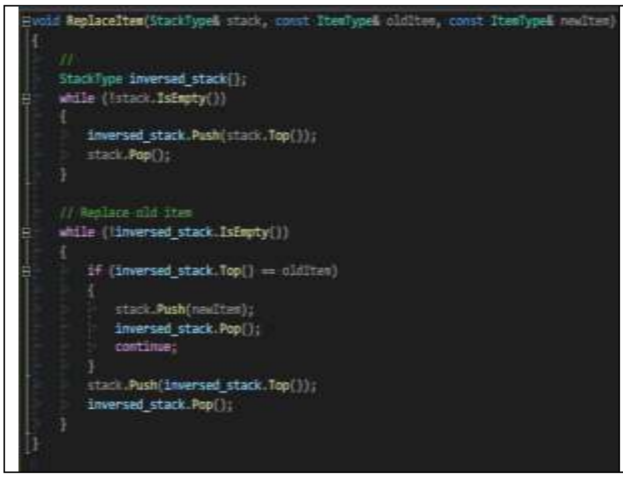
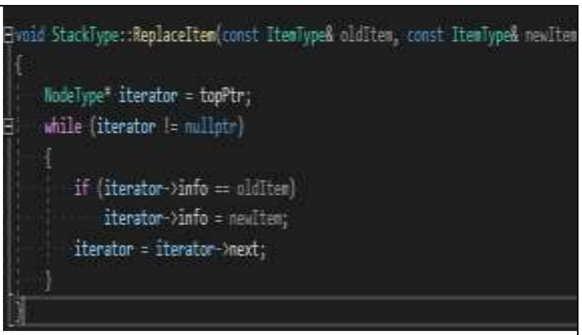
2020105695 김희성

Exercise1

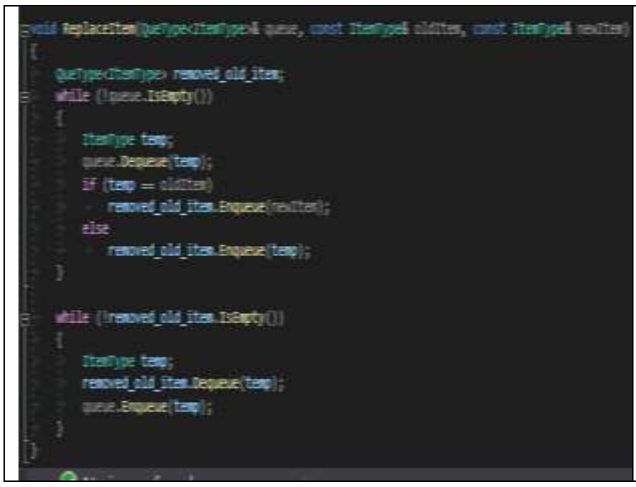


```cpp
void ReplaceItem(StackType& stack, const ItemType& oldItem, const ItemType& newItem)
{
    //
    StackType inversed_stack{};
    while (!stack.IsEmpty())
    {
        inversed_stack.Push(stack.Top());
        stack.Pop();
    }

    // Replace old item
    while (!inversed_stack.IsEmpty())
    {
        if (inversed_stack.Top() == oldItem)
        {
            stack.Push(newItem);
            inversed_stack.Pop();
            continue;
        }
        stack.Push(inversed_stack.Top());
        inversed_stack.Pop();
    }
}
```

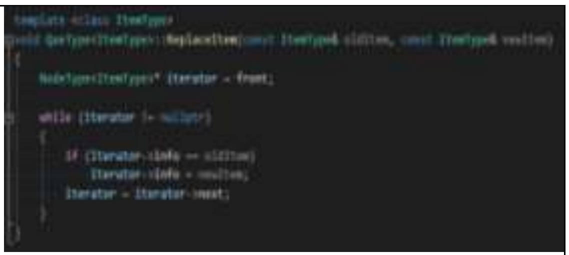


```cpp
void StackType::ReplaceItem(const ItemType& oldItem, const ItemType& newItem
{
    NodeType* iterator = topPtr;
    while (iterator != nullptr)
    {
        if (iterator->info == oldItem)
            iterator->info = newItem;
        iterator = iterator->next;
    }
}
```

Exercise2

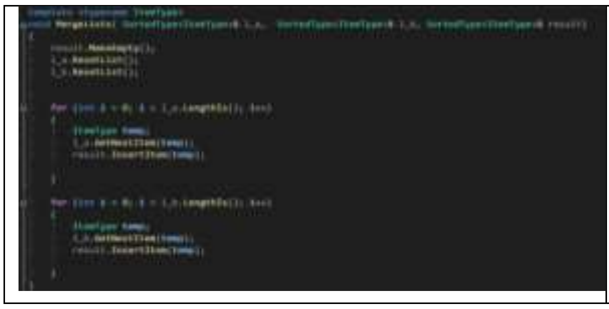


```cpp
void ReplaceItem(QueType<ItemType>& queue, const ItemType& oldItem, const ItemType& newItem)
{
    QueType<ItemType> removed_old_item;
    while (!queue.IsEmpty())
    {
        ItemType temp;
        queue.Dequeue(temp);
        if (temp == oldItem)
            removed_old_item.Enqueue(newItem);
        else
            removed_old_item.Enqueue(temp);
    }

    while (!removed_old_item.IsEmpty())
    {
        ItemType temp;
        removed_old_item.Dequeue(temp);
        queue.Enqueue(temp);
    }
}
```

