1. (**실습**) 스택은 자료의 삽입(push)과 삭제(pop)가 한쪽 끝(top)에서만 발생한다. 다음은 LinkedList 클래스의 addFirst, removeFirst, getFirst 메소드를 스택의 push, pop, peek으로 사용한 예시 코드이다. 스택이 비어 있는지 검사하기 위해 LinkedList의 isEmpty 메소드를 사용하였으며, 스택 크기 확인을 위해 LinkedList의 size 메소드를 사용하였다. 아래 코드를 입력하고 실행하면서 스택 및 LinkedList의 사용을 익히시오.

   - **실습**: LinkedList의 addFirst, removeFirst, getFirst 대신 addLast, removeLast, getLast 메소드를 사용하여 아래 코드를 재실행해 보고 어떤 차이가 있는지 확인하시오.

```
public class Test {
	public static void main(String[] args) {
		LinkedList<Integer>	stack=new LinkedList<>(); //  이중연결리스트 구현
		stack.addFirst(1); // list.push(1)
		stack.addFirst(2); // list.push(2)
		stack.addFirst(3); // list.push(3)
		System.out.println("스택 전체 내용: "+stack);
		System.out.println("스택 크기="+stack.size());
		System.out.println("스택이 비어 있는가? "+stack.isEmpty());
		int		v;
		v=stack.getFirst();
		System.out.println("스택 top 위치 자료 확인="+v);
		v=stack.removeFirst(); // list.pop()
		System.out.println("스택 top 위치 자료 제거="+v);
		System.out.println("스택 전체 내용: "+stack);
		System.out.println("스택 top 위치 자료 제거="+stack.pop());
		System.out.println("스택 전체 내용: "+stack);
		System.out.println("스택 top 위치 자료 제거="+stack.pop());
		System.out.println("스택 전체 내용: "+stack);
		System.out.println("스택이 비어 있는가? "+stack.isEmpty());
	}
}
```

2. (스택 동작 예시) 다음은 Stack 클래스를 사용하여 스택의 동작을 예시한 코드이다. 아래 코드를 입력하고 실행해 보면서 스택 및 Stack 클래스의 동작을 이해하도록 하시오.

```
public class Test {
	public static void main(String[] args) {
		Stack<String>		stack=new Stack<>();
		stack.push("월");
		stack.push("화");
		stack.push("수");
		System.out.println("스택 전체 내용: "+stack);
		System.out.println("스택 크기="+stack.size());
		System.out.println("스택이 비어 있는가? "+stack.isEmpty());
		String	v;
		v=stack.peek(); // 스택의 top에 있는 자료를 제거하지 않고 그 값을 반환
		System.out.println("스택 top 위치 자료 확인="+v);
		v=stack.pop(); // 스택의 top에 있는 자료를 제거한 후 그 값을 반환
		System.out.println("스택 top 위치 자료 제거="+v);
		System.out.println("스택 전체 내용: "+stack);
		System.out.println("스택 top 위치 자료 제거="+stack.pop());
		System.out.println("스택 전체 내용: "+stack);
		System.out.println("스택 top 위치 자료 제거="+stack.pop());
		System.out.println("스택 전체 내용: "+stack);
		System.out.println("스택이 비어 있는가? "+stack.isEmpty());
	}
}
```

3. (**실습**: 스택 기반 문자열 역순 출력) 다음은 자바 클래스 Stack을 이용하여 문자열 s를 역순 출력하는 코드이다. 이 코드를 완성하시오.

