AnyType.ts

type AnyType= Complex | Vector | Matrix | number;

export default AnyType;

Vector.ts

class Vector{

values: AnyType[];

constructor(value:AnyType[] = []){

…

}

}

class Matrix{

values: AnyType[][]; constructor(value:AnyType[][] = []){

}

…

}

AnyCalculator.ts

type AnyCalculator = RealCalculator | ComplexCalculator| VectorCalculator| MatrixCalculator;

export default AnyCalculator;

Создадим для всех калькуляторов общий интерфейс. В папке калькуляторов создадим класс ICalculator.

ICalculator.ts

export enum EOperand{

add=’add’,

sub=’sub’,

mult=’mult’,

div=’div’,

prod=’prod’,

pow=’pow’,

one=’one’,

zero=’zero’,

}

export default interface ICalculator<T>{

add(a:T, b:T):T;

sub(a:T, b:T):T;

div(a?:T, b?:T):T|null;

prod(a:T, p:number):T;

pow(a:T, p:number):T;

one(a?:T|number):T;…

}

RealCalculator.ts

class RealCalculator implements ICalculator<number>{

add(a:number, b:number):number{

return a+b;

}

…

}

VectorCalculator.ts

class VectorCalculator implements ICalculator<Vector>{

calc:ICalculator<AnyType>;

constructor(calc:ICalculator<AnyType>){

this.calc=calc;

}

add(a: Vector, b: Vector): Vector {

…

}

div(){return null}

…

}

Calculator.ts

class Calculator implements ICalculator<AnyType>{

complex(re?:number, im?:number):Complex{

return new Complex(re,im);

}

vector(values?: AnyType[]):Vector{

return new Vector(values);

}

matrix(…){…}

getComplex(str:string):Complex{…}

getVector(str:string):Vector{…}

getMatrix(str:string):Matrix{…}

get(elem?:AnyType):ICalculator<AnyType>{

if (elem instanceof Matrix){

return new MatrixCalculator(this.get(elem.values[0][0]);

}

…

return new RealCalculator();

}

[EOperand.add](a:AnyType, b:AnyType):AnyType{

return this.get(a).add(a,b);

}

Также для всех остальных

Для метода zero и one будет посложнее(но разберетесь).

}

Создадим хук useCalculator.ts

useCalculator.ts

export default function useCalculator(

refA:React.RefObject<HTMLTextAreaElement>,

refB:React.RefObject<HTMLTextAreaElement>,

refC:React.RefObject<HTMLTextAreaElement>

):(operand:EOperand)=>void{

const calc = new Calculator();

return (operand:EOperand)={

if (refA && refB && refC){

const A=refA.current?.value ||’’;

const B=refB.current?.value ||’’;

if (operand===EOperand.prod || operand === EOperand.pow){

refC.current.value=calc[operand](

calc.getValue(A),

parseFloat(B))?.toString() || ’’;

return;

}

refC.current.value=calc[operand](

calc.getValue(A),

calc.getValue(B))?.toString() || ’’;

)

}

}

}

Calc.tsx

const Calc:React.FC = () => {

const refA = useRef<HTMLTextAreaElement>(null);

const refB = useRef<HTMLTextAreaElement>(null);

const refC = useRef<HTMLTextAreaElement>(null);

const calc = useCalculator(refA,refB,refC);

return (<>

<textarea ref={refA}/>

<textarea ref={refB}/>

<textarea ref={refC}/>

<div>

<button onClick={()=>calc(EOperand.add)}>+</button>

<button onClick={()=>calc(EOperand.sub)}>-</button>

Со всеми так

</div>

<>);

}

export default Calc;