Tuguldur Sosorbaram **2. Assignment / Task №4** **5th of May**
EERDEY
Sutu0318@gmail.com
Group 4

TASK

4. Layers of gases are given, with certain type (ozone, oxygen, carbon dioxide) and thickness, affected by atmospheric variables (thunderstorm, sunshine, other effects). When a part of one layer changes into another layer due to an atmospheric variable, the newly transformed layer ascends and engrosses the first identical type of layer of gases over it. In case there is no identical layer above, it creates a new layer on the top of the atmosphere. In the following we declare, how the different types of layers react to the different variables by changing their type and thickness. No layer can have a thickness less than 0.5 km, unless it ascends to the identical-type upper layer. In case there is no identical one, the layer perishes.

	thunderstorm	**sunshine**	**other**
ozone	-	-	5% turns to oxygen
oxygen	50% turns to ozone	5% turns to ozone	10% turns to carbon dioxide
carbon dioxide	-	5% turns to oxygen	-

The program reads data from a text file. The first line of the file contains a single integer N indicating the number of layers. Each of the following N lines contains the attributes of a layer separated by spaces: type and thickness. The type is identified by a character: Z – ozone, X – oxygen, C – carbon dioxide. The last line of the file represents the atmospheric variables in the form of a sequence of characters: T – thunderstorm, S – sunshine, O – others. In case the simulation is over, it continues from the beginning.

The program should continue the simulation until the number of layers is the triple of the initial number of layers or is less than three. The program should print all attributes of the layers by simulation rounds!

The program should ask for a filename, then print the content of the input file. You can assume that the input file is correct.

ANALYSIS

Independent objects in the task are the Layers of gasses. There is 3 different type of layers which is Oxygen, Ozone and Carbon Dioxide. As atmospheric variable happens layers of gasses change or transform into other layer then rise to merge or perish.

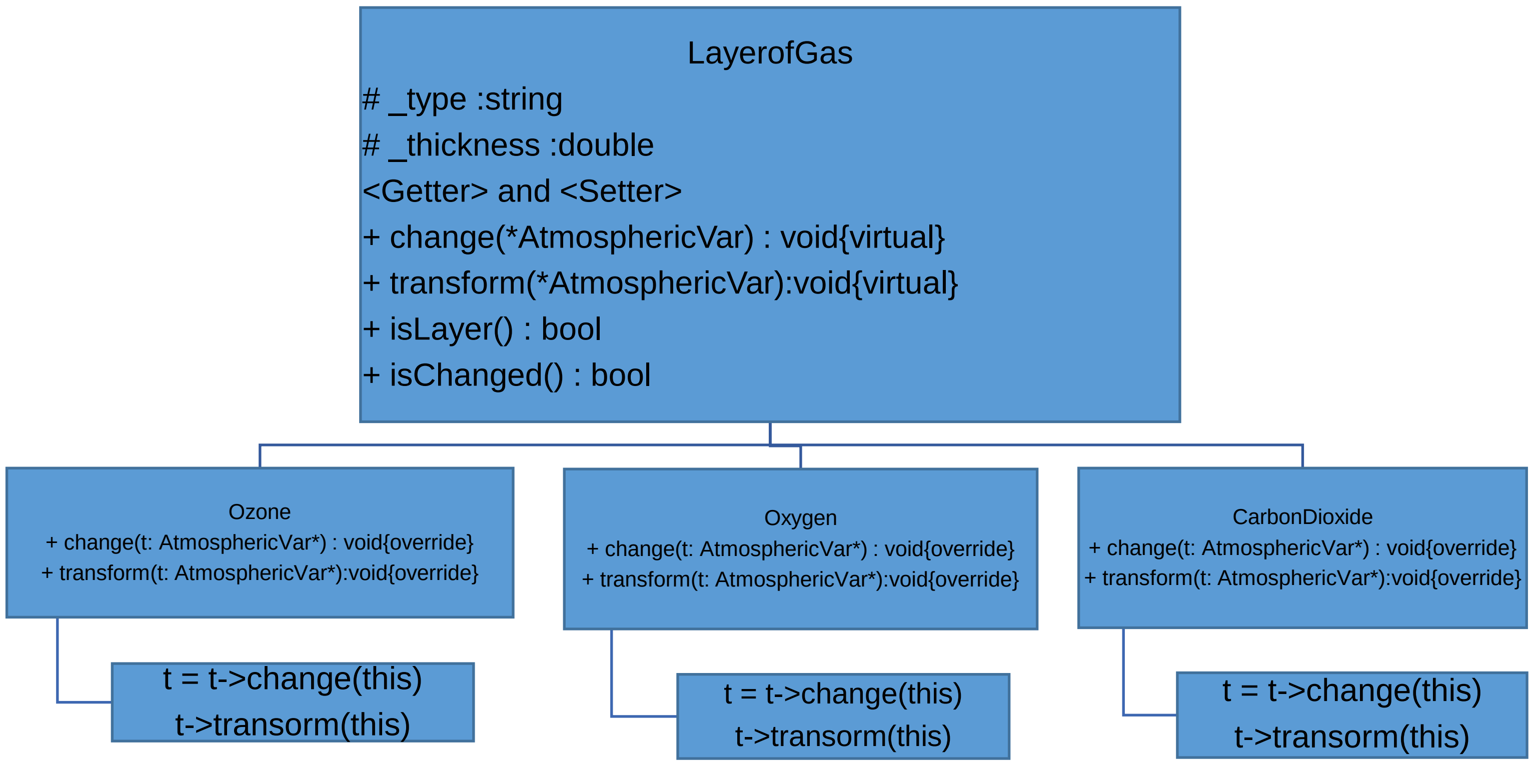
We can see how different atmospheric variables affect other layers from the task instruction table.

