자료구조응용
06. 스택과 큐

1. [동적할당배열을 이용한 환형큐(circular queue)] 동적할당 배열을 이용한 환형큐를 생성하고 실행 예와 같이 수행되는 프로그램을 작성하라. 이를 위해, addq, deleteq, queueFull, queueEmpty qprint(queue의 내용을 출력) 함수를 구현하여야 한다. queueFull 함수는 queue capacity를 두 배로 확장한다.

[자료형과 함수의 정의]

```
typedef struct {
        int id;                          // unique id
        char name[MAX_NAME_SIZE];        //last name
} element;
element *queue;
int capacity = 2;
int rear = 0;
int front = 0;
```

```
element deleteq()
{/* remove front element from the queue */
   element item;
      if (front == rear)
         return queueEmpty(); /* return an error key */
      front = (front+1) % MAX_QUEUE_SIZE;
      return queue[front];
}
```

Program 3.8: Delete from a circular queue

```
void addq(element item)
{/* add an item to the queue */
   rear = (rear+1) % capacity;
   if (front == rear)
      queueFull(); /* double capacity */
   queue[rear] = item;
}
```

Program 3.9: Add to a circular queue

```
void queueFull()
{   int start;
    /* allocate an array with twice the capacity */
    element* newQueue;
    MALLOC(newQueue, 2 * capacity * sizeof(*queue));

    /* copy from queue to newQueue */
    start = (front+1)  % capacity;
    if (start  <  2)
        /* no wrap around */
        copy(queue+start, queue+start+capacity-1, newQueue);
    else
    {/* queue wraps around */
        copy(queue+start, queue+capacity, newQueue);
        copy(queue, queue+rear+1, newQueue+capacity-start);
    }

    /* switch to newQueue */
    front = 2 * capacity - 1;
    rear = capacity - 2;
    capacity *= 2;
    free(queue);
    queue = newQueue;
}
```

Program 3.10: Doubling queue capacity

[구현 조건]

① 사용자입력으로부터 데이터 추출을 위해 gets_s, strtok_s, strcmp, sscanf_s, strlen 등을 사용

② 전역변수 capacity, front, rear의 초기값은 각각 2, 0, 0

③ **addq, deleteq 함수를 참조**

(단, deleteq의 MAX_QUEUE_SIZE를 capacity로 수정함)

④ **circular queue**를 **전역변수 element *queue;**로 선언

⑤ main에서 동적할당으로 capacity 2의 초기queue를 생성함

⑥ **copy 함수**를 **직접 정의**해야 함

[실행 예]

```
 <<  circular queue operations using dynamic allocated array, where the initial capacity is 2>>
add 1 Jung
delete

**************************************************
add 1 kim
add 2 Park
queue capacity is doubled,
current queue capacity is 4.
add 3 Jone
delete
qprint
2, Park
3, Jone

add 4 Korea
add 5 America
queue capacity is doubled,
current queue capacity is 8.
add 6 Song
qprint
2, Park
3, Jone
4, Korea
5, America
6, Song

add 7 Cu
add 8 Me
delete
delete
qprint
4, Korea
5, America
6, Song
7, Cu
8, Me

add 9 Food
add 10 Pen
qprint
4, Korea
5, America
6, Song
7, Cu
8, Me
9, Food
10, Pen
```

2. Program 3.12 Maze search function을 사용하는 미로탐색 프로그램을 작성하여라.

[프로그램 설명]

다음 ① ~ ⑤의 변수는 모두 전역으로 선언되며 정적할당을 사용함.

