

자료구조 프로그래밍 과제1 제출

호크마교양대학 장에서

<Maze with Point>

1) 주석이 달린 소스 코드

```
#include <stdio.h> // stdio.h. 헤더 파일을 선언한다.
#define MAX_STACK_SIZE 1000 // MAX_STACK_SIZE의 크기를 1000으로 정의한다.
#define MAZE_SIZE 15 // MAZE_SIZE의 크기를 15로 정의한다.

typedef struct {
    short r;
    short c;
} Element;
// Element 구조체에 두 short r, c를 선언하였다.

typedef struct {
    Element data[MAX_STACK_SIZE];
    int top;
} StackType;
// StackType 구조체에 data[1000]과 top 변수를 선언하였다.

void init_stack(StackType* s) { // 빈 스택을 만드는 함수 init_stack을 선언한다.
    s->top = -1; // 스택 s의 top은 -1이 된다.
}

int isEmpty(StackType* s) { // 스택이 비어 있는지를 판단하는 함수 isEmpty를 선언한다.
    if (s->top == -1)
        return 1; // 만약 스택 s의 top이 -1이면 스택은 비어 있는 것이고 1을
    리턴한다.
    else
        return 0; // 그게 아니라면 함수는 0을 리턴한다.
}

int isFull(StackType* s) { // 스택이 가득 차 있는지를 판단하는 함수 isFull을 선언한다.
    if (s->top == MAX_STACK_SIZE - 1)
        return 1; // 만약 스택 s의 top 값이 MAX_STACK_SIZE-1이면 스택이 가득 차
    있는 것이고 함수는 1을 리턴한다.
    else
        return 0; // 그게 아니라면 함수는 0을 리턴한다.
}

void Push(StackType* s, Element k) { // 스택 s에 Element k를 넣는 함수 Push를 선언한다.
    if (isFull(s)); // 만약 스택이 가득 차 있다면, 함수를 종료한다.
    else s->data[++(s->top)] = k; // 그게 아니라면, (스택 s의 top) + 1값을 인덱스로
    가지는 스택 s의 데이터에 k를 집어넣는다.
}

Element Pop(StackType* s) { // 스택 s에서 데이터 하나를 빼오는 함수 Pop을 선언한다.
```

```
    if (isEmpty(s)) {
        return;
    } // 만약 스택이 비어있다면 함수를 종료한다.
    else {
        return s->data[(s->top)--];
    } // 그게 아니라면, s->data[(s->top)]를 리턴한다. 그 후, 스택 s에서의 top 값은 top-
    1이 된다.
}
```

Element here = { 1, 0 }; // Element here을 선언한다. here은 { 1, 0 } 이다.
Element entry = { 1, 0 }; // Element entry를 선언한다. entry는 { 1, 0 } 이다.

```
char maze[MAZE_SIZE + 2][MAZE_SIZE + 2] = {
    {'1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1'},
    {'1','e','0','2','1','1','1','1','1','1','1','1','1','1','1','1','1'},
    {'1','1','0','1','1','1','2','1','1','1','1','2','0','0','0','3','1'},
    {'1','1','0','1','0','0','0','1','0','1','3','1','1','0','1','1','1'},
    {'1','1','0','0','0','2','1','1','0','0','1','1','1','1','0','1','1'},
    {'1','1','0','1','0','1','1','1','0','0','0','0','0','0','0','0','1'},
    {'1','1','3','1','0','2','0','1','0','1','1','0','1','0','1','1','1'},
    {'1','0','0','0','0','0','0','0','0','0','0','1','2','0','1','1','1'},
    {'1','2','1','1','1','1','2','1','1','1','0','0','0','0','0','0','1'},
    {'1','1','1','1','1','1','1','1','0','0','0','1','1','1','1','3','1'},
    {'1','1','1','1','1','1','1','3','1','0','1','0','0','1','1','1','1'},
    {'1','1','1','1','1','1','1','0','0','0','1','0','1','1','1','1','1'},
    {'1','1','1','1','1','1','1','3','0','1','1','0','3','1','1','1','2'},
    {'1','1','3','1','1','1','1','0','1','2','0','1','0','0','0','0','1'},
    {'1','0','0','0','0','0','0','0','1','1','0','1','3','1','1','1','1'},
    {'1','1','1','1','1','1','1','1','1','1','1','0','0','0','0','x','1'},
    {'1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1','1'}
}; // MAZE[17][17]을 선언한다.
```