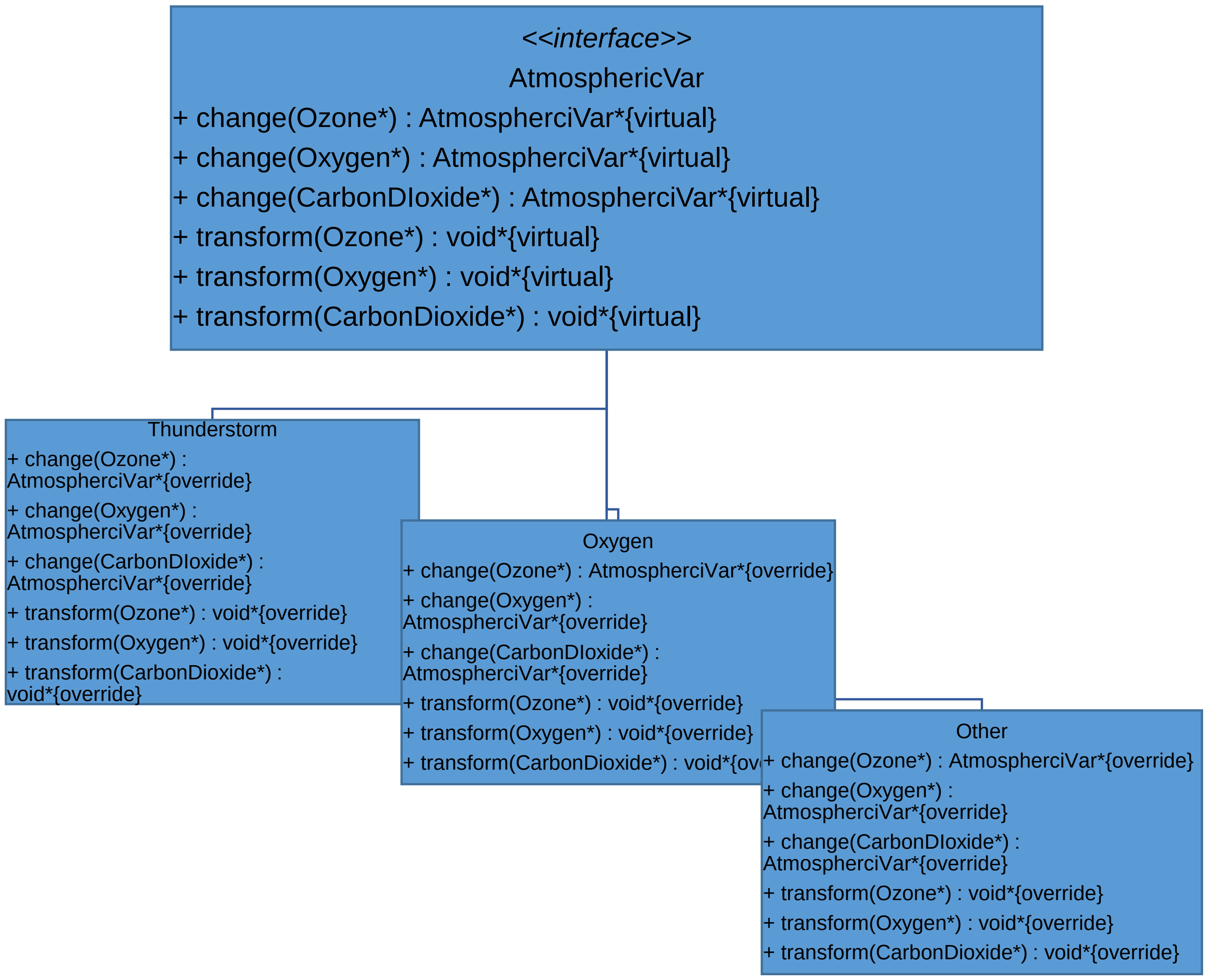
PLAN

For layers of gas:

