자료구조

과제 4

사용언어 : c언어

물리학과 20182326 이선민

문제 1. 미로 탐색 - 스택

- 이번 미로 탐색 문제에서 스택의 기능들은 수업시간에 배운 내용을 그대로 적용하여 구현하였다.
- 스택에 담는 내용은 이중배열의 가로,세로 인덱스와 갈 수 있는 방향 문자를 동시에 담기 위해서 node 구조체를 선언하여 node를 스택에 담았다.
- Get_top이라는 함수를 새롭게 추가하여 요소를 Pop하지 않고 가져올 수 있도록 하였다.

문제 1. 미로 탐색

- 먼저 main함수에서 미로를 읽어온다.
- 미로를 읽은 뒤 맨 첫줄에서 row 와 column을 읽어와 따로 변수에 저장해 주었고 전체 미로는 maze라는 이중배열에 동적할당을 활용하여 저장하였다.
- 저장 방법은 그냥 미로의 크기만큼 동적할당을 받아 전체 미로를 그대로 저장하였다.
- 그리고 저장한 미로를 find_road 라는 함수에 매개변수로 넘겨 길을 찾았다.

문제 1. 미로 탐색 – find_road

- 함수로 들어오면 무한 반복문으로 들어가게된다.
- 우선 prev_dir 이라는 변수를 선언하였다. Prev_dir은 만약 이전에 위치에서 오른쪽으로 이동했다면 prev_dir = ‘r’이다.
- 오른쪽으로 이동한 이후의 위치에서는 이전 위치가 왼쪽이 된다. 따라서 왼쪽으로는 가지 않는다.
- 따라서 만약 prev_dir = ‘r’이면 왼쪽이 아닌 다른 3개의 방향, 오른쪽, 위, 아래를 벽이 있는지 없는 지 체크하면 된다.
- 벽이 있는지 없는 지 체크하는 함수는 check_go함수로 따로 만들었다.

문제 1. 미로 탐색 –check_go

- Check go 함수는 현재 위치 인덱스와 미로, 그리고 갈 방향을 매개변수로 받아서 벽이 아닌 공백일 경우 True를 반환하고 그 외에는 false를 반환한다.

문제 1. 미로 탐색 – find_road

- 만약 그 방향으로 갈 수 있다면 road_dir 배열에 갈 수 있는 방향을 담고 count를 1개 올려준다.
- 그렇게 switch문을 지나면 if문이 나온다. 여기서는 만약 현재 위치가 도착 지점일 경우를 따져준다.
- 도착지점이고 만약 이게 처음 찾은 길이라면 answer라는 stack에 현재까지 지나온 길이 저장되어 있다. 이 answer에 저장된 길을 save_ans라는 이중배열로 옮겨 저장한다. (굳이 다시 저장하는 이유는 후에 설명하겠다.)
- 그 다음 road_count를 1 올려주고 만약 stack 비어있다면 무한 반복문을 빠져나온다.
- 그렇지 않다면 stack에 저장된 위치를 pop하여 해당하는 위치로 이동한다. (새로운 길을 찾기 위해)

문제 1. 미로 탐색

- 그 다음 if문은 길이 없을 경우이다.
- 사방이 막혀 있거나, 다시 처음 위치로 되돌아왔을 경우,
- 이 경우에도 stack이 비어있으면 무한 반복문을 빠져나간다.
- 그게 아니라면 stack에 저장된 위치를 pop하여 해당하는 위치로 이동한다.
- Answer (정답이 되는 길) 스택에 저장된 위치들을 다시 되돌아갈 위치 부분까지 전부 pop한다. 길이 없기때문에 정답이 아니니까

문제 1. 미로 탐색

- 그리고 현재 위치에서 갈 수 있는 방향이 1개 이상일 경우 , count > 1인 경우, 아까 갈 수 있는 방향을 체크하면서 road_dir 배열에 방향을 저장했었다. 첫번째 방향을 제외한 나머지 갈 수 있는 방향과, 현재 위치를 pstack에 저장한다.
- 그리고 저장하지 않은 나머지 한개의 방향으로 이동한다.
- 갈 방향에 따라 인덱스가 다르게 변화한다.
- 갈 수 있는 방향이 1개 인 경우 stack에 push 없이 그냥 이동한다.

문제 1. 미로 탐색

- 그렇게 되면 무한 반복문이 끝난다.
- 무한 반복문이 끝나는 조건은 두개였다.
- 길을 다 찾은 후 첫번째 찾은 길만 answer스택에서 save_ans배열로 옮겨 담은 후 pstack(이동 포인트를 담은 스택)이 비어있게 되면 종료.
- 길이 없는데 pstack 이 비어있게 되면 종료.
- 그렇게 빠져 나와서 이제 미로를 정답이 되는 길과 함께 출력한다.
- 이중 반복문을 통해 출력하는 데 save_ans에 정답이 되는 길 인덱스가 저장되어 있으므로 특정 포인트가 save_ans에 들어있으면 ‘o’를 출력한다.

