1. 다음은 클래스 Node의 정의문과 Node 객체 생성 예를 보인 것이다. 아래에 보인 클래스 Node는 이 자료의 이후 코드들에서 반복 사용된다.

```
public class Node {
	int		data;
	Node	next;
	public Node(int data) {
		this.data=data;
	}
}
public class Test {
	public static void main(String[] args) {
		Node	node=new Node(99);
		System.out.println(node.data); // 99  출력
		System.out.println(node.next); // null  출력
	}
}
```

2. (연결리스트 출력) 다음은 head->1->2->3의 연결리스트를 생성한 후 출력하는 코드이다. 아래 코드를 입력하고 실행하면서 그 내용을 학습하시오.

```
public class Test {
	public static void main(String[] args) {
		Node	head=null;
		Node	n1=new Node(1);
		Node	n2=new Node(2);
		Node	n3=new Node(3);
		head=n1;
		n1.next=n2;
		n2.next=n3;
		for (Node p=head; p != null; p=p.next) System.out.print(p.data+"->");
	}
}
```

3. (연결리스트 첫 위치 삽입) 다음은 연결리스트의 첫 위치에 7을 삽입하는 코드이다. 연결리스트가 비어 있는 경우(head가 null인 경우)에도 동작한다. 아래 코드를 입력하고 실행하면서 그 내용을 학습하시오.

- 삽입 전: head->1->2->3
- 삽입 후: head->7->1->2->3

```
public class Test {
	public static void main(String[] args) {
			Node	head=null;
			head=new Node(1);
			head.next=new Node(2);
			head.next.next=new Node(3);
			printList(head);

			Node	newNode=new Node(7);
			newNode.next=head;
			head=newNode;
			printList(head);
	}
	private static void printList(Node head) {
			for (Node p=head; p != null; p=p.next) System.out.print(p.data+"->");
			System.out.println();
	}
}
```

4. (연결리스트 끝 위치 삽입) 다음은 연결리스트의 끝 위치에 7을 삽입하는 코드이다. 연결리스트가 비어 있는 경우에도 동작한다. 아래 코드를 입력하고 실행하면서 그 내용을 학습하시오.

- 삽입 전: head->1->2->3
- 삽입 후: head->1->2->3->7

```
public class Test {
	public static void main(String[] args) {
		Node	head=null;
		head=new Node(1);
		head.next=new Node(2);
		head.next.next=new Node(3);
		printList(head);

		Node	newNode=new Node(7);
		if(head!=null){
			Node	p=head;
			while(p.next!=null) p=p.next;
			p.next=newNode;
		}
		else head=newNode; // 연결리스트가 비어 있는 경우
		printList(head);
	}
	private static void printList(Node head) {
		for (Node p=head; p != null; p=p.next) System.out.print(p.data+"->");
		System.out.println();
	}
}
```

5. (연결리스트 내 특정 자료 위치 삽입) 다음은 연결리스트 내 특정 자료 위치에 7을 삽입하는 코드이다. 삽입 위치 자료 값에 해당하는 searchValue 값을 1,2,9 등으로 변경하면서 실행해 보시오. 연결리스트가 비어 있는 경우나 삽입 위치에 해당하는 자료 값이 존재하지 않는 경우 "does not exist" 오류 메시지가 출력된다. 아래 코드를 입력하고 실행하면서 그 내용을 학습하시오.

- 자료 3 위치 삽입 전: head->1->2->3
- 자료 3 위치 삽입 후: head->1->2->7->3

```
public class Test {
	public static void main(String[] args) {
		Node	head=null;
		head=new Node(1);
		head.next=new Node(2);
		head.next.next=new Node(3);
		printList(head);

		int	searchValue=3; // 1,2,9 등으로 변경하면서 실행해 볼 것.

		Node	newNode=new Node(7);
		Node	p=head, prev=null;
		while(p!=null && p.data!=searchValue){ prev=p; p=p.next; }
		if(p==null) System.out.println(searchValue+" does not exist");
		else{
			if(prev!=null) prev.next=newNode;
			else	head=newNode;
			newNode.next=p;
		}
		printList(head);
	}
	private static void printList(Node head) {
		for (Node p=head; p != null; p=p.next) System.out.print(p.data+"->");
		System.out.println();
	}
}
```

6. (연결리스트 첫 위치 자료 삭제) 다음은 연결리스트 내 첫 위치 자료를 삭제하는 코드이다. 아래 코드를 입력하고 실행하면서 그 내용을 학습하시오.

- 삭제 전: head->1->2->3
- 삭제 후: head->2->3

```
public class Test {
	public static void main(String[] args) {
		Node	head=null;
		head=new Node(1);
		head.next=new Node(2);
		head.next.next=new Node(3);
		printList(head);

		if(head!=null){
			Node	p=head;
			head=head.next;
			p.next=null;
		}
		printList(head);
	}
	private static void printList(Node head) {
		for (Node p=head; p != null; p=p.next) System.out.print(p.data+"->");
		System.out.println();
	}
}
```

7. (연결리스트 끝 위치 자료 삭제) 다음은 연결리스트 내 끝 위치 자료를 삭제하는 코드이다. 아래 코드를 입력하고 실행하면서 그 내용을 학습하시오.