// 이때, 미로가 15X15임에도 불구하고 인덱스를 17로 두고 첫째와 마지막 행/열의 값을 모두 1로 둔 것은 모든 점에서 위/아래/오른쪽/왼쪽으로 움직일 수 있도록 하기 위해서이다.

```
void push_loc(StackType* s, int r, int c) { // {r,c}를 스택 s에 집어넣는 push_loc 함수를
    생성한다.
    if (r < 0 || c < 0) return; // 만약 r, c가 0보다 작다면 함수를 종료한다.
    if (maze[r][c] != '1' && maze[r][c] != '.') { // 만약 좌표값이 이미 지나온 길이나
    벽을 가르키지 않는다면,
        Element tmp; // Element tmp를 선언한다.
        tmp.r = r; // tmp의 r값에 r을 대입한다.
        tmp.c = c; // tmp의 c값에 c를 대입한다.
        Push(s, tmp); // push 함수를 이용해 스택 s에 tmp를 집어넣는다.
    }
}
```

```
void maze_print(char maze[MAZE_SIZE + 2][MAZE_SIZE + 2]) { // maze_print 함수는 미로 지도를
    출력해 주기 위한 함수이다.
    printf("\n");
    for (int r = 1; r < MAZE_SIZE + 1; r++) {
        for (int c = 1; c < MAZE_SIZE + 1; c++) {
            // 미로는 15X15인데 maze는 17X17이므로, 본인이 코드를 작성할 때
            임의로 추가한 맨 윗쪽과 오른쪽 행/열을 제외하고 미로 지도를 출력한다.
            printf("%c", maze[r][c]); // maze[r][c]의 char값을 출력한다.
```

```

    }
    printf("\n");
}

int main(void) { // main 함수를 선언한다.
    StackType s; // StackType s를 선언한다.
    int r, c;
    int t = 0, w = 0; // int r, c, t, w를 선언하고, t와 w를 0으로 초기화한다.

    init_stack(&s); // 스택 s를 init_stack 함수를 이용해 초기화한다.
    here = entry;

    while (maze[here.r][here.c] != 'x') { // while 문을 이용해 출구를 찾을 때까지 루프를
        반복한다. 출구('x')에 도달하면 루프를 탈출한다.
            r = here.r; // r에 here.r 값을 집어넣는다.
            c = here.c; // c에 here.c값을 집어넣는다.
            if (maze[r][c] == '2') { // 만약 maze[r][c]가 2라면, (보물을 찾았다면)
                t += 1; // t = t+1이 된다.
            }
            else if (maze[r][c] == '3') { // 만약 maze[r][c]가 3이라면, (함정을
                밟았다면)
                    w += 1; // w = w+1이 된다.
                }
            maze[r][c] = '.'; // 이제 maze[r][c]는 이미 지나간 곳이므로 '.'으로 표기를
                바꿔 준다.

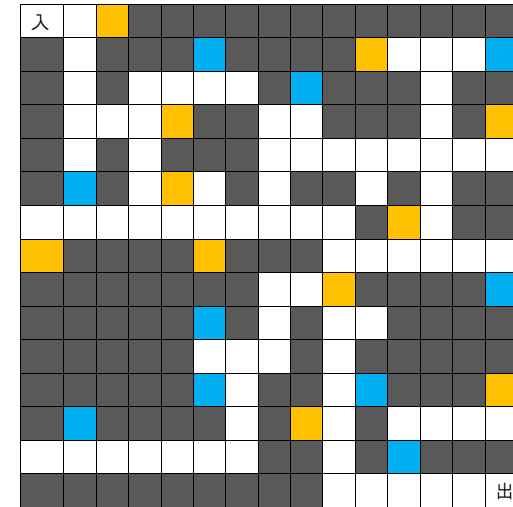
            maze_print(maze); // 미로 지도를 출력한다.
            push_loc(&s, r - 1, c); // 사용자의 위치에서 오른쪽 경로를 스택에 저장한다.
            push_loc(&s, r + 1, c); // 사용자의 위치에서 왼쪽 경로를 스택에 저장한다.
            push_loc(&s, r, c - 1); // 사용자의 위치에서 아래쪽 경로를 스택에 저장한다.
            push_loc(&s, r, c + 1); // 사용자의 위치에서 위쪽 경로를 스택에 저장한다.

            if (!isEmpty(&s)) { // 만약 스택이 비어있다면,
                printf("실패\n"); // "실패"를 출력한다.
                return;
            }
            else { // 만약 스택이 비어있지 않다면,
                here = Pop(&s); // here에 스택 맨 위의 값을 pop한다.
            }
        }
        printf("성공!\n"); // while문을 빠져나오면, 성공을 출력한다.
        printf("취득한 보물의 갯수는: %d\n이동 중 만난 함정의 갯수는: %d\n총 보물
        점수는: %d\n", t, w, t - w);
        // 보물 갯수, 함정 갯수, 보물 점수를 출력한다. 보물은 1점, 함정은 -1점이므로 보물
        점수는 t-w가 된다.

        return 0; // 0을 리턴하고 함수를 종료한다.
    }
}

