The difference between the implementations of a composition and an aggregation.

Composition means that the composer object will totally handle the component object’s lifetime. Once the composer object is destroyed, the component object will also be destroyed.

However, aggregation means that the composer doesn’t handle the component object’s lifetime. Even the composer is destroyed, the component object will not be influenced.

Composition:

In MessagePack 1 arg constructor

this->message = new Message[numOfMessage];

In messagePack destructor

MessagePack::~MessagePack() {

delete[] this->message;

}

Aggregation:

In Notifications class, we don’t need to allocate or delete the memory of Message Object.

I use array of pointer to handle the Message object.

const Message\*\* ptrMessagePtr;

this->ptrMessagePtr = new const Message \* [maxNumAddress];

2. The difference between the implementations of move and copy functions in a composition and an aggregation.

In composition, I use pointer of Message to hold the Message object

As a result, I need to use deep copy to move or copy the object.

It needs to directly work with the address of the Message object.

if (this != &src)

{

delete[] this->message;

this->numOfMsg = src.numOfMsg;

this->message = new Message[this->numOfMsg];

for (size\_t i = 0; i < this->numOfMsg; i++)

{

this->message[i] = src.message[i];

}

}

However, in the aggregation, I use dynamically allocated array of pointers to handle the message object.

Still, I need to use deep copy to move or copy the address.

The only difference is that in this case, it copies the address of pointer that point to another pointer.

if (this != &src)

{

delete[] this->ptrMessagePtr;

this->currentNumAddress = src.currentNumAddress;

this->maxNumAddress = src.maxNumAddress;

this->ptrMessagePtr = new const Message \* [this->maxNumAddress];

for (size\_t i = 0; i < this->maxNumAddress; i++)

{

this->ptrMessagePtr[i] = src.ptrMessagePtr[i];

}

}

Quiz 4

Full mark in quiz 4