## **Angular Input Output Example**

You can define the properties for components you create and make them available across the whole angular app. In this Angular 12 Input Output tutorial, we will display the parent component’s data into a child component and the child component’s data into the parent component.

For that, we need to create two components. So go to the terminal and make two components by typing the following command. Remember, we are using [Angular CLI](https://appdividend.com/2020/02/18/how-to-update-angular-cli-to-version-9-angular-9-cli-upgrade/) to generate the new component.

## **Angular @Input**

Angular input decorator is just telling Angular, hey, when you find a property binding with this name, map it to my component property of this other name. Or, if I don’t give you an alias, use my component property name.

Now, even though Angular supports this name alias, it is a recommended practice to avoid using that approach by default. And try and use the class property name instead.

The Decorator marks the class field as an input property and supplies configuration metadata.

The input property is bound to the DOM property in the template. Therefore, during change detection, Angular automatically updates a data property with a DOM property’s value.

## **Angular @Output**

The Decorator that marks a class field as an output property and supplies configuration metadata. The DOM property bound to an output property is automatically updated during change detection.

You can supply the optional name to use in the templates when a component is instantiated that maps to the name of the bound property. By default, the original name of the required property is used for output binding.

Now, let’s understand this with an example. But, first, let’s create a new Angular project.

## **Step 1: Install the Angular Project via AngularCLI.**

First, we need to install Angular CLI globally in our system by typing the following command.

npm install -g @angular/cli

Now, fire the following command to create a project.

ng **new** **inout**

## **Step 2: Create parent and child components.**

Go to the terminal and type the following command.

ng g **c** parent

ng g **c** child

So, it will create an individual folder. Type the following command to start the Angular development server.

ng serve *--open*

It will open up the browser at the **port: 4200**.

Right now, only the **app.component.ts**component is rendered in the browser. If we want to render our parent component, we need to include it in an **app.component.html**file as an HTML tag.

<div style="text-align:center">

<h1>

Welcome to {{ title }}!

</h1>

<app-parent></app-parent>

</div>

Now, if you see in the browser, you can see the parent component renders. “**parent works!!** “

## **Step 3: Define HTML for parent component.**

Write the following code in the **parent.component.html**file.

<h3>Parent **Component**</h3>

<**label**>Bitcoin price</**label**>

<input **type**="text" />

<p>Value **of** child **component** **is**: </p>

First, we pass the data from the parent component to the child component. Here is the scenario, when the user types the bitcoin price in the text box, we can see its worth in the child component.

The same scenario applies to the child component. When the child component starts entering the price, it will display in the parent component.

## **Step 4: Define HTML for child components.**

Write the following code in the **child.component.html**file.

<h3>Child **Component**</h3>

<**label**>Child **Component**</**label**>

<input **type**="text" />

<p>Value **of** parent **component** **is**: </p>

As we know, this is a child component, so we need to include the **<app-child>** tag into the parent component. So our parent component HTML looks like this.

<h3>Parent Component</h3>

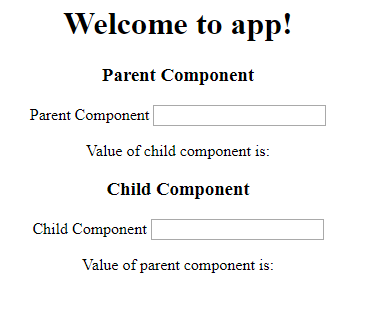
<label>Parent Component</label>

<input type="text" />

<p>Value of child component is: </p>

<app-child></app-child>

So, our application looks like this.



## **Step 5: Use Input to display parent component value**

Create a reference to the input text of the parent component. To edit the following lines in the **parent,component.html**file.

<**input** type="text" **#pcomponent** (keyup)="0"/>

<app-child [PData]="pcomponent.value"></app-child>

First, I have defined the reference for the input tag and then set the event listener. Then, when a user types something in the textbox, it will pass the value as a property to the child component.

The child component is ready to receive the property via the **@Input Decorator**. So this is the first use case of Inputs in Angular.

File **child.component.ts**file looks like this.

*// child.component.ts*

**import** { Component, OnInit, Input } **from** '@angular/core';

**@Component**({

selector: 'app-child',

templateUrl: './child.component.html',

styleUrls: ['./child.component.css']

})

**export** **class** ChildComponent **implements** OnInit {

**@Input**() PData: number;

**constructor**() { }

ngOnInit() {

}

}

You can see this component’s property is PData, which is the same property we have written in the **parent.component.html**file.