```

2) 미로 지도

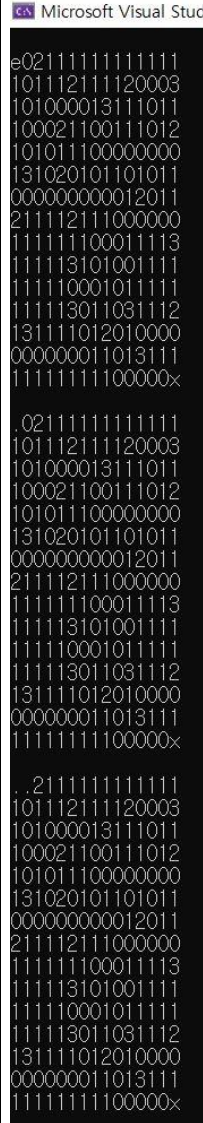

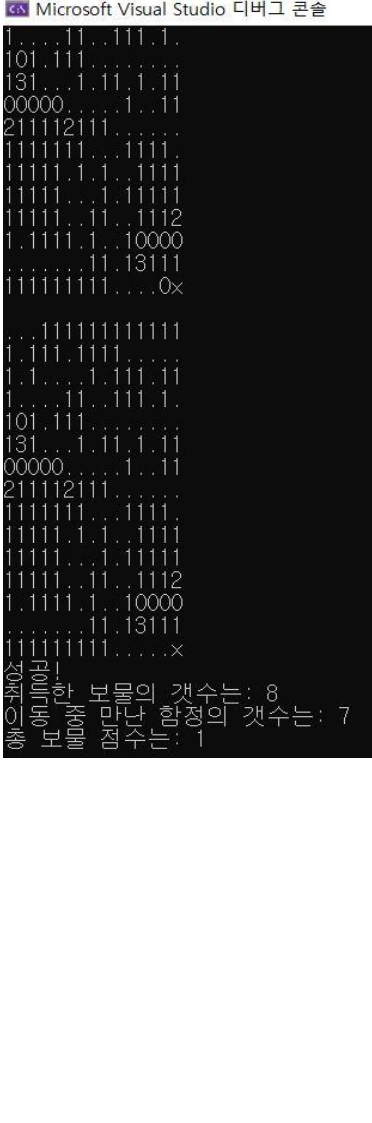


길: 0, 벽: 1, 보물: 2, 장애물: 3

시작점: e, 도착점: x

e	0	2	1	1	1	1	1	1	1	1	1	1	1	1
1	0.	1	1	1	3	1	1	1	1	2	0	0	0	3
1	0.	1	0	0	0	0	1	3	1	1	1	0	1	1
1	0	0	0.	2	1	1	0	0	1	1	1	0	1	2
1	0	1	0.	1	1	1	0	0	0	0	0	0	0	0
1	3	1	.0	2	0	1	0	1	1	0	1	0	1	1
0	0	0	0	0	0	0	0	0	0	1	2	0	1	1
2	1	1	1	1	2	1	1	1	0	0	0	0	0	0
1	1	1	1	1	1	1	0	0	0	1	1	1	1	3
1	1	1	1	1	3	1	0	1	0	0	1	1	1	1
1	1	1	1	1	0	0	0	1	0	1	1	1	1	1
1	1	1	1	1	3	0	1	1	0	3	1	1	1	2
1	3	1	1	1	1	0	1	2	0	1	0	0	0	0
0	0	0	0	0	0	0	1	1	0	1	3	1	1	1
1	1	1	1	1	1	1	1	1	0	0	0	0	0	x

3) 실행화면 캡처

시작	중간	최종 종료 화면
		

<Advanced Stack>

1) 주석이 달린 소스 코드

```

- #include<stdio.h> // stdio.h 헤더 파일을 선언한다.
- #include<stdlib.h> // malloc, free를 쓰기 위해 stdlib.h 헤더 파일을 선언한다.
- #include<string.h> // strcpy를 쓰기 위해 string.h 헤더 파일을 선언한다.
- #define MAX_STACK_SIZE 10 // MAX_STACK_SIZE의 크기를 10으로 정의한다.
-
- typedef char* element; // char*을 element로 선언한다.
-
- typedef struct {