- 실행결과: .다이울서 는도수 의국민한대

```
import java.util.Stack;

public class Test {
	public static void main(String[] args) {
			String	s="대한민국의 수도는 서울이다.";
			Stack<Character> stack=new Stack<>();
			for (int i = 0; i < s.length(); i++) {
				char	c=s.charAt(i);


			}


	}
}
```

4. (**실습**: 스택 기반 문자열 역순 출력) 다음은 자바 클래스 LinkedList를 스택으로 이용하여 문자열 s를 역순 출력하는 코드이다. 이 코드를 완성하시오. (addFirst, removeFirst를 push, pop으로 활용, 혹은 addLast, removeLast를 push, pop으로 활용)

- 실행결과: .다이울서 는도수 의국민한대

```
public class Test {
	public static void main(String[] args) {
			String	s="대한민국의 수도는 서울이다.";
			LinkedList<Character>		stack=new LinkedList<>();


	}
}
```

5. (**실습**: 스택 활용 괄호쌍매칭판단) 다음은 문자열 s 내 괄호쌍매칭여부를 스택을 이용하여 판단한 후 그 결과를 true, false로 반환하는 코드이다. 이 코드를 완성하시오. 괄호 문자로는 '('와 ')'만 다루시오.

```
public class Test {
	public static void main(String[] args) {
		String	s="(1+3*(4+5))/(6*(7+8))";
		System.out.println(checkParen(s));
	}
	private static boolean checkParen(String s) {
		Stack<Character>	stack=new Stack<>();
		for (Character c : s.toCharArray()){



		}


	}
}
```

6. (**실습**: 스택 활용 괄호쌍매칭판단) 다음은 문자열 s 내 괄호쌍매칭여부를 스택을 이용하여 판단한 후 그 결과를 true, false로 반환하는 코드이다. 이 코드를 완성하시오. 괄호 문자로는 (,),{,},[,]를 모두 처리할 수 있도록 하시오.

```
public class Test {
	public static void main(String[] args) {
		String	s="부산({Busan}[(Boo)(saan)])시청({City}{hall}[(See)(Cheong)])"; // true
		System.out.println(checkParen(s));
	}
	private static boolean checkParen(String s) {
		Stack<Character>	stack=new Stack<>();




	}
}
```

7. (후위표현식 계산 과정 예시) 다음은 스택을 이용한 후위표현식 **3 4 5 + ***의 계산 과정을 예시한 코드이다. 아래 코드를 입력하고 실행해 보시오.

```
public class Test {
	public static void main(String[] args) {
		int		n1, n2;
		Stack<Integer>		stack=new Stack<>();
		stack.push(3); //  피연산자  3 => push
		stack.push(4); //  피연산자  4 => push
		stack.push(5); //  피연산자  5 => push
		//  연산자  +
		n2=stack.pop();
		n1=stack.pop();
		stack.push(n1+n2);
		//  연산자  *
		n2=stack.pop();
		n1=stack.pop();
		stack.push(n1*n2);
		System.out.println("계산결과="+stack.pop());
	}
}
```

8. (StringTokenizer 클래스) 다음는 StringTokenizer 클래스를 사용하여 문자열 e 내의 토큰들을 :, -, ' ' 문자들을 기준으로 분리하여 출력하는 코드이다. 아래 코드를 입력하고 실행하여 StringTokenizer 클래스의 사용을 익히시오.

```
public class Test {
	public static void main(String[] args) {
		String		e="2018-09-17 17:23:49";
		StringTokenizer		stok=new StringTokenizer(e, "- :");
		//StringTokenizer		stok=new StringTokenizer(e, "- :", true);
		while (stok.hasMoreTokens()) {
			System.out.println(stok.nextToken());
		}
	}
}
```

9. (String 클래스의 split 메소드) 다음는 String 클래스의 split 메소드를 사용하여 문자열 e 내의 토큰들을 whitespace 문자열을 기준으로 분리하여 출력하는 코드이다. 아래 코드를 입력하고 실행하여 split 메소드의 사용을 익히시오. whitespace 문자에는 일반적으로 ' ', '₩t', '₩r', '₩n' 이 포함된다.

```
public class Test {
	public static void main(String[] args) {
		String		e="3 1 2 / *";
		String		v[]=e.split("\\s+");
		System.out.println(Arrays.toString(v));
	}
}
```