Finally, our **child.component.html**file looks like this. We need to add interpolation to display the parent data in the child component.

<h3>Child Component</h3>

<label>Child Component</label>

<input type="text" />

<p>Value of parent component is: {{ PData }}</p>

Now, if you type the parent text box, then its value print in the child component. Thus, all is done through the parent to child node via input property.

## **Step 6: Pass value from child to parent component.**

Passing the data from the child component to the parent component is a little bit tricky. In this scenario, the child component does not have any reference to the parent component.

So, in this case, we need to emit an event from the child component, and the parent component will listen to it and receive the data via event and display it.

First, create a reference to the Input in the child component and attach an event listener to it.

<h3>Child Component</h3>

<label>Child Component</label>

<input type="text" #ccomponent (keyup)="onChange(ccomponent.value)"/>

<p>Value of parent component is: {{ PData }}</p>

Write the onChange function in the **child.component.ts**file.

*// child.component.ts*

**import** { Component, OnInit, Input, Output, EventEmitter } **from** '@angular/core';

**@Component**({

selector: 'app-child',

templateUrl: './child.component.html',

styleUrls: ['./child.component.css']

})

**export** **class** ChildComponent **implements** OnInit {

**@Input**() PData: number;

**@Output**() childEvent = **new** EventEmitter();

**constructor**() { }

onChange(value) {

**this**.childEvent.emit(value);

}

ngOnInit() {

}

}

When the user types anything in the textbox of the child component, it will start emitting the value from the child component. So we just need to listen to that event emitter and display the passed value in the parent component.

Use an event binding in the **parent.component.html**file and listen for the event emitter.

<**app**-child [PData]="pcomponent.value" (childEvent)="CData=$event"></**app**-child>

We need to define CData into the **parent.component.ts**file.

*// parent.component.ts*

**public** CData: number;

Finally, by interpolation, we can display its value in the **parent.component.html** file.

// parent.component.html

<h3>Parent Component</h3>

<label>Parent Component</label>

<input type="text" #pcomponent (keyup)="0"/>

<p>Value of child component is: {{ CData }}</p>

<app-child [PData]="pcomponent.value" (childEvent)="CData=$event"></app-child>

I am writing the following four files if you find any confusion throughout this tutorial.

*// parent.component.ts*

**import** { Component, OnInit } **from** '@angular/core';

**@Component**({

selector: 'app-parent',

templateUrl: './parent.component.html',

styleUrls: ['./parent.component.css']

})

**export** **class** ParentComponent **implements** OnInit {

**public** CData: number;

**constructor**() { }

ngOnInit() {

}

}

*// child.component.ts*

**import** { Component, OnInit, Input, Output, EventEmitter } **from** '@angular/core';

**@Component**({

selector: 'app-child',

templateUrl: './child.component.html',

styleUrls: ['./child.component.css']

})

**export** **class** ChildComponent **implements** OnInit {

**@Input**() PData: number;

**@Output**() childEvent = **new** EventEmitter();

**constructor**() { }

onChange(value) {

**this**.childEvent.emit(value);

}

ngOnInit() {

}

}

// child.component.html

<h3>Child Component</h3>

<label>Child Component</label>

<input type="text" #ccomponent (keyup) = "onChange(ccomponent.value)"/>

<p>Value of parent component is: {{ PData }}</p>

@Input() And @Output() Decorator In Angular

## **@Input Decorator**

@Input is a decorator to mark a property as an input.  @Input is used to define an input property, to achieve component property binding.  @Inoput decorator is used to pass data (property binding) from parent to child component.  The component property should be annotated with @Input decorator to act as input property.

Let's explore it practically.