-     element data[MAX_STACK_SIZE];
-     int top;
- } StackType;
-
- // StackType 구조체에 data[10]과 top 변수를 선언하였다.
-
- int empty_stack(StackType* s) { // 스택이 비어 있는지를 판단하는 함수
-     empty_stack를 선언한다.
-     if (s->top == -1)
-         return 1; // 만약 스택이 비어 있다면, 1을 반환한다.
-     else
-         return 0; // 만약 스택이 비어 있지 않다면, 0을 반환한다.
- }
-
- int full_stack(StackType* s) { // 스택이 가득 차 있는지를 판단하는 함수 full_stack를
-     선언한다.
-     if (s->top == MAX_STACK_SIZE - 1)
-         return 1; // 만약 스택이 가득 채워져 있다면, 1을 반환한다.
-     else
-         return 0; // 만약 스택이 가득 채워져 있지 않다면, 0을 반환한다.
- }
-
- void push_stack(StackType* s, element k) { // 스택 s에 element k를 집어넣는 함수
-     push_stack를 선언한다.
-     if (full_stack(s))
-         return; // 만약 스택이 가득 차 있다면 함수가 종료된다.
-     else {
-         s->data[(s->top)++] = k;
-     } // 만약 스택이 가득 차 있지 않다면, 구조체 s의 (top+1)을 인덱스로 하는 구조체 s의
-     element data에 k가 대입된다.
- }
-
- element pop_stack(StackType* s) { // 스택 s에서 하나의 값을 꺼내는 함수 pop_stack를
-     선언한다.
-     if (empty_stack(s))