10. (**실습**: 스택 활용 후위표현식 계산기) 다음은 문자열 e에 저장된 후위표현식의 계산 결과를 출력하는 코드이다. 후위표현식에서 연산자는 +,-,*,/만 다루고, 각 토큰(연산자 및 피연산자)들은 하나 이상의 공백 문자로 분리되어 있으며, 주어진 후위표현식에는 오류가 없으며 가정하시오. 이 코드를 완성하시오.

```
public class Test {
	public static void main(String[] args) {
		String	e="3 1 2 / *";
		System.out.println(postfixEval(e));
	}
	private static double postfixEval(String e) {
		Stack<Double>		stack=new Stack<>();
		for (String token : e.split("\\s+")) {



		}
		return stack.pop();
	}
}
```

11. (**실습**: 스택 활용 후위표현식 계산기) 다음은 문자열 e에 저장된 후위표현식의 계산 결과를 출력하는 코드이다. 후위표현식에서 연산자는 +,-,*,/만 다루고, 각 피연산자들은 하나 이상의 공백 문자로 분리되어 있으며(피연산자와 연산자 사이 및 연산자들 사이에는 공백이 없을 수 있음에 주의할 것), 주어진 후위표현식에는 오류가 없으며 가정하시오. 이 코드를 완성하시오.

```
public class Test {
	public static void main(String[] args) {
		String	e="34 12 25/*";
		System.out.println(postfixEval(e));
	}
	private static double postfixEval(String e) {
		Stack<Double>		stack=new Stack<>();



	}
}
```

12. (스택 개념: 배열 구현) 아래 코드는 스택을 배열로 간단 구현한 코드이다.

- top은 스택에 마지막으로 삽입된 자료의 위치 값을 보관한다. (초기값 -1)
- 스택에 새로운 자료 삽입(push) 시, top을 1 증가시킨 후 top 위치에 자료를 저장한다.
- 스택에서 자료 추출(pop) 시, top 위치의 자료를 추출하고, top을 1 감소시킨다.

아래 코드로는 스택 사용 상의 제약이 있다. (공백 스택 및 포화 스택 판단 필요)

- 공백 스택: top의 값이 -1이면 공백 스택이다.
- 포화 스택: top의 값이 스택 최대 크기보다 1 적은 값이면 포화 스택이다.

```
public class Test {
	public static void main(String[] args) {
		int stack[] = new int[5]; // 크기 5의 스택 생성
		int top = -1;
		// 스택에 5, 9, 1을 차례로 삽입(push)
		stack[++top] = 5;
		stack[++top] = 9;
		stack[++top] = 1;
		System.out.println(Arrays.toString(stack)+", top="+top+", 스택 크기="+(top+1));
		System.out.println("스택이 비어 있는가? "+(top==-1));
		System.out.println("스택 peek: "+stack[top]); // 스택 top 확인 peek
		System.out.println("스택 pop: "+stack[top--]); // 스택에서 자료 추출(pop)
		System.out.println(Arrays.toString(stack)+", top="+top+", 스택 크기="+(top+1));
		System.out.println("스택 pop: "+stack[top--]); // 스택에서 자료 추출(pop)
		System.out.println(Arrays.toString(stack)+", top="+top+", 스택 크기="+(top+1));
		System.out.println("스택 pop: "+stack[top--]); // 스택에서 자료 추출(pop)
		System.out.println(Arrays.toString(stack)+", top="+top+", 스택 크기="+(top+1));
		System.out.println("스택이 비어 있는가? "+(top==-1));
	}
}
```

13. (**실습**: 배열 스택 기반 문자열 역순 출력) 다음은 문자열 s를 역순 출력하는 코드이다. Stack, LinkedList 클래스를 사용하지 말고 이전 문제에서 예시한 배열 기반 스택을 사용하여 아래 코드를 완성하시오.

- 실행결과: .다이울서 는도수 의국민한대

```
public class Test {
	public static void main(String[] args) {
		String	s="대한민국의 수도는 서울이다.";
		char	stack[] = new char[s.length()];
		int top = -1;

	}
}
```

14. (스택 개념: 배열 구현) 아래 코드는 스택 클래스를 배열로 간단 구현한 코드이다. 이대로는 사용 상의 제약이 있다.