It will be saved in vector of pointers of class LayerofGas which is base for 3 concrete class. Those are Ozone, Oxygen and CarbonDioxide. All 3 concrete class have common properties in base class which is _type (type of layer saved as string), _thickness(thickness of layer saved as a double).
Also there are common methods for Layer of gasses.
getType(), getThickness() = getters for protected properties.
addThickness(), transformType() = setters for protected properties
isLayer() = Layer with thickness less than 0.5km can not be layer so it checks that.
isChanged() = Checks if current Layer of Gas is changed during this atmospheric variable.
Change() and transform() = are pure virtual functions since both of the methods effect depends on what kind of atmospheric variable is happening.
For atmospheric variable:

It will be also saved in vector of pointers to class AtmosphericVar which is also base for 3 concrete inherited class: Thunderstorm, Sunshine and Other. Every Concrete atmosphericVar class has 6 different methods.
3 for changing LayerofGas thickness and 3 for transforming type of LayerofGas according to AtmosphericVar. Methods will be virtual in base class but overridden in the 3 separate concrete classes according to layerofGasses.


LayerofGas
# _type :string
# _thickness :double
<Getter> and <Setter>
+ change(*AtmosphericVar) : void{virtual}
+ transform(*AtmosphericVar):void{virtual}
+ isLayer() : bool
+ isChanged() : bool
Ozone
+ change(t: AtmosphericVar*) : void{override}
+ transform(t: AtmosphericVar*):void{override}
Oxygen
+ change(t: AtmosphericVar*) : void{override}
+ transform(t: AtmosphericVar*):void{override}
CarbonDioxide
+ change(t: AtmosphericVar*) : void{override}
+ transform(t: AtmosphericVar*):void{override}
t = t->change(this)
t->transorm(this)
t = t->change(this)
t->transorm(this)
t = t->change(this)
t->transorm(this)



<<interface>>
AtmosphericVar
+ change(Ozone*) : AtmospherciVar*{virtual}
+ change(Oxygen*) : AtmospherciVar*{virtual}
+ change(CarbonDIoxide*) : AtmospherciVar*{virtual}
+ transform(Ozone*) : void*{virtual}
+ transform(Oxygen*) : void*{virtual}
+ transform(CarbonDioxide*) : void*{virtual}
Thunderstorm
+ change(Ozone*) : AtmospherciVar*{override}
+ change(Oxygen*) : AtmospherciVar*{override}
+ change(CarbonDIoxide*) : AtmospherciVar*{override}
+ transform(Ozone*) : void*{override}
+ transform(Oxygen*) : void*{override}
+ transform(CarbonDioxide*) : void*{override}
Oxygen
+ change(Ozone*) : AtmospherciVar*{override}
+ change(Oxygen*) : AtmospherciVar*{override}
+ change(CarbonDIoxide*) : AtmospherciVar*{override}
+ transform(Ozone*) : void*{override}
+ transform(Oxygen*) : void*{override}
+ transform(CarbonDioxide*) : void*{ov
Other
+ change(Ozone*) : AtmospherciVar*{override}
+ change(Oxygen*) : AtmospherciVar*{override}
+ change(CarbonDIoxide*) : AtmospherciVar*{override}
+ transform(Ozone*) : void*{override}
+ transform(Oxygen*) : void*{override}
+ transform(CarbonDioxide*) : void*{override}

Methods change() and transform() expects parameter atmosphericVar* as a visitor and calls the methods which corresponds to type of LayerofGas.

Specification:

A = Effects^m: AtmosphericVar* , Layers^n: LayerofGas*

Pre = Effects = Effects$_0$ $\wedge$ Layers = Layers$_0$

Post = Effects = Effects$_0$ $\wedge$ $\sum$ $\forall$i$\in$[i..m] : $\forall$j$\in$[j..n] : change(Layers[j], Effects[i])

$\wedge \sum_{k=0}^{n}(\text{transform}(\text{Layers}[k]))$

isChanged()

After each every atmospheric effect gas layer changes thickness and some of it transforms into new layer. If thickness does not change I don't have to transform any layer. So 1 atmospheric layer effecting on all layers should be 2 summation enum. First one is change and then if it is changed I create new layer same as the current and transform it and after that transformed layer should rise.