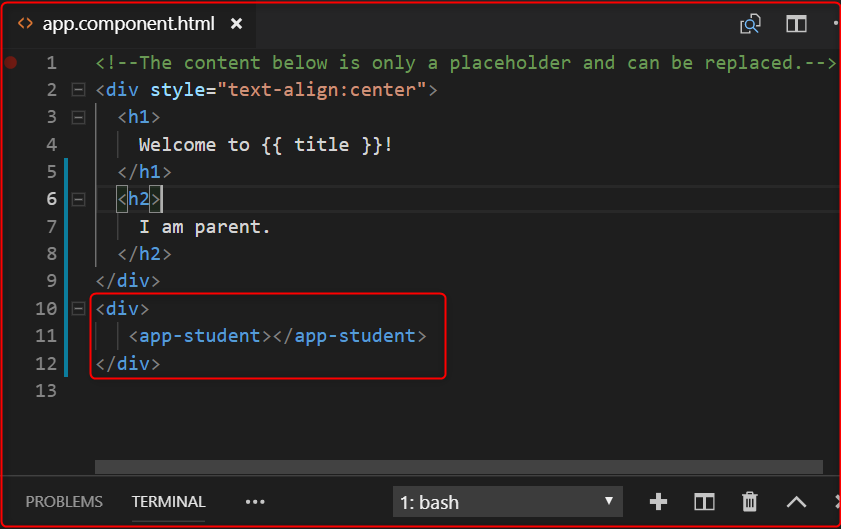
I have created an angular application which is AngApp.  I have created two components.  They are app components and student components.  I will transfer the data from parent to child component, using @Input decorator.  I am assuming my, app-component is the parent component and student-component is the child component.

To make the parent-child relation, keep the instance (selector of student component) of student component inside the template URL (app.component.html) of app component.

Open app.component.html:  Inside this file, we keep an instance of student component.

**Example**

1. <div> <app-student></app-student></div>



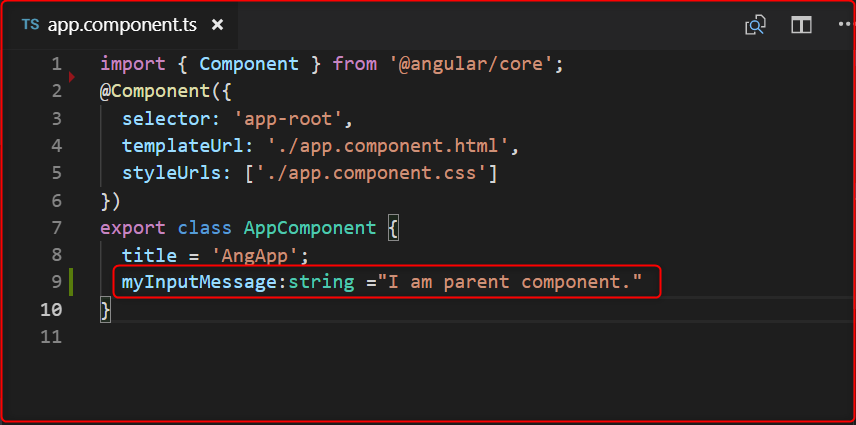
Listing 1.0

In the above image, the selected area is the child component.

Now, we want to send the message from parent to child component.

Let's open the parent component's .ts file (app.component.ts) and declare a variable inside AppComponent class, to store the message.  This message is received by the child component.

1. myInputMessage:string ="I am the parent comppnent"

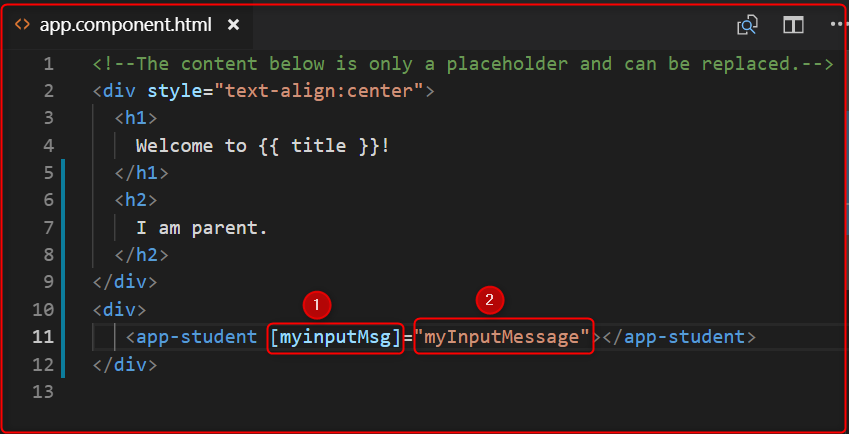


1. **import** { Component, Input, OnInit } from '@angular/core';
3. @Component({
4. selector: 'app-student',
5. templateUrl: './student.component.html',
6. styleUrls: ['./student.component.css']
7. })
9. **export** **class** StudentComponent **implements** OnInit {
10. @Input() myinputMsg:string;
12. constructor() { }
14. ngOnInit() {
15. console.log(**this**.myinputMsg);
16. }
18. }

In the above image, we have declared a variable( myInputMessage) shown in the selected area.

