1. (**실습**) 큐(queue)는 자료의 삽입(push)과 삭제(pop)가 서로 다른 양 끝(rear와 front)에서 발생한다. 다음은 LinkedList 클래스의 addLast, removeFirst, getFirst 메소드를 큐의 enqueue, dequeue, peek로 사용한 예시 코드이다. 큐가 비어 있는지 검사하기 위해 LinkedList의 isEmpty 메소드를 사용하였으며, 큐 크기 확인을 위해 LinkedList의 size 메소드를 사용하였다. 아래 코드를 입력하고 실행하면서 큐 및 LinkedList의 사용을 익히시오.

   - **실습**: LinkedList의 addLast, removeFirst, getFirst 대신 addFirst, removeLast, getLast 메소드를 사용하여 아래 코드를 재실행해 보시오.

```
public class Test {
	public static void main(String[] args) {
		LinkedList<String>	queue=new LinkedList<>();
		queue.addLast("한국"); // 큐 rear에 자료 삽입, queue.add("한국");
		queue.addLast("미국"); // 큐 rear에 자료 삽입
		queue.addLast("독일"); // 큐 rear에 자료 삽입
		System.out.println("큐 전체 내용: "+queue);
		System.out.println("큐 크기="+queue.size());
		System.out.println("큐가 비어 있는가? "+queue.isEmpty());
		String	v=null;
		v=queue.getFirst();
		System.out.println("큐 front 자료 확인="+v);
		v=queue.removeFirst(); // String v=queue.remove();
		System.out.println("큐 front 추출 값="+v);
		System.out.println("큐 전체 내용: "+queue);
		System.out.println("큐 front 추출 값="+queue.removeFirst());
		System.out.println("큐 전체 내용: "+queue);
		System.out.println("큐 front 추출 값="+queue.removeFirst());
		System.out.println("큐 전체 내용: "+queue);
		System.out.println("큐가 비어 있는가? "+queue.isEmpty());
	}
}
```

2. (리스트 노드 제거) 다음은 리스트 내 특정 위치 자료를 삭제하는 예시 코드이다. 아래 코드를 입력하고 실행하면서 리스트 내 자료 삭제를 학습하시오.

- 실행 결과:
[1, 2, 3, 4, 5]
- 제거된 값=3
[1, 2, 4, 5]

```
public class Test {
	public static void main(String[] args) {
			LinkedList<Integer>	queue=new LinkedList<>();
			for (int i=1; i <=5; i++) queue.add(i); // 1 2 3 4 5
			System.out.println(queue);
			int		v=queue.remove(2);
			System.out.println("제거된 값="+v);
			System.out.println(queue);
	}
}
```

3. (**실습**: 원형 리스트) 다음은 1~7까지 수들이 원형으로 연결되었다고 가정하고 curPos 위치부터 k번째 노드 값을 출력하는 코드이다. curPos가 0인 경우 k번째 노드 값은 4가 출력되며, curPos가 5인 경우 2가 출력되어야 한다. 아래 코드의 오류를 수정하시오.

- 실행 결과:
4
2

```
public class Test {
	public static void main(String[] args) {
			int		n=7, k=4;
			LinkedList<Integer>	queue=new LinkedList<>();
			for (int i=1; i <=n; i++) queue.add(i); // 1 2 3 4 5 6 7
			int		curPos=0;
			System.out.println(queue.get(curPos+k-1)); // curPos부터 k번째 노드 값 => 4
			curPos=5;
			System.out.println(queue.get(curPos+k-1)); // curPos부터 k번째 노드 값 => 2
	}
}
```

4. (**실습**: 요세푸스 문제) 요세푸스 문제는, 원형으로 둘러선 사람 n명에 대해 특정 위치부터 시작하여 한 방향으로 k번째 사람을 원형에서 제외하는 작업을 반복한다고 할 때 마지막으로 제외되는 사람의 최초 위치를 알아내는 것이다. 아래 예시와 같이 n=7, k=3인 경우 요세푸스 문제의 해는 4이다. 구현 방법 #1, #2를 참조하여 요세푸스 문제를 해결하는 아래 코드를 완성하시오. (참조: https://en.wikipedia.org/wiki/Josephus_problem)

   A. 구현 방법 #1: 최초 1~n까지 수들을 큐에 삽입한 후, "큐 크기가 2이상인 동안, 큐에서 노드 하나를 삭제하고 삭제한 노드를 다시 큐에 삽입하는 작업을 k-1회 수행한 다음 큐에서 노드 하나를 삭제하는 작업"을 반복 수행한다. (참조: https://rosettacode.org/wiki/Josephus_problem)

   B. 구현 방법 #2: 1~n까지 수들의 연결리스트를 원형 큐로 고려하여 "큐 크기가 2이상인 동안, 직전 제거 위치부터 시작하여 k번째 위치 노드를 삭제하는 작업"을 반복 수행한다. (참조: https://rosettacode.org/wiki/Josephus_problem)

1 2 3 4 5 6 7 (3 제거)
1 2 4 5 6 7 (6 제거)
1 2 4 5 7 (2 제거)
1 4 5 7 (7 제거)
1 4 5 (5 제거)
1 4 (1 제거)
4

```
public class Test {
	public static void main(String[] args) {
		System.out.println(solveJosephusProblem(7,3));
	}
	private static int solveJosephusProblem(int n, int k) {
		return 0;
	}
}
```

5. (실습) 다음 코드는 길이 30의 문자열 s에 저장된 문자들을 5행 6열의 이차원배열 maze에 저장한 후 출력하는 코드이다. 이 코드를 완성하시오.