```

```

-         return NULL; // 만약 스택이 비어 있다면 아무 값도 반환하지 않고 함수가
종료된다.
-     else {
-         element a = s->data[(s->top)--]; // 구조체 s의 top을 인덱스로 하는 구조체
s의 element data를 값으로 갖는 element a를 선언한다. 그 후, 구조체 s의 top에 top-1이
대입된다.
-         return a; // a를 리턴한다.
-     }
- }
- }

- typedef struct {
-     StackType sOne;
-     StackType sTwo;
- } QueueType; // QueueType 구조체에 StackType 두 개를 선언하였다.
-
- void init_queue(QueueType* q) { // 새로운 큐 q를 만드는 함수 init_queue를 선언한다.
-     q->sOne.top = -1;
-     q->sTwo.top = -1;
- } // q 안의 두 스택의 top값을 모두 -1로 만들어 큐를 초기화시킨다.
-
- void dequeue(QueueType* q) { // q에서 값을 출력하는 함수 dequeue를 선언한다.
-     if (empty_stack(&q->sTwo)) { // 만약 q 안의 스택 sTwo가 비어 있다면,
-         while (!empty_stack(&(q->sOne))) { // q안의 스택 sOne에 아무 데이터도 남아
있지 않을 때까지
-             push_stack(&(q->sTwo), pop_stack(&q->sOne)); // 스택 sOne에 있는
값을 pop한 뒤 스택 sTwo에 push한다.
-         }
-     }
-     printf("%s\n", pop_stack(&(q->sTwo))); // 큐 q 안의 스택 sTwo에서 하나의 데이터를
출력한다.
- }
-
- void enqueue(QueueType* q, element k) { // 큐 q에 element k를 집어넣는 함수
enqueue를 선언한다.
-     push_stack(&(q->sOne), k); // push_stack 함수를 이용해 큐 q 안의 스택 sOne에 k를
집어넣는다.
- }
-
- void print_stack(StackType* s) { // 스택 s에 있는 모든 값을 출력하는 함수
print_stack를 선언한다.
-     if (empty_stack(s)); // 만약 스택 s가 비어 있다면, 함수를 종료한다.
-     else { // 만약 스택 s가 비어 있지 않다면,
-         element copy[20]; // 20개의 인덱스를 가지는 element copy를 선언한다.
-         int j = s->top + 1; // 스택 s의 top 값에 1을 더한 값을 갖는 int j를
선언한다.
-         for (int k = 0; k < j; k++) {
-             strcpy(copy, s->data[k]); // strcpy를 이용하여 copy에 s->data[k]에