문제 1. 미로 탐색

- 길이 없으면 save_ans가 비어있으니 그냥 미로만 출력되게 된다.
- 미로를 전부 출력한 후에는 모두 몇개의 길을 찾았는지 출력하고 (road_count 변수에 저장되어 있다.)
- Pstack 과 answer스택을 free 해준뒤 함수가 종료된다.
- Find_road함수가 끝나면 미로를 저장했던 이중배열을 free해주고 미로 파일을 닫고 프로그램을 종료한다.

문제 1. 미로 탐색 – 출력 결과

```
[~/2022_1/Data_structure/Assignment/assignment_4]$ gcc maze_explore.c
[~/2022_1/Data_structure/Assignment/assignment_4]$ ./a.out maze1.txt
PUSH(0,0)
PUSH(1,6)
POP(1,6)
PUSH(0,6)
PUSH(3,7)
PUSH(3,8)
PUSH(4,8)
PUSH(6,8)
POP(6,8)
PUSH(6,7)
PUSH(8,6)
POP(8,6)
POP(6,7)
PUSH(8,3)
PUSH(9,3)
PUSH(10,5)
POP(10,5)
POP(9,3)
PUSH(9,0)
POP(9,0)
POP(8,3)
PUSH(5,3)
POP(5,3)
PUSH(4,3)
PUSH(5,0)
POP(5,0)
PUSH(2,2)
POP(2,2)
POP(4,3)
POP(4,8)
POP(3,8)
POP(3,7)
POP(0,6)
POP(0,0)
PUSH(2,2)
```

```
POP(2,2)
PUSH(5,0)
POP(5,0)
PUSH(4,3)
PUSH(5,3)
POP(5,3)
PUSH(8,3)
PUSH(6,7)
PUSH(6,8)
POP(6,8)
PUSH(4,8)
POP(4,8)
PUSH(3,8)
PUSH(3,7)
PUSH(0,6)
PUSH(1,6)
POP(1,6)
POP(0,6)
POP(3,7)
POP(3,8)
POP(6,7)
PUSH(8,6)
POP(8,6)
POP(8,3)
PUSH(9,3)
PUSH(10,5)
POP(10,5)
POP(9,3)
PUSH(9,0)
POP(9,0)
POP(4,3)
```

문제 1. 미로탐색 – 출력 결과

문제 1. 미로탐색 – 출력 결과

```
[~/2022_1/Data_structure/Assignment/assignment_4]$ ./a.out maze2.txt
PUSH(0,1)
PUSH(4,3)
PUSH(4,4)
PUSH(1,5)
PUSH(0,8)
POP(0,8)
POP(1,5)
POP(4,4)
POP(4,3)
PUSH(4,2)
POP(4,2)
POP(0,1)
PUSH(6,0)
PUSH(6,1)
POP(6,1)
PUSH(7,3)
PUSH(9,5)
POP(9,5)
PUSH(5,6)
POP(5,6)
PUSH(5,8)
PUSH(7,9)
PUSH(8,9)
POP(8,9)
POP(7,9)
POP(5,8)
POP(7,3)
POP(6,0)
```

```
+-------+---+-------+
|S o    |   |       | 0
+-+ +-+ | + +---+ + |
|o o| | | |       | | 1
| +-+ | | | +-----+-+
|o o| | | | |o o o o| 2
+-+ | | | +-+ +-+-+ |
|o o| | |   |o o| |o| 3
| +-+ + +-+ +-+ | | |
|o|         |o o| |o| 4
| +-----+ +-+ +-+ + |
|o|o o o| |o o  |o o| 5
| + +-+ | | +-+-+ +-+
|o o  |o| |o| |o o| | 6
| +---+ +-+ | | +-+ |
| |   |o o|o| |o o o| 7
| + + | + | | +---+ |
|   | | |o|o|     o| 8
+---+ | | + +-----+ |
|     | |o o     |E| 9
+-----+-+---------+-+
 0 1 2 3 4 5 6 7 8 9
모두 1개의 길을 찾았습니다.
```