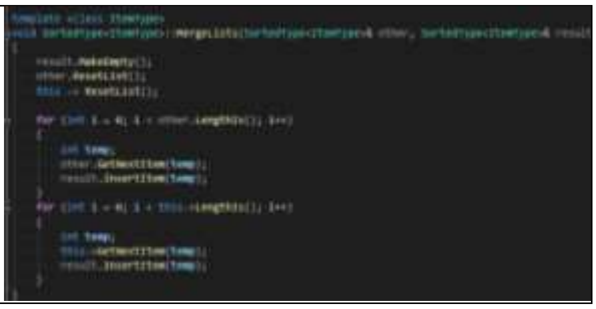


```cpp
template <class ItemType>
void QueType<ItemType>::ReplaceItem(const ItemType& oldItem, const ItemType& newItem)
{
    NodeType<ItemType>* iterator = front;

    while (iterator != nullptr)
    {
        if (iterator->info == oldItem)
            iterator->info = newItem;
        iterator = iterator->next;
    }
}
```

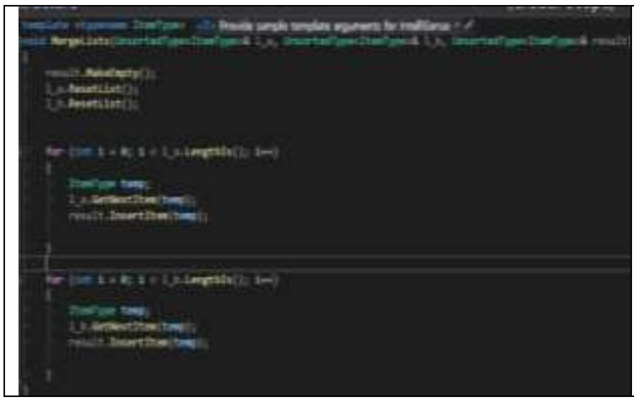
Exercise3.



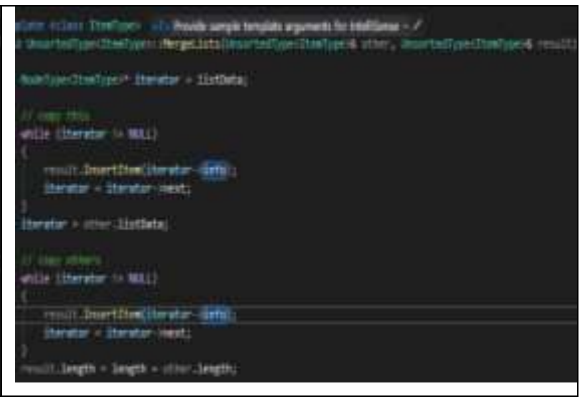
```
template <typename ItemType>
void MergeLists( SortedType<ItemType>& l_a,  SortedType<ItemType>& l_b, SortedType<ItemType>& result)
{
    result.MakeEmpty();
    l_a.ResetList();
    l_b.ResetList();


    for (int i = 0; i < l_a.LengthIs(); i++)
    {
        ItemType temp;
        l_a.GetNextItem(temp);
        result.InsertItem(temp);

    }

    for (int i = 0; i < l_b.LengthIs(); i++)
    {
        ItemType temp;
        l_b.GetNextItem(temp);
        result.InsertItem(temp);

    }
}
```



```
template <class ItemType>
void SortedType<ItemType>::MergeLists(SortedType<ItemType>& other, SortedType<ItemType>& result
{
    result.MakeEmpty();
    other.ResetList();
    this -> ResetList();

    for (int i = 0; i < other.LengthIs(); i++)
    {
        int temp;
        other.GetNextItem(temp);
        result.InsertItem(temp);
    }
    for (int i = 0; i < this->LengthIs(); i++)
    {
        int temp;
        this->GetNextItem(temp);
        result.InsertItem(temp);
    }
}
```