- 삭제 전: head->1->2->3
- 삭제 후: head->1->2

```
public class Test {
	public static void main(String[] args) {
		Node	head=null;
		head=new Node(1);
		head.next=new Node(2);
		head.next.next=new Node(3);
		printList(head);

		if(head!=null){
			Node	p=head, prev=null;
			while(p.next!=null){ prev=p; p=p.next; } // p를 마지막 노드로, prev를 마지막 직전 노드로 이동
			if(prev!=null) prev.next=null; // 리스트 크기가 2개 이상인 경우
			else head=null; // 리스트 크기가 1인 경우
		}

		printList(head);
	}
	private static void printList(Node head) {
		for (Node p=head; p != null; p=p.next) System.out.print(p.data+"->");
		System.out.println();
	}
}
```

8. (연결리스트 내 특정 자료 삭제) 다음은 연결리스트 내 특정 자료를 삭제하는 코드이다. 삭제할 자료 값에 해당하는 searchValue 값을 1,3,9 등으로 변경하면서 실행해 보시오. 연결리스트가 비어 있는 경우나 삭제할 자료가 존재하지 않는 경우 "does not exist" 오류 메시지가 출력된다. 아래 코드를 입력하고 실행하면서 그 내용을 학습하시오.

- 삭제 전: head->1->2->3
- 삭제 후: head->1->3

```
public class Test {
	public static void main(String[] args) {
		Node	head=null;
		head=new Node(1);
		head.next=new Node(2);
		head.next.next=new Node(3);
		printList(head);

		int	searchValue=2; // 1,2,9 등으로 변경하면서 실행해 볼 것.

		Node	p=head, prev=null;
		while(p!=null && p.data!=searchValue){ prev=p; p=p.next; }
		if(p==null) System.out.println(searchValue+" does not exist");
		else{
			if(prev!=null) prev.next=p.next;
			else	head=p.next;
			p.next=null;
		}
		printList(head);
	}
	private static void printList(Node head) {
		for (Node p=head; p != null; p=p.next) System.out.print(p.data+"->");
		System.out.println();
	}
}
```

9. (클래스 기반 연결리스트 구현) 다음은 연결리스트 클래스 SimpleList를 구현한 코드이다. SimpleList의 addFirst(int data) 메소드는 연결리스트의 첫 위치에 새로운 자료를 삽입하는 메소드이다. 아래 코드를 입력하고 실행하면서 그 내용을 학습하시오.

```
public class Node {
	int		data;
	Node	next;
	public Node(int data) {
		this.data=data;
	}
}

public class SimpleList {
	Node	head;
	public void addFirst(int data) {
		Node	newNode=new Node(data);
		newNode.next=head;
		head=newNode;
	}
	@Override
	public String toString() {
		String	v="";
		for (Node p=head; p!=null; p=p.next){
			if(v.length()>0) v+="->";
			v+=p.data;
		}
		return v;
	}
}

public class Test {
	public static void main(String[] args) {
		SimpleList list=new SimpleList();
		list.addFirst(1);
		list.addFirst(2);
		list.addFirst(3);
		System.out.println(list);
	}
}
```

10. (Generic 클래스 기반 연결리스트 구현) 다음은 generic 클래스 기반의 연결리스트를 구현한 코드이다. 아래 코드를 입력하고 실행하면서 그 내용을 학습하시오.

```
public class Node<T> {
	T		data;
	Node<T>	next;
	public Node(T data) {
		this.data=data;
	}
}
public class SimpleList<T> {
	Node<T> head;

	public void addFirst(T data) {
		Node<T> newNode = new Node<>(data);
		newNode.next = head;
		head = newNode;
	}

	@Override
	public String toString() {
		String v = "";
		for (Node<T> p = head; p != null; p = p.next) {
			if (v.length() > 0)
				v += "->";
			v += p.data;
		}
		return v;
	}
}
public class Test {
	public static void main(String[] args) {
			SimpleList<Integer>		listA=new SimpleList<>();
			listA.addFirst(1);
			listA.addFirst(2);
			listA.addFirst(3);
			System.out.println(listA);

			SimpleList<String>		listB=new SimpleList<>();
			listB.addFirst("Korea");
			listB.addFirst("Japan");
			listB.addFirst("China");
			System.out.println(listB);

			SimpleList<Boolean>		listC=new SimpleList<>();
			listC.addFirst(true);
			listC.addFirst(true);
			listC.addFirst(false);
			System.out.println(listC);

	}
}
```

11. (**실습**: removeFirst()) 아래 SimpleList의 removeFirst()는 연결리스트의 0번째 자료를 삭제하는 메소드이다. 아래 코드가 정상 실행되도록 removeFirst()를 완성하시오.