Analogy with summation enumerator:

Change() function:

enor(E)	i = 1 .. n
f(e)	change(Layers[i], track)
s	Layers
H, +, 0	Layers* , ⊕, <>

Transform() function:

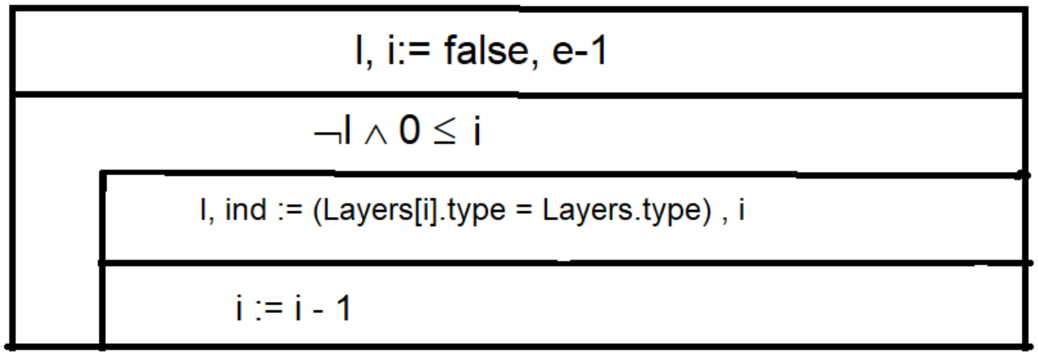
enor(E)	i = 1 .. n
f(e)	<transform (Layers[i], track)> if isChanged(Layers[i])
s	Layers
H, +, 0	Layers* , ⊕, <>

After I figured out how to change and transform the Layers there was something had to be done.
That was transformed layer should rise and engross with identical layer or create new layer on top of layers if there is no identical layer above. Above means before in Layers vector. So I created 2 different function. addNewLayer() and SearchLayer() both takes vector of LayerofGas* and 1 LayerofGas as a parameter.

SearchLayer() takes 1 more parameter than addNewLayer() which is integer to define interval to search from the vector since I had to find the layer above the transformed layer.

SearchLayer()
Analogy with Linear Search interval:

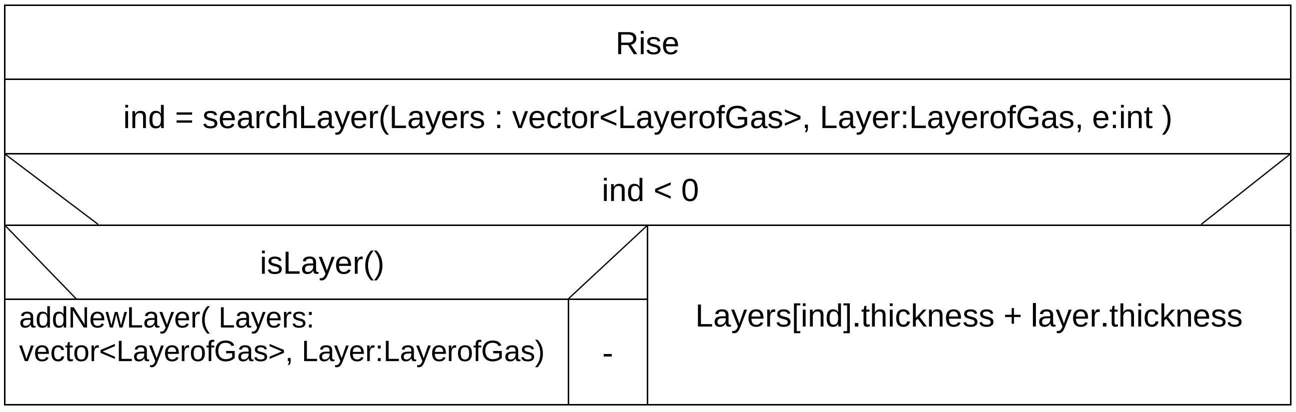
m,n	e,0
Cond(i)	Layers[i].type = Layer.type



```
l, i:= false, e-1
¬l ∧ 0 ≤ i
    l, ind := (Layers[i].type = Layers.type) , i
    i := i - 1
```