- 스택에 자료 삽입 시 스택의 가용 공간 유무에 대한 검사 부재
- 스택에서 자료 추출 시 스택이 비어 있는 상태인지에 대한 검사 부재

- **실습**: 아래 코드에 isEmpty(), isFull(), peek() 메소드를 추가해 보시오.

```
public class SimpleStack {
	int		stack[];
	int		top=-1;
	public SimpleStack(int size) {
			stack=new int[size];
	}
	public void push(int data) {
			stack[++top]=data;
	}
	public int pop() {
			return stack[top--];
	}
	@Override
	public String toString() {
			return "top="+top+", stack="+Arrays.toString(stack);
	}
}

public class Test {
	public static void main(String[] args) {
			SimpleStack		stack=new SimpleStack(10); // 크기 10의 Stack 생성

			// 스택에 5 9 1 순차 삽입
			stack.push(5);
			stack.push(9);
			stack.push(1);

			System.out.println(stack); // 스택 출력

			int		data=stack.pop(); // 스택에서 자료 추출
			System.out.println("Data deleted from stack:"+data);

			System.out.println(stack); // 스택 출력
	}
}
```

15. (배열 크기 변경) 다음은 배열을 확장 및 축소하는 예시 코드이다. 아래 코드를 입력하고 실행하면서 가변 배열 처리를 익히시오.

- 실행결과:

```
[1, 2, 3, 4]
[1, 2, 3, 4, 0, 0, 0, 0]
[1, 2]
```

```
public class Test {
	public static void main(String[] args) {
		int	n[]={1,2,3,4};
		System.out.println(Arrays.toString(n));
		int	m[]=Arrays.copyOf(n, n.length*2); // 2배 확장
		n=m;
		System.out.println(Arrays.toString(n));
		n=Arrays.copyOf(n, (int) (n.length*0.25)); // 1/4로 축소
		System.out.println(Arrays.toString(n));
	}
}
```

16. (**실습**: 배열 크기 변경) 다음은 배열을 확장, 축소하는 코드이다. arrayCopy는 int 배열에 대해 이전 문제의 Arrays.copyOf() 함수와 같은 동작을 수행하는 함수이다. 아래 코드를 완성하시오.

- 실행결과:

```
[1, 2, 3, 4]
[1, 2, 3, 4, 0, 0, 0, 0]
[1, 2]
```

```
public class Test {
	public static void main(String[] args) {
		int	n[]={1,2,3,4};
		System.out.println(Arrays.toString(n));
		int	m[]=arrayCopy(n, n.length*2); // 2배 확장
		n=m;
		System.out.println(Arrays.toString(n));
		n=arrayCopy(n, (int) (n.length*0.25)); // 1/4로 축소
		System.out.println(Arrays.toString(n));
	}

	private static int[] arrayCopy(int[] n, int newSize) {




	}
}
```

17. (**실습**: 동적 배열 기반 스택 클래스 구현) 아래 코드는 스택 클래스를 동적 배열로 구현한 코드이다. 스택 full인 경우 배열의 크기를 2배로 증가시킨다. 스택 포화를 검사하는 full() 메소드와 스택이 비어 있는지 검사하는 empty() 메소드가 구현되어 있다.

