {
                found = TRUE;
            }
            else if (!maze[next_row][next_col]
&&!mark[next_row][next_col]) {
                mark[next_row][next_col] =1;
                position.row = row;
                position.col = col;
                position.dir =++dir;
                push(*head, position);
                row = next_row;
                col = next_col;
                dir =0;
            }
            else ++dir;
        }
    }
    if (found) {
        /*
        printf("The path is : \n");
        printf("row col\n");
        print_dlist_reverse(head);
        printf("%2d%5d\n", row, col);
        */
        DListNode* ptr;
        if ((fp2 = fopen("path.txt", "w")) ==NULL) {
            printf("파일 쓰기 오류! \n");
            exit(1);
        }
        for (ptr = (*head)->llink; ptr != (*head); ptr = ptr ->llink) {
            fprintf(fp2, "%2d%5d\n", ptr ->data.row, ptr ->data.col);
        }
        fprintf(fp2, "%2d%5d\n", row, col);
        fclose(fp2);
    }
}

```



```

        printf("path.txt 파일이 생성되었습니다.\n");
    }
    else {
        printf("The maze dose not have a path\n");
    }
}
int main(void) {
    DListNode* head = (DListNode *)malloc(sizeof(DListNode));
    init(head);
    path(&head);
    return 0;
}

```

– 실행 결과

Microsoft Visual Studio 디버그 콘솔

```

path.txt 파일이 생성되었습니다.
C:\Users\juho3\Desktop\LinkedList\Debug\LinkedList.exe (프로세스 9772개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...

```

maze - Windows 메모장

```

1 1 1 1 1 1 1 1 1 1
1 0 1 1 1 1 1 0 1 1
1 1 0 0 0 1 0 1 1 1
1 0 0 0 1 0 0 0 1 1
1 1 0 0 0 0 1 1 1 1
1 0 1 0 0 1 0 0 0 1
1 1 0 1 0 0 1 0 1 1
1 0 1 1 1 1 1 0 0 1
1 0 1 1 0 0 0 1 0 1
1 1 1 1 1 1 1 1 1 1

```

path - Windows 메모장

```

1 1
2 2
2 3
2 4
3 5
2 6
3 7
3 6
4 5
5 6
5 7
5 8
6 7
7 8
8 8

```

Microsoft Visual Studio 디버그 콘솔

```

The maze dose not have a path
C:\Users\juho3\Desktop\LinkedList\Debug\LinkedList.exe (프로세스 30408개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...

```

maze - Windows 메모장

```

1 1 1 1 1 1 1 1 1 1
1 0 1 1 1 1 1 0 1 1
1 1 0 0 0 1 0 1 1 1
1 0 0 0 1 1 0 0 1 1
1 1 0 0 0 1 1 1 1 1
1 0 1 0 0 1 1 0 0 1
1 1 0 1 0 0 1 0 1 1
1 0 1 1 1 1 1 0 0 1
1 0 1 1 0 0 0 1 0 1
1 1 1 1 1 1 1 1 1 1

```

path - Windows 메모장

```


```