문제 1. 미로탐색 – 출력 결과

```
[~/2022_1/Data_structure/Assignment/assignment_4]$ ./a.out maze3.txt
PUSH(0,0)
PUSH(1,6)
POP(1,6)
PUSH(0,6)
PUSH(3,7)
POP(3,7)
POP(0,6)
POP(0,0)
PUSH(2,2)
POP(2,2)
PUSH(5,0)
POP(5,0)
PUSH(4,3)
PUSH(5,3)
POP(5,3)
PUSH(8,3)
PUSH(6,7)
PUSH(6,8)
POP(6,8)
POP(6,7)
PUSH(8,6)
POP(8,6)
POP(8,3)
PUSH(9,3)
PUSH(10,5)
POP(10,5)
POP(9,3)
PUSH(9,0)
POP(9,0)
POP(4,3)
```

```
+-------+-------+-+
|S      |       | | 0
| +---+ +---+ + | |
|o o o| |     | | | 1
+---+ | + +-+ | | |
|o o o|   |   | | | 2
| +-+ +-+-+ +-+ + |
|o| | | | | |     | 3
| | +-+ | +-+ +---+
|o| |o o|     |   | 4
| + | + +---+-+ + |
|o  |o|o    |   | | 5
| +-+ | +---+---+ |
|o o o|o|o o o o  | 6
+-+---+ | +---+ + |
| |o o o|o|o o o| | 7
| | +---+ | +---+-+
| |o o o o|o o o o| 8
| +---+ +-+-+ +-+ |
|     |   | | | |o| 9
| +-+ | + | +-+ | |
|   | | | |     |o| 10
+-+ | + | + +---+ |
|   |   |   |    E| 11
+---+---+---+-----+
 0 1 2 3 4 5 6 7 8
모두 1개의 길을 찾았습니다.
```

문제 2. 큐 운영하기

- 큐 구현은 수업시간에 배운 코드를 참고 하였다.
- 우선 큐를 구조체로 선언했다. 큐에는 문자를 담을 배열과 front인덱스, rear인덱스가 저장되어 있다.
- 원형 큐를 구현하였는데 full과 empty를 구분하기 위해 한자리는 제외하고 큐를 사용하였다.
- 따라서 총 20 글자까지 넣을 수 있도록 하기 위해 큐 전체 크기는 21로 하였다.
- 큐를 새로 생성하는 create(), 큐에 요소를 넣는 enqueue, 큐에서 요소를 꺼내는 dequeue, 큐의 현재 상태를 출력해주는 current_queue를 구현하였다.

문제 2. 큐 운영하기

- Enqueue에서는 큐가 가득 찼을 경우 가득 찼다는 메시지를 출력하고 함수를 종료한다.
- 그 외에는 큐에 데이터를 넣고 큐에 추가하였다는 메시지를 출력한다.
- Dequeue에서는 큐가 비어있을 경우 큐가 비어있다는 메시지를 출력하고 함수를 종료한다.
- 그 외에는 큐에서 데이터를 꺼내고 데이터를 꺼냈다는 메시지를 출력한다.
- Current_queue는 현재 큐의 상태를 출력해준다.

문제 2. 큐 운영하기

- Main에서는 data를 입력받아 큐에 추가하거나 꺼내는 작업을 한다.
- 무한 반복문을 도는 데 만약 -1을 입력하면 프로그램을 종료하도록 구현했다.
- 0을 입력하면 큐 현재 상태 출력
- 알파벳을 입력하면 큐에 넣고
- 숫자를 입력하면 숫자만큼 큐에서 데이터를 꺼내 출력한다.
- 입력받은 문자열이 알파벳인지, 숫자인지 판별하는 is_all_digit과 is_all_alpha함수를 따로 구현하여 활용하였다.

문제 2. 큐 운영하기 – 출력 결과.

```
[~/2022_1/Data_structure/Assignment/assignment_4]$ ./a.out
시스템이 시작됩니다. 종료를 원할 시 -1을 입력하세요.
>>> 0
QUEUE = (0)
>>> ABC
(SYSTEM) ADDQUEUE(A) F=0 R=1
(SYSTEM) ADDQUEUE(B) F=0 R=2
(SYSTEM) ADDQUEUE(C) F=0 R=3
>>> DE
(SYSTEM) ADDQUEUE(D) F=0 R=4
(SYSTEM) ADDQUEUE(E) F=0 R=5
>>> 3
DELETEQUEUE() = A, F=1 R=5
DELETEQUEUE() = B, F=2 R=5
DELETEQUEUE() = C, F=3 R=5
>>> 0
QUEUE = DE (2)
>>> FG
(SYSTEM) ADDQUEUE(F) F=3 R=6
(SYSTEM) ADDQUEUE(G) F=3 R=7
>>> 5
DELETEQUEUE() = D, F=4 R=7
DELETEQUEUE() = E, F=5 R=7
DELETEQUEUE() = F, F=6 R=7
DELETEQUEUE() = G, F=7 R=7
DELETEQUEUE() FAIL. QueueEmpty
>>> -1
```