- **실습**: SimpleStack 클래스에 peek()을 구현하시오. peek()은 pop()과 같으나 top의 위치를 변경하지 않는다.

```
public class SimpleStack {
	int		DefaultSize=1;
	int		MaxSize;
	int		stack[];
	int		top=-1;
	public SimpleStack() {
			stack=new int[DefaultSize];
			MaxSize=DefaultSize;
	}
	public void push(int data) {
			if(full()){
					MaxSize*=2;
					stack=Arrays.copyOf(stack, MaxSize);
			}
			stack[++top]=data;
	}
	private boolean full() {
			return top==MaxSize-1;
	}
	@Override
	public String toString() {
			return "top="+top+", stack="+Arrays.toString(stack);
	}
	public int pop() {
			if(empty()) throw new RuntimeException("stack empty");
			return stack[top--];
	}
	private boolean empty() {
			return top==-1;
	}
}
public class Test {
	public static void main(String[] args) {
			SimpleStack		stack=new SimpleStack();

			for (int i = 0; i < 10; i++){
					stack.push(i);
					System.out.println(stack); // 스택 출력
			}
			for (int i = 0; i < 10; i++){
					int		data=stack.pop(); // 스택에서 자료 추출
					System.out.println("Data deleted from stack:"+data);
					System.out.println(stack); // 스택 출력
			}
	}
}
```

18. (Generic 스택 클래스 구현: 동적 배열 기반) 아래 코드는 Generic 스택 클래스를 동적 배열로 구현한 코드이다.

```
public class SimpleStack<T> {
	int		DefaultSize=1;
	int		MaxSize;
	Object	stack[];
	int		top=-1;
	public SimpleStack() {
			stack=new Object[DefaultSize];
			MaxSize=DefaultSize;
	}
	public void push(Object data) {
			if(full()){
				MaxSize*=2;
				stack=Arrays.copyOf(stack, MaxSize);
			}
			stack[++top]=data;
	}
	private boolean full() {
			return top==MaxSize-1;
	}
	@Override
	public String toString() {
			return "top="+top+", stack="+Arrays.toString(stack);
	}
	@SuppressWarnings("unchecked")
	public T pop() {
			if(empty()) throw new RuntimeException("stack empty");
			return (T) stack[top--];
	}
	private boolean empty() {
			return top==-1;
	}
}
public class Test {
	public static void main(String[] args) {
			SimpleStack<Integer>		stack1=new SimpleStack<>();
			stack1.push(1);
			stack1.push(2);
			stack1.push(3);
			System.out.println(stack1);

			SimpleStack<String>  stack2=new SimpleStack<>();
			stack2.push("K");
			stack2.push("J");
			stack2.push("C");
			System.out.println(stack2);
	}
}
```

19. (**실습**: stock span problem) n개 일별 주가(stock price)에 대해 특정 day의 span은 현재 day의 주가보다 크지 않은 연속된 이전 day들(현재 day 포함)의 최대 개수이다. stack span problem은 n개 일별 주가에 대해 각 day의 span을 계산하는 것이다(아래 예 참조). stock span problem을 해결하는 아래 코드를 완성하시오. (참고: https://en.wikibooks.org/wiki/Data_Structures/Stacks_and_Queues#The_Stock_Span_Problem)

A. 구현 방법 #1 (시간복잡도 $O(n^2)$): n개 일별 주가가 배열에 저장되어 있다고 가정할 때 배열 인덱스를 day 식별자로 고려하여, 각 day에 대해 현재 day의 주가보다 크지 않은 주가를 보인 연속된 이전 day들의 수(현재 day 포함)를 현재 day의 span에 저장한다.

B. 구현 방법 #2 (시간복잡도 O(n)): n개 일별 주가가 배열에 저장되어 있다고 가정할 때 배열 인덱스를 day 식별자로 고려하여, 시작 day부터 "각 day에 대해 현재 day의 주가보다 큰 주가를 보이는 day가 스택의 top에서 발견될 때까지 스택을 pop한 후, 현재 day와 스택의 top에 있는 day와의 일수 차이를 현재 day의 span에 저장한 다음, 현재 day를 스택에 push하는 작업"을 반복한다.

- 일별 주가:
100,90,80,70,85,95,110,120

- stock span:
1, 1, 1, 1, 3, 5, 7, 8

```
public class Test {
	public static void main(String[] args) {
			int		price[]= {100,90,80,70,85,95,110,120};
			int		span[]=new int[price.length];



			System.out.println(Arrays.toString(span));
	}
}
```

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.