- removeFirst 실행 전 연결리스트: 4->3->2->1->0
- removeFirst 실행 후 연결리스트: 3->2->1->0

```
public class SimpleList {
	Node	head;
	public void addFirst(int data) {
		Node	newNode=new Node(data);
		newNode.next=head;
		head=newNode;
	}
	public void removeFirst() {




	}
	@Override
	public String toString() { ... } // 이전과 동일
}
public class Test {
	public static void main(String[] args) {
		SimpleList list=new SimpleList();
		for (int i=1; i <=5; i++) list.addFirst(i);
		System.out.println(list);
		list.removeFirst();
		System.out.println(list);
	}
}
```

12. (**실습**: size(), get()) 아래 SimpleList의 size()는 연결리스트에 저장된 자료의 전체 개수를 반환하는 메소드이다. 또한 SimpleList의 get(int index)는 연결리스트의 index번째 노드의 data 값을 반환하는 메소드이다. 아래 Test 클래스는 연결리스트 4->3->2->1->0를 생성한 후 get() 메소드를 통해 각 노드의 자료를 가져와 출력하는 코드이다. 아래 코드가 정상 동작하도록 SimpleList를 수정하시오.

- 아래 코드 실행결과: 4 3 2 1 0

```
public class SimpleList {
	Node	head;
	public void addFirst(int data) {
		Node	newNode=new Node(data);
		newNode.next=head;
		head=newNode;
	}
	public int size() {




	}
	public String get(int i) {




	}
	@Override
	public String toString() { ... } // 이전과 동일
}
public class Test {
	public static void main(String[] args) {
		SimpleList list=new SimpleList();
		for (int i = 0; i < 5; i++) list.addFirst(i);
		for (int i = 0; i < list.size(); i++) System.out.print(list.get(i)+" ");
	}
}
```

13. (**실습**: addLast()) 아래 코드가 연결리스트 1->2->3을 생성하도록 다음 두 가지 구현 방법 각각에 대해 아래 SimpleList를 수정하시오. addLast(int data)는 연결리스트의 끝 위치에 새로운 자료 data를 추가하는 메소드이다. 두 구현 방법의 시간 복잡도는 각각 어떻게 되는가?

- 구현 방법 1: tail 필드를 추가하지 않고, head 필드만 사용하여 구현
- 구현 방법 2: tail 필드를 추가하고, tail 필드를 사용하여 구현

```
public class SimpleList {
	Node	head;
	public void addLast(int i) {




	}
	@Override
	public String toString() { ... } // 이전과 동일
}
public class Test {
	public static void main(String[] args) {
			SimpleList list=new SimpleList();
			list.addLast(1);
			list.addLast(2);
			list.addLast(3);
			System.out.println(list);
	}
}
```

14. (**실습**: removeLast()) SimpleList의 removeLast는 연결리스트의 끝 위치 자료를 삭제하는 메소드이다. 아래 코드가 정상 동작하도록 SimpleList에 removeLast를 구현하시오. 지금까지 구현한 SimpleList와 같은 단일연결리스트에서 구현된 removeLast는 어떤 단점이 있는가?

```
public class SimpleList {
	Node	head;
	public void addLast(int i) {




	}
	public void removeLast() {




	}
	@Override
	public String toString() { ... } //  이전과  동일
}
public class Test {
	public static void main(String[] args) {
			SimpleList list=new SimpleList();
			for (int i = 0; i < 5; i++) list.addLast(i);
			System.out.println(list);
			list.removeLast();
			System.out.println(list);
	}
}
```

15. (이중연결리스트) 다음은 이중연결리스트 head->77<->99<-tail을 생성하는 코드이다. next, prev 링크는 각각 순방향, 역방향 연결을 구성한다. 아래 코드를 입력하고 실행하면서 그 내용을 학습하시오.

```
public class Test {
	public static void main(String[] args) {
		class Node {
			int		data;
			Node		prev, next;
			public Node(int data) {
				this.data=data;
			}
		}
		Node		n1=new Node(1);
		Node		n2=new Node(2);
		Node		head=n1;
		Node		tail=n2;
		n1.next=n2;
		n2.prev=n1;
		System.out.print("head");
		for (Node p=head; p != null; p=p.next) System.out.print("->"+p.data);
		System.out.print("\ntail");
		for (Node p=tail; p != null; p=p.prev) System.out.print("->"+p.data);
	}
}
```

16. (**실습**) 다음과 같이 동작하는 SimpleList를 이중연결리스트로 구현하시오.

- 실행결과:

```
2->1->9->8
1->9
```

```
public class Test {
	public static void main(String[] args) {
			SimpleList list=new SimpleList();
			list.addFirst(1);
			list.addFirst(2);
			list.addLast(9);
			list.addLast(8);
			System.out.println(list);
			list.removeFirst();
			list.removeLast();
			System.out.println(list);
	}
}
```

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.