① maze

- 입구는 left top, 출구는 right bottom으로 가정
- 미로 데이터입력 ("input.txt", 최대 10×10 행렬로 파일입력)

```
4 5
0 0 0 1 1
1 1 0 0 0
1 0 1 1 1
1 0 1 0 0
```

entrance (1, 1), exit (4, 5)

② mark

③ move

```
typedef struct {
        short int vert;
        short int horiz;
        } offsets;
offsets move[8];
```

Name	Dir	*move[dir].vert*	*move[dir].horiz*
N	0	−1	0
NE	1	−1	1
E	2	0	1
SE	3	1	1
S	4	1	0
SW	5	1	−1
W	6	0	−1
NW	7	−1	−1

④ stack

```
#define MAX_STACK_SIZE 100
typedef struct {
          short int row;
          short int col;
          short int dir;
        } element;
element stack[MAX_STACK_SIZE];
int top = -1;
```

⑤ 기호상수 : #define TRUE 1
#define FALSE 0

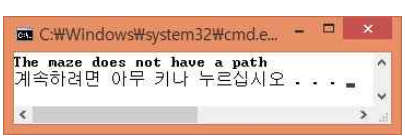
전역변수 : int EXIT_ROW, EXIT_COL;

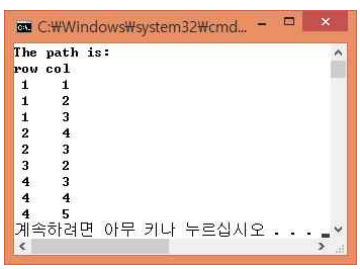
```
void path(void)
{/* output a path through the maze if such a path exists */
   int i, row, col, nextRow, nextCol, dir, found = FALSE;
   element position;
   mark[1][1] = 1; top = 0;
   stack[0].row = 1;  stack[0].col = 1;  stack[0].dir = 1;
   while (top > -1 && !found) {
      position = pop();
      row = position.row;  col = position.col;
      dir = position.dir;
      while (dir <  8 && !found) {
         /* move in direction dir */
         nextRow = row + move[dir].vert;
         nextCol = col + move[dir].horiz;
         if (nextRow == EXIT_ROW && nextCol == EXIT_COL)
            found = TRUE;
         else if ( !maze[nextRow][nextCol] &&
         ! mark[nextRow][nextCol]) {
            mark[nextRow][nextCol] = 1;
            position.row = row; position.col = col;
            position.dir = ++dir;
            push(position);
            row = nextRow; col = nextCol; dir = 0;
         }
         else ++dir;
      }
   }
   if (found) {
      printf("The path is:\n");
      printf("row  col\n");
      for (i = 0; i <= top; i++)
         printf("%2d%5d",stack[i].row, stack[i].col);
      printf("%2d%5d\n",row,col);
      printf("%2d%5d\n",EXIT_ROW,EXIT_COL);
   }
   else printf("The maze does not have a path\n");
}
```

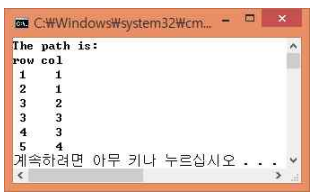
Program 3.12: Maze search function

[실행 예]

4 5 0 0 0 1 1 1 1 0 0 0 1 0 1 1 1 1 0 1 0 0	4 5 0 0 0 1 1 1 1 0 0 0 1 0 1 1 1 1 0 0 0 0	5 4 0 1 1 1 0 1 1 0 1 0 0 1 1 1 0 1 1 0 1 0







■ 제출 형식

- 공학인증 시스템(ABEEK)에 과제를 올릴 때 제목:
- 1차 제출: 학번_이름_DS_06(1), 2차 제출: 학번_이름_DS_06(2)
- 솔루션 이름 : DS_06
- 프로젝트 이름 : 1, 2
- 실행화면을 캡쳐하여 한글파일에 추가 후 솔루션 폴더에 포함.
- 한글 파일명 : 학번_이름.hwp
- 솔루션 폴더를 압축하여 제출할 것.
- 솔루션 압축 파일 명:

  1차 제출: 학번_이름_DS_06(1).zip, 2차 제출: 학번_이름_DS_06(2).zip

- 제출은 2회 걸쳐 가능(수정 시간 기준으로 처리)