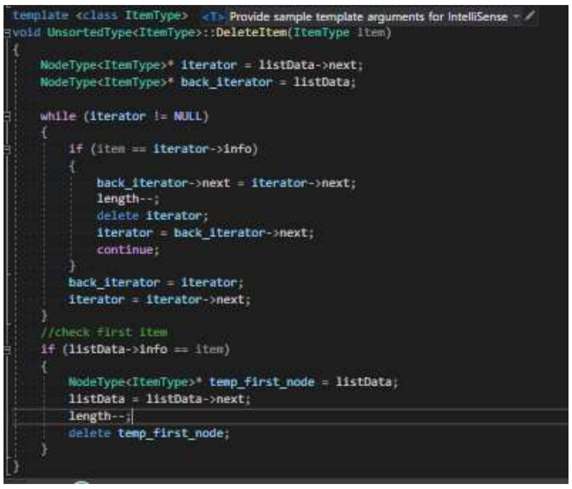


```
template <typename ItemType>  <T> Provide sample template arguments for IntelliSense
void MergeLists(UnsortedType<ItemType>& l_a, UnsortedType<ItemType>& l_b, UnsortedType<ItemType>& result)
{
    result.MakeEmpty();
    l_a.ResetList();
    l_b.ResetList();


    for (int i = 0; i < l_a.LengthIs(); i++)
    {
        ItemType temp;
        l_a.GetNextItem(temp);
        result.InsertItem(temp);

    }
    for (int i = 0; i < l_b.LengthIs(); i++)
    {
        ItemType temp;
        l_b.GetNextItem(temp);
        result.InsertItem(temp);

    }
}
```



```
plate <class ItemType>  <T> Provide sample template arguments for IntelliSense
d UnsortedType<ItemType>::MergeLists(UnsortedType<ItemType>& other, UnsortedType<ItemType>& result)

NodeType<ItemType>* iterator = listData;

// copy this
while (iterator != NULL)
{
    result.InsertItem(iterator->info);
    iterator = iterator->next;
}
iterator = other.listData;

// copy others
while (iterator != NULL)
{
    result.InsertItem(iterator->info);
    iterator = iterator->next;
}
result.length = length + other.length;
```

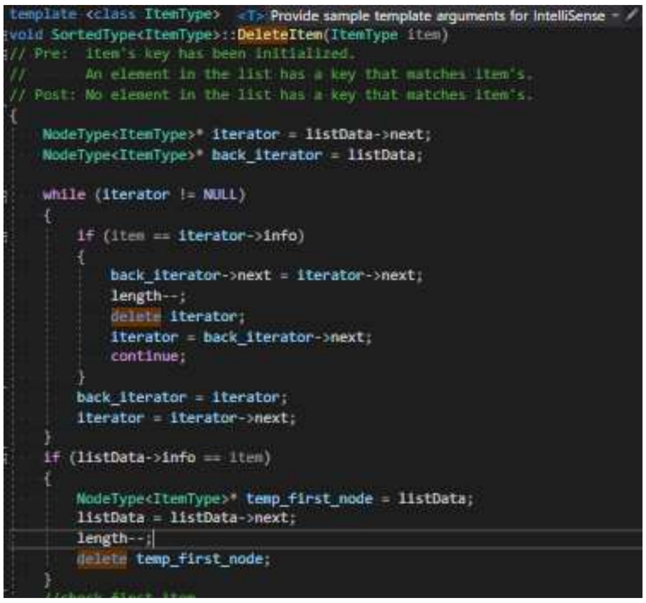
Exercise4



```
template <class ItemType>  <T> Provide sample template arguments for IntelliSense
void UnsortedType<ItemType>::DeleteItem(ItemType item)
{
    NodeType<ItemType>* iterator = listData->next;
    NodeType<ItemType>* back_iterator = listData;

    while (iterator != NULL)
    {
        if (item == iterator->info)
        {
            back_iterator->next = iterator->next;
            length--;
            delete iterator;
            iterator = back_iterator->next;
            continue;
        }
        back_iterator = iterator;
        iterator = iterator->next;
    }
    //check first item
    if (listData->info == item)
    {
        NodeType<ItemType>* temp_first_node = listData;
        listData = listData->next;
        length--;
        delete temp_first_node;
    }
}
```

동일한 값을 가질 경우 모두 없애고

Exercise 5



```
template <class ItemType>   <T> Provide sample template arguments for IntelliSense
void SortedType<ItemType>::DeleteItem(ItemType item)
// Pre:  item's key has been initialized.
//       An element in the list has a key that matches item's.
// Post: No element in the list has a key that matches item's.
{
    NodeType<ItemType>* iterator = listData->next;
    NodeType<ItemType>* back_iterator = listData;

    while (iterator != NULL)
    {
        if (item == iterator->info)
        {
            back_iterator->next = iterator->next;
            length--;
            delete iterator;
            iterator = back_iterator->next;
            continue;
        }
        back_iterator = iterator;
        iterator = iterator->next;
    }
    if (listData->info == item)
    {
        NodeType<ItemType>* temp_first_node = listData;
        listData = listData->next;
        length--;
        delete temp_first_node;
    }
```