React useReducer로 복잡한 상태 관리하기

**React**에서는 다음 상태가 이전 상태에 의존하거나 여러 상태를 관리해야할 경우에는 **useReducer** 사용을 권장하고 있습니다.

**useReducer**는 새로운 상태 값을 만들어 낼 때, 기존 상태 값이 필요한 경우와 여러 상태를 관리할 때 유용합니다. 하지만 복잡한 상태를 다뤄보지 않은 경우 **useReducer**를 이용해 상태를 변경하는 것이 생소하고 불편하게 느껴질 수 있습니다.

따라서 본 실습을 통해 **useReducer**를 사용하면 복잡한 상태관리가 어떻게 쉬워지는지 느껴봅시다.

useReducer는 다음과 같이 사용합니다.

const [state, dispatch] = useReducer(reducer, initialArg, init);

Copy

useState와 생김새가 비슷합니다. 현재 상태를 리턴하고 또, Setter 함수가 아닌 **dispatch**라는 함수를 리턴하는 것에 주목해봅시다.

dispatch({ type: "Increase", payload: 10});

Copy

**dispatch** 함수 안에는 action 객체를 전달합니다. **type**과 **payload**를 넣어서 상태 변경을 요청하는 형태입니다. 상태를 관리하는 일종의 관행이라고 생각하시면 됩니다. **payload**는 상태를 변경하는데 영향을 주는 데이터를 말합니다.

**useReducer** 함수에 인자를 살펴봅시다. **reducer**라는 상태 변경 설명서와 같은 함수를 **첫번째 인자**로 받고, **두번째**와 **세번째**는 **초기값**을 받습니다.

보통 일반적으로 **두번째 인자**에 초기값을 넣어주는 것으로 사용합니다. **세번째 인자**는 *특정 로직*을 처리 후 초기값을 설정할 때 사용합니다.

**reducer** 함수는 현재 상태와 **action** 객체를 받습니다. 상태를 변경하는 일종의 규칙을 **type**이라 하고 해당 **action.type**이 들어오면 상태를 변경합니다. 주의할 점은 이전 값을 참조하여 변경하는 것이 아닌 **새로운 값**을 반환해야합니다.

보통 객체를 이용하여 상태를 관리하는데, 리액트는 **객체의 참조값**을 비교하여 이전 상태와 다음 상태를 비교합니다. 따라서 상태를 변경할 때는 **새로운 참조값**을 가진 객체로 변경해야합니다. 그래야 기존 값이 다르다고 인식하여 화면 **render**가 다시 일어납니다.

따라서 다음과 같이 스프레드 문법이 자주 사용됩니다.

function reducer(state, action) {

switch (action.type) {

case "Increase":

*// 기존 상태를 스프레드 문법으로 새로운 객체에 넣고 age 맴버변수만 변경합니다.*

return {...state, number: state.number + action.payload};

default:

throw new Error();

}

}

Copy

정리해보자면 **useContext**로 하위에 있는 컴포넌트들에게 현재 **State**, 상태를 변경할 수 있는 **dispatch** 함수를 전달합니다.  
이전 실습에서 진행한 방식과 같이 전달받아 사용합니다.

그리고 상태를 변경하는 방식만 **useState**에서 **useReducer**로 변경했습니다. 여기서 추가로 작성한 것은 **reducer**라는 상태를 변경하는 규칙을 조금 더 상세히 적은 것입니다.

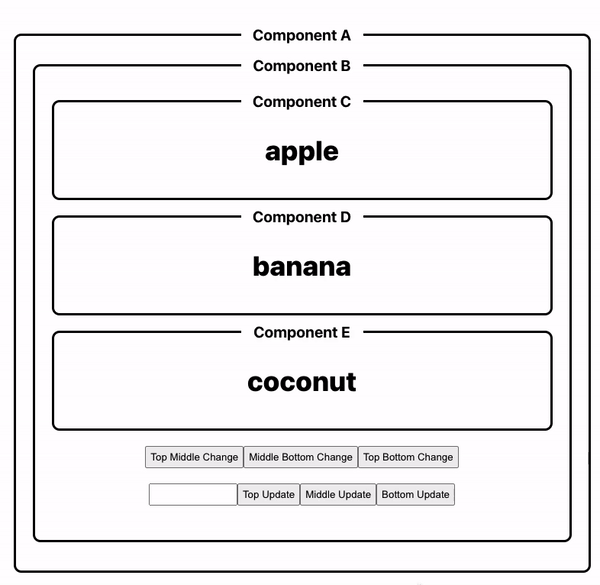
따라서 **reducer**라는 상태 변경 설명서와 이를 사용할 수 있는 도구인 **dispatch** 함수에 설명서에 나온대로 **action.type**과 **action.payload**를 전달해 상태를 변경할 수 있습니다.

그럼 지시사항을 보며 천천히 따라해봅시다.

지시사항

1. **App.js**에서 **initial** 상수에 초기값을 완성해주세요.
   * **top: apple**
   * **middle: banana**
   * **bottom: coconut**
2. 다음 사항에 맞추어 **reducer**를 작성해주세요. 서로 바꿀 수 있는 기능을 제공하도록 합니다.
   * **top-middle-change**: top ↔ middle
   * **middle-bottom-change**: middle ↔ bottom
   * **top-bottom-change**: top ↔ bottom
   * **top-update**: 이전 ↔ action.payload
   * **middle-update**: 이전 ↔ action.payload
   * **bottom-update**: 이전 ↔ action.payload
3. **useReducer**를 사용해 **state**, **dispatch**를 **value**에 넣어서 **Provider**에 전달해주세요.
4. **B.jsx**에서 **Context**로 부터 **dispatch**를 받아서 **button**에 **action** 객체를 전달해주세요.
   * **payload**에는 **useRef**로 **input**에 입력된 텍스트 값을 전달합니다.

실행 결과



Tips!

* 채점을 위해 지시사항을 따라서 단어를 작성해주세요. (**apple**, **banana**, **coconut**)
* **Component A**에 **changed**는 채점을 위한 것입니다. 지시사항에 맞춰서 실습을 진행해주세요.