```

있는 데이터값을 복사한다.

```

-         printf("%s ", copy);
-     } // 스택 s에 있는 모든 데이터를 출력한다.
- }
-     printf("\n");
- }
-
- void printallstack(QueueType* q) { // 큐 q를 구성하는 두 스택에 있는 데이터값을 모두
출력하는 함수 printallstack를 선언한다.
-     printf("입력 스택이 가지고 있는 데이터는: ");
-     print_stack(&q->sOne); // 함수 print_stack를 이용해 큐 q의 스택 sOne에 있는 모든
데이터값을 출력한다.
-     printf("출력 스택이 가지고 있는 데이터는: ");
-     print_stack(&q->sTwo); // 함수 print_stack를 이용해 큐 q의 스택 sTwo에 있는 모든
데이터값을 출력한다.
-     printf("\n");
- }
-
- int main(void) { // main 함수를 선언한다.
-     QueueType QQQ; // QueueType QQQ를 선언한다.
-     init_queue(&QQQ); // QQQ를 초기화한다.
-     int n; // int n을 선언한다.
-     int k = 0; // int k를 선언한다. k는 0으로 초기화된 값이다.
-     printf("입력할 데이터의 수는?: ");
-     scanf_s("%d", &n); // 입력할 데이터의 수 n을 scanf_s로 사용자로부터 입력받는다.
-     element city; // element city를 선언한다.
-     for (int i = 0; i < n; i++) { // for 루프는 n번 반복된다.
-         printf("도시 이름: ");
-         city = (element)malloc(sizeof(char) * MAX_STACK_SIZE); // element city에
대해 malloc 함수를 이용해 동적할당을 해 준다.
-         scanf("%s", city); // 사용자로부터 city를 입력받는다.
-         enqueue(&QQQ, city); // enqueue 함수를 이용하여 city를 큐 QQQ에 저장한다.
-         printallstack(&QQQ); // printallstack 함수를 이용하여 enqueue 실행 후 큐
QQQ를 구성하는 스택들의 모든 데이터값을 출력한다.
-         if (i > 0 && i % 5 == 0) { // for문이 반복되는 동안 i가 0보다 크고, 5의
배수일 때마다,
-             dequeue(&QQQ); // dequeue를 이용해 큐 QQQ에서 하나의 값을 출력한다.
-             printallstack(&QQQ); // printallstack 함수를 이용하여 dequeue 실행
후 큐 QQQ를 구성하는 스택들의 모든 데이터값을 출력한다.
-             k++; // k = k+1이 된다.
-         }
-     }
-     printf("\n");
-     for (int i = 0; i < (10 - k); i++) { // for문이 10-k번만큼 반복된다. 10-k번인 이유는
조건에 의해 이 프로그램에서 출력이 10번만 되어야 하기 때문이다.
-         dequeue(&QQQ); // dequeue를 이용해 큐 QQQ에서 하나의 값을 출력한다.
-         printallstack(&QQQ); // printallstack 함수를 이용하여 dequeue 실행 후 큐

```

```

    QQQ를 구성하는 스택들의 모든 데이터값을 출력한다.
-   }
-   free(city); // 동적으로 할당되었던 메모리 블록을 시스템에 반납한다.
-   return 0; // 0을 리턴하고 main 함수를 종료한다.
-   }

```

2) 실행 결과

Microsoft Visual Studio 디버그 콘솔

```

입력 할 데이터의 수는?: 13
도시 이름: Seoul
인력 스택이 가지고 있는 데이터는: Seoul
출력 스택이 가지고 있는 데이터는:

도시 이름: Madrid
인력 스택이 가지고 있는 데이터는: Seoul Madrid
출력 스택이 가지고 있는 데이터는:

도시 이름: Singapore
인력 스택이 가지고 있는 데이터는: Seoul Madrid Singapore
출력 스택이 가지고 있는 데이터는:

도시 이름: Victoria
인력 스택이 가지고 있는 데이터는: Seoul Madrid Singapore Victoria
출력 스택이 가지고 있는 데이터는:

도시 이름: Kingstown
인력 스택이 가지고 있는 데이터는: Seoul Madrid Singapore Victoria Kingstown
출력 스택이 가지고 있는 데이터는:

도시 이름: WashingtonD.C.
인력 스택이 가지고 있는 데이터는: Seoul Madrid Singapore Victoria Kingstown WashingtonD.C.
출력 스택이 가지고 있는 데이터는:

Seoul
인력 스택이 가지고 있는 데이터는:
출력 스택이 가지고 있는 데이터는: WashingtonD.C. Kingstown Victoria Singapore Madrid

도시 이름: Jakarta
인력 스택이 가지고 있는 데이터는: Jakarta
출력 스택이 가지고 있는 데이터는: WashingtonD.C. Kingstown Victoria Singapore Madrid

도시 이름: Budapest
인력 스택이 가지고 있는 데이터는: Jakarta Budapest
출력 스택이 가지고 있는 데이터는: WashingtonD.C. Kingstown Victoria Singapore Madrid

도시 이름: Rome
인력 스택이 가지고 있는 데이터는: Jakarta Budapest Rome
출력 스택이 가지고 있는 데이터는: WashingtonD.C. Kingstown Victoria Singapore Madrid

도시 이름: Tokyo
인력 스택이 가지고 있는 데이터는: Jakarta Budapest Rome Tokyo
출력 스택이 가지고 있는 데이터는: WashingtonD.C. Kingstown Victoria Singapore Madrid

도시 이름: Berlin
인력 스택이 가지고 있는 데이터는: Jakarta Budapest Rome Tokyo Berlin
출력 스택이 가지고 있는 데이터는: WashingtonD.C. Kingstown Victoria Singapore Madrid