Now, let's open parent component views (app.component.html) pass this variable inside child component instance, which is passed inside parent component.

1. <div>
2. <app-student [myinputMsg]="myInputMessage"></app-student>
3. </div>



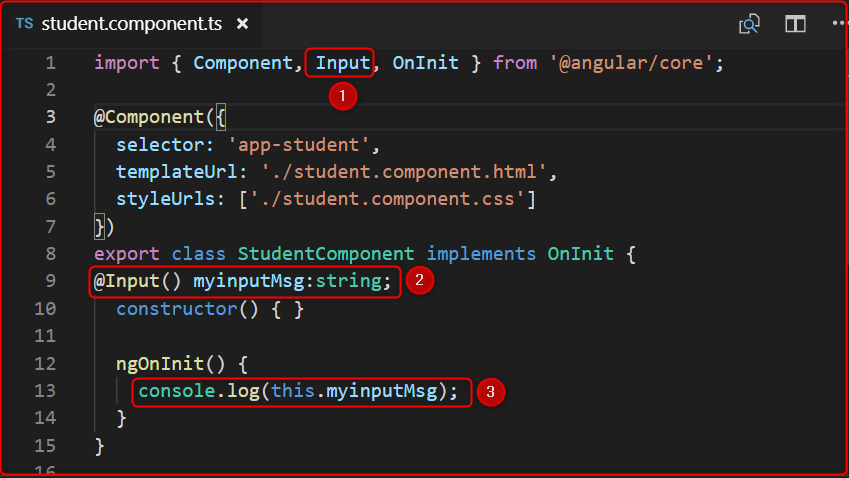
The above image, represents 2 points.  Let's explain each of the points.

1. Denotes those variables which will be used by child component (student component) with @Input decorator to fetch the message from parent component and,
2. denotes those variables which are passed the parent component message to child component.

Now, open the child component's .ts file (student.component.ts) and import Input decorator, using the myinputMsg variable with @Input decorator and print it inside constructor or ngOnInit().

1. **import** { Component, Input, OnInit } from '@angular/core';
3. @Component({
4. selector: 'app-student',
5. templateUrl: './student.component.html',
6. styleUrls: ['./student.component.css']
7. })
9. **export** **class** StudentComponent **implements** OnInit {
10. @Input() myinputMsg:string;
12. constructor() { }
14. ngOnInit() {
15. console.log(**this**.myinputMsg);
16. }
18. }

**Output**



**Output**

In image 4, represent 3 points. Let's explain each of the points.

1. First import the Input decorator, which is provided by angular and full path is @anuglar/core.
2. Use @Input decorator and declare those variables which are passed by parent component Html(app.component.html) file's point 1.  When we declare those variable (myinputMsg) with @Input decorator it automatically fetches the value of the parent component variable with the help of @Input decorator.
3. Print the values of this variable inside constructor or ngOnInit(). We have used inside ngOnInit().

Let's output,



## **@Output Decorator**

@Output decorator is used to pass the data from child to parent component.  @Output decorator binds a property of a component, to send data from one component to the calling component.  @Output binds a property of the type of angular EventEmitter class.

To transfer the data from child to parent component, we use @Output decorator.

Lets's Open the child component' .ts file (student.component.ts).

For use the @Output decorator we have to import, two important decorators, they are Output and EventEmitter.

## **EventEmitter**

Use in components with the @Output directive to emit custom events synchronously or asynchronously, and register handlers for those events by subscribing to an instance.

1. **import** { Component, Input, Output,EventEmitter, OnInit } from '@angular/core';

Now, create any variable with @Output decorator.