- 실행결과:

```
012345
678901
234567
890123
456789
```

```
public class Test {
	public static void main(String[] args) {
		int		R=5, C=6;
		String	s="012345678901234567890123456789";
		char		maze[][]=new char[R][C];


		for (int i = 0; i < maze.length; i++){
			for (int j = 0; j < maze[i].length; j++) System.out.print(maze[i][j]);
			System.out.println();
		}
	}
}
```

6. (BFS 기반 미로 완전 탐색) 다음은 큐를 이용하여 미로 내 이동 가능한 모든 위치를 방문하는 코드이다. 미로는 이차원배열 maze로 표현되며 이동 가능, 불가능 위치에 각각 '0', '#'의 값이 저장되어 있다. 미로 탐색의 시작 위치는 maze[0][0]으로 가정한다. 큐를 이용한 미로 완전 탐색 과정은 다음과 같다. (아래 코드는 "프로그래밍 콘테스트 챌린징, Akiba 등 공저, 로드북, 2011, p51~53"의 코드를 참조하여 작성되었음)

- Step-1: 시작 위치 (0,0)를 방문 표시 후(maze[0][0]에 'v'를 저장), 시작 위치를 공백 큐에 삽입한다.
- Step-2: 큐가 비어 있지 않은 동안, "큐에서 가장 오래 전에 방문했던 위치를 추출하여 그 위치의 상하좌우 위치들 중 이동가능한 미방문 위치들을 방문 표시 후 큐에 삽입"하는 절차를 반복한다.

```
public class Test {
	public static void main(String[] args) {
		class Point {
			int x,y;
			public Point(int x, int y) { this.x=x; this.y=y; }
		}
		int		R=6, C=8;
		String	input=
				"0000000#"+
				"##0#0#0#"+
				"0000000#"+
				"0#####0#"+
				"00000###"+
				"###00000";
		char	maze[][]=new char[R][C];
		for (int i = 0; i < input.length(); i++) maze[i/C][i%C]=input.charAt(i);
		LinkedList<Point>	queue=new LinkedList<>();
		maze[0][0]='v'; // visited
		queue.addLast(new Point(0,0));
		int		dx[]={0,0,1,-1}, dy[]={1,-1,0,0};
		while(!queue.isEmpty()){
			printMaze(maze);
			Point	p=queue.removeFirst();
			for (int i = 0; i < dx.length; i++) {
				int		x=p.x+dx[i], y=p.y+dy[i];
				if(x<0 || x>=R || y<0 || y>=C || maze[x][y]=='#' || maze[x][y]=='v') continue;
				maze[x][y]='v';
				queue.addLast(new Point(x,y));
			}
		}
	}
	private static void printMaze(char[][] maze) {
		for (int i = 0; i < maze.length; i++) System.out.println(maze[i]);
		System.out.println();
	}
}
```

7. (**실습**: 미로 이동 거리 계산) 아래 사각행렬은 시작 위치 (0,0)으로부터 해당 위치로 이동한 총 거리(시작 위치로의 이동 거리를 1로 가정)를 표시한 것이다. 미로 종료 위치를 (R-1,C-1)이라고 할 때, 아래 사각행렬은 미로의 시작위치로부터 종료위치까지의 최단거리가 17임을 보여준다. 이전 문제의 미로 탐색 코드를 수정하여 다음과 같은 사각행렬이 출력되도록 하시오.

```
01 02 03 04 05 06 07 00
00 00 04 00 06 00 08 00
07 06 05 06 07 08 09 00
08 00 00 00 00 00 10 00
09 10 11 12 13 00 00 00
00 00 00 13 14 15 16 17
```

8. (**실습**: 미로 최단 경로 탐색) 이전 문제의 미로 탐색 코드를 수정하여 아래와 같이 시작위치 (0,0)로부터 종료위치 (5,7)까지의 최단 경로를 추가 출력하도록 하시오.

```
01 02 03 04 05 06 07 00
00 00 04 00 06 00 08 00
07 06 05 06 07 08 09 00
08 00 00 00 00 00 10 00
09 10 11 12 13 00 00 00
00 00 00 13 14 15 16 17

(0,0)(0,1)(0,2)(1,2)(2,2)(2,1)(2,0)(3,0)(4,0)(4,1)(4,2)(4,3)(5,3)(5,4)(5,5)(5,6)(5,7)
```

9. (배열 기반 큐 클래스 구현) 아래 코드는 배열로 구현된 큐 클래스의 동작을 간단 구현한 코드이다.