```

Microsoft Visual Studio 디버그 콘솔

```

Madrid
인력 스택이 가지고 있는 데이터는: Jakarta Budapest Rome Tokyo Berlin
출력 스택이 가지고 있는 데이터는: WashingtonD.C. Kingstown Victoria Singapore

도시 이름: Vienna
인력 스택이 가지고 있는 데이터는: Jakarta Budapest Rome Tokyo Berlin Vienna
출력 스택이 가지고 있는 데이터는: WashingtonD.C. Kingstown Victoria Singapore

도시 이름: Manama
인력 스택이 가지고 있는 데이터는: Jakarta Budapest Rome Tokyo Berlin Vienna Manama
출력 스택이 가지고 있는 데이터는: WashingtonD.C. Kingstown Victoria Singapore

Singapore
인력 스택이 가지고 있는 데이터는: Jakarta Budapest Rome Tokyo Berlin Vienna Manama
출력 스택이 가지고 있는 데이터는: WashingtonD.C. Kingstown Victoria

Victoria
인력 스택이 가지고 있는 데이터는: Jakarta Budapest Rome Tokyo Berlin Vienna Manama
출력 스택이 가지고 있는 데이터는: WashingtonD.C. Kingstown

Kingstown
인력 스택이 가지고 있는 데이터는: Jakarta Budapest Rome Tokyo Berlin Vienna Manama
출력 스택이 가지고 있는 데이터는: WashingtonD.C.

WashingtonD.C.
인력 스택이 가지고 있는 데이터는: Jakarta Budapest Rome Tokyo Berlin Vienna Manama
출력 스택이 가지고 있는 데이터는:

Jakarta
인력 스택이 가지고 있는 데이터는:
출력 스택이 가지고 있는 데이터는: Manama Vienna Berlin Tokyo Rome Budapest

Budapest
인력 스택이 가지고 있는 데이터는:
출력 스택이 가지고 있는 데이터는: Manama Vienna Berlin Tokyo Rome

Rome
인력 스택이 가지고 있는 데이터는:
출력 스택이 가지고 있는 데이터는: Manama Vienna Berlin Tokyo

Tokyo
인력 스택이 가지고 있는 데이터는:
출력 스택이 가지고 있는 데이터는: Manama Vienna Berlin

```

3) 프로그램 설명

한 개의 큐는 두 개의 스택으로 이루어져 있다. 큐가 받는 입력은 sOne 스택에만 저장되고, 출력 명령은 sTwo 스택에서만 시행된다. 큐 출력 명령이 시행되었을 때 sTwo 스택이 비어 있다면 sOne 스택이 빌 때까지 sOne 스택에서 pop 한 값을 sTwo 스택으로 push 하는 과정이 반복된다. 프로그램 처음에 데이터를 몇 번 입력받을 것인지 사용자에게 받는다. (n 값) n 만큼 데이터를 집어넣는 과정에서, 5 번 입력 후에는 데이터를 1 번 출력해 준다. (이때, 1 번 출력해 줄 때마다 int k=0 이 1 씩 늘어난다.) 모든 입력이 끝난 후, 나머지 데이터를 (10-k 번) 출력한다. 그러므로 최종적으로 10 번 데이터를 출력한 셈이 된다. 모든 입력, 출력 후에는 입력 스택과 출력 스택이 가지고 있는 데이터를 보여 주도록 하였다.

(자세한 설명은 주석 참조)