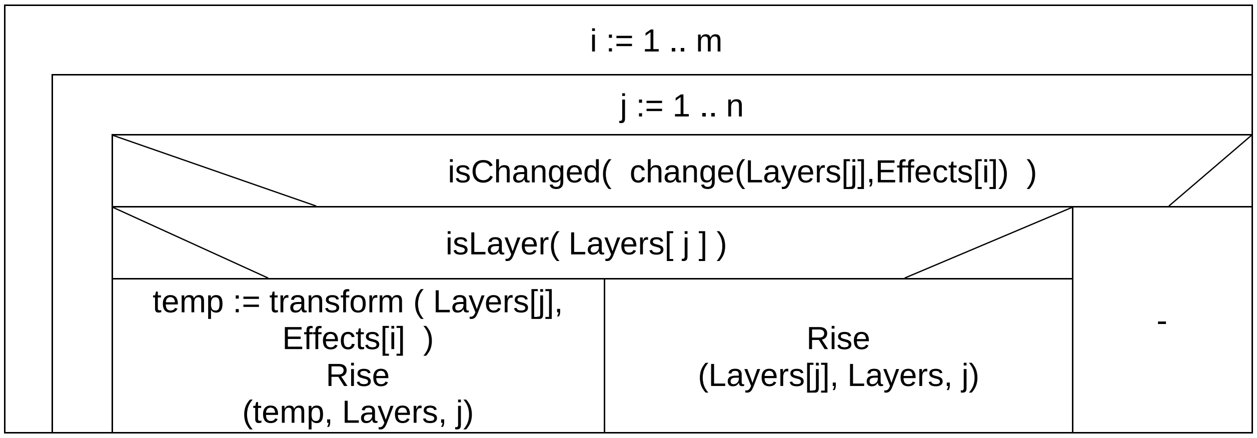
So after this if I combine this two function it should be ready to handle new transformed layer. Every time new Layer transforms I have to searchLayer and if found in the vector addThickness() if it is not found I have to addNewLayer().
In case of addNewLayer() I used built in function vector.insert but thing is since Layersn vector size is increased by 1 on the top I had to increase index by 1 everytime I use that function.
(1 thing to remember is everytime I add new Layer to Layers I have to make sure it is isLayer = true)

Rise:



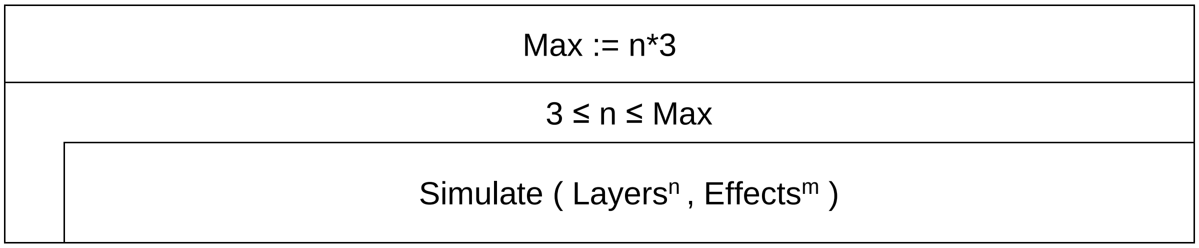
```
Rise
ind = searchLayer(Layers : vector<LayerofGas>, Layer:LayerofGas, e:int )
ind < 0
    isLayer()
        addNewLayer( Layers: vector<LayerofGas>, Layer:LayerofGas)
    else
        -
else
    Layers[ind].thickness + layer.thickness
```

After this if I merge it with change() and transform() from top it should be 1 whole simulation.



```
i := 1 .. m
    j := 1 .. n
        isChanged(  change(Layers[j],Effects[i])  )
            isLayer( Layers[ j ] )
                temp := transform ( Layers[j], Effects[i]  )
                Rise
                (temp, Layers, j)
            else
                Rise
                (Layers[j], Layers, j)
        else
            -
```

The program should continue the simulation until the number of layers is the triple of the initial number of layers or is less than three. The program should print all attributes of the layers by simulation rounds.
So I create integer max which is triple the initial number of Layers.
And create loop which does the simulation as long as the layers count is between 3 and max.



```
Max := n*3
3 ≤ n ≤ Max
    Simulate ( $Layers^n$ , $Effects^m$ )
```

TEST CASES:

Grey box test cases:
Change and transform (Summation)

1. length-based:
    - zero layer
    - one layer (3 different layers)
    - more layer
2. Order of layers:
    - When transforming:
        - no identical layer above
        - 1 identical layer above
        - multiple identical layer above
3. Thickness of layers:
    -when changed layer’s thickness is less than 0.5km
    -when transformed layer’s thickness is less than 0.5km

Change and transform with different type of atmospheric variable

- 9 different change cases
- 6 x 2 x 2 different transform cases