1. @Output() myOutput:EventEmitter<string>= **new** EventEmitter();

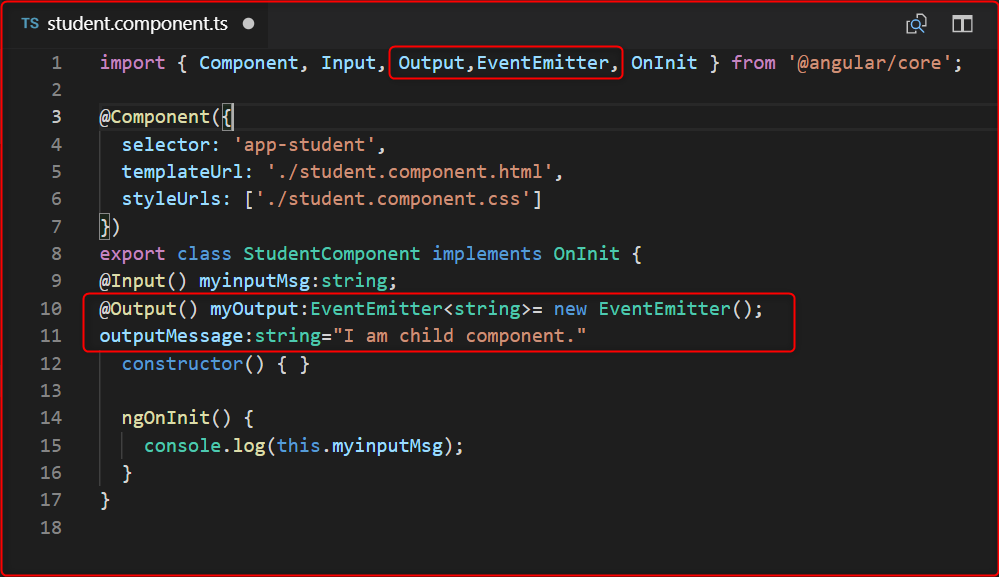
Here in the place of string, we can pass different types of DataTypes.

After that create a variable to store and pass the message to the parent component.

1. outputMessage:string="I am child component."

**Code**

1. **import** { Component, Input, Output,EventEmitter, OnInit } from '@angular/core';
3. @Component({
4. selector: 'app-student',
5. templateUrl: './student.component.html',
6. styleUrls: ['./student.component.css']
7. })
8. **export** **class** StudentComponent **implements** OnInit {
9. @Input() myinputMsg:string;
10. @Output() myOutput:EventEmitter<string>= **new** EventEmitter();
11. outputMessage:string="I am child component."
12. constructor() { }
14. ngOnInit() {
15. console.log(**this**.myinputMsg);
16. }
17. }



Send the value of output message, to the parent component.  Then we create a button and click on this button.  We will send the values to the parent component.

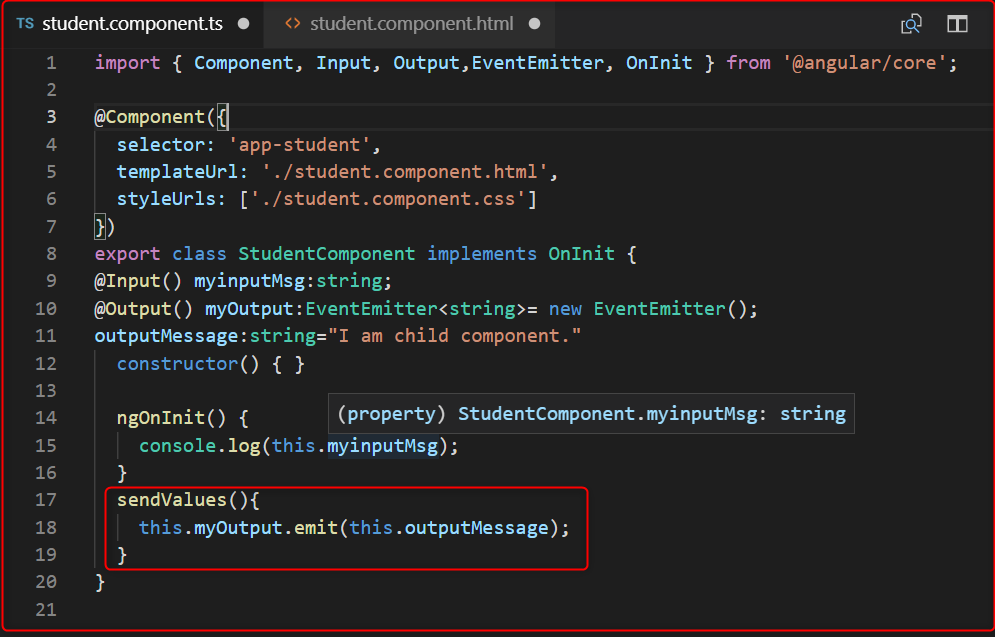
Let's open the child component Html page and create a button and click event of this button.  We then send the values.