- rear는 큐에 삽입된 가장 최근 자료의 위치이다. (초기값 -1)
- front는 큐에 삽입된 가장 오래된 자료 직전 위치이다. (초기값 -1)
- 큐에 자료 삽입(enqueue,add) 시 rear를 1 증가시킨 후 rear 위치에 자료를 저장한다.
- 큐에서 자료 추출(dequeue,remove) 시 front를 1 증가시킨 후 front 위치의 자료를 추출한다.
- 공백 큐 판단: rear와 front의 값이 같으면 큐가 비어 있는 상태이다.
- 포화 큐 판단: rear가 최대 큐 크기보다 1 적은 값이면 큐가 포화 상태이다.
- 아래 코드는 최초 설정된 큐의 크기를 초과하는 자료를 삽입하지 못하는 한계가 있다. 큐 포화 시 front가 -1을 초과하는 경우 여분의 공간 확보를 위해 배열 내 전체 자료를 왼쪽으로 이동시키고 front, rear 값을 재설정할 필요가 있다.

```
public class SimpleQueue {
	int		rear=-1, front=-1, MaxSize=5;
	char		queue[];
	public SimpleQueue() {
			queue=new char[MaxSize];
	}
	public void add(char data) {
			if(full()) throw new RuntimeException("queue full");
			queue[++rear]=data;
	}
	public int remove() {
			if(empty()) throw new RuntimeException("queue empty");
			return queue[++front];
	}
	private boolean full() { return rear==MaxSize-1; }
	private boolean empty() { return rear==front; }
	@Override
	public String toString() {
			return "front="+front+", rear="+rear+", "+Arrays.toString(queue);
	}
}
  public class Test {
	public static void main(String[] args) {
			SimpleQueue		queue=new SimpleQueue();
			System.out.println("크기 5 큐 초기 상태: "+queue);
			queue.add('A');		System.out.println("자료 A 삽입 후: "+queue);
			queue.add('B');		System.out.println("자료 B 삽입 후: "+queue);
			queue.add('C');		System.out.println("자료 C 삽입 후: "+queue);
			System.out.println("큐에서 자료 추출: "+(char)queue.remove());
			System.out.println("큐에서 자료 추출: "+(char)queue.remove());
			System.out.println("큐에서 자료 추출: "+(char)queue.remove());
			queue.add('D');		System.out.println("자료 D 삽입 후: "+queue);
			queue.add('E');		System.out.println("자료 E 삽입 후: "+queue);
	}
}
```

10. (**실습**) 위 코드를 수정하여, 아래 실행 예와 같이 포화 큐에 새로운 자료 삽입 시 큐의 크기를 두 배 확장한 후 배열 내 전체 자료를 왼쪽으로 이동시키고 새로운 자료를 삽입하도록 하시오.

```
크기 1 큐 초기 상태: front=-1, rear=-1, [ ]
자료 A 삽입 후: front=-1, rear=0, [A]
자료 B 삽입 후: front=-1, rear=1, [A, B]
자료 C 삽입 후: front=-1, rear=2, [A, B, C,    ]
큐에서 자료 추출: A
큐에서 자료 추출: B
큐에서 자료 추출: C
자료 D 삽입 후: front=2, rear=3, [A, B, C, D]
자료 E 삽입 후: front=-1, rear=1, [D, E, C, D,    ,    ,    ,    ]
```

```
public class Test {
	public static void main(String[] args) {
		SimpleQueue	queue=new SimpleQueue();
		System.out.println("크기 1 큐 초기 상태: "+queue);
		queue.add('A');	System.out.println("자료 A 삽입 후: "+queue);
		queue.add('B');	System.out.println("자료 B 삽입 후: "+queue);
		queue.add('C');	System.out.println("자료 C 삽입 후: "+queue);

		System.out.println("큐에서 자료 추출: "+(char)queue.remove());
		System.out.println("큐에서 자료 추출: "+(char)queue.remove());
		System.out.println("큐에서 자료 추출: "+(char)queue.remove());
		queue.add('D');	System.out.println("자료 D 삽입 후: "+queue);
		queue.add('E');	System.out.println("자료 E 삽입 후: "+queue);
	}
}
```

References

- C로 쓴 자료구조론 (Fundamentals of Data Structures in C, Horowitz et al.). 이석호 역. 사이텍 미디어. 1993.
- 쉽게 배우는 알고리즘: 관계 중심의 사고법. 문병로. 한빛아카데미. 2013.
- C언어로 쉽게 풀어 쓴 자료구조. 천인국 외 2인. 생능출판사. 2017.
- 김윤명. (2008). 뇌를 자극하는 Java 프로그래밍. 한빛미디어.
- 남궁성. 자바의 정석. 도우출판.
- 김윤명. (2010). 뇌를 자극하는 JSP & Servlet. 한빛미디어.