**student.component.html**

1. <button (click)="sendValues"> Send Data </button>

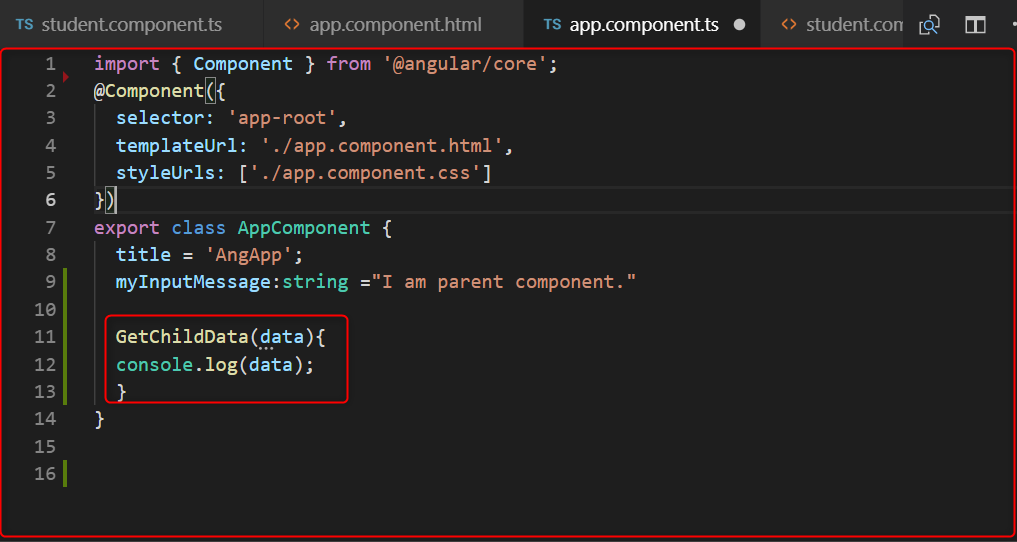
 Now fire the click on student.component.ts.

1. sendValues(){
2. **this**.myOutput.emit(**this**.outputMessage);
3. }

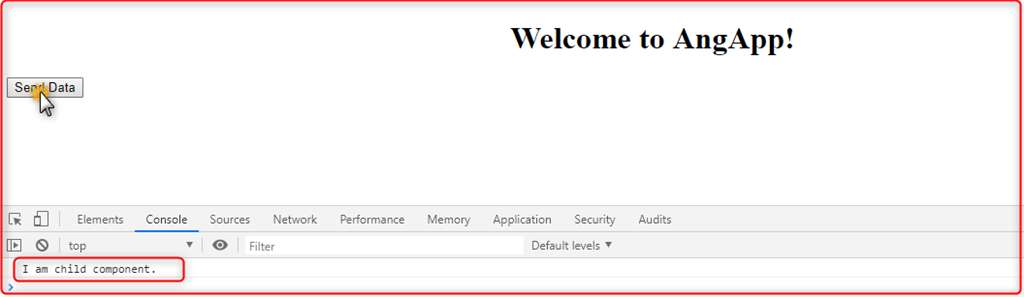


Now, to fetch the value we have to go app.component.html file and use the below code.

1. <app-student [myinputMsg]="myInputMessage" (myOutput) ="GetChildData($event)"></app-student>



1. **function** which is GetChildData() on parent component' .ts file, **for** fetch the data from child component.
2. Open the app.component.ts:
4. Code:
6. GetChildData(data){
7. console.log(data);
8. }



npm install bootstrap --save

npm install jquery --save

npm install popper.js --save

Example for Inputoutput decorator in Angular

ng new InputOutputExample

cd InputOutputExample

ng g c customerList

ng g c customerDetail

ng g class customer

Customer

export class Customer {

customerNo: number=0;

name: string="";

address: string="";

city: string="";

state: string="